IsaSAT: Heuristics and Code Generation

Mathias Fleury, Jasmin Blanchette, Peter Lammich

January 20, 2020

Contents

theory *IsaSAT-Literals*
imports *Watched-Literals.WB-More-Refinement HOL−Word.More-Word*
Watched-Literals.Watched-Literals-Watch-List
Entailment-Definition.Partial-Herbrand-Interpretation
Isabelle-LLVM.Bits-Natural
begin

Chapter 1

Refinement of Literals

1.1 Literals as Natural Numbers

1.1.1 Definition

lemma *Pos-div2-iff*:
  ‹*Pos* ((*bb* :: *nat*) *div 2*) = *b* ⟷ *is-pos b* ∧ (*bb* = *2* ∗ *atm-of b* ∨ *bb* = *2* ∗ *atm-of b* + *1*)›
  by (*cases b*) *auto*
lemma *Neg-div2-iff*:
  ‹*Neg* ((*bb* :: *nat*) *div 2*) = *b* ⟷ *is-neg b* ∧ (*bb* = *2* ∗ *atm-of b* ∨ *bb* = *2* ∗ *atm-of b* + *1*)›
  by (*cases b*) *auto*

Modeling *nat literal* via the transformation associating (*2*::′*a*) ∗ *n* or (*2*::′*a*) ∗ *n* + (*1*::′*a*) has some advantages over the transformation to positive or negative integers: 0 is not an issue. It is also a bit faster according to Armin Biere.

fun *nat-of-lit* :: ‹*nat literal* ⇒ *nat*› **where**
  ‹*nat-of-lit* (*Pos L*) = *2*∗*L*›
| ‹*nat-of-lit* (*Neg L*) = *2*∗*L* + *1*›

lemma *nat-of-lit-def*: ‹*nat-of-lit L* = (*if is-pos L then 2* ∗ *atm-of L else 2* ∗ *atm-of L* + *1*)›
  by (*cases L*) *auto*

fun *literal-of-nat* :: ‹*nat* ⇒ *nat literal*› **where**
  ‹*literal-of-nat n* = (*if even n then Pos* (*n div 2*) *else Neg* (*n div 2*))›

lemma *lit-of-nat-nat-of-lit*[*simp*]: ‹*literal-of-nat* (*nat-of-lit L*) = *L*›
  by (*cases L*) *auto*

lemma *nat-of-lit-lit-of-nat*[*simp*]: ‹*nat-of-lit* (*literal-of-nat n*) = *n*›
  by *auto*

lemma *atm-of-lit-of-nat*: ‹*atm-of* (*literal-of-nat n*) = *n div 2*›
  by *auto*

There is probably a more "closed" form from the following theorem, but it is unclear if that is useful or not.

lemma *uminus-lit-of-nat*:
  ‹− (*literal-of-nat n*) = (*if even n then literal-of-nat* (*n*+*1*) *else literal-of-nat* (*n*−*1*))›
  by (*auto elim*!: *oddE*)

lemma *literal-of-nat-literal-of-nat-eq*[*iff*]: ‹*literal-of-nat x* = *literal-of-nat xa* ⟷ *x* = *xa*›

by *auto presburger*+

definition *nat-lit-rel* :: ‹(*nat* × *nat literal*) *set*› **where**
‹*nat-lit-rel* = *br literal-of-nat* (λ-. *True*)›

lemma *ex-literal-of-nat*: ‹∃ *bb*. *b* = *literal-of-nat bb*›
by (*cases b*)
(*auto simp*: *nat-of-lit-def split*: *if-splits*; *presburger*; *fail*)+

1.1.2 Lifting to annotated literals

fun *pair-of-ann-lit* :: ‹(′*a*, ′*b*) *ann-lit* ⇒ ′*a literal* × ′*b option*› **where**
‹*pair-of-ann-lit* (*Propagated L D*) = (*L*, *Some D*)›
| ‹*pair-of-ann-lit* (*Decided L*) = (*L*, *None*)›

fun *ann-lit-of-pair* :: ‹′*a literal* × ′*b option* ⇒ (′*a*, ′*b*) *ann-lit*› **where**
‹*ann-lit-of-pair* (*L*, *Some D*) = *Propagated L D*›
| ‹*ann-lit-of-pair* (*L*, *None*) = *Decided L*›

lemma *ann-lit-of-pair-alt-def*:
‹*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))›
by (*cases D*) *auto*

lemma *ann-lit-of-pair-pair-of-ann-lit*: ‹*ann-lit-of-pair* (*pair-of-ann-lit L*) = *L*›
by (*cases L*) *auto*

lemma *pair-of-ann-lit-ann-lit-of-pair*: ‹*pair-of-ann-lit* (*ann-lit-of-pair L*) = *L*›
by (*cases L*; *cases* ‹*snd L*›) *auto*

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ‹*literal-of-nat b* = *L* ⟷ *b* = *nat-of-lit L*›
by (*auto simp del*: *literal-of-nat.simps*)

lemma *nat-of-lit-eq-iff*[*iff*]: ‹*nat-of-lit xa* = *nat-of-lit x* ⟷ *x* = *xa*›
apply (*cases x*; *cases xa*) **by** *auto presburger*+

definition *ann-lit-rel*:: ‹(′*a* × *nat*) *set* ⇒ (′*b* × *nat option*) *set* ⇒
((′*a* × ′*b*) × (*nat*, *nat*) *ann-lit*) *set*› **where**
ann-lit-rel-internal-def:
‹*ann-lit-rel R R*′ = {(*a*, *b*). ∃ *c d*. (*fst a*, *c*) ∈ *R* ∧ (*snd a*, *d*) ∈ *R*′ ∧
b = *ann-lit-of-pair* (*literal-of-nat c*, *d*)}›

1.2 Conflict Clause

definition *the-is-empty* **where**
‹*the-is-empty D* = *Multiset.is-empty* (*the D*)›

1.3 Atoms with bound

definition *uint32-max* :: *nat* **where**
‹*uint32-max* ≡ *2^32−1*›

definition *uint64-max* :: *nat* **where**
‹*uint64-max* ≡ *2^64−1*›

definition *sint32-max* :: *nat* **where**

‹$sint32\text{-}max \equiv 2\hat{\ }31-1$›

definition *sint64-max* :: *nat* **where**
‹$sint64\text{-}max \equiv 2\hat{\ }63-1$›

lemma *uint64-max-uint-def*: ‹$unat\ (-1 :: 64\ Word.word) = uint64\text{-}max$›
proof −
have ‹$unat\ (-1 :: 64\ Word.word) = unat\ (-\ Numeral1 :: 64\ Word.word)$›
unfolding *numeral.numeral-One* **..**
also have ‹$\ldots = uint64\text{-}max$›
unfolding *unat-bintrunc-neg*
apply (*simp add*: *uint64-max-def*)
apply (*subst numeral-eq-Suc*; *subst bintrunc.Suc*; *simp*)+
done
finally show *?thesis* **.**
qed

1.4 Operations with set of atoms.

context
fixes $\mathcal{A}_{in}$ :: ‹*nat multiset*›
begin

abbreviation D_0 :: ‹(*nat* × *nat literal*) *set*› **where**
‹$D_0 \equiv (\lambda L.\ (nat\text{-}of\text{-}lit\ L,\ L))$ ‘ $set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}_{in})$›

definition *length-ll-f* **where**
‹*length-ll-f W L* = *length* (*W L*)›

The following lemma was necessary at some point to prove the existence of some list.

lemma *ex-list-watched*:
fixes *W* :: ‹*nat literal* ⇒ *'a list*›
shows ‹$\exists\, aa.\ \forall\, x \in \# \mathcal{L}_{all}\ \mathcal{A}_{in}.\ nat\text{-}of\text{-}lit\ x < length\ aa \wedge aa\ !\ nat\text{-}of\text{-}lit\ x = W\ x$›
(**is** ‹∃ *aa*. *?P aa*›)
proof −
define D' **where** ‹$D' = D_0$›
define $\mathcal{L}_{all}{}'$ **where** ‹$\mathcal{L}_{all}{}' = \mathcal{L}_{all}$›
define D'' **where** ‹$D'' = mset\text{-}set\ (snd$ ‘ $D')$›
let *?f* = ‹(λ*L a*. *a*[*nat-of-lit L*:= *W L*])›
interpret *comp-fun-commute ?f*
apply *standard*
apply (*case-tac* ‹*y* = *x*›)
apply (*solves simp*)
apply (*intro ext*)
apply (*subst* (*asm*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*subst* (*asm*)(*3*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*clarsimp simp only*: *comp-def intro*!: *list-update-swap*)
done
define *aa* **where**
‹$aa \equiv fold\text{-}mset\ ?f\ (replicate\ (1+Max\ (nat\text{-}of\text{-}lit$ ‘ snd ‘ $D'))\ [])\ (mset\text{-}set\ (snd$ ‘ $D'))$›
have *length-fold*: ‹*length* (*fold-mset* (λ*L a*. *a*[*nat-of-lit L* := *W L*]) *l M*) = *length l*› **for** *l M*
by (*induction M*) *auto*
have *length-aa*: ‹$length\ aa = Suc\ (Max\ (nat\text{-}of\text{-}lit$ ‘ snd ‘ $D'))$›
unfolding *aa-def* D''*-def*[*symmetric*] **by** (*simp add*: *length-fold*)

have *H*: ‹$x \in\# \mathcal{L}_{all}' \Longrightarrow$
length l $\geq$ *Suc* (*Max* (*nat-of-lit* ‘ *set-mset* ($\mathcal{L}_{all}'$))) $\Longrightarrow$
fold-mset (λL *a*. *a*[*nat-of-lit L* := *W L*]) *l* (*remdups-mset* ($\mathcal{L}_{all}'$)) ! *nat-of-lit x* = *W x*›
for *x l* $\mathcal{L}_{all}'$
unfolding $\mathcal{L}_{all}'$-*def*[*symmetric*]
apply (*induction* $\mathcal{L}_{all}'$ *arbitrary*: *l*)
subgoal by *simp*
subgoal for *xa Ls l*
apply (*case-tac* ‹(*nat-of-lit* ‘ *set-mset Ls*) = {}›)
apply (*solves simp*)
apply (*auto simp*: *less-Suc-eq-le length-fold*)
done
done
have *H′*: ‹*aa* ! *nat-of-lit x* = *W x*› **if** ‹$x \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$› **for** *x*
using *that* **unfolding** *aa-def D′-def*
by (*auto simp*: *D′-def image-image remdups-mset-def*[*symmetric*]
less-Suc-eq-le intro!: *H*)
have ‹*?P aa*›
by (*auto simp*: *D′-def image-image remdups-mset-def*[*symmetric*]
less-Suc-eq-le length-aa H′)
then show *?thesis*
by *blast*
qed

definition *isasat-input-bounded* **where**
[*simp*]: ‹*isasat-input-bounded* = ($\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$. *nat-of-lit L* $\leq$ *uint32-max*)›

definition *isasat-input-nempty* **where**
[*simp*]: ‹*isasat-input-nempty* = (*set-mset* $\mathcal{A}_{in} \neq$ {})›

definition *isasat-input-bounded-nempty* **where**
‹*isasat-input-bounded-nempty* = (*isasat-input-bounded* $\wedge$ *isasat-input-nempty*)›

1.5 Set of atoms with bound

context
assumes *in*-$\mathcal{L}_{all}$-*less-uint32-max*: ‹*isasat-input-bounded*›
begin

lemma *in*-$\mathcal{L}_{all}$-*less-uint32-max′*: ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in} \Longrightarrow$ *nat-of-lit L* $\leq$ *uint32-max*›
using *in*-$\mathcal{L}_{all}$-*less-uint32-max* **by** *auto*

lemma *in*-$\mathcal{A}_{in}$-*less-than-uint32-max-div-2*:
‹$L \in\# \mathcal{A}_{in} \Longrightarrow L \leq$ *uint32-max div 2*›
using *in*-$\mathcal{L}_{all}$-*less-uint32-max′*[*of* ‹*Neg L*›]
unfolding *Ball-def atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *in*-$\mathcal{L}_{all}$-*atm-of-in-atms-of-iff*
by (*auto simp*: *uint32-max-def*)

lemma *simple-clss-size-upper-div2′*:
assumes
lits: ‹*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*› **and**
in-$\mathcal{L}_{all}$-*less-uint32-max*: ‹$\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$. *nat-of-lit L* < *uint32-max* − *1*›
shows ‹*size C* $\leq$ *uint32-max div 2*›

proof −
  let *?C* = ‹*atm-of* ‘# *C*›
  have ‹*distinct-mset ?C*›
  proof (*rule ccontr*)
    assume ‹¬ *?thesis*›
    then obtain *K* **where** ‹¬*count* (*atm-of* ‘# *C*) *K* ≤ *Suc 0*›
      unfolding *distinct-mset-count-less-1*
      by *auto*
    then have ‹*count* (*atm-of* ‘# *C*) *K* ≥ *2*›
      by *auto*
    then obtain *L L′ C′* **where**
      C: ‹*C* = {#*L*, *L′*#} + *C′*› **and** *L-L′*: ‹*atm-of L* = *atm-of L′*›
      by (*auto dest*!: *count-image-mset-multi-member-split-2*)
    then show *False*
      using *dist tauto* **by** (*auto simp*: *atm-of-eq-atm-of tautology-add-mset*)
  qed
  then have *card*: ‹*size ?C* = *card* (*set-mset ?C*)›
    using *distinct-mset-size-eq-card* **by** *blast*
  have *size*: ‹*size ?C* = *size C*›
    using *dist tauto*
    by (*induction C*) (*auto simp*: *tautology-add-mset*)
  have *m*: ‹*set-mset ?C* ⊆ {*0*..<*uint32-max div 2*}›
  proof
    fix *L*
    assume ‹*L* ∈ *set-mset ?C*›
    then have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
    using *lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹*Pos L* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
      using *lits* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
    then have ‹*nat-of-lit* (*Pos L*) < *uint32-max* − *1*›
      using *in-*$\mathcal{L}_{all}$*-less-uint32-max* **by** (*auto simp*: *atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹*L* < *uint32-max div 2*›
      by (*auto simp*: *atm-of-lit-in-atms-of*
        in-all-lits-of-m-ain-atms-of-iff subset-iff uint32-max-def)
    then show ‹*L* ∈ {*0*..<*uint32-max div 2*}›
      by (*auto simp*: *atm-of-lit-in-atms-of uint32-max-def*
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
  qed
  moreover have ‹*card* . . . = *uint32-max div 2*›
    by *auto*
  ultimately have ‹*card* (*set-mset ?C*) ≤ *uint32-max div 2*›
    using *card-mono*[*OF* - *m*] **by** *auto*
  then show *?thesis*
    unfolding *card*[*symmetric*] *size* **.**
qed

lemma *simple-clss-size-upper-div2*:
  assumes
    lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
    dist: ‹*distinct-mset C*› **and**
    tauto: ‹¬*tautology C*›
  shows ‹*size C* ≤ *1* + *uint32-max div 2*›
proof −

let *?C* = ‹*atm-of* ‘# *C*›
have ‹*distinct-mset ?C*›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *K* **where** ‹¬*count* (*atm-of* ‘# *C*) *K* ≤ *Suc 0*›
unfolding *distinct-mset-count-less-1*
by *auto*
then have ‹*count* (*atm-of* ‘# *C*) *K* ≥ *2*›
by *auto*
then obtain *L L′ C′* **where**
C: ‹*C* = {#*L*, *L′*#} + *C′*› **and** *L-L′*: ‹*atm-of L* = *atm-of L′*›
by (*auto dest*!: *count-image-mset-multi-member-split-2*)
then show *False*
using *dist tauto* **by** (*auto simp*: *atm-of-eq-atm-of tautology-add-mset*)
qed
then have *card*: ‹*size ?C* = *card* (*set-mset ?C*)›
using *distinct-mset-size-eq-card* **by** *blast*
have *size*: ‹*size ?C* = *size C*›
using *dist tauto*
by (*induction C*) (*auto simp*: *tautology-add-mset*)
have *m*: ‹*set-mset ?C* ⊆ {*0*..*uint32-max div 2*}›
proof
fix *L*
assume ‹*L* ∈ *set-mset ?C*›
then have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
using *lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
then have ‹*Neg L* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›
using *lits* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
then have ‹*nat-of-lit* (*Neg L*) ≤ *uint32-max*›
using *in-*$\mathcal{L}_{all}$*-less-uint32-max* **by** (*auto simp*: *atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
then have ‹*L* ≤ *uint32-max div 2*›
by (*auto simp*: *atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff uint32-max-def)
then show ‹*L* ∈ {*0* .. *uint32-max div 2*}›
by (*auto simp*: *atm-of-lit-in-atms-of uint32-max-def*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
qed
moreover have ‹*card* . . . = *1* + *uint32-max div 2*›
by *auto*
ultimately have ‹*card* (*set-mset ?C*) ≤ *1* + *uint32-max div 2*›
using *card-mono*[*OF - m*] **by** *auto*
then show *?thesis*
unfolding *card*[*symmetric*] *size* **.**
qed

lemma *clss-size-uint32-max*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*›
shows ‹*size C* ≤ *uint32-max* + *2*›
proof −
let *?posC* = ‹*filter-mset is-pos C*›
let *?negC* = ‹*filter-mset is-neg C*›
have *C*: ‹*C* = *?posC* + *?negC*›

apply (*subst multiset-partition*[*of* - *is-pos*])
by *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *?posC*›
by (*rule literals-are-in-*$\mathcal{L}_{in}$*-mono*[*OF lits*]) *auto*
moreover have ‹*distinct-mset ?posC*›
by (*rule distinct-mset-mono*[*OF -dist*]) *auto*
ultimately have *pos*: ‹*size ?posC* ≤ *1* + *uint32-max div 2*›
by (*rule simple-clss-size-upper-div2*) (*auto simp*: *tautology-decomp*)

have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *?negC*›
by (*rule literals-are-in-*$\mathcal{L}_{in}$*-mono*[*OF lits*]) *auto*
moreover have ‹*distinct-mset ?negC*›
by (*rule distinct-mset-mono*[*OF -dist*]) *auto*
ultimately have *neg*: ‹*size ?negC* ≤ *1* + *uint32-max div 2*›
by (*rule simple-clss-size-upper-div2*) (*auto simp*: *tautology-decomp*)

show *?thesis*
apply (*subst C*)
apply (*subst size-union*)
using *pos neg* **by** *linarith*
qed

lemma *clss-size-upper*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ *C*› **and**
dist: ‹*distinct-mset C*› **and**
in-$\mathcal{L}_{all}$*-less-uint32-max*: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* < *uint32-max* − *1*›
shows ‹*size C* ≤ *uint32-max*›
proof −
let *?A* = ‹*remdups-mset* (*atm-of* ‘# *C*)›
have [*simp*]: ‹*distinct-mset* (*poss ?A*)› ‹*distinct-mset* (*negs ?A*)›
by (*simp-all add*: *distinct-image-mset-inj inj-on-def*)

have ‹*C* ⊆# *poss ?A* + *negs ?A*›
apply (*rule distinct-subseteq-iff* [*THEN iffD1*])
subgoal by (*auto simp*: *dist distinct-mset-add disjunct-not-in*)
subgoal by (*auto simp*: *dist distinct-mset-add disjunct-not-in*)
subgoal
apply *rule*
using *literal.exhaust-sel* **by** (*auto simp*: *image-iff*)
done
have [*simp*]: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*poss ?A*)› ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*negs ?A*)›
using *lits*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-negs-remdups-mset literals-are-in-*$\mathcal{L}_{in}$*-poss-remdups-mset*)

have ‹¬ *tautology* (*poss ?A*)› ‹¬ *tautology* (*negs ?A*)›
by (*auto simp*: *tautology-decomp*)
then have ‹*size* (*poss ?A*) ≤ *uint32-max div 2*› **and** ‹*size* (*negs ?A*) ≤ *uint32-max div 2*›
using *simple-clss-size-upper-div2'*[*of* ‹*poss ?A*›]
simple-clss-size-upper-div2'[*of* ‹*negs ?A*›] *in-*$\mathcal{L}_{all}$*-less-uint32-max*
by *auto*
then have ‹*size C* ≤ *uint32-max div 2* + *uint32-max div 2*›
using ‹*C* ⊆# *poss* (*remdups-mset* (*atm-of* ‘# *C*)) + *negs* (*remdups-mset* (*atm-of* ‘# *C*))›
size-mset-mono **by** *fastforce*
then show *?thesis* **by** (*auto simp*: *uint32-max-def*)
qed

```
lemma
  assumes
    lits: ‹literals-are-in-ℒin-trail 𝒜in M› and
    n-d: ‹no-dup M›
  shows
    literals-are-in-ℒin-trail-length-le-uint32-max:
      ‹length M ≤ Suc (uint32-max div 2)› and
    literals-are-in-ℒin-trail-count-decided-uint32-max:
      ‹count-decided M ≤ Suc (uint32-max div 2)› and
    literals-are-in-ℒin-trail-get-level-uint32-max:
      ‹get-level M L ≤ Suc (uint32-max div 2)›
proof –
  have ‹length M = card (atm-of ‘ lits-of-l M)›
    using no-dup-length-eq-card-atm-of-lits-of-l[OF n-d] .
  moreover have ‹atm-of ‘ lits-of-l M ⊆ set-mset 𝒜in›
    using lits unfolding literals-are-in-ℒin-trail-atm-of by auto
  ultimately have ‹length M ≤ card (set-mset 𝒜in)›
    by (simp add: card-mono)
  moreover {
    have ‹set-mset 𝒜in ⊆ {0 ..< (uint32-max div 2) + 1}›
      using in-𝒜in-less-than-uint32-max-div-2 by (fastforce simp: in-ℒall-atm-of-in-atms-of-iff
          Ball-def atms-of-ℒall-𝒜in uint32-max-def)
    from subset-eq-atLeast0-lessThan-card[OF this] have ‹card (set-mset 𝒜in) ≤ uint32-max div 2 + 1›
      .
  }
  ultimately show ‹length M ≤ Suc (uint32-max div 2)›
    by linarith
  moreover have ‹count-decided M ≤ length M›
    unfolding count-decided-def by auto
  ultimately show ‹count-decided M ≤ Suc (uint32-max div 2)› by simp
  then show ‹get-level M L ≤ Suc (uint32-max div 2)›
    using count-decided-ge-get-level[of M L]
    by simp
qed

lemma length-trail-uint32-max-div2:
  fixes M :: ‹(nat, 'b) ann-lits›
  assumes
    M-ℒall: ‹∀ L∈set M. lit-of L ∈# ℒall 𝒜in› and
    n-d: ‹no-dup M›
  shows ‹length M ≤ uint32-max div 2 + 1›
proof –
  have dist-atm-M: ‹distinct-mset {#atm-of (lit-of x). x ∈# mset M#}›
    using n-d by (metis distinct-mset-mset-distinct mset-map no-dup-def)
  have incl: ‹atm-of ‘# lit-of ‘# mset M ⊆# remdups-mset (atm-of ‘# ℒall 𝒜in)›
    apply (subst distinct-subseteq-iff[THEN iffD1])
    using assms dist-atm-M
    by (auto 5 5 simp: Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct
        atm-of-eq-atm-of)
  have inj-on: ‹inj-on nat-of-lit (set-mset (remdups-mset (ℒall 𝒜in)))›
    by (auto simp: inj-on-def)
  have H: ‹xa ∈# ℒall 𝒜in ⟹ atm-of xa ≤ uint32-max div 2› for xa
    using in-ℒall-less-uint32-max
    by (cases xa) (auto simp: uint32-max-def)
  have ‹remdups-mset (atm-of ‘# ℒall 𝒜in) ⊆# mset [0..< 1 + (uint32-max div 2)]›
```

apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *H distinct-image-mset-inj*[*OF inj-on*]
by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))›
by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))) ≤ *1* + *uint32-max div 2*›
by *auto*
from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* ≤ *size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)))›
unfolding *uint32-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
with *2* **show** *?thesis*
by (*auto simp*: *uint32-max-def*)
qed

end

end

1.6 Instantion for code generation

instantiation *literal* :: (*default*) *default*
begin

definition *default-literal* **where**
‹*default-literal* = *Pos default*›
instance by *standard*

end

instantiation *fmap* :: (*type*, *type*) *default*
begin

definition *default-fmap* **where**
‹*default-fmap* = *fmempty*›
instance by *standard*

end

1.6.1 Literals as Natural Numbers

definition *propagated* **where**
‹*propagated L C* = (*L*, *Some C*)›

definition *decided* **where**
‹*decided L* = (*L*, *None*)›

definition *uminus-lit-imp* :: ‹*nat* ⇒ *nat*› **where**
‹*uminus-lit-imp L* = *bitXOR L 1*›

lemma *uminus-lit-imp-uminus*:
‹(*RETURN o uminus-lit-imp*, *RETURN o uminus*) ∈
nat-lit-rel $\rightarrow_f$ ⟨*nat-lit-rel*⟩*nres-rel*›
unfolding *bitXOR-1-if-mod-2 uminus-lit-imp-def*

by (*intro frefI nres-relI*) (*auto simp*: *uminus-lit-imp-def case-prod-beta p2rel-def br-def nat-lit-rel-def split*: *option.splits*, *presburger*)

1.6.2 State Conversion

Functions and Types:

More Operations

1.6.3 Code Generation

More Operations

definition *literals-to-update-wl-empty* :: ‹*nat twl-st-wl* ⇒ *bool*› **where**
‹*literals-to-update-wl-empty* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *Q* = {#})›

lemma *in-nat-list-rel-list-all2-in-set-iff*:
‹(*a*, *aa*) ∈ *nat-lit-rel* ⟹
list-all2 ($\lambda x\ x'$. (x, x') ∈ *nat-lit-rel*) *b ba* ⟹
a ∈ *set b* ⟷ *aa* ∈ *set ba*›
apply (*subgoal-tac* ‹*length b* = *length ba*›)
subgoal
apply (*rotate-tac 2*)
apply (*induction b ba rule*: *list-induct2*)
apply (*solves simp*)
apply (*auto simp*: *p2rel-def nat-lit-rel-def br-def*, *presburger*)[]
done
subgoal using *list-all2-lengthD* **by** *auto*
done

definition *is-decided-wl* **where**
‹*is-decided-wl L* ⟷ *snd L* = *None*›

lemma *ann-lit-of-pair-if*:
‹*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))›
by (*cases D*) *auto*

definition *get-maximum-level-remove* **where**
‹*get-maximum-level-remove M D L* = *get-maximum-level M* (*remove1-mset L D*)›

lemma *in-list-all2-ex-in*: ‹*a* ∈ *set xs* ⟹ *list-all2 R xs ys* ⟹ ∃ *b* ∈ *set ys*. *R a b*›
apply (*subgoal-tac* ‹*length xs* = *length ys*›)
apply (*rotate-tac 2*)
apply (*induction xs ys rule*: *list-induct2*)
apply ((*solves auto*)+)[*2*]
using *list-all2-lengthD* **by** *blast*

definition *find-decomp-wl-imp* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat literal* ⇒ (*nat*, *nat*) *ann-lits nres*› **where**
‹*find-decomp-wl-imp* = (λM_0 *D L*. *do* {
let lev = *get-maximum-level* M_0 (*remove1-mset* (−*L*) *D*);
let k = *count-decided* M_0;
(-, *M*) ←
$WHILE_T$$^{\lambda(j,\ M).\ j\ =\ count\text{-}decided\ M\ \wedge\ j\ \geq\ lev\ \wedge\ \qquad (M\ =\ []\ \longrightarrow\ j\ =\ lev)\ \wedge\ \qquad (\exists M'.\ M_0\ =\ M'\ @\ M\ \wedge\ (j\ =}$
(λ(*j*, *M*). *j* > *lev*)

(λ(*j*, *M*). *do* {
ASSERT(*M* ≠ []);
if is-decided (*hd M*)
then RETURN (*j*−*1*, *tl M*)
else RETURN (*j*, *tl M*)}
)
(*k*, M_0);
RETURN M
})›

lemma *ex-decomp-get-ann-decomposition-iff*:
‹(∃ *M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)) ⟷
(∃ *M2*. *M* = *M2* @ *Decided K* # *M1*)›
using *get-all-ann-decomposition-ex* **by** *fastforce*

lemma *count-decided-tl-if*:
‹*M* ≠ [] ⟹ *count-decided* (*tl M*) = (*if is-decided* (*hd M*) *then count-decided M* − *1 else count-decided M*)›
by (*cases M*) *auto*

lemma *count-decided-butlast*:
‹*count-decided* (*butlast xs*) = (*if is-decided* (*last xs*) *then count-decided xs* − *1 else count-decided xs*)›
by (*cases xs rule*: *rev-cases*) (*auto simp*: *count-decided-def*)

definition *find-decomp-wl′* **where**
‹*find-decomp-wl′* =
(λ(*M*::(*nat*, *nat*) *ann-lits*) (*D*::*nat clause*) (*L*::*nat literal*).
SPEC(λ*M1*. ∃ *K M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
get-level M K = *get-maximum-level M* (*D* − {#−*L*#}) + *1*))›

definition *get-conflict-wl-is-None* :: ‹*nat twl-st-wl* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *is-None D*)›

lemma *get-conflict-wl-is-None*: ‹*get-conflict-wl S* = *None* ⟷ *get-conflict-wl-is-None S*›
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-def split*: *option.splits*)

lemma *watched-by-nth-watched-app′*:
‹*watched-by S K* = ((*snd o snd o snd o snd o snd o snd o snd o snd*) *S*) *K*›
by (*cases S*) (*auto*)

lemma *hd-decided-count-decided-ge-1*:
‹*x* ≠ [] ⟹ *is-decided* (*hd x*) ⟹ *Suc 0* ≤ *count-decided x*›
by (*cases x*) *auto*

definition (**in** −) *find-decomp-wl-imp′* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clause-l list* ⇒ *nat* ⇒
nat clause ⇒ *nat clauses* ⇒ *nat clauses* ⇒ *nat lit-queue-wl* ⇒
(*nat literal* ⇒ *nat watched*) ⇒ - ⇒ (*nat*, *nat*) *ann-lits nres*› **where**
‹*find-decomp-wl-imp′* = (λ*M N U D NE UE W Q L*. *find-decomp-wl-imp M D L*)›

definition *is-decided-hd-trail-wl* **where**
‹*is-decided-hd-trail-wl S* = *is-decided* (*hd* (*get-trail-wl S*))›

definition *is-decided-hd-trail-wll* :: ‹*nat twl-st-wl* ⇒ *bool nres*› **where**
‹*is-decided-hd-trail-wll* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*).
RETURN (*is-decided* (*hd M*))
)›

lemma *Propagated-eq-ann-lit-of-pair-iff*:
  ‹*Propagated x21 x22 = ann-lit-of-pair* (*a*, *b*) ⟷ *x21 = a* ∧ *b = Some x22*›
  by (*cases b*) *auto*

lemma *set-mset-all-lits-of-mm-atms-of-ms-iff*:
  ‹*set-mset* (*all-lits-of-mm A*) = *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ⟷ *atms-of-ms* (*set-mset A*) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
  by (*force simp add*: *atms-of-s-def in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def*
      atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *atms-of-def atm-of-eq-atm-of uminus*-$\mathcal{A}_{in}$-*iff*
      eq-commute[*of* ‹*set-mset* (*all-lits-of-mm* -)› ‹*set-mset* ($\mathcal{L}_{all}$ -)›]
      dest: *multi-member-split*)

definition *card-max-lvl* **where**
  ‹*card-max-lvl M C* ≡ *size* (*filter-mset* (λ*L*. *get-level M L = count-decided M*) *C*)›

lemma *card-max-lvl-add-mset*: ‹*card-max-lvl M* (*add-mset L C*) =
  (*if get-level M L = count-decided M then 1 else 0*) +
    card-max-lvl M C›
  by (*auto simp*: *card-max-lvl-def*)

lemma *card-max-lvl-empty*[*simp*]: ‹*card-max-lvl M* {#} = *0*›
  by (*auto simp*: *card-max-lvl-def*)

lemma *card-max-lvl-all-poss*:
   ‹*card-max-lvl M C = card-max-lvl M* (*poss* (*atm-of* ‘# *C*))›
  unfolding *card-max-lvl-def*
  apply (*induction C*)
  subgoal by *auto*
  subgoal for *L C*
    using *get-level-uminus*[*of M L*]
    by (*cases L*) (*auto*)
  done

lemma *card-max-lvl-distinct-cong*:
  assumes
    ‹⋀*L*. *get-level M* (*Pos L*) = *count-decided M* ⟹ (*L* ∈ *atms-of C*) ⟹ (*L* ∈ *atms-of C'*)› **and**
    ‹⋀*L*. *get-level M* (*Pos L*) = *count-decided M* ⟹ (*L* ∈ *atms-of C'*) ⟹ (*L* ∈ *atms-of C*)› **and**
    ‹*distinct-mset C*› ‹¬*tautology C*› **and**
    ‹*distinct-mset C'*› ‹¬*tautology C'*›
  shows ‹*card-max-lvl M C = card-max-lvl M C'*›
proof −
  have [*simp*]: ‹*NO-MATCH* (*Pos x*) *L* ⟹ *get-level M L = get-level M* (*Pos* (*atm-of L*))› **for** *x L*
    by (*simp add*: *get-level-def*)
  have [*simp*]: ‹*atm-of L* ∉ *atms-of C'* ⟷ *L* ∉# *C'* ∧ −*L* ∉# *C'*› **for** *L C'*
    by (*cases L*) (*auto simp*: *atm-iff-pos-or-neg-lit*)
  then have [*iff*]: ‹*atm-of L* ∈ *atms-of C'* ⟷ *L* ∈# *C'* ∨ −*L* ∈# *C'*› **for** *L C'*
    by *blast*
  have *H*: ‹*distinct-mset* {#*L* ∈# *poss* (*atm-of* ‘# *C*). *get-level M L = count-decided M*#}›
    if ‹*distinct-mset C*› ‹¬*tautology C*› **for** *C*
    using *that* **by** (*induction C*) (*auto simp*: *tautology-add-mset atm-of-eq-atm-of*)
  show *?thesis*
    apply (*subst card-max-lvl-all-poss*)
    apply (*subst* (*2*) *card-max-lvl-all-poss*)
    unfolding *card-max-lvl-def*
    apply (*rule arg-cong*[*of* - - *size*])
    apply (*rule distinct-set-mset-eq*)

subgoal by (*rule H*) (*use assms* **in** *fast*)+
subgoal by (*rule H*) (*use assms* **in** *fast*)+
subgoal using *assms* **by** (*auto simp*: *atms-of-def imageI image-iff*) *blast*+
done
qed

end
theory *IsaSAT-Arena*
imports
Watched-Literals.WB-More-Refinement-List
IsaSAT-Literals
begin

Chapter 2

The memory representation: Arenas

We implement an "arena" memory representation: This is a flat representation of clauses, where all clauses and their headers are put one after the other. A lot of the work done here could be done automatically by a C compiler (see paragraph on Cadical below).

While this has some advantages from a performance point of view compared to an array of arrays, it allows to emulate pointers to the middle of array with extra information put before the pointer. This is an optimisation that is considered as important (at least according to Armin Biere).

In Cadical, the representation is done that way although it is implicit by putting an array into a structure (and rely on UB behaviour to make sure that the array is "inlined" into the structure). Cadical also uses another trick: the array is but inside a union. This union contains either the clause or a pointer to the new position if it has been moved (during GC-ing). There is no way for us to do so in a type-safe manner that works both for *uint64* and *nat* (unless we know some details of the implementation). For *uint64*, we could use the space used by the headers. However, it is not clear if we want to do do, since the behaviour would change between the two types, making a comparison impossible. This means that half of the blocking literals will be lost (if we iterate over the watch lists) or all (if we iterate over the clauses directly).

The order in memory is in the following order:

1. the saved position (was optional in cadical too; since sr-19, not optional);
2. the status;
3. the activity;
4. the LBD;
5. the size;
6. the clause.

Remark that the information can be compressed to reduce the size in memory:

1. the saved position can be skipped for short clauses;
2. the LBD will most of the time be much shorter than a 32-bit integer, so only an approximation can be kept and the remaining bits be reused;
3. the activity is not kept by cadical (to use instead a MTF-like scheme).

As we are already wasteful with memory, we implement the first optimisation. Point two can be implemented automatically by a (non-standard-compliant) C compiler.

In our case, the refinement is done in two steps:

1. First, we refine our clause-mapping to a big list. This list contains the original elements. For type safety, we introduce a datatype that enumerates all possible kind of elements.

2. Then, we refine all these elements to uint32 elements.

In our formalisation, we distinguish active clauses (clauses that are not marked to be deleted) from dead clauses (that have been marked to be deleted but can still be accessed). Any dead clause can be removed from the addressable clauses (*vdom* for virtual domain). Remark that we actually do not need the full virtual domain, just the list of all active position (TODO?).

Remark that in our formalisation, we don't (at least not yet) plan to reuse freed spaces (the predicate about dead clauses must be strengthened to do so). Due to the fact that an arena is very different from an array of clauses, we refine our data structure by hand to the long list instead of introducing refinement rules. This is mostly done because iteration is very different (and it does not change what we had before anyway).

Some technical details: due to the fact that we plan to refine the arena to uint32 and that our clauses can be tautologies, the size does not fit into uint32 (technically, we have the bound *uint32-max + 1*). Therefore, we restrict the clauses to have at least length 2 and we keep *length C − 2* instead of *length C* (same for position saving). If we ever add a preprocessing path that removes tautologies, we could get rid of these two limitations.

To our own surprise, using an arena (without position saving) was exactly as fast as the our former resizable array of arrays. We did not expect this result since:

1. First, we cannot use *uint32* to iterate over clauses anymore (at least no without an additional trick like considering a slice).

2. Second, there is no reason why MLton would not already use the trick for array.

(We assume that there is no gain due the order in which we iterate over clauses, which seems a reasonnable assumption, even when considering than some clauses will subsume the previous one, and therefore, have a high chance to be in the same watch lists).

We can mark clause as used. This trick is used to implement a MTF-like scheme to keep clauses.

2.1 Status of a clause

datatype *clause-status* = *IRRED* | *LEARNED* | *DELETED*

instantiation *clause-status* :: *default*
begin

definition *default-clause-status* **where** ⟨*default-clause-status* = *DELETED*⟩
instance by *standard*

end

2.2 Definition

The following definitions are the offset between the beginning of the clause and the specific headers before the beginning of the clause. Remark that the first offset is not always valid. Also remark that the fields are *before* the actual content of the clause.

definition *POS-SHIFT* :: *nat* **where**
⟨*POS-SHIFT = 5*⟩

definition *STATUS-SHIFT* :: *nat* **where**
⟨*STATUS-SHIFT = 4*⟩

definition *ACTIVITY-SHIFT* :: *nat* **where**
⟨*ACTIVITY-SHIFT = 3*⟩

definition *LBD-SHIFT* :: *nat* **where**
⟨*LBD-SHIFT = 2*⟩

definition *SIZE-SHIFT* :: *nat* **where**
⟨*SIZE-SHIFT = 1*⟩

definition *MAX-LENGTH-SHORT-CLAUSE* :: *nat* **where**
[*simp*]: ⟨*MAX-LENGTH-SHORT-CLAUSE = 4*⟩

definition *is-short-clause* **where**
[*simp*]: ⟨*is-short-clause C* $\longleftrightarrow$ *length C* $\leq$ *MAX-LENGTH-SHORT-CLAUSE*⟩

abbreviation *is-long-clause* **where**
⟨*is-long-clause C* $\equiv$ $\neg$*is-short-clause C*⟩

definition *header-size* :: ⟨*nat clause-l* $\Rightarrow$ *nat*⟩ **where**
⟨*header-size C* = (*if is-short-clause C then 4 else 5*)⟩

lemmas *SHIFTS-def = POS-SHIFT-def STATUS-SHIFT-def ACTIVITY-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def*

In an attempt to avoid unfolding definitions and to not rely on the actual value of the positions of the headers before the clauses.

lemma *arena-shift-distinct*:
⟨$i > 3 \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *LBD-SHIFT*⟩
⟨$i > 3 \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *ACTIVITY-SHIFT*⟩
⟨$i > 3 \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩
⟨$i > 3 \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *ACTIVITY-SHIFT*⟩
⟨$i > 3 \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩
⟨$i > 3 \Longrightarrow i -$ *ACTIVITY-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩

⟨$i > 4 \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i > 4 \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i > 4 \Longrightarrow i -$ *ACTIVITY-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i > 4 \Longrightarrow i -$ *STATUS-SHIFT* $\neq i -$ *POS-SHIFT*⟩

⟨$i > 3 \Longrightarrow j > 3 \Longrightarrow i -$ *SIZE-SHIFT* $= j -$ *SIZE-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i > 3 \Longrightarrow j > 3 \Longrightarrow i -$ *LBD-SHIFT* $= j -$ *LBD-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i > 4 \Longrightarrow j > 4 \Longrightarrow i -$ *ACTIVITY-SHIFT* $= j -$ *ACTIVITY-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i > 3 \Longrightarrow j > 3 \Longrightarrow i -$ *STATUS-SHIFT* $= j -$ *STATUS-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i > 4 \Longrightarrow j > 4 \Longrightarrow i -$ *POS-SHIFT* $= j -$ *POS-SHIFT* $\longleftrightarrow i = j$⟩

⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *LBD-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *ACTIVITY-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *ACTIVITY-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow i -$ *ACTIVITY-SHIFT* $\neq i -$ *STATUS-SHIFT*⟩

⟨$i \geq$ *header-size* $C \Longrightarrow$ *is-long-clause* $C \Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow$ *is-long-clause* $C \Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow$ *is-long-clause* $C \Longrightarrow i -$ *ACTIVITY-SHIFT* $\neq i -$ *POS-SHIFT*⟩
⟨$i \geq$ *header-size* $C \Longrightarrow$ *is-long-clause* $C \Longrightarrow i -$ *STATUS-SHIFT* $\neq i -$ *POS-SHIFT*⟩

⟨$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i -$ *SIZE-SHIFT* $= j -$ *SIZE-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i -$ *LBD-SHIFT* $= j -$ *LBD-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i -$ *ACTIVITY-SHIFT* $= j -$ *ACTIVITY-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow i -$ *STATUS-SHIFT* $= j -$ *STATUS-SHIFT* $\longleftrightarrow i = j$⟩
⟨$i \geq$ *header-size* $C \Longrightarrow j \geq$ *header-size* $C' \Longrightarrow$ *is-long-clause* $C \Longrightarrow$ *is-long-clause* $C' \Longrightarrow$
$i -$ *POS-SHIFT* $= j -$ *POS-SHIFT* $\longleftrightarrow i = j$⟩
unfolding *POS-SHIFT-def STATUS-SHIFT-def ACTIVITY-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def header-size-def*
by (*auto split*: *if-splits simp*: *is-short-clause-def*)

lemma *header-size-ge0*[*simp*]: ⟨$0 <$ *header-size x1*⟩
by (*auto simp*: *header-size-def*)

datatype *arena-el* =
is-Lit: *ALit* (*xarena-lit*: ⟨*nat literal*⟩) |
is-LBD: *ALBD* (*xarena-lbd*: *nat*) |
is-Act: *AActivity* (*xarena-act*: *nat*) |
is-Size: *ASize* (*xarena-length*: *nat*) |
is-Pos: *APos* (*xarena-pos*: *nat*) |
is-Status: *AStatus* (*xarena-status*: *clause-status*) (*xarena-used*: *bool*)

type-synonym *arena* = ⟨*arena-el list*⟩

definition *xarena-active-clause* :: ⟨*arena* $\Rightarrow$ *nat clause-l* $\times$ *bool* $\Rightarrow$ *bool*⟩ **where**
⟨*xarena-active-clause arena* = (λ(C, *red*).
(*length* $C \geq 2$ $\wedge$
header-size C + *length* C = *length arena* $\wedge$
(*is-long-clause* $C \longrightarrow$ (*is-Pos* (*arena*!(*header-size* $C -$ *POS-SHIFT*)) $\wedge$
xarena-pos(*arena*!(*header-size* $C -$ *POS-SHIFT*)) $\leq$ *length* $C - 2$))) $\wedge$
is-Status(*arena*!(*header-size* $C -$ *STATUS-SHIFT*)) $\wedge$
(*xarena-status*(*arena*!(*header-size* $C -$ *STATUS-SHIFT*)) = *IRRED* $\longleftrightarrow$ *red*) $\wedge$
(*xarena-status*(*arena*!(*header-size* $C -$ *STATUS-SHIFT*)) = *LEARNED* $\longleftrightarrow \neg$*red*) $\wedge$
is-LBD(*arena*!(*header-size* $C -$ *LBD-SHIFT*)) $\wedge$
is-Act(*arena*!(*header-size* $C -$ *ACTIVITY-SHIFT*)) $\wedge$
is-Size(*arena*!(*header-size* $C -$ *SIZE-SHIFT*)) $\wedge$
xarena-length(*arena*!(*header-size* $C -$ *SIZE-SHIFT*)) + 2 = *length* C $\wedge$
drop (*header-size* C) *arena* = *map ALit* C
)⟩

As ($N \propto i$, *irred* N i) is automatically simplified to *the* (*fmlookup* N i), we provide an alternative definition that uses the result after the simplification.

lemma *xarena-active-clause-alt-def*:
‹*xarena-active-clause arena* (*the* (*fmlookup N i*)) ⟷ (
(*length* (*N*∝*i*) ≥ *2* ∧
header-size (*N*∝*i*) + *length* (*N*∝*i*) = *length arena* ∧
(*is-long-clause* (*N*∝*i*) ⟶ (*is-Pos* (*arena*!(*header-size* (*N*∝*i*) − *POS-SHIFT*)) ∧
xarena-pos(*arena*!(*header-size* (*N*∝*i*) − *POS-SHIFT*)) ≤ *length* (*N*∝*i*) − *2*)) ∧
is-Status(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) ∧
(*xarena-status*(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) = *IRRED* ⟷ *irred N i*) ∧
(*xarena-status*(*arena*!(*header-size* (*N*∝*i*) − *STATUS-SHIFT*)) = *LEARNED* ⟷ ¬*irred N i*) ∧
is-LBD(*arena*!(*header-size* (*N*∝*i*) − *LBD-SHIFT*)) ∧
is-Act(*arena*!(*header-size* (*N*∝*i*) − *ACTIVITY-SHIFT*)) ∧
is-Size(*arena*!(*header-size* (*N*∝*i*) − *SIZE-SHIFT*)) ∧
xarena-length(*arena*!(*header-size* (*N*∝*i*) − *SIZE-SHIFT*)) + *2* = *length* (*N*∝*i*) ∧
drop (*header-size* (*N*∝*i*)) *arena* = *map ALit* (*N*∝*i*)
))›
proof −
have *C*: ‹*the* (*fmlookup N i*) = (*N* ∝ *i*, *irred N i*)›
by *simp*
show *?thesis*
apply (*subst C*)
unfolding *xarena-active-clause-def prod.case*
by *meson*
qed

The extra information is required to prove "separation" between active and dead clauses. And it is true anyway and does not require any extra work to prove. TODO generalise LBD to extract from every clause?

definition *arena-dead-clause* :: ‹*arena* ⇒ *bool*› **where**
‹*arena-dead-clause arena* ⟷
is-Status(*arena*!(*4* − *STATUS-SHIFT*)) ∧ *xarena-status*(*arena*!(*4* − *STATUS-SHIFT*)) = *DELETED* ∧
is-LBD(*arena*!(*4* − *LBD-SHIFT*)) ∧
is-Act(*arena*!(*4* − *ACTIVITY-SHIFT*)) ∧
is-Size(*arena*!(*4* − *SIZE-SHIFT*))
›

When marking a clause as garbage, we do not care whether it was used or not.

definition *extra-information-mark-to-delete* **where**
‹*extra-information-mark-to-delete arena i* = *arena*[*i* − *STATUS-SHIFT* := *AStatus DELETED False*]›

This extracts a single clause from the complete arena.

abbreviation *clause-slice* **where**
‹*clause-slice arena N i* ≡ *Misc.slice* (*i* − *header-size* (*N*∝*i*)) (*i* + *length*(*N*∝*i*)) *arena*›

abbreviation *dead-clause-slice* **where**
‹*dead-clause-slice arena N i* ≡ *Misc.slice* (*i* − *4*) *i arena*›

We now can lift the validity of the active and dead clauses to the whole memory and link it the mapping to clauses and the addressable space.

In our first try, the predicated *xarena-active-clause* took the whole arena as parameter. This however turned out to make the proof about updates less modular, since the slicing already takes care to ignore all irrelevant changes.

definition *valid-arena* :: ‹*arena* ⇒ *nat clauses-l* ⇒ *nat set* ⇒ *bool*› **where**
‹*valid-arena arena N vdom* ⟷

$(\forall i \in\# \ dom\text{-}m\ N.\ i < length\ arena \wedge i \geq header\text{-}size\ (N{\propto}i) \wedge$
$xarena\text{-}active\text{-}clause\ (clause\text{-}slice\ arena\ N\ i)\ (the\ (fmlookup\ N\ i))) \wedge$
$(\forall i \in vdom.\ i \notin\# \ dom\text{-}m\ N \longrightarrow (i < length\ arena \wedge i \geq 4 \wedge$
$arena\text{-}dead\text{-}clause\ (dead\text{-}clause\text{-}slice\ arena\ N\ i)))$
›

lemma *valid-arena-empty*: ‹*valid-arena* [] *fmempty* {}›
unfolding *valid-arena-def*
by *auto*

definition *arena-status* **where**
‹$arena\text{-}status\ arena\ i = xarena\text{-}status\ (arena!(i - STATUS\text{-}SHIFT))$›

definition *arena-used* **where**
‹$arena\text{-}used\ arena\ i = xarena\text{-}used\ (arena!(i - STATUS\text{-}SHIFT))$›

definition *arena-length* **where**
‹$arena\text{-}length\ arena\ i = 2 + xarena\text{-}length\ (arena!(i - SIZE\text{-}SHIFT))$›

definition *arena-lbd* **where**
‹$arena\text{-}lbd\ arena\ i = xarena\text{-}lbd\ (arena!(i - LBD\text{-}SHIFT))$›

definition *arena-act* **where**
‹$arena\text{-}act\ arena\ i = xarena\text{-}act\ (arena!(i - ACTIVITY\text{-}SHIFT))$›

definition *arena-pos* **where**
‹$arena\text{-}pos\ arena\ i = 2 + xarena\text{-}pos\ (arena!(i - POS\text{-}SHIFT))$›

definition *arena-lit* **where**
‹$arena\text{-}lit\ arena\ i = xarena\text{-}lit\ (arena!i)$›

definition *op-incr-mod32* $n \equiv (n{+}1 :: nat)\ mod\ 2\hat{}32$

definition *arena-incr-act* **where**
‹$arena\text{-}incr\text{-}act\ arena\ i = arena[i - ACTIVITY\text{-}SHIFT := AActivity\ (op\text{-}incr\text{-}mod32\ (xarena\text{-}act\ (arena!(i - ACTIVITY\text{-}SHIFT))))]$›

2.3 Separation properties

The following two lemmas talk about the minimal distance between two clauses in memory. They are important for the proof of correctness of all update function.

lemma *minimal-difference-between-valid-index*:
assumes ‹$\forall i \in\# \ dom\text{-}m\ N.\ i < length\ arena \wedge i \geq header\text{-}size\ (N{\propto}i) \wedge$
$xarena\text{-}active\text{-}clause\ (clause\text{-}slice\ arena\ N\ i)\ (the\ (fmlookup\ N\ i))$› **and**
‹$i \in\# \ dom\text{-}m\ N$› **and** ‹$j \in\# \ dom\text{-}m\ N$› **and** ‹$j > i$›
shows ‹$j - i \geq length\ (N{\propto}i) + header\text{-}size\ (N{\propto}j)$›
proof (*rule ccontr*)
assume *False*: ‹¬ *?thesis*›
let *?Ci* = ‹*the* (*fmlookup N i*)›
let *?Cj* = ‹*the* (*fmlookup N j*)›
have
1: ‹$xarena\text{-}active\text{-}clause\ (clause\text{-}slice\ arena\ N\ i)\ (N \propto i,\ irred\ N\ i)$› **and**
2: ‹$xarena\text{-}active\text{-}clause\ (clause\text{-}slice\ arena\ N\ j)\ (N \propto j,\ irred\ N\ j)$› **and**
i-le: ‹$i < length\ arena$› **and**

i-ge: ‹$i \geq$ *header-size*$(N \propto i)$›**and**
j-le: ‹$j <$ *length arena*› **and**
j-ge: ‹$j \geq$ *header-size*$(N \propto j)$›
using *assms*
by *auto*

have *Ci*: ‹$?Ci = (N \propto i,$ *irred N i*$)$› **and** *Cj*: ‹$?Cj = (N \propto j,$ *irred N j*$)$›
by *auto*

have
eq: ‹*Misc.slice i* $(i +$ *length* $(N \propto i))$ *arena* $=$ *map ALit* $(N \propto i)$› **and**
‹*length* $(N \propto i) -$ *Suc 0* $<$ *length* $(N \propto i)$› **and**
length-Ni: ‹*length* $(N \propto i) \geq 2$›
using *1 i-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
apply *simp-all*
apply *force*
done

from *arg-cong*[*OF this*(*1*), *of* ‹$\lambda n.\ n$! (*length* $(N \propto i) - 1)$›] *this*(*2*−)
have *lit*: ‹*is-Lit* (*arena* ! $(i +$ *length*$(N \propto i) - 1))$›
using *i-le i-ge* **by** (*auto simp*: *map-nth slice-nth*)
have
Cj2: ‹$2 \leq$ *length* $(N \propto j)$›
using *2 j-le j-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
header-size-def
by *simp*
have *headerj*: ‹*header-size* $(N \propto j) \geq 4$›
unfolding *header-size-def* **by** (*auto split*: *if-splits*)
then have [*simp*]: ‹*header-size* $(N \propto j) -$ *POS-SHIFT* $<$ *length* $(N \propto j) +$ *header-size* $(N \propto j)$›
using *Cj2*
by *linarith*
have [*simp*]:
‹*is-long-clause* $(N \propto j) \longrightarrow j +$ (*header-size* $(N \propto j) -$ *POS-SHIFT*) $-$ *header-size* $(N \propto j) = j -$ *POS-SHIFT*›
‹$j +$ (*header-size* $(N \propto j) -$ *STATUS-SHIFT*) $-$ *header-size* $(N \propto j) = j -$ *STATUS-SHIFT*›
‹$j +$ (*header-size* $(N \propto j) -$ *SIZE-SHIFT*) $-$ *header-size* $(N \propto j) = j -$ *SIZE-SHIFT*›
‹$j +$ (*header-size* $(N \propto j) -$ *LBD-SHIFT*) $-$ *header-size* $(N \propto j) = j -$ *LBD-SHIFT*›
‹$j +$ (*header-size* $(N \propto j) -$ *ACTIVITY-SHIFT*) $-$ *header-size* $(N \propto j) = j -$ *ACTIVITY-SHIFT*›
using *Cj2 headerj* **unfolding** *POS-SHIFT-def STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def ACTIVITY-SHIFT-def*
by (*auto simp*: *header-size-def*)

have
pos: ‹*is-long-clause* $(N \propto j) \longrightarrow$ *is-Pos* (*arena* ! $(j -$ *POS-SHIFT*))› **and**
st: ‹*is-Status* (*arena* ! $(j -$ *STATUS-SHIFT*))› **and**
size: ‹*is-Size* (*arena* ! $(j -$ *SIZE-SHIFT*))› **and**
lbd: ‹*is-LBD* (*arena* ! $(j -$ *LBD-SHIFT*))› **and**
act: ‹*is-Act* (*arena* ! $(j -$ *ACTIVITY-SHIFT*))›
using *2 j-le j-ge Cj2 headerj*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
by (*simp-all add*: *slice-nth*)
have *False* **if** *ji*: ‹$j - i \geq$ *length* $(N \propto i)$›
proof −
have *Suc3*: ‹$3 =$ *Suc* (*Suc* (*Suc 0*))›

by *auto*
have *Suc4*: ‹$4 = Suc\ (Suc\ (Suc\ (Suc\ 0)))$›
by *auto*
have *Suc5*: ‹$5 = Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))$›
by *auto*
have *j-i-1*[*iff*]:
‹$j - 1 = i + length\ (N \propto i) - 1 \longleftrightarrow j = i + length\ (N \propto i)$›
‹$j - 2 = i + length\ (N \propto i) - 1 \longleftrightarrow j = i + length\ (N \propto i) + 1$›
‹$j - 3 = i + length\ (N \propto i) - 1 \longleftrightarrow j = i + length\ (N \propto i) + 2$›
‹$j - 4 = i + length\ (N \propto i) - 1 \longleftrightarrow j = i + length\ (N \propto i) + 3$›
‹$j - 5 = i + length\ (N \propto i) - 1 \longleftrightarrow j = i + length\ (N \propto i) + 4$›
using *False that j-ge i-ge length-Ni* **unfolding** *Suc4 Suc5 header-size-def numeral-2-eq-2*
by (*auto split*: *if-splits*)
have *H4*: ‹$Suc\ (j - i) \leq length\ (N \propto i) + 4 \Longrightarrow j - i = length\ (N \propto i) \vee$
$j - i = length\ (N \propto i) + 1 \vee j - i = length\ (N \propto i) + 2 \vee j - i = length\ (N \propto i) + 3$›
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
have *H5*: ‹$Suc\ (j - i) \leq length\ (N \propto i) + 5 \Longrightarrow j - i = length\ (N \propto i) \vee$
$j - i = length\ (N \propto i) + 1 \vee j - i = length\ (N \propto i) + 2 \vee j - i = length\ (N \propto i) + 3 \vee$
$(is\text{-}long\text{-}clause\ (N \propto j) \wedge j = i{+}length\ (N \propto i) + 4)$›
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
consider
‹$is\text{-}long\text{-}clause\ (N \propto j)$› ‹$j - POS\text{-}SHIFT = i + length(N{\propto}i) - 1$› |
‹$j - STATUS\text{-}SHIFT = i + length(N{\propto}i) - 1$› |
‹$j - LBD\text{-}SHIFT = i + length(N{\propto}i) - 1$› |
‹$j - ACTIVITY\text{-}SHIFT = i + length(N{\propto}i) - 1$› |
‹$j - SIZE\text{-}SHIFT = i + length(N{\propto}i) - 1$›
using *False ji j-ge i-ge length-Ni*
unfolding *header-size-def not-less-eq-eq STATUS-SHIFT-def SIZE-SHIFT-def*
LBD-SHIFT-def ACTIVITY-SHIFT-def le-Suc-eq POS-SHIFT-def j-i-1
apply (*cases* ‹$is\text{-}short\text{-}clause\ (N \propto j)$›)
subgoal
using *H4* **by** *auto*
subgoal
using *H5* **by** *auto*
done
then show *False*
using *lit pos st size lbd act*
by *cases auto*
qed
moreover have *False* **if** *ji*: ‹$j - i < length\ (N{\propto}i)$›
proof −
from *arg-cong*[*OF eq*, *of* ‹$\lambda xs.\ xs\ !\ (j{-}i{-}1)$›]
have ‹$is\text{-}Lit\ (arena\ !\ (j{-}1))$›
using *that j-le i-le* ‹$j > i$›
by (*auto simp*: *slice-nth*)
then show *False*
using *size* **unfolding** *SIZE-SHIFT-def* **by** *auto*
qed
ultimately show *False*
by *linarith*
qed

lemma *minimal-difference-between-invalid-index*:
assumes ‹*valid-arena arena N vdom*› **and**

$\langle i \in\# \ dom\text{-}m\ N\rangle$ **and** $\langle j \notin\# \ dom\text{-}m\ N\rangle$ **and** $\langle j \geq i\rangle$ **and** $\langle j \in vdom\rangle$
shows $\langle j - i \geq length\ (N{\propto}i) + 4\rangle$
proof (*rule ccontr*)
assume *False*: $\langle\neg\ ?thesis\rangle$
let $?Ci = \langle the\ (fmlookup\ N\ i)\rangle$
let $?Cj = \langle the\ (fmlookup\ N\ j)\rangle$
have
1: ⟨*xarena-active-clause* (*clause-slice arena N i*) ($N \propto i$, *irred N i*)⟩ **and**
2: ⟨*arena-dead-clause* (*dead-clause-slice arena N j*)⟩ **and**
i-le: $\langle i < length\ arena\rangle$ **and**
i-ge: $\langle i \geq header\text{-}size(N{\propto}i)\rangle$**and**
j-le: $\langle j < length\ arena\rangle$ **and**
j-ge: $\langle j \geq 4\rangle$
using *assms* **unfolding** *valid-arena-def*
by *auto*

have *Ci*: ⟨$?Ci = (N \propto i,$ *irred N i*)⟩ **and** *Cj*: ⟨$?Cj = (N \propto j,$ *irred N j*)⟩
by *auto*

have
eq: ⟨*Misc.slice i* (i + *length* ($N \propto i$)) *arena* = *map ALit* ($N \propto i$)⟩ **and**
⟨*length* ($N \propto i$) − *Suc 0* < *length* ($N \propto i$)⟩ **and**
length-Ni: $\langle length\ (N{\propto}i) \geq 2\rangle$ **and**
pos: ⟨*is-long-clause* ($N \propto i$) $\longrightarrow$
is-Pos (*arena* ! (i − *POS-SHIFT*))⟩ **and**
status: ⟨*is-Status* (*arena* ! (i − *STATUS-SHIFT*))⟩ **and**
lbd: ⟨*is-LBD* (*arena* ! (i − *LBD-SHIFT*))⟩ **and**
act: ⟨*is-Act* (*arena* ! (i − *ACTIVITY-SHIFT*))⟩ **and**
size: ⟨*is-Size* (*arena* ! (i − *SIZE-SHIFT*))⟩ **and**
st-init: ⟨(*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *IRRED*) = (*irred N i*)⟩ **and**
st-learned: ⟨(*xarena-status* (*arena* ! (i − *STATUS-SHIFT*)) = *LEARNED*) = (¬ *irred N i*)⟩
using *1 i-ge i-le*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
unfolding *STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def*
apply (*simp-all add*: *header-size-def slice-nth split*: *if-splits*)
apply *force+*
done

have
st: ⟨*is-Status* (*arena* ! (j − *STATUS-SHIFT*))⟩ **and**
del: ⟨*xarena-status* (*arena* ! (j − *STATUS-SHIFT*)) = *DELETED*⟩
using *2 j-le j-ge* **unfolding** *arena-dead-clause-def STATUS-SHIFT-def*
by (*simp-all add*: *header-size-def slice-nth*)
consider
⟨j − *STATUS-SHIFT* ≥ i⟩ |
⟨j − *STATUS-SHIFT* < i⟩
using *False* $\langle j \geq i\rangle$ **unfolding** *STATUS-SHIFT-def*
by *linarith*
then show *False*
proof *cases*
case *1*
then have ⟨j − *STATUS-SHIFT* < i + *length* ($N{\propto}i$)⟩
using $\langle j \geq i\rangle$ *False j-ge*
unfolding *not-less-eq-eq STATUS-SHIFT-def*
by *simp*
with *arg-cong*[*OF eq*, *of* ⟨$\lambda n.\ n$! (j − *STATUS-SHIFT* − i)⟩]

have *lit*: ‹*is-Lit* (*arena* ! (*j* − *STATUS-SHIFT*))›
  using *1* ‹*j* ≥ *i*› *i-le i-ge j-ge* **by** (*auto simp*: *map-nth slice-nth STATUS-SHIFT-def*)
with *st*
show *False* **by** *auto*
next
case *2*
then consider
‹*j* − *STATUS-SHIFT* = *i* − *STATUS-SHIFT*› |
‹*j* − *STATUS-SHIFT* = *i* − *LBD-SHIFT*› |
‹*j* − *STATUS-SHIFT* = *i* − *ACTIVITY-SHIFT*› |
‹*j* − *STATUS-SHIFT* = *i* − *SIZE-SHIFT*› |
‹*is-long-clause* (*N* ∝ *i*)› **and** ‹*j* − *STATUS-SHIFT* = *i* − *POS-SHIFT*›
using ‹*j* ≥ *i*›
unfolding *STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def*
by *force*
then show *False*
apply *cases*
subgoal using *st status st-init st-learned del* **by** *auto*
subgoal using *st lbd* **by** *auto*
subgoal using *st act* **by** *auto*
subgoal using *st size* **by** *auto*
subgoal using *st pos* **by** *auto*
done
qed
qed

At first we had the weaker $(1::'a) \leq i - j$ which we replaced by $(4::'a) \leq i - j$. The former however was able to solve many more goals due to different handling between $1::'a$ (which is simplified to *Suc 0*) and $4::'a$ (whi::natch is not). Therefore, we replaced $4::'a$ by *Suc* (*Suc* (*Suc* (*Suc 0*)))

lemma *minimal-difference-between-invalid-index2*:
assumes ‹*valid-arena arena N vdom*› **and**
‹*i* ∈# *dom-m N*› **and** ‹*j* ∉# *dom-m N*› **and** ‹*j* < *i*› **and** ‹*j* ∈ *vdom*›
shows ‹*i* − *j* ≥ *Suc* (*Suc* (*Suc* (*Suc 0*)))› **and**
‹*is-long-clause* (*N* ∝ *i*) ⟹ *i* − *j* ≥ *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))›
proof −
let *?Ci* = ‹*the* (*fmlookup N i*)›
let *?Cj* = ‹*the* (*fmlookup N j*)›
have
1: ‹*xarena-active-clause* (*clause-slice arena N i*) (*N* ∝ *i*, *irred N i*)› **and**
2: ‹*arena-dead-clause* (*dead-clause-slice arena N j*)› **and**
i-le: ‹*i* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N*∝*i*)›**and**
j-le: ‹*j* < *length arena*› **and**
j-ge: ‹*j* ≥ *4*›
using *assms* **unfolding** *valid-arena-def*
by *auto*

have *Ci*: ‹*?Ci* = (*N* ∝ *i*, *irred N i*)› **and** *Cj*: ‹*?Cj* = (*N* ∝ *j*, *irred N j*)›
by *auto*

have
eq: ‹*Misc.slice i* (*i* + *length* (*N* ∝ *i*)) *arena* = *map ALit* (*N* ∝ *i*)› **and**
‹*length* (*N* ∝ *i*) − *Suc 0* < *length* (*N* ∝ *i*)› **and**
length-Ni: ‹*length* (*N*∝*i*) ≥ *2*› **and**

```
  pos: ‹is-long-clause (N ∝ i) ⟶
     is-Pos (arena ! (i - POS-SHIFT))› and
  status: ‹is-Status (arena ! (i - STATUS-SHIFT))› and
  lbd: ‹is-LBD (arena ! (i - LBD-SHIFT))› and
  act: ‹is-Act (arena ! (i - ACTIVITY-SHIFT))› and
  size: ‹is-Size (arena ! (i - SIZE-SHIFT))› and
  st-init: ‹(xarena-status (arena ! (i - STATUS-SHIFT)) = IRRED) ⟷ (irred N i)› and
  st-learned: ‹ (xarena-status (arena ! (i - STATUS-SHIFT)) = LEARNED) ⟷ ¬irred N i›
  using 1 i-ge i-le
  unfolding xarena-active-clause-def extra-information-mark-to-delete-def prod.case
  unfolding STATUS-SHIFT-def LBD-SHIFT-def ACTIVITY-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
   apply (simp-all add: header-size-def slice-nth split: if-splits)
  apply force+
  done

have
  st: ‹is-Status (arena ! (j - STATUS-SHIFT))› and
  del: ‹xarena-status (arena ! (j - STATUS-SHIFT)) = DELETED› and
  lbd': ‹is-LBD (arena ! (j - LBD-SHIFT))› and
  act': ‹is-Act (arena ! (j - ACTIVITY-SHIFT))› and
  size': ‹is-Size (arena ! (j - SIZE-SHIFT))›
  using 2 j-le j-ge unfolding arena-dead-clause-def SHIFTS-def
  by (simp-all add: header-size-def slice-nth)
have 4: ‹4 = Suc (Suc (Suc (Suc 0)))›  and 5: ‹5 = Suc (Suc (Suc (Suc (Suc 0))))›
  by auto
have [simp]: ‹a < 4 ⟹ j - Suc a = i - Suc 0 ⟷ i = j - a› for a
  using ‹i > j› j-ge i-ge
  by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq )
have [simp]: ‹Suc i - j = Suc a ⟷ i - j = a› for a
  using ‹i > j› j-ge i-ge
  by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq)


show 1: ‹i - j ≥ Suc (Suc (Suc (Suc 0)))› (is ?A)
proof (rule ccontr)
  assume False: ‹¬?A›
  consider
      ‹i - STATUS-SHIFT = j - STATUS-SHIFT› |
      ‹i - STATUS-SHIFT = j - LBD-SHIFT› |
      ‹i - STATUS-SHIFT = j - ACTIVITY-SHIFT› |
      ‹i - STATUS-SHIFT = j - SIZE-SHIFT›
    using False ‹i > j› j-ge i-ge unfolding SHIFTS-def header-size-def 4
    by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq )
  then show False
    apply cases
    subgoal using st status st-init st-learned del by auto
    subgoal using status lbd' by auto
    subgoal using status act' by auto
    subgoal using status size' by auto
    done
qed

show ‹i - j ≥ Suc (Suc (Suc (Suc (Suc 0))))› (is ?A)
  if long: ‹is-long-clause (N ∝ i)›
proof (rule ccontr)
  assume False: ‹¬?A›
```

have [*simp*]: ‹$a < 5 \Longrightarrow a' < 4 \Longrightarrow i - Suc\ a = j - Suc\ a' \longleftrightarrow i - a = j - a'$› **for** *a a'*
using ‹$i > j$› *j-ge i-ge long*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
have ‹$i - j = Suc\ (Suc\ (Suc\ (Suc\ 0)))$›
using *1* ‹$i > j$› *False j-ge i-ge long* **unfolding** *SHIFTS-def header-size-def 4*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
then have ‹$i - POS\text{-}SHIFT = j - SIZE\text{-}SHIFT$›
using *1* ‹$i > j$› *j-ge i-ge long* **unfolding** *SHIFTS-def header-size-def 4 5*
by (*auto split*: *if-splits simp*: *not-less-eq-eq le-Suc-eq*)
then show *False*
using *pos long size'*
by *auto*
qed
qed

lemma *valid-arena-in-vdom-le-arena*:
assumes ‹*valid-arena arena N vdom*› **and** ‹$j \in vdom$›
shows ‹$j <$ *length arena*› **and** ‹$j \geq 4$›
using *assms* **unfolding** *valid-arena-def*
by (*cases* ‹$j \in\#$ *dom-m N*›; *auto simp*: *header-size-def*
dest!: *multi-member-split split*: *if-splits*; *fail*)+

lemma *valid-minimal-difference-between-valid-index*:
assumes ‹*valid-arena arena N vdom*› **and**
‹$i \in\#$ *dom-m N*› **and** ‹$j \in\#$ *dom-m N*› **and** ‹$j > i$›
shows ‹$j - i \geq$ *length* $(N \propto i)$ + *header-size* $(N \propto j)$›
by (*rule minimal-difference-between-valid-index*[*OF - assms*(*2*−*4*)])
(*use assms*(*1*) **in** ‹*auto simp*: *valid-arena-def*›)

Updates

Mark to delete **lemma** *clause-slice-extra-information-mark-to-delete*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \in\#$ *dom-m N*› **and**
dom: ‹$\forall i \in\#$ *dom-m N*. $i <$ *length arena* $\wedge\ i \geq$ *header-size* $(N \propto i)\ \wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*extra-information-mark-to-delete arena i*) *N ia* =
(*if ia* = *i then extra-information-mark-to-delete* (*clause-slice arena N ia*) (*header-size* $(N \propto i)$)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹$ia \geq$ *header-size*$(N \propto ia)$› ‹$ia <$ *length arena*› **and**
i-ge: ‹$i \geq$ *header-size*$(N \propto i)$› ‹$i <$ *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ‹$ia < i$›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *clause-slice-extra-information-mark-to-delete-dead*:
assumes
i: ‹$i \in\#$ *dom-m N*› **and**
ia: ‹$ia \notin\#$ *dom-m N*› ‹$ia \in$ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*extra-information-mark-to-delete arena i*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹$ia \geq 4$› ‹$ia <$ *length arena*› **and**
i-ge: ‹$i \geq$ *header-size*($N \propto i$)› ‹$i <$ *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹$ia < i$›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap*
arena-dead-clause-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-extra-information-mark-to-delete*[*simp*]:
‹*length* (*extra-information-mark-to-delete arena i*) = *length arena*›
unfolding *extra-information-mark-to-delete-def* **by** *auto*

lemma *valid-arena-mono*: ‹*valid-arena ab ar vdom1* $\Longrightarrow$ *vdom2* $\subseteq$ *vdom1* $\Longrightarrow$ *valid-arena ab ar vdom2*›
unfolding *valid-arena-def*
by *fast*

lemma *valid-arena-extra-information-mark-to-delete*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) (*insert i vdom*)›
proof −
let *?arena* = ‹*extra-information-mark-to-delete arena i*›
have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
‹$\bigwedge ia.$ $ia \notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ $ia = i \vee (i \neq ia \wedge ia \notin\#$ *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have
dom: ‹$\forall i \in\#$*dom-m N*.
$i <$ *length arena* $\wedge$
header-size ($N \propto i$) $\leq i \wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹$\bigwedge i.$ $i \in\#$*dom-m N* $\Longrightarrow$
$i <$ *length arena* $\wedge$
header-size ($N \propto i$) $\leq i \wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹$\bigwedge i.$ $i \in$*vdom* $\longrightarrow$ $i \notin\#$ *dom-m N* $\longrightarrow$ $4 \leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹$ia \in\#$*dom-m* (*fmdrop i N*) $\Longrightarrow$
$ia <$ *length ?arena* $\wedge$
header-size (*fmdrop i N* $\propto$ *ia*) $\leq ia \wedge$
xarena-active-clause (*clause-slice ?arena* (*fmdrop i N*) *ia*) (*the* (*fmlookup* (*fmdrop i N*) *ia*))› **for** *ia*

using *dom'*[*of ia*] *clause-slice-extra-information-mark-to-delete*[*OF i - dom*, *of ia*]
by *auto*
moreover have ‹*ia* $\neq$ *i* $\longrightarrow$ *ia*$\in$*insert i vdom* $\longrightarrow$
ia $\notin\#$ *dom-m* (*fmdrop i N*) $\longrightarrow$
4 $\leq$ *ia* $\wedge$ *arena-dead-clause*
(*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-extra-information-mark-to-delete-dead*[*OF i - - arena*, *of ia*]
by *auto*
moreover have ‹*4* $\leq$ *i* $\wedge$ *arena-dead-clause*
(*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *i*)›
using *dom'*[*of i*, *OF i*]
unfolding *arena-dead-clause-def xarena-active-clause-alt-def*
extra-information-mark-to-delete-def **apply** −
by (*simp-all add*: *SHIFTS-def header-size-def Misc.slice-def drop-update-swap min-def*
split: *if-splits*)
force+
ultimately show *?thesis*
using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

lemma *valid-arena-extra-information-mark-to-delete'*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* $\in\#$ *dom-m N*›
shows ‹*valid-arena* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *vdom*›
using *valid-arena-extra-information-mark-to-delete*[*OF assms*]
by (*auto intro*: *valid-arena-mono*)

Removable from addressable space **lemma** *valid-arena-remove-from-vdom*:
assumes ‹*valid-arena arena N* (*insert i vdom*)›
shows ‹*valid-arena arena N vdom*›
using *assms valid-arena-def*
by (*auto dest*!: *in-diffD*)

Update activity **definition** *update-act* **where**
‹*update-act C act arena* = *arena*[*C* − *ACTIVITY-SHIFT* := *AActivity act*]›

lemma *clause-slice-update-act*:
assumes
i: ‹*i* $\in\#$ *dom-m N*› **and**
ia: ‹*ia* $\in\#$ *dom-m N*› **and**
dom: ‹$\forall$ *i* $\in\#$ *dom-m N*. *i* < *length arena* $\wedge$ *i* $\geq$ *header-size* (*N*$\propto$*i*) $\wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*update-act i act arena*) *N ia* =
(*if ia* = *i then update-act* (*header-size* (*N*$\propto$*i*)) *act* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* $\geq$ *header-size*(*N* $\propto$ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* $\geq$ *header-size*(*N* $\propto$ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*

by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap ACTIVITY-SHIFT-def update-act-def Misc.slice-def header-size-def split*: *if-splits*)
qed

lemma *length-update-act*[*simp*]:
‹*length* (*update-act i act arena*) = *length arena*›
by (*auto simp*: *update-act-def*)

lemma *clause-slice-update-act-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-act i act arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap arena-dead-clause-def update-act-def ACTIVITY-SHIFT-def Misc.slice-def header-size-def split*: *if-splits*)
qed

lemma *xarena-active-clause-update-act-same*:
assumes
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*update-act* (*header-size* (*N*∝*i*)) *act* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
using *assms*
by (*cases* ‹*is-short-clause* (*N* ∝ *i*)›)
(*simp-all add*: *xarena-active-clause-alt-def update-act-def SHIFTS-def Misc.slice-def header-size-def*)

lemma *valid-arena-update-act*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*›
shows ‹*valid-arena* (*update-act i act arena*) *N vdom*›
proof −
let *?arena* = ‹*update-act i act arena*›
have [*simp*]: ‹*i* ∉# *remove1-mset i* (*dom-m N*)›
‹⋀*ia*. *ia* ∉# *remove1-mset i* (*dom-m N*) ⟷ *ia* =*i* ∨ (*i* ≠ *ia* ∧ *ia* ∉# *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have

dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *4* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-act*[*OF i - dom*, *of ia act*]
by *auto*
moreover have ‹*ia* = *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-act*[*OF i - dom*, *of ia act*] *i*
by (*simp add*: *xarena-active-clause-update-act-same*)
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
4 ≤ *ia* ∧ *arena-dead-clause*
(*dead-clause-slice* (*update-act i act arena*) (*fmdrop i N*) *ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-update-act-dead*[*OF i - - arena*, *of ia*] *i*
by *auto*
ultimately show *?thesis*
using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

Update LBD **definition** *update-lbd* **where**
‹*update-lbd C lbd arena* = *arena*[*C* − *LBD-SHIFT* := *ALBD lbd*]›

lemma *clause-slice-update-lbd*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*update-lbd i lbd arena*) *N ia* =
(*if ia* = *i then update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*

minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-lbd-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-update-lbd*[*simp*]:
‹*length* (*update-lbd i lbd arena*) = *length arena*›
by (*auto simp*: *update-lbd-def*)

lemma *clause-slice-update-lbd-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-lbd i lbd arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-lbd-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *xarena-active-clause-update-lbd-same*:
assumes
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
using *assms*
by (*cases* ‹*is-short-clause* (*N* ∝ *i*)›)
(*simp-all add*: *xarena-active-clause-alt-def update-lbd-def SHIFTS-def Misc.slice-def*
header-size-def)

lemma *valid-arena-update-lbd*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*›
shows ‹*valid-arena* (*update-lbd i lbd arena*) *N vdom*›
proof −
let *?arena* = ‹*update-lbd i lbd arena*›
have [*simp*]: ‹*i* ∉# *remove1-mset i* (*dom-m N*)›
‹⋀*ia*. *ia* ∉# *remove1-mset i* (*dom-m N*) ⟷ *ia* = *i* ∨ (*i* ≠ *ia* ∧ *ia* ∉# *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)

have
dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *4* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*]
by *auto*
moreover have ‹*ia* = *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*] *i*
by (*simp add*: *xarena-active-clause-update-lbd-same*)
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
4 ≤ *ia* ∧ *arena-dead-clause*
(*dead-clause-slice* (*update-lbd i lbd arena*) (*fmdrop i N*) *ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-update-lbd-dead*[*OF i - - arena*, *of ia*] *i*
by *auto*
ultimately show *?thesis*
using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

Update saved position **definition** *update-pos-direct* **where**
‹*update-pos-direct C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos pos*]›

definition *arena-update-pos* **where**
‹*arena-update-pos C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos* (*pos* − *2*)]›

lemma *arena-update-pos-alt-def*:
‹*arena-update-pos C i N* = *update-pos-direct C* (*i* − *2*) *N*›
by (*auto simp*: *arena-update-pos-def update-pos-direct-def*)

lemma *clause-slice-update-pos*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)›
shows
‹*clause-slice* (*update-pos-direct i pos arena*) *N ia* =
(*if ia* = *i then update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N ia*)

else clause-slice arena N ia)⟩
proof −
have *ia-ge*: ⟨*ia* ≥ *header-size*(*N* ∝ *ia*)⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*
show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge long*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *clause-slice-update-pos-dead*:
assumes
i: ⟨*i* ∈# *dom-m N*⟩ **and**
ia: ⟨*ia* ∉# *dom-m N*⟩ ⟨*ia* ∈ *vdom*⟩ **and**
dom: ⟨*valid-arena arena N vdom*⟩ **and**
long: ⟨*is-long-clause* (*N* ∝ *i*)⟩
shows
⟨*arena-dead-clause* (*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)⟩
proof −
have *ia-ge*: ⟨*ia* ≥ *4*⟩ ⟨*ia* < *length arena*⟩ **and**
i-ge: ⟨*i* ≥ *header-size*(*N* ∝ *i*)⟩ ⟨*i* < *length arena*⟩
using *dom ia i long* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge long*
by (*cases* ⟨*ia* < *i*⟩)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *xarena-active-clause-update-pos-same*:
assumes
⟨*i* ≥ *header-size* (*N* ∝ *i*)⟩ **and**
⟨*i* < *length arena*⟩ **and**
⟨*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))⟩ **and**
long: ⟨*is-long-clause* (*N* ∝ *i*)⟩ **and**
⟨*pos* ≤ *length* (*N* ∝ *i*) − *2*⟩
shows ⟨*xarena-active-clause* (*update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))⟩
using *assms*
by (*simp-all add*: *update-pos-direct-def SHIFTS-def Misc.slice-def*
header-size-def xarena-active-clause-alt-def)

lemma *length-update-pos*[*simp*]:
⟨*length* (*update-pos-direct i pos arena*) = *length arena*⟩
by (*auto simp*: *update-pos-direct-def*)

lemma *valid-arena-update-pos*:
  assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*› **and**
    long: ‹*is-long-clause* ($N \propto i$)›**and**
    pos: ‹$pos \leq$ *length* ($N \propto i$) $-$ *2*›
  shows ‹*valid-arena* (*update-pos-direct i pos arena*) *N vdom*›
proof −
  let *?arena* = ‹*update-pos-direct i pos arena*›
  have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
    ‹$\bigwedge$*ia*. $ia \notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ $ia = i \vee (i \neq ia \wedge ia \notin\#$ *dom-m N*)›
    using *assms distinct-mset-dom*[*of N*]
    by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
  have
    dom: ‹$\forall i \in\#$*dom-m N*.
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    dom': ‹$\bigwedge i$. $i \in\#$*dom-m N* $\Longrightarrow$
        $i <$ *length arena* $\wedge$
        header-size ($N \propto i$) $\leq i \wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
    vdom: ‹$\bigwedge i$. $i \in vdom \longrightarrow i \notin\#$ *dom-m N* $\longrightarrow$ *4* $\leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N*
i)›
    using *assms* **unfolding** *valid-arena-def* **by** *auto*
  have ‹$ia \in\#$*dom-m N* $\Longrightarrow ia \neq i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom, of ia pos*] *long*
    by *auto*
  moreover have ‹$ia = i \Longrightarrow$
        $ia <$ *length ?arena* $\wedge$
        header-size ($N \propto ia$) $\leq ia \wedge$
        xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
    using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom, of ia pos*] *i long pos*
    by (*simp add*: *xarena-active-clause-update-pos-same*)
  moreover have ‹$ia \in vdom \longrightarrow$
        $ia \notin\#$ *dom-m N* $\longrightarrow$
        4 $\leq ia \wedge$ *arena-dead-clause*
         (*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*)› **for** *ia*
    using *vdom*[*of ia*] *clause-slice-update-pos-dead*[*OF i - - arena, of ia*] *i long*
    by *auto*
  ultimately show *?thesis*
    using *assms* **unfolding** *valid-arena-def*
    by *auto*
qed

Swap literals   **definition** *swap-lits* **where**
  ‹*swap-lits C i j arena* = *swap arena* ($C + i$) ($C + j$)›

lemma *clause-slice-swap-lits*:
  assumes
    i: ‹$i \in\#$ *dom-m N*› **and**
    ia: ‹$ia \in\#$ *dom-m N*› **and**
    dom: ‹$\forall i \in\#$ *dom-m N*. $i <$ *length arena* $\wedge$ $i \geq$ *header-size* ($N \propto i$) $\wedge$
        xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**

k: ‹$k < $ *length* $(N \propto i)$› **and**
l: ‹$l < $ *length* $(N \propto i)$›
shows
‹*clause-slice* (*swap-lits i k l arena*) *N ia* =
(*if ia* = *i then swap-lits* (*header-size* $(N\propto i)$) *k l* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*$(N \propto ia)$› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*$(N \propto i)$› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge k l*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
swap-lits-def SHIFTS-def swap-def ac-simps
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-swap-lits*[*simp*]:
‹*length* (*swap-lits i k l arena*) = *length arena*›
by (*auto simp*: *swap-lits-def*)

lemma *clause-slice-swap-lits-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›**and**
k: ‹$k < $ *length* $(N \propto i)$› **and**
l: ‹$l < $ *length* $(N \propto i)$›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*swap-lits i k l arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*$(N \propto i)$› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge k l*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *xarena-active-clause-swap-lits-same*:
assumes
‹*i* ≥ *header-size* $(N \propto i)$› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›**and**
k: ‹$k < $ *length* $(N \propto i)$› **and**

l: ‹*l* < *length* (*N* ∝ *i*)›
shows ‹*xarena-active-clause* (*clause-slice* (*swap-lits i k l arena*) *N i*)
(*the* (*fmlookup* (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *i*))›
using *assms*
unfolding *xarena-active-clause-alt-def*
by (*cases* ‹*is-short-clause* (*N* ∝ *i*)›)
(*simp-all add*: *swap-lits-def SHIFTS-def min-def swap-nth-if map-swap swap-swap*
header-size-def ac-simps is-short-clause-def split: *if-splits*)

lemma *is-short-clause-swap*[*simp*]: ‹*is-short-clause* (*swap* (*N* ∝ *i*) *k l*) = *is-short-clause* (*N* ∝ *i*)›
by (*auto simp*: *header-size-def is-short-clause-def split*: *if-splits*)

lemma *header-size-swap*[*simp*]: ‹*header-size* (*swap* (*N* ∝ *i*) *k l*) = *header-size* (*N* ∝ *i*)›
by (*auto simp*: *header-size-def split*: *if-splits*)

lemma *valid-arena-swap-lits*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows ‹*valid-arena* (*swap-lits i k l arena*) (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *vdom*›
proof −
let *?arena* = ‹*swap-lits i k l arena*›
have [*simp*]: ‹*i* ∉# *remove1-mset i* (*dom-m N*)›
‹⋀*ia*. *ia* ∉# *remove1-mset i* (*dom-m N*) ⟷ *ia* =*i* ∨ (*i* ≠ *ia* ∧ *ia* ∉# *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have
dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom′: ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *4* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N*
i)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom′*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *k l*
by *auto*
moreover have ‹*ia* = *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*)
(*the* (*fmlookup* (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *ia*))›
for *ia*
using *dom′*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *i k l*
xarena-active-clause-swap-lits-same[*OF - - - k l*, *of arena*]
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
4 ≤ *ia* ∧ *arena-dead-clause* (*dead-clause-slice* (*swap-lits i k l arena*) (*fmdrop i N*) *ia*)›

for *ia*
using *vdom[of ia] clause-slice-swap-lits-dead[OF i - - arena, of ia] i k l*
by *auto*
ultimately show *?thesis*
using *i k l arena* **unfolding** *valid-arena-def*
by *auto*
qed

Learning a clause **definition** *append-clause-skeleton* **where**
‹*append-clause-skeleton pos st used act lbd C arena* =
(*if is-short-clause C then*
arena @ (*AStatus st used*) # *AActivity act* # *ALBD lbd* #
ASize (*length C* − *2*) # *map ALit C*
else arena @ *APos pos* # (*AStatus st used*) # *AActivity act* #
ALBD lbd # *ASize* (*length C* − *2*) # *map ALit C*)›

definition *append-clause* **where**
‹*append-clause b C arena* =
append-clause-skeleton 0 (*if b then IRRED else LEARNED*) *False 0* (*length C* − *2*) *C arena*›

lemma *arena-active-clause-append-clause*:
assumes
‹$i \geq$ *header-size* ($N \propto i$)› **and**
‹$i <$ *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N i*)
(*the* (*fmlookup N i*))›
proof −
have ‹*drop* (*header-size* ($N \propto i$)) (*clause-slice arena N i*) = *map ALit* ($N \propto i$)› **and**
‹*header-size* ($N \propto i$) $\leq i$› **and**
‹$i <$ *length arena*›
using *assms*
unfolding *xarena-active-clause-alt-def*
by *auto*
from *arg-cong*[*OF this*(*1*), *of length*] *this*(*2*−)
have ‹i + *length* ($N \propto i$) $\leq$ *length arena*›
unfolding *xarena-active-clause-alt-def*
by (*auto simp add*: *slice-len-min-If header-size-def is-short-clause-def split*: *if-splits*)
then have ‹*clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N i* =
clause-slice arena N i›
by (*auto simp add*: *append-clause-skeleton-def*)
then show *?thesis*
using *assms* **by** *simp*
qed

lemma *length-append-clause*[*simp*]:
‹*length* (*append-clause-skeleton pos st used act lbd C arena*) =
length arena + *length C* + *header-size C*›
‹*length* (*append-clause b C arena*) = *length arena* + *length C* + *header-size C*›
by (*auto simp*: *append-clause-skeleton-def header-size-def*
append-clause-def)

lemma *arena-active-clause-append-clause-same*: ‹$2 \leq$ *length C* $\Longrightarrow$ *st* $\neq$ *DELETED* $\Longrightarrow$
pos $\leq$ *length C* − *2* $\Longrightarrow$
b $\longleftrightarrow$ (*st* = *IRRED*) $\Longrightarrow$
xarena-active-clause

(*Misc.slice* (*length arena*) (*length arena* + *header-size C* + *length C*)
 (*append-clause-skeleton pos st used act lbd C arena*))
 (*the* (*fmlookup* (*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
 (*length arena* + *header-size C*)))›
unfolding *xarena-active-clause-alt-def append-clause-skeleton-def*
by (*cases st*)
 (*auto simp*: *header-size-def slice-start0 SHIFTS-def slice-Cons split*: *if-splits*)

lemma *clause-slice-append-clause*:
 assumes
 ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
 dom: ‹*valid-arena arena N vdom*› **and**
 ‹*arena-dead-clause* (*dead-clause-slice* (*arena*) *N ia*)›
 shows
 ‹*arena-dead-clause* (*dead-clause-slice* (*append-clause-skeleton pos st used act lbd C arena*) *N ia*)›
proof −
 have *ia-ge*: ‹*ia* ≥ *4*› ‹*ia* < *length arena*›
 using *dom ia* **unfolding** *valid-arena-def*
 by *auto*
 then have ‹*dead-clause-slice* (*arena*) *N ia* =
 dead-clause-slice (*append-clause-skeleton pos st used act lbd C arena*) *N ia*›
 by (*auto simp add*: *extra-information-mark-to-delete-def drop-update-swap*
 append-clause-skeleton-def
 arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
 Misc.slice-def header-size-def split: *if-splits*)
 then show *?thesis*
 using *assms* **by** *simp*
qed

lemma *valid-arena-append-clause-skeleton*:
 assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ *2*› **and**
 b: ‹*b* ⟷ (*st* = *IRRED*)› **and** *st*: ‹*st* ≠ *DELETED*› **and**
 pos: ‹*pos* ≤ *length C* − *2*›
 shows ‹*valid-arena* (*append-clause-skeleton pos st used act lbd C arena*)
 (*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
 (*insert* (*length arena* + *header-size C*) *vdom*)›
proof −
 let *?arena* = ‹*append-clause-skeleton pos st used act lbd C arena*›
 let *?i*= ‹*length arena* + *header-size C*›
 let *?N* = ‹(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)›
 let *?vdom* = ‹*insert* (*length arena* + *header-size C*) *vdom*›
 have
 dom: ‹∀ *i*∈#*dom-m N*.
 i < *length arena* ∧
 header-size (*N* ∝ *i*) ≤ *i* ∧
 xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
 dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
 i < *length arena* ∧
 header-size (*N* ∝ *i*) ≤ *i* ∧
 xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
 vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *i* ≤ *length arena* ∧ *4* ≤ *i* ∧
 arena-dead-clause (*dead-clause-slice arena N i*)›
 using *assms* **unfolding** *valid-arena-def* **by** *auto*
 have [*simp*]: ‹*?i* ∉# *dom-m N*›
 using *dom'*[*of ?i*]

by *auto*
have ‹*ia*∈#*dom-m N* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *arena-active-clause-append-clause*[*of N ia arena*]
by *auto*
moreover have ‹*ia* = *?i* ⟹
ia < *length ?arena* ∧
header-size (*?N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena ?N ia*) (*the* (*fmlookup ?N ia*))› **for** *ia*
using *dom'*[*of ia*] *le-C arena-active-clause-append-clause-same*[*of C st pos b arena used*]
b st pos
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶ *ia* < *length* (*?arena*) ∧
4 ≤ *ia* ∧ *arena-dead-clause* (*Misc.slice* (*ia* − *4*) *ia* (*?arena*))› **for** *ia*
using *vdom*[*of ia*] *clause-slice-append-clause*[*of ia N vdom arena pos st used act lbd C*, *OF - - arena*]
le-C b st
by *auto*
ultimately show *?thesis*
unfolding *valid-arena-def*
by *auto*
qed

lemma *valid-arena-append-clause*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C* ≥ *2*›
shows ‹*valid-arena* (*append-clause b C arena*)
(*fmupd* (*length arena* + *header-size C*) (*C*, *b*) *N*)
(*insert* (*length arena* + *header-size C*) *vdom*)›
using *valid-arena-append-clause-skeleton*[*OF assms*(*1*,*2*),
of b ‹*if b then IRRED else LEARNED*›]
by (*auto simp*: *append-clause-def*)

Refinement Relation

definition *status-rel*:: (*nat* × *clause-status*) *set* **where**
‹*status-rel* = {(*0*, *IRRED*), (*1*, *LEARNED*), (*3*, *DELETED*)}›

definition *bitfield-rel* **where**
‹*bitfield-rel n* = {(*a*, *b*). *b* ⟷ *a AND* (*2* ^ *n*) > *0*}›

definition *arena-el-relation* **where**
‹*arena-el-relation x el* = (*case el of*
AStatus n b ⇒ (*x AND 0b11*, *n*) ∈ *status-rel* ∧ (*x*, *b*) ∈ *bitfield-rel 2*
| *APos n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ASize n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ALBD n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *AActivity n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ALit n* ⇒ (*x*, *n*) ∈ *nat-lit-rel*
)›

definition *arena-el-rel* **where**
arena-el-rel-interal-def: ‹*arena-el-rel* = {(*x*, *el*). *arena-el-relation x el*}›

lemmas *arena-el-rel-def* = *arena-el-rel-interal-def*[*unfolded arena-el-relation-def*]

Preconditions and Assertions for the refinement

The following lemma expresses the relation between the arena and the clauses and especially shows the preconditions to be able to generate code.

The conditions on *arena-status* are in the direction to simplify proofs: If we would try to go in the opposite direction, we could rewrite ¬ *irred N i* into *arena-status arena i* ≠ *LEARNED*, which is a weaker property.

The inequality on the length are here to enable simp to prove inequalities *Suc 0* < *arena-length arena C* automatically. Normally the arithmetic part can prove it from *2* ≤ *arena-length arena C*, but as this inequality is simplified away, it does not work.

lemma *arena-lifting*:
  assumes *valid*: ‹*valid-arena arena N vdom*› **and**
   i: ‹*i* ∈# *dom-m N*›
  shows
    ‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
    ‹*i* < *length arena*›
    ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
    ‹*length* (*N* ∝ *i*) = *arena-length arena i*›
    ‹*j* < *length* (*N* ∝ *i*) ⟹ *N* ∝ *i* ! *j* = *arena-lit arena* (*i* + *j*)› **and**
    ‹*j* < *length* (*N* ∝ *i*) ⟹ *is-Lit* (*arena* ! (*i*+*j*))› **and**
    ‹*j* < *length* (*N* ∝ *i*) ⟹ *i* + *j* < *length arena*› **and**
    ‹*N* ∝ *i* ! *0* = *arena-lit arena i*› **and**
    ‹*is-Lit* (*arena* ! *i*)› **and**
    ‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*› **and**
    ‹*is-long-clause* (*N* ∝ *i*) ⟹ *is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› **and**
    ‹*is-long-clause* (*N* ∝ *i*) ⟹ *arena-pos arena i* ≤ *arena-length arena i*› **and**
    ‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› **and**
    ‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› **and**
    ‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))› **and**
    ‹*SIZE-SHIFT* ≤ *i*› **and**
    ‹*LBD-SHIFT* ≤ *i*›
    ‹*ACTIVITY-SHIFT* ≤ *i*› **and**
    ‹*arena-length arena i* ≥ *2*› **and**
    ‹*arena-length arena i* ≥ *Suc 0*› **and**
    ‹*arena-length arena i* ≥ *0*› **and**
    ‹*arena-length arena i* > *Suc 0*› **and**
    ‹*arena-length arena i* > *0*› **and**
    ‹*arena-status arena i* = *LEARNED* ⟷ ¬*irred N i*› **and**
    ‹*arena-status arena i* = *IRRED* ⟷ *irred N i*› **and**
    ‹*arena-status arena i* ≠ *DELETED*› **and**
    ‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* (*N* ∝ *i*)›
proof −
  have
    dom: ‹⋀*i*. *i*∈#*dom-m N* ⟹
      i < *length arena* ∧
      header-size (*N* ∝ *i*) ≤ *i* ∧
      xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
    using *valid* **unfolding** *valid-arena-def*
    by *blast*+

  have
    i-le: ‹*i* < *length arena*› **and**
    i-ge: ‹*header-size* (*N* ∝ *i*) ≤ *i*› **and**
    xi: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))›

using *dom*[*OF i*] **by** *fast*+

have
ge2: ‹*2* ≤ *length* (*N* ∝ *i*)› **and**
‹*header-size* (*N* ∝ *i*) + *length* (*N* ∝ *i*) = *length* (*clause-slice arena N i*)› **and**
pos: ‹*is-long-clause* (*N* ∝ *i*) ⟶
is-Pos (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *POS-SHIFT*)) ∧
xarena-pos (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *POS-SHIFT*))
≤ *length* (*N* ∝ *i*) − *2*› **and**
status: ‹*is-Status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*))› **and**
init: ‹(*xarena-status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*)) =
IRRED) =
irred N i› **and**
learned: ‹(*xarena-status*
(*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*)) =
LEARNED) =
(¬ *irred N i*)› **and**
lbd: ‹*is-LBD* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *LBD-SHIFT*))› **and**
act: ‹*is-Act* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *ACTIVITY-SHIFT*))› **and**
size: ‹*is-Size* (*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *SIZE-SHIFT*))› **and**
size': ‹*Suc* (*Suc* (*xarena-length*
(*clause-slice arena N i* !
(*header-size* (*N* ∝ *i*) − *SIZE-SHIFT*)))) =
length (*N* ∝ *i*)› **and**
clause: ‹*Misc.slice i* (*i* + *length* (*N* ∝ *i*)) *arena* = *map ALit* (*N* ∝ *i*)›
using *xi i-le i-ge* **unfolding** *xarena-active-clause-alt-def arena-length-def*
by *simp-all*
have [*simp*]:
‹*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *LBD-SHIFT*) = *ALBD* (*arena-lbd arena i*)›
‹*clause-slice arena N i* ! (*header-size* (*N* ∝ *i*) − *STATUS-SHIFT*) =
AStatus (*arena-status arena i*) (*arena-used arena i*)›
using *size size' i-le i-ge ge2 lbd status size'*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def arena-used-def*
by (*auto simp*: *SHIFTS-def slice-nth*)
have *HH*:
‹*arena-length arena i* = *length* (*N* ∝ *i*)› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
using *size size' i-le i-ge ge2 lbd status size' ge2*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def*
by (*cases* ‹*arena* ! (*i* − *Suc 0*)›; *auto simp*: *SHIFTS-def slice-nth*; *fail*)+
then show ‹*length* (*N* ∝ *i*) = *arena-length arena i*› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
using *i-le i-ge size' size ge2 HH* **unfolding** *numeral-2-eq-2*
by (*simp-all split*:)
show ‹*arena-length arena i* ≥ *2*›
‹*arena-length arena i* ≥ *Suc 0*› **and**
‹*arena-length arena i* ≥ *0*› **and**
‹*arena-length arena i* > *Suc 0*› **and**
‹*arena-length arena i* > *0*›
using *ge2* **unfolding** *HH* **by** *auto*
show
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*›
using *i-le i-ge* **by** *auto*
show *is-lit*: ‹*is-Lit* (*arena* ! (*i*+*j*))› ‹*N* ∝ *i* ! *j* = *arena-lit arena* (*i* + *j*)›
if ‹*j* < *length* (*N* ∝ *i*)›

for *j*
using *arg-cong*[*OF clause*, *of* ‹λ*xs*. *xs* ! *j*›] *i-le i-ge that*
by (*auto simp*: *slice-nth arena-lit-def*)

show *i-le-arena*: ‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*›
using *arg-cong*[*OF clause*, *of length*] *i-le i-ge*
by (*auto simp*: *arena-lit-def slice-len-min-If*)
show ‹*is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› and
‹*arena-pos arena i* ≤ *arena-length arena i*›
if ‹*is-long-clause* (*N* ∝ *i*)›
using *pos ge2 i-le i-ge that* unfolding *arena-pos-def HH*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› and
‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› and
‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))›
using *lbd act ge2 i-le i-ge status* unfolding *arena-pos-def*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*SIZE-SHIFT* ≤ *i*› and ‹*LBD-SHIFT* ≤ *i*› and
‹*ACTIVITY-SHIFT* ≤ *i*›
using *i-ge* unfolding *header-size-def SHIFTS-def* by (*auto split*: *if-splits*)
show ‹*j* < *length* (*N* ∝ *i*) ⟹ *i* + *j* < *length arena*›
using *i-le-arena* by *linarith*
show
‹*N* ∝ *i* ! *0* = *arena-lit arena i*› and
‹*is-Lit* (*arena* ! *i*)›
using *is-lit*[*of 0*] *ge2* by *fastforce*+
show
‹*arena-status arena i* = *LEARNED* ⟷ ¬*irred N i* ›and
‹*arena-status arena i* = *IRRED* ⟷ *irred N i*› and
‹*arena-status arena i* ≠ *DELETED*›
using *learned init* unfolding *arena-status-def*
by (*auto simp*: *arena-status-def*)
show
‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* (*N* ∝ *i*)›
apply (*subst list-eq-iff-nth-eq*, *intro conjI allI*)
subgoal
using *HH i-le-arena i-le*
by (*auto simp*: *slice-nth slice-len-min-If*)
subgoal for *j*
using *HH i-le-arena i-le is-lit*[*of j*]
by (*cases* ‹*arena* ! (*i* + *j*)›)
(*auto simp*: *slice-nth slice-len-min-If*
arena-lit-def)
done
qed

lemma *arena-dom-status-iff*:
assumes *valid*: ‹*valid-arena arena N vdom*› and
i: ‹*i* ∈ *vdom*›
shows
‹*i* ∈# *dom-m N* ⟷ *arena-status arena i* ≠ *DELETED*› (is ‹*?eq*› is ‹*?A* ⟷ *?B*›) and
‹*is-LBD* (*arena* ! (*i* − *LBD-SHIFT*))› (is *?lbd*) and
‹*is-Act* (*arena* ! (*i* − *ACTIVITY-SHIFT*))› (is *?act*) and
‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))› (is *?stat*) and
‹*4* ≤ *i*› (is *?ge*)

```
proof -
  have H1: ?eq ?lbd ?act ?stat ?ge
    if ‹?A›
  proof -
    have
      ‹xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))› and
      i-ge: ‹header-size (N ∝ i) ≤ i› and
      i-le: ‹i < length arena›
      using assms that unfolding valid-arena-def by blast+
    then have ‹is-Status (clause-slice arena N i ! (header-size (N ∝ i) - STATUS-SHIFT))› and
      ‹(xarena-status (clause-slice arena N i ! (header-size (N ∝ i) - STATUS-SHIFT)) = IRRED) =
       irred N i› and
      ‹(xarena-status (clause-slice arena N i ! (header-size (N ∝ i) - STATUS-SHIFT)) = LEARNED)
=
        (¬ irred N i)› and
      ‹is-LBD (clause-slice arena N i ! (header-size (N ∝ i) - LBD-SHIFT))› and
      ‹is-Act (clause-slice arena N i ! (header-size (N ∝ i) - ACTIVITY-SHIFT)) ›
      unfolding xarena-active-clause-alt-def arena-status-def
      by blast+
    then show ?eq and ?lbd and ?act and ?stat and ?ge
      using i-ge i-le that
      unfolding xarena-active-clause-alt-def arena-status-def
      by (auto simp: SHIFTS-def header-size-def slice-nth split: if-splits)
  qed
  moreover have H2: ?eq
    if ‹?B›
  proof -
    have ?A
    proof (rule ccontr)
      assume ‹i ∉# dom-m N›
      then have
        ‹arena-dead-clause (Misc.slice (i - 4) i arena)› and
        i-ge: ‹4 ≤ i› and
        i-le: ‹i < length arena›
        using assms unfolding valid-arena-def by blast+
      then show False
        using ‹?B›
        unfolding arena-dead-clause-def
        by (auto simp: arena-status-def slice-nth SHIFTS-def)
    qed
    then show ?eq
      using arena-lifting[OF valid, of i] that
      by auto
  qed
  moreover have ?lbd ?act ?stat ?ge if ‹¬?A›
  proof -
    have
      ‹arena-dead-clause (Misc.slice (i - 4) i arena)› and
      i-ge: ‹4 ≤ i› and
      i-le: ‹i < length arena›
      using assms that unfolding valid-arena-def by blast+
    then show ?lbd ?act ?stat ?ge
      unfolding arena-dead-clause-def
      by (auto simp: SHIFTS-def slice-nth)
  qed
  ultimately show ?eq and ?lbd and ?act and ?stat and ?ge
```

by *blast+*
qed

lemma *valid-arena-one-notin-vdomD*:
⟨$valid\text{-}arena\ M\ N\ vdom \Longrightarrow Suc\ 0 \notin vdom$⟩
using *arena-dom-status-iff*[*of M N vdom 1*]
by *auto*

This is supposed to be used as for assertions. There might be a more "local" way to define it, without the need for an existentially quantified clause set. However, I did not find a definition which was really much more useful and more practical.

definition *arena-is-valid-clause-idx* :: ⟨$arena \Rightarrow nat \Rightarrow bool$⟩ **where**
⟨$arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ arena\ i \longleftrightarrow$
$(\exists N\ vdom.\ valid\text{-}arena\ arena\ N\ vdom \wedge i \in\#\ dom\text{-}m\ N)$⟩

This precondition has weaker preconditions is restricted to extracting the status (the other headers can be extracted but only garbage is returned).

definition *arena-is-valid-clause-vdom* :: ⟨$arena \Rightarrow nat \Rightarrow bool$⟩ **where**
⟨$arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom\ arena\ i \longleftrightarrow$
$(\exists N\ vdom.\ valid\text{-}arena\ arena\ N\ vdom \wedge i \in vdom)$⟩

lemma *SHIFTS-alt-def*:
⟨$POS\text{-}SHIFT = Suc\ (Suc\ (Suc\ (Suc\ (Suc\ 0))))$⟩
⟨$STATUS\text{-}SHIFT = Suc\ (Suc\ (Suc\ (Suc\ 0)))$⟩
⟨$ACTIVITY\text{-}SHIFT = Suc\ (Suc\ (Suc\ 0))$⟩
⟨$LBD\text{-}SHIFT = Suc\ (Suc\ 0)$⟩
⟨$SIZE\text{-}SHIFT = Suc\ 0$⟩
by (*auto simp*: *SHIFTS-def*)

definition *arena-is-valid-clause-idx-and-access* :: ⟨$arena \Rightarrow nat \Rightarrow nat \Rightarrow bool$⟩ **where**
⟨$arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\text{-}and\text{-}access\ arena\ i\ j \longleftrightarrow$
$(\exists N\ vdom.\ valid\text{-}arena\ arena\ N\ vdom \wedge i \in\#\ dom\text{-}m\ N \wedge j < length\ (N \propto i))$⟩

This is the precondition for direct memory access: $N\ !\ i$ where $i = j + (j - i)$ instead of $N \propto j\ !\ (i - j)$.

definition *arena-lit-pre* **where**
⟨$arena\text{-}lit\text{-}pre\ arena\ i \longleftrightarrow$
$(\exists j.\ i \geq j \wedge arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\text{-}and\text{-}access\ arena\ j\ (i - j))$⟩

definition *arena-lit-pre2* **where**
⟨$arena\text{-}lit\text{-}pre2\ arena\ i\ j \longleftrightarrow$
$(\exists N\ vdom.\ valid\text{-}arena\ arena\ N\ vdom \wedge i \in\#\ dom\text{-}m\ N \wedge j < length\ (N \propto i))$⟩

definition *swap-lits-pre* **where**
⟨$swap\text{-}lits\text{-}pre\ C\ i\ j\ arena \longleftrightarrow C + i < length\ arena \wedge C + j < length\ arena$⟩

definition *update-lbd-pre* **where**
⟨$update\text{-}lbd\text{-}pre = (\lambda((C,\ lbd),\ arena).\ arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ arena\ C)$⟩

definition *get-clause-LBD-pre* **where**
⟨$get\text{-}clause\text{-}LBD\text{-}pre = arena\text{-}is\text{-}valid\text{-}clause\text{-}idx$⟩

Saved position **definition** *get-saved-pos-pre* **where**

⟨*get-saved-pos-pre arena C* ⟷ *arena-is-valid-clause-idx arena C* ∧
arena-length arena C > *MAX-LENGTH-SHORT-CLAUSE*⟩

definition *isa-update-pos-pre* **where**
⟨*isa-update-pos-pre* = (λ((*C*, *pos*), *arena*). *arena-is-valid-clause-idx arena C* ∧ *pos* ≥ *2* ∧
pos ≤ *arena-length arena C* ∧ *arena-length arena C* > *MAX-LENGTH-SHORT-CLAUSE*)⟩

definition *mark-garbage-pre* **where**
⟨*mark-garbage-pre* = (λ(*arena*, *C*). *arena-is-valid-clause-idx arena C*)⟩

definition *arena-act-pre* **where**
⟨*arena-act-pre* = *arena-is-valid-clause-idx*⟩

lemma *length-clause-slice-list-update*[*simp*]:
⟨*length* (*clause-slice* (*arena*[*i* := *x*]) *a b*) = *length* (*clause-slice arena a b*)⟩
by (*auto simp*: *Misc.slice-def*)

definition *arena-decr-act* **where**
⟨*arena-decr-act arena i* = *arena*[*i* − *ACTIVITY-SHIFT* :=
AActivity (*xarena-act* (*arena*!(*i* − *ACTIVITY-SHIFT*)) *div 2*)]⟩

lemma *length-arena-decr-act*[*simp*]:
⟨*length* (*arena-decr-act arena C*) = *length arena*⟩
by (*auto simp*: *arena-decr-act-def*)

definition *mark-used* **where**
⟨*mark-used arena i* =
arena[*i* − *STATUS-SHIFT* := *AStatus* (*xarena-status* (*arena*!(*i* − *STATUS-SHIFT*))) *True*]⟩

lemma *length-mark-used*[*simp*]: ⟨*length* (*mark-used arena C*) = *length arena*⟩
by (*auto simp*: *mark-used-def*)

lemma *valid-arena-mark-used*:
assumes *C*: ⟨*C* ∈# *dom-m N*⟩ **and** *valid*: ⟨*valid-arena arena N vdom*⟩
shows
⟨*valid-arena* (*mark-used arena C*) *N vdom*⟩
proof −
let *?arena* = ⟨*mark-used arena C*⟩
have *act*: ⟨∀ *i*∈#*dom-m N*.
i < *length* (*arena*) ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))⟩ **and**
dead: ⟨⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)⟩ **and**
C-ge: ⟨*header-size* (*N* ∝ *C*) ≤ *C*⟩ **and**
C-le: ⟨*C* < *length arena*⟩ **and**
C-act: ⟨*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))⟩
using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ⟨*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)⟩ **and**

[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *STATUS-SHIFT*) =
AStatus (*xarena-status* (*clause-slice arena N C* ! (*header-size* ($N \propto C$) − *STATUS-SHIFT*)))
True› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* ($N \propto C$) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* ($N \propto C$) $\Longrightarrow$ *clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* ($N \propto C$) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *ACTIVITY-SHIFT*) =
clause-slice arena N C ! (*header-size* ($N \propto C$) − *ACTIVITY-SHIFT*)› **and**
[*simp*]: ‹*Misc.slice C* (*C* + *length* ($N \propto C$)) *?arena* =
Misc.slice C (*C* + *length* ($N \propto C$)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def mark-used-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) $\Longrightarrow$
xarena-active-clause (*clause-slice* (*mark-used arena C*) *N i*) (*the* (*fmlookup N i*))›
if ‹$i \in\#$ *dom-m N*›
for *i*
using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *mark-used-def header-size-def STATUS-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i ?arena*)›
if ‹$i \in$ *vdom*›‹$i \notin\#$ *dom-m N*›‹*arena-dead-clause* (*Misc.slice* ($i - 4$) *i arena*)›
for *i*
proof −
have *i-ge*: ‹$i \geq 4$› ‹$i <$ *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *mark-used-def header-size-def STATUS-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

definition *mark-unused* **where**
‹*mark-unused arena i* =
arena[i − *STATUS-SHIFT* := *AStatus* (*xarena-status* (*arena*!(i − *STATUS-SHIFT*))) *False*]›

lemma *length-mark-unused*[*simp*]: ‹*length* (*mark-unused arena C*) = *length arena*›
  by (*auto simp*: *mark-unused-def*)

lemma *valid-arena-mark-unused*:
  assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
  shows
   ‹*valid-arena* (*mark-unused arena C*) *N vdom*›
proof −
  let *?arena* = ‹*mark-unused arena C*›
  have *act*: ‹∀ *i*∈#*dom-m N*.
     i < *length* (*arena*) ∧
     header-size (*N* ∝ *i*) ≤ *i* ∧
     xarena-active-clause (*clause-slice arena N i*)
      (*the* (*fmlookup N i*))› **and**
    dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
          4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
    C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
    C-le: ‹*C* < *length arena*› **and**
    C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
      (*the* (*fmlookup N C*))›
    using *assms*
    by (*auto simp*: *valid-arena-def*)
  have
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
          clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
          AStatus (*xarena-status* (*clause-slice arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)))
            False› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
          clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
   [*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
          clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
   [*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
   [*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*) =
          clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*)› **and**
   [*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
     Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
    using *C-le C-ge* **unfolding** *SHIFTS-def mark-unused-def header-size-def*
    by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

  have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
    using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
    by *simp*

  then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
     xarena-active-clause (*clause-slice* (*mark-unused arena C*) *N i*) (*the* (*fmlookup N i*))›
    if ‹*i* ∈# *dom-m N*›
    for *i*
    using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
      minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
      that C-ge
    by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
      (*auto simp*: *mark-unused-def header-size-def STATUS-SHIFT-def*
      split: *if-splits*)

have *2*:
⟨*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i* *?arena*)⟩
if ⟨*i* ∈ *vdom*⟩⟨*i* ∉# *dom-m N*⟩⟨*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i* *arena*)⟩
for *i*
proof −
have *i-ge*: ⟨*i* ≥ *4*⟩ ⟨*i* < *length arena*⟩
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ⟨*C* < *i*⟩; *cases* ⟨*C* > *i*⟩)
(*auto simp*: *mark-unused-def header-size-def STATUS-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

definition *marked-as-used* :: ⟨*arena* ⇒ *nat* ⇒ *bool*⟩ **where**
⟨*marked-as-used arena C* = *xarena-used* (*arena* ! (*C* − *STATUS-SHIFT*))⟩

definition *marked-as-used-pre* **where**
⟨*marked-as-used-pre* = *arena-is-valid-clause-idx*⟩

lemma *valid-arena-vdom-le*:
assumes ⟨*valid-arena arena N ovdm*⟩
shows ⟨*finite ovdm*⟩ **and** ⟨*card ovdm* ≤ *length arena*⟩
proof −
have *incl*: ⟨*ovdm* ⊆ {*4*..< *length arena*}⟩
apply *auto*
using *assms valid-arena-in-vdom-le-arena* **by** *blast*+
from *card-mono*[*OF - this*] **show** ⟨*card ovdm* ≤ *length arena*⟩ **by** *auto*
have ⟨*length arena* ≥ *4* ∨ *ovdm* = {}⟩
using *incl* **by** *auto*
with *card-mono*[*OF - incl*] **have** ⟨*ovdm* ≠ {} ⟹ *card ovdm* < *length arena*⟩
by *auto*
from *finite-subset*[*OF incl*] **show** ⟨*finite ovdm*⟩ **by** *auto*
qed

lemma *valid-arena-vdom-subset*:
assumes ⟨*valid-arena arena N* (*set vdom*)⟩ **and** ⟨*distinct vdom*⟩
shows ⟨*length vdom* ≤ *length arena*⟩
proof −
have ⟨*set vdom* ⊆ {*0* ..< *length arena*}⟩
using *assms* **by** (*auto simp*: *valid-arena-def*)
from *card-mono*[*OF - this*] **show** *?thesis* **using** *assms* **by** (*auto simp*: *distinct-card*)
qed

lemma *valid-arena-arena-incr-act*:
assumes *C*: ⟨*C* ∈# *dom-m N*⟩ **and** *valid*: ⟨*valid-arena arena N vdom*⟩

shows
‹*valid-arena* (*arena-incr-act arena C*) *N vdom*›
proof −
let *?arena* = ‹*arena-incr-act arena C*›
have *act*: ‹∀ *i*∈#*dom-m N*.
i < *length* (*arena*) ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
4 ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)› **and**
C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
C-le: ‹*C* < *length arena*› **and**
C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))›
using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* (*N* ∝ *C*) ⟹ *clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*is-Act* (*clause-slice ?arena N C* ! (*header-size* (*N* ∝ *C*) − *ACTIVITY-SHIFT*))› **and**
[*simp*]: ‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *?arena* =
Misc.slice C (*C* + *length* (*N* ∝ *C*)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def arena-incr-act-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) ⟹
xarena-active-clause (*clause-slice* (*arena-incr-act arena C*) *N i*) (*the* (*fmlookup N i*))›
if ‹*i* ∈# *dom-m N*›
for *i*
using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *arena-incr-act-def header-size-def ACTIVITY-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i ?arena*)›
if ‹*i* ∈ *vdom*›‹*i* ∉# *dom-m N*›‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)›
for *i*
proof −
have *i-ge*: ‹*i* ≥ *4*› ‹*i* < *length arena*›
using *that valid* **unfolding** *valid-arena-def*

by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹$C < i$›; *cases* ‹$C > i$›)
(*auto simp*: *arena-incr-act-def header-size-def ACTIVITY-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def arena-incr-act-def*)
qed

lemma *valid-arena-arena-decr-act*:
assumes *C*: ‹$C \in\#$ *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*arena-decr-act arena C*) *N vdom*›
proof −
let *?arena* = ‹*arena-decr-act arena C*›
have *act*: ‹$\forall\, i{\in}\#$*dom-m N*.
$i <$ *length* (*arena*) $\wedge$
header-size ($N \propto i$) $\leq i \wedge$
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
dead: ‹$\bigwedge i.\ i \in$ *vdom* $\Longrightarrow i \notin\#$ *dom-m N* $\Longrightarrow i <$ *length arena* $\wedge$
$4 \leq i \wedge$ *arena-dead-clause* (*Misc.slice* ($i - 4$) *i arena*)› **and**
C-ge: ‹*header-size* ($N \propto C$) $\leq C$› **and**
C-le: ‹$C <$ *length arena*› **and**
C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))›
using *assms*
by (*auto simp*: *valid-arena-def*)
have
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *LBD-SHIFT*) =
clause-slice arena N C ! (*header-size* ($N \propto C$) − *LBD-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *STATUS-SHIFT*) =
clause-slice arena N C ! (*header-size* ($N \propto C$) − *STATUS-SHIFT*)› **and**
[*simp*]: ‹*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *SIZE-SHIFT*) =
clause-slice arena N C ! (*header-size* ($N \propto C$) − *SIZE-SHIFT*)› **and**
[*simp*]: ‹*is-long-clause* ($N \propto C$) $\Longrightarrow$ *clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *POS-SHIFT*)
=
clause-slice arena N C ! (*header-size* ($N \propto C$) − *POS-SHIFT*)› **and**
[*simp*]: ‹*length* (*clause-slice ?arena N C*) = *length* (*clause-slice arena N C*)› **and**
[*simp*]: ‹*is-Act* (*clause-slice ?arena N C* ! (*header-size* ($N \propto C$) − *ACTIVITY-SHIFT*))› **and**
[*simp*]: ‹*Misc.slice C* ($C +$ *length* ($N \propto C$)) *?arena* =
Misc.slice C ($C +$ *length* ($N \propto C$)) *arena*›
using *C-le C-ge* **unfolding** *SHIFTS-def arena-decr-act-def header-size-def*
by (*auto simp*: *Misc.slice-def drop-update-swap split*: *if-splits*)

have ‹*xarena-active-clause* (*clause-slice ?arena N C*) (*the* (*fmlookup N C*))›
using *C-act C-le C-ge* **unfolding** *xarena-active-clause-alt-def*
by *simp*

then have *1*: ‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*)) $\Longrightarrow$
xarena-active-clause (*clause-slice* (*arena-decr-act arena C*) *N i*) (*the* (*fmlookup N i*))›

if ‹*i* ∈# *dom-m N*›
for *i*
using *minimal-difference-between-valid-index*[*of N arena C i*, *OF act*]
minimal-difference-between-valid-index[*of N arena i C*, *OF act*] *assms*
that C-ge
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *arena-decr-act-def header-size-def ACTIVITY-SHIFT-def*
split: *if-splits*)

have *2*:
‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i ?arena*)›
if ‹*i* ∈ *vdom*›‹*i* ∉# *dom-m N*›‹*arena-dead-clause* (*Misc.slice* (*i* − *4*) *i arena*)›
for *i*
proof −
have *i-ge*: ‹*i* ≥ *4*› ‹*i* < *length arena*›
using *that valid* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *dead*[*of i*] *that C-le C-ge*
minimal-difference-between-invalid-index[*OF valid*, *of C i*]
minimal-difference-between-invalid-index2[*OF valid*, *of C i*]
by (*cases* ‹*C* < *i*›; *cases* ‹*C* > *i*›)
(*auto simp*: *arena-decr-act-def header-size-def ACTIVITY-SHIFT-def C*
split: *if-splits*)
qed
show *?thesis*
using *1 2 valid*
by (*auto simp*: *valid-arena-def*)
qed

lemma *length-arena-incr-act*[*simp*]:
‹*length* (*arena-incr-act arena C*) = *length arena*›
by (*auto simp*: *arena-incr-act-def*)

2.4 MOP versions of operations

2.4.1 Access to literals

definition *mop-arena-lit* **where**
‹*mop-arena-lit arena s* = *do* {
ASSERT(*arena-lit-pre arena s*);
RETURN (*arena-lit arena s*)
}›

lemma *arena-lit-pre-le-lengthD*: ‹*arena-lit-pre arena C* ⟹ *C* < *length arena*›
apply (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*)
using *arena-lifting*(*7*) *nat-le-iff-add* **by** *auto*

definition *mop-arena-lit2* :: ‹*arena* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*mop-arena-lit2 arena i j* = *do* {
ASSERT(*arena-lit-pre arena* (*i*+*j*));
let s = *i*+*j*;
RETURN (*arena-lit arena s*)
}›

named-theorems *mop-arena-lit* ‹*Theorems on mop−forms of arena constants*›

lemma *mop-arena-lit-itself*:
‹*mop-arena-lit arena* $k' \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*) $\Longrightarrow$ *mop-arena-lit arena* $k' \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*)›
‹*mop-arena-lit2 arena* i' $k' \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*) $\Longrightarrow$ *mop-arena-lit2 arena* i' $k' \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*)›
.

lemma [*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹$i \in\#$ *dom-m N*›
shows
‹$k = i+j \Longrightarrow j <$ *length* ($N \propto i$) $\Longrightarrow$ *mop-arena-lit arena* $k \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*)›
‹$i=i' \Longrightarrow j=j' \Longrightarrow j <$ *length* ($N \propto i$) $\Longrightarrow$ *mop-arena-lit2 arena* i' $j' \leq$ *SPEC*($\lambda c.$ (c, $N \propto i!j$) $\in$ *Id*)›
using *assms* **apply** (*auto simp*: *arena-lifting mop-arena-lit-def mop-arena-lit2-def Let-def*
intro!: *ASSERT-leI*)
apply (*metis arena-is-valid-clause-idx-and-access-def arena-lifting*(*4*) *arena-lit-pre-def diff-add-inverse le-add1*)+
done

lemma *mop-arena-lit2*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(C, C') $\in$ *nat-rel*› ‹(i, i') $\in$ *nat-rel*›
shows
‹*mop-arena-lit2 arena C i* $\leq \Downarrow$*Id* (*mop-clauses-at N* C' i')›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-lit2-def mop-clauses-at-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *C*])

definition *mop-arena-lit2'* :: ‹*nat set* $\Rightarrow$ *arena* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat literal nres*› **where**
‹*mop-arena-lit2' vdom* = *mop-arena-lit2*›

lemma *mop-arena-lit2'*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(C, C') $\in$ *nat-rel*› ‹(i, i') $\in$ *nat-rel*›
shows
‹*mop-arena-lit2' vdom arena C i* $\leq \Downarrow$*Id* (*mop-clauses-at N* C' i')›
using *mop-arena-lit2*[*OF assms*]
unfolding *mop-arena-lit2'-def*
.

lemma *arena-lit-pre2-arena-lit*[*dest*]:
‹*arena-lit-pre2 N i j* $\Longrightarrow$ *arena-lit-pre N* ($i+j$)›
by (*auto simp*: *arena-lit-pre-def arena-lit-pre2-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *i*])

2.4.2 Swapping of literals

definition *mop-arena-swap* **where**
‹*mop-arena-swap C i j arena* = *do* {

  ASSERT(*swap-lits-pre C i j arena*);
  RETURN (*swap-lits C i j arena*)
}›

lemma *mop-arena-swap*[*mop-arena-lit*]:
 assumes *valid*: ‹*valid-arena arena N vdom*› **and**
  i: ‹$(C,\ C') \in$ *nat-rel*› ‹$(i,\ i') \in$ *nat-rel*› ‹$(j,\ j') \in$ *nat-rel*›
 shows
  ‹*mop-arena-swap C i j arena* $\leq \Downarrow\{(N',\ N).$ *valid-arena* N' *N vdom*$\}$ (*mop-clauses-swap N* C' i' j')›
 using *assms* **unfolding** *mop-clauses-swap-def mop-arena-swap-def swap-lits-pre-def*
 by *refine-rcg*
  (*auto simp*: *arena-lifting valid-arena-swap-lits*)

2.4.3 Position Saving

definition *mop-arena-pos* :: ‹*arena* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-pos arena C* = *do* {
 ASSERT(*get-saved-pos-pre arena C*);
 RETURN (*arena-pos arena C*)
}›

definition *mop-arena-length* :: ‹*arena-el list* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length arena C* = *do* {
 ASSERT(*arena-is-valid-clause-idx arena C*);
 RETURN (*arena-length arena C*)
}›

2.4.4 Clause length

lemma *mop-arena-length*:
 ‹(*uncurry mop-arena-length*, *uncurry* (*RETURN oo* ($\lambda N\ c.$ *length* ($N \propto c$)))) $\in$
  $[\lambda(N,\ i).\ i \in\#$ *dom-m* $N]_f$ $\{(N,\ N').$ *valid-arena N* N' *vdom*$\}$ $\times_f$ *nat-rel* $\rightarrow$ ⟨*nat-rel*⟩*nres-rel*›
 unfolding *mop-arena-length-def*
 by (*intro frefI nres-relI*)
  (*auto 5 3 intro*!: *ASSERT-leI simp*: *append-ll-def arena-is-valid-clause-idx-def*
   arena-lifting)

definition *mop-arena-lbd* **where**
 ‹*mop-arena-lbd arena C* = *do* {
  ASSERT(*get-clause-LBD-pre arena C*);
  RETURN(*arena-lbd arena C*)
 }›

definition *mop-arena-status* **where**
 ‹*mop-arena-status arena C* = *do* {
  ASSERT(*arena-is-valid-clause-vdom arena C*);
  RETURN(*arena-status arena C*)
 }›

definition *mop-marked-as-used* **where**
 ‹*mop-marked-as-used arena C* = *do* {
  ASSERT(*marked-as-used-pre arena C*);
  RETURN(*marked-as-used arena C*)
 }›

definition *arena-other-watched* :: ‹*arena* ⇒ *nat literal* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**

```
‹arena-other-watched S L C i = do {
    ASSERT(i < 2 ∧ arena-lit S (C + i) = L ∧ arena-lit-pre2 S C i ∧
      arena-lit-pre2 S C (1−i));
    mop-arena-lit2 S C (1 − i)
  }›

end
theory WB-More-Word
  imports HOL−Word.More-Word Isabelle-LLVM.Bits-Natural
begin

lemma nat-uint-XOR: ‹nat (uint (a XOR b)) = nat (uint a) XOR nat (uint b)›
  if len: ‹LENGTH('a) > 0›
  for a b :: ‹'a ::len0 Word.word›
proof −
  have 1: ‹uint ((word-of-int:: int ⇒ 'a Word.word)(uint a)) = uint a›
    by (subst (2) word-of-int-uint[of a, symmetric]) (rule refl)
  have H: ‹nat (bintrunc n (a XOR b)) = nat (bintrunc n a XOR bintrunc n b)›
    if ‹n> 0› for n and a :: int and b :: int
    using that
  proof (induction n arbitrary: a b)
    case 0
    then show ?case by auto
  next
    case (Suc n) note IH = this(1) and Suc = this(2)
    then show ?case
    proof (cases n)
      case (Suc m)
      moreover have
        ‹nat (bintrunc m (bin-rest (bin-rest a) XOR bin-rest (bin-rest b)) BIT
            ((bin-last (bin-rest a) ∨ bin-last (bin-rest b)) ∧
             (bin-last (bin-rest a) ⟶ ¬ bin-last (bin-rest b))) BIT
            ((bin-last a ∨ bin-last b) ∧ (bin-last a ⟶ ¬ bin-last b))) =
         nat ((bintrunc m (bin-rest (bin-rest a)) XOR bintrunc m (bin-rest (bin-rest b))) BIT
              ((bin-last (bin-rest a) ∨ bin-last (bin-rest b)) ∧
               (bin-last (bin-rest a) ⟶ ¬ bin-last (bin-rest b))) BIT
              ((bin-last a ∨ bin-last b) ∧ (bin-last a ⟶ ¬ bin-last b)))›
        (is ‹nat (?n1 BIT ?b) = nat (?n2 BIT ?b)›)
      proof −
        have a1: nat ?n1 = nat ?n2
          using IH Suc by auto
        have f2: 0 ≤ ?n2
          by (simp add: bintr-ge0)
        have 0 ≤ ?n1
          using bintr-ge0 by auto
        then have ?n2 = ?n1
          using f2 a1 by presburger
        then show ?thesis by simp
      qed
      ultimately show ?thesis by simp
    qed simp
  qed
   have ‹nat (bintrunc LENGTH('a) (a XOR b)) = nat (bintrunc LENGTH('a) a XOR bintrunc
LENGTH('a) b)› for a b
    using len H[of ‹LENGTH('a)› a b] by auto
  then have ‹nat (uint (a XOR b)) = nat (uint a XOR uint b)›
```

by *transfer*
then show *?thesis*
unfolding *bitXOR-nat-def* **by** *auto*
qed
lemma *bitXOR-1-if-mod-2-int*: ‹*bitOR L 1 = (if L mod 2 = 0 then L + 1 else L)*› **for** *L* :: *int*
apply (*rule bin-rl-eqI*)
unfolding *bin-rest-OR bin-last-OR*
apply (*auto simp*: *bin-rest-def bin-last-def*)
done

lemma *bitOR-1-if-mod-2-nat*:
‹*bitOR L 1 = (if L mod 2 = 0 then L + 1 else L)*›
‹*bitOR L (Suc 0) = (if L mod 2 = 0 then L + 1 else L)*› **for** *L* :: *nat*
proof −
have *H*: ‹*bitOR L 1 = L + (if bin-last (int L) then 0 else 1)*›
unfolding *bitOR-nat-def*
apply (*auto simp*: *bitOR-nat-def bin-last-def*
bitXOR-1-if-mod-2-int)
done
show ‹*bitOR L 1 = (if L mod 2 = 0 then L + 1 else L)*›
unfolding *H*
apply (*auto simp*: *bitOR-nat-def bin-last-def*)
apply *presburger+*
done
then show ‹*bitOR L (Suc 0) = (if L mod 2 = 0 then L + 1 else L)*›
by *simp*
qed

lemma *bin-pos-same-XOR3*:
‹*a XOR a XOR c = c*›
‹*a XOR c XOR a = c*› **for** *a c* :: *int*
by (*metis bin-ops-same*(*3*) *int-xor-assoc int-xor-zero*)+

lemma *bin-pos-same-XOR3-nat*:
‹*a XOR a XOR c = c*›
‹*a XOR c XOR a = c*› **for** *a c* :: *nat*
unfolding *bitXOR-nat-def* **by** (*auto simp*: *bin-pos-same-XOR3*)

end
theory *IsaSAT-Literals-LLVM*
imports *WB-More-Word IsaSAT-Literals Watched-Literals.WB-More-IICF-LLVM*
begin

lemma *inline-ho*[*llvm-inline*]: *doM* { *f* ← *return f*; *m f* } = *m f* **for** *f* :: - ⇒ - **by** *simp*

lemma *RETURN-comp-5-10-hnr-post*[*to-hnr-post*]:
(*RETURN ooooo f5*)$*a*$*b*$*c*$*d*$*e* = *RETURN*$(*f5*$*a*$*b*$*c*$*d*$*e*)
(*RETURN oooooo f6*)$*a*$*b*$*c*$*d*$*e*$*f* = *RETURN*$(*f6*$*a*$*b*$*c*$*d*$*e*$*f*)
(*RETURN ooooooo f7*)$*a*$*b*$*c*$*d*$*e*$*f*$*g* = *RETURN*$(*f7*$*a*$*b*$*c*$*d*$*e*$*f*$*g*)
(*RETURN oooooooo f8*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h* = *RETURN*$(*f8*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*)
(*RETURN ooooooooo f9*)$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i* = *RETURN*$(*f9*$*a*$*b*$*c*$*d*$*e*$*f*$*g*$*h*$*i*)

(*RETURN oooooooooo f10*)\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j* = *RETURN*\$(*f10*\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*)
(*RETURN* o_{11} *f11*)\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k* = *RETURN*\$(*f11*\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*)
(*RETURN* o_{12} *f12*)\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l* = *RETURN*\$(*f12*\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l*)
(*RETURN* o_{13} *f13*)\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l*\$*m* = *RETURN*\$(*f13*\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l*\$*m*)
(*RETURN* o_{14} *f14*)\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l*\$*m*\$*n* = *RETURN*\$(*f14*\$*a*\$*b*\$*c*\$*d*\$*e*\$*f*\$*g*\$*h*\$*i*\$*j*\$*k*\$*l*\$*m*\$*n*)
by *simp-all*

definition [*simp*,*llvm-inline*]: *case-prod-open* ≡ *case-prod*
lemmas *fold-case-prod-open* = *case-prod-open-def*[*symmetric*]

lemma *case-prod-open-arity*[*sepref-monadify-arity*]:
case-prod-open ≡ λ_2*fp p*. *SP case-prod-open*\$(λ_2*a b*. *fp*\$*a*\$*b*)\$*p*
by (*simp-all only*: *SP-def APP-def PROTECT2-def RCALL-def*)

lemma *case-prod-open-comb*[*sepref-monadify-comb*]:
⋀*fp p*. *case-prod-open*\$*fp*\$*p* ≡ *Refine-Basic.bind*\$(*EVAL*\$*p*)\$(λ_2*p*. (*SP case-prod-open*\$*fp*\$*p*))
by (*simp-all*)

lemma *case-prod-open-plain-comb*[*sepref-monadify-comb*]:
EVAL\$(*case-prod-open*\$(λ_2*a b*. *fp a b*)\$*p*) ≡
Refine-Basic.bind\$(*EVAL*\$*p*)\$(λ_2*p*. *case-prod-open*\$(λ_2*a b*. *EVAL*\$(*fp a b*))\$*p*)
apply (*rule eq-reflection*, *simp split*: *list.split prod.split option.split*)+
done

lemma *hn-case-prod-open'*[*sepref-comb-rules*]:
assumes *FR*: Γ ⊢ *hn-ctxt* (*prod-assn P1 P2*) *p' p* ∗∗ Γ*1*
assumes *Pair*: ⋀*a1 a2 a1' a2'*. ⟦*p'*=(*a1'*,*a2'*)⟧
⟹ *hn-refine* (*hn-ctxt P1 a1' a1* ∗∗ *hn-ctxt P2 a2' a2* ∗∗ Γ*1*) (*f a1 a2*)
(Γ*2 a1 a2 a1' a2'*) *R* (*f' a1' a2'*)
assumes *FR2*: ⋀*a1 a2 a1' a2'*. Γ*2 a1 a2 a1' a2'* ⊢ *hn-ctxt P1' a1' a1* ∗∗ *hn-ctxt P2' a2' a2* ∗∗ Γ*1'*
shows *hn-refine* Γ (*case-prod-open f p*) (*hn-ctxt* (*prod-assn P1' P2'*) *p' p* ∗∗ Γ*1'*)
R (*case-prod-open*\$(λ_2*a b*. *f' a b*)\$*p'*) (**is** *?G* Γ)
unfolding *autoref-tag-defs PROTECT2-def*
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *prod-assn-pair-conv*[*THEN prod-assn-ctxt*])
apply (*rule hn-refine-cons*[*OF - Pair - entails-refl*])
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* **using** *FR2*
by (*simp add*: *hn-ctxt-def*)

lemma *ho-prod-open-move*[*sepref-preproc*]: *case-prod-open* (λ*a b x*. *f x a b*) = (λ*p x*. *case-prod-open* (*f x*) *p*)
by (*auto*)

definition *tuple4 a b c d* ≡ (*a*,*b*,*c*,*d*)
definition *tuple7 a b c d e f g* ≡ *tuple4 a b c* (*tuple4 d e f g*)
definition *tuple13 a b c d e f g h i j k l m* ≡ (*tuple7 a b c d e f* (*tuple7 g h i j k l m*))

lemmas *fold-tuples* = *tuple4-def*[*symmetric*] *tuple7-def*[*symmetric*] *tuple13-def*[*symmetric*]

sepref-register *tuple4 tuple7 tuple13*

sepref-def *tuple4-impl* [*llvm-inline*] **is** *uncurry3* (*RETURN oooo tuple4*) ::
$A1^d *_a A2^d *_a A3^d *_a A4^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4$
unfolding *tuple4-def* **by** *sepref*

sepref-def *tuple7-impl* [*llvm-inline*] **is** *uncurry6* (*RETURN ooooooo tuple7*) ::
$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7$
unfolding *tuple7-def* **by** *sepref*

sepref-def *tuple13-impl* [*llvm-inline*] **is** *uncurry12* (*RETURN* o_{13} *tuple13*) ::
$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d *_a A8^d *_a A9^d *_a A10^d *_a A11^d *_a A12^d *_a A13^d$
$\rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7 \times_a A8 \times_a A9 \times_a A10 \times_a A11 \times_a A12 \times_a A13$
unfolding *tuple13-def* **by** *sepref*

lemmas *fold-tuple-optimizations* = *fold-tuples fold-case-prod-open*

lemma *sint64-max-refine*[*sepref-import-param*]: (*0x7FFFFFFFFFFFFFFF*, *sint64-max*)∈*snat-rel′ TYPE*(*64*)
apply (*auto simp*: *snat-rel-def snat.rel-def in-br-conv sint64-max-def snat-invar-def*)
apply (*auto simp*: *snat-def*)
done

lemma *sint32-max-refine*[*sepref-import-param*]: (*0x7FFFFFFF*, *sint32-max*)∈*snat-rel′ TYPE*(*32*)
apply (*auto simp*: *snat-rel-def snat.rel-def in-br-conv sint32-max-def snat-invar-def*)
apply (*auto simp*: *snat-def*)
done

lemma *uint64-max-refine*[*sepref-import-param*]: (*0xFFFFFFFFFFFFFFFF*, *uint64-max*)∈*unat-rel′ TYPE*(*64*)
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv uint64-max-def*)
done

lemma *uint32-max-refine*[*sepref-import-param*]: (*0xFFFFFFFF*, *uint32-max*)∈*unat-rel′ TYPE*(*32*)
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv uint32-max-def*)
done

lemma *convert-fref*:
WB-More-Refinement.fref = *Sepref-Rules.frefnd*
WB-More-Refinement.freft = *Sepref-Rules.freftnd*
unfolding *WB-More-Refinement.fref-def Sepref-Rules.fref-def*
by *auto*

no-notation *WB-More-Refinement.fref* ($[-]_f$ - → - [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (- $\rightarrow_f$ - [*60*,*60*] *60*)

abbreviation *uint32-nat-assn* ≡ *unat-assn′ TYPE*(*32*)

abbreviation *uint64-nat-assn* ≡ *unat-assn′ TYPE(64)*

abbreviation *sint32-nat-assn* ≡ *snat-assn′ TYPE(32)*
abbreviation *sint64-nat-assn* ≡ *snat-assn′ TYPE(64)*

lemmas [*sepref-bounds-simps*] =
uint32-max-def sint32-max-def
uint64-max-def sint64-max-def

lemma *is-up′-32-64*[*simp*,*intro*!]: *is-up′ UCAST(32 → 64)* **by** (*simp add*: *is-up′*)
lemma *is-down′-64-32*[*simp*,*intro*!]: *is-down′ UCAST(64 → 32)* **by** (*simp add*: *is-down′*)

lemma *ins-idx-upcast64*:
l[*i*:=*y*] = *op-list-set l* (*op-unat-snat-upcast TYPE(64) i*) *y*
l!*i* = *op-list-get l* (*op-unat-snat-upcast TYPE(64) i*)
by *simp-all*

type-synonym ′*a array-list32* = (′*a*,*32*)*array-list*
type-synonym ′*a array-list64* = (′*a*,*64*)*array-list*

abbreviation *arl32-assn* ≡ *al-assn′ TYPE(32)*
abbreviation *arl64-assn* ≡ *al-assn′ TYPE(64)*

type-synonym ′*a larray32* = (′*a*,*32*) *larray*
type-synonym ′*a larray64* = (′*a*,*64*) *larray*

abbreviation *larray32-assn* ≡ *larray-assn′ TYPE(32)*
abbreviation *larray64-assn* ≡ *larray-assn′ TYPE(64)*

definition *unat-lit-rel* == *unat-rel′ TYPE(32) O nat-lit-rel*
lemmas [*fcomp-norm-unfold*] = *unat-lit-rel-def*[*symmetric*]

abbreviation *unat-lit-assn* :: ‹*nat literal* ⇒ *32 word* ⇒ *assn*› **where**
‹*unat-lit-assn* ≡ *pure unat-lit-rel*›

2.4.5 Atom-Of

type-synonym *atom-assn* = *32 word*

definition *atom-rel* ≡ *b-rel* (*unat-rel′ TYPE(32)*) (*λx. x*<*2^31*)
abbreviation *atom-assn* ≡ *pure atom-rel*

lemma *atom-rel-alt*: *atom-rel* = *unat-rel′ TYPE(32) O nbn-rel* (*2^31*)
by (*auto simp*: *atom-rel-def*)

interpretation *atom*: *dflt-pure-option-private 2^32−1 atom-assn ll-icmp-eq* (*2^32−1*)
apply *unfold-locales*
subgoal
unfolding *atom-rel-def*
apply (*simp add*: *pure-def fun-eq-iff pred-lift-extract-simps*)
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv unat-minus-one-word*)

done
subgoal proof *goal-cases*
case *1*
interpret *llvm-prim-arith-setup* **.**
show *?case* **unfolding** *bool.assn-def* **by** *vcg′*
qed
subgoal by *simp*
done

lemma *atm-of-refine*: $(\lambda x.\ x\ div\ 2\ ,\ atm\text{-}of) \in nat\text{-}lit\text{-}rel \to nat\text{-}rel$
by (*auto simp*: *nat-lit-rel-def in-br-conv*)

sepref-def *atm-of-impl* **is** [] *RETURN o* $(\lambda x{::}nat.\ x\ div\ 2)$
:: $uint32\text{-}nat\text{-}assn^k \to_a atom\text{-}assn$
unfolding *atom-rel-def b-assn-pure-conv*[*symmetric*]
apply (*rule hfref-bassn-resI*)
subgoal by *sepref-bounds*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemmas [*sepref-fr-rules*] = *atm-of-impl.refine*[*FCOMP atm-of-refine*]

definition *Pos-rel* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*Pos-rel n* = *2* ∗ *n*›

lemma *Pos-refine-aux*: (*Pos-rel*,*Pos*)∈*nat-rel* → *nat-lit-rel*
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*)

lemma *Neg-refine-aux*: $(\lambda x.\ 2{*}x + 1, Neg) \in nat\text{-}rel \to nat\text{-}lit\text{-}rel$
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*)

sepref-def *Pos-impl* **is** [] *RETURN o Pos-rel* :: $atom\text{-}assn^d \to_a uint32\text{-}nat\text{-}assn$
unfolding *atom-rel-def Pos-rel-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-def *Neg-impl* **is** [] *RETURN o* $(\lambda x.\ 2{*}x{+}1)$ :: $atom\text{-}assn^d \to_a uint32\text{-}nat\text{-}assn$
unfolding *atom-rel-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemmas [*sepref-fr-rules*] =
Pos-impl.refine[*FCOMP Pos-refine-aux*]
Neg-impl.refine[*FCOMP Neg-refine-aux*]

sepref-def *atom-eq-impl* **is** *uncurry* (*RETURN oo* (=)) :: $atom\text{-}assn^d *_a atom\text{-}assn^d \to_a bool1\text{-}assn$
unfolding *atom-rel-def*
by *sepref*

definition *value-of-atm* :: ‹*nat* ⇒ *nat*› **where**

[*simp*]: ‹*value-of-atm A* = *A*›

lemma *value-of-atm-rel*: ‹(λ*x*. *x*, *value-of-atm*) ∈ *nat-rel* → *nat-rel*›
  by (*auto*)

sepref-def *value-of-atm-impl*
  is [] ‹*RETURN o* (λ*x*. *x*)›
  :: ‹$atom\text{-}assn^d \rightarrow_a unat\text{-}assn'\ TYPE(32)$›
  unfolding *value-of-atm-def atom-rel-def*
  by *sepref*

lemmas [*sepref-fr-rules*] = *value-of-atm-impl.refine*[*FCOMP value-of-atm-rel*]

definition *index-of-atm* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*index-of-atm A* = *value-of-atm A*›

lemma *index-of-atm-rel*: ‹(λ*x*. *value-of-atm x*, *index-of-atm*) ∈ *nat-rel* → *nat-rel*›
  by (*auto*)

sepref-def *index-of-atm-impl*
  is [] ‹*RETURN o* (λ*x*. *value-of-atm x*)›
  :: ‹$atom\text{-}assn^d \rightarrow_a snat\text{-}assn'\ TYPE(64)$›
  unfolding *index-of-atm-def*
  apply (*rewrite at - eta-expand*)
  apply (*subst annot-unat-snat-upcast*[**where** ′*l*=*64*])
  by *sepref*

lemmas [*sepref-fr-rules*] = *index-of-atm-impl.refine*[*FCOMP index-of-atm-rel*]

lemma *annot-index-of-atm*: ‹*xs* ! *x* = *xs* ! *index-of-atm x*›
  ‹*xs* [*x* := *a*] = *xs* [*index-of-atm x* := *a*]›
  by *auto*

definition *index-atm-of* **where**
[*simp*]: ‹*index-atm-of L* = *index-of-atm* (*atm-of L*)›

context fixes *x y* :: *nat* **assumes** *NO-MATCH* (*index-of-atm y*) *x* **begin**
  lemmas *annot-index-of-atm*′ = *annot-index-of-atm*[**where** *x*=*x*]
end

method-setup *annot-all-atm-idxs* = ‹*Scan.succeed* (*fn ctxt* => *SIMPLE-METHOD*′
    let
      val ctxt = *put-simpset HOL-basic-ss ctxt*
      val ctxt = *ctxt addsimps* @{*thms annot-index-of-atm*′}
      val ctxt = *ctxt addsimprocs* [@{*simproc NO-MATCH*}]
    in
      simp-tac ctxt
    end
)›

lemma *annot-index-atm-of*[*def-pat-rules*]:
  ‹*nth*$*xs*$(*atm-of*$*x*) ≡ *nth*$*xs*$(*index-atm-of*$*x*)›
  ‹*list-update*$*xs*$(*atm-of*$*x*)$*a* ≡ *list-update*$*xs*$(*index-atm-of*$*x*)$*a*›
  by *auto*

sepref-def *index-atm-of-impl*
is ⟨*RETURN o index-atm-of*⟩
:: ⟨$unat\text{-}lit\text{-}assn^d \rightarrow_a snat\text{-}assn'$ *TYPE*(*64*)⟩
unfolding *index-atm-of-def*
by *sepref*

lemma *nat-of-lit-refine-aux*: ((λ*x*. *x*), *nat-of-lit*) ∈ *nat-lit-rel* → *nat-rel*
by (*auto simp*: *nat-lit-rel-def in-br-conv*)

sepref-def *nat-of-lit-rel-impl* **is** [] *RETURN o* (λ*x*::*nat*. *x*) :: $uint32\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$
apply (*rewrite annot-unat-snat-upcast*[**where** ′*l*=*64*])
by *sepref*
lemmas [*sepref-fr-rules*] = *nat-of-lit-rel-impl.refine*[*FCOMP nat-of-lit-refine-aux*]

lemma *uminus-refine-aux*: (λ*x*. *x XOR 1*, *uminus*) ∈ *nat-lit-rel* → *nat-lit-rel*
apply (*auto simp*: *nat-lit-rel-def in-br-conv bitXOR-1-if-mod-2*[*simplified*])
subgoal by *linarith*
subgoal by (*metis dvd-minus-mod even-Suc-div-two odd-Suc-minus-one*)
done

sepref-def *uminus-impl* **is** [] *RETURN o* (λ*x*::*nat*. *x XOR 1*) :: $uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemmas [*sepref-fr-rules*] = *uminus-impl.refine*[*FCOMP uminus-refine-aux*]

lemma *lit-eq-refine-aux*: ((=), (=)) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*; *auto?*; *presburger*)

sepref-def *lit-eq-impl* **is** [] *uncurry* (*RETURN oo* (=)) :: $uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a$ *bool1-assn*
by *sepref*

lemmas [*sepref-fr-rules*] = *lit-eq-impl.refine*[*FCOMP lit-eq-refine-aux*]

lemma *is-pos-refine-aux*: (λ*x*. *x AND 1* = *0*, *is-pos*) ∈ *nat-lit-rel* → *bool-rel*
by (*auto simp*: *nat-lit-rel-def in-br-conv bitAND-1-mod-2*[*simplified*] *split*: *if-splits*)

sepref-def *is-pos-impl* **is** [] *RETURN o* (λ*x*. *x AND 1* = *0*) :: $uint32\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemmas [*sepref-fr-rules*] = *is-pos-impl.refine*[*FCOMP is-pos-refine-aux*]

end
theory *IsaSAT-Arena-LLVM*
imports *IsaSAT-Arena IsaSAT-Literals-LLVM*
WB-More-Word
begin

2.5 Code Generation

no-notation *WB-More-Refinement.fref* ([-]$_f$ - → - [0,60,60] 60)
no-notation *WB-More-Refinement.freft* (- →$_f$ - [60,60] 60)

lemma *protected-bind-assoc*: *Refine-Basic.bind*\$(*Refine-Basic.bind*\$*m*\$($\lambda_2 x.$ *f x*))\$($\lambda_2 y.$ *g y*) = *Refine-Basic.bind*\$*m*\$(λ_2
Refine-Basic.bind\$(*f x*)\$($\lambda_2 y.$ *g y*)) **by** *simp*

lemma *convert-swap*: *WB-More-Refinement-List.swap* = *More-List.swap*
unfolding *WB-More-Refinement-List.swap-def More-List.swap-def* **..**

Code Generation

definition *arena-el-impl-rel* ≡ *unat-rel′ TYPE(32) O arena-el-rel*
lemmas [*fcomp-norm-unfold*] = *arena-el-impl-rel-def*[*symmetric*]
abbreviation *arena-el-impl-assn* ≡ *pure arena-el-impl-rel*

Arena Element Operations **context**
notes [*simp*] = *arena-el-rel-def*
notes [*split*] = *arena-el.splits*
notes [*intro*!] = *frefI*
begin

Literal

lemma *xarena-lit-refine1*: (λ*eli. eli*, *xarena-lit*) ∈ [*is-Lit*]$_f$ *arena-el-rel* → *nat-lit-rel* **by** *auto*
sepref-def *xarena-lit-impl* [*llvm-inline*] **is** [] *RETURN o* (λ*eli. eli*) :: *uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-lit-impl.refine*[*FCOMP xarena-lit-refine1*]

lemma *ALit-refine1*: (λ*x. x*,*ALit*) ∈ *nat-lit-rel* → *arena-el-rel* **by** *auto*
sepref-def *ALit-impl* [*llvm-inline*] **is** [] *RETURN o* (λ*x. x*) :: *uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn* **by**
sepref
lemmas [*sepref-fr-rules*] = *ALit-impl.refine*[*FCOMP ALit-refine1*]

LBD

lemma *xarena-lbd-refine1*: (λ*eli. eli*, *xarena-lbd*) ∈ [*is-LBD*]$_f$ *arena-el-rel* → *nat-rel* **by** *auto*
sepref-def *xarena-lbd-impl* [*llvm-inline*] **is** [] *RETURN o* (λ*eli. eli*) :: *uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-lbd-impl.refine*[*FCOMP xarena-lbd-refine1*]

lemma *ALBD-refine1*: (λ*eli. eli*, *ALBD*) ∈ *nat-rel* → *arena-el-rel* **by** *auto*
sepref-def *xarena-ALBD-impl* [*llvm-inline*] **is** [] *RETURN o* (λ*eli. eli*) :: *uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-ALBD-impl.refine*[*FCOMP ALBD-refine1*]

Activity

lemma *xarena-act-refine1*: (λ*eli. eli*, *xarena-act*) ∈ [*is-Act*]$_f$ *arena-el-rel* → *nat-rel* **by** *auto*
sepref-def *xarena-act-impl* [*llvm-inline*] **is** [] *RETURN o* (λ*eli. eli*) :: *uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-act-impl.refine*[*FCOMP xarena-act-refine1*]

lemma *AAct-refine1*: $(\lambda x.\ x, AActivity) \in nat\text{-}rel \to arena\text{-}el\text{-}rel$ **by** *auto*
sepref-def *AAct-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda x.\ x)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$ **by** *sepref*
lemmas [*sepref-fr-rules*] = *AAct-impl.refine*[*FCOMP AAct-refine1*]

Size

lemma *xarena-length-refine1*: $(\lambda eli.\ eli, xarena\text{-}length) \in [is\text{-}Size]_f\ arena\text{-}el\text{-}rel \to nat\text{-}rel$ **by** *auto*
sepref-def *xarena-len-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda eli.\ eli)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$ **by** *sepref*
lemmas [*sepref-fr-rules*] = *xarena-len-impl.refine*[*FCOMP xarena-length-refine1*]

lemma *ASize-refine1*: $(\lambda x.\ x, ASize) \in nat\text{-}rel \to arena\text{-}el\text{-}rel$ **by** *auto*
sepref-def *ASize-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda x.\ x)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$ **by** *sepref*
lemmas [*sepref-fr-rules*] = *ASize-impl.refine*[*FCOMP ASize-refine1*]

Position

lemma *xarena-pos-refine1*: $(\lambda eli.\ eli, xarena\text{-}pos) \in [is\text{-}Pos]_f\ arena\text{-}el\text{-}rel \to nat\text{-}rel$ **by** *auto*
sepref-def *xarena-pos-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda eli.\ eli)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$ **by** *sepref*
lemmas [*sepref-fr-rules*] = *xarena-pos-impl.refine*[*FCOMP xarena-pos-refine1*]

lemma *APos-refine1*: $(\lambda x.\ x, APos) \in nat\text{-}rel \to arena\text{-}el\text{-}rel$ **by** *auto*
sepref-def *APos-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda x.\ x)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$ **by** *sepref*
lemmas [*sepref-fr-rules*] = *APos-impl.refine*[*FCOMP APos-refine1*]

Status

definition *status-impl-rel* $\equiv$ *unat-rel'* $TYPE(32)$ *O status-rel*
lemmas [*fcomp-norm-unfold*] = *status-impl-rel-def*[*symmetric*]
abbreviation *status-impl-assn* $\equiv$ *pure status-impl-rel*

lemma *xarena-status-refine1*: $(\lambda eli.\ eli\ AND\ 0b11, xarena\text{-}status) \in [is\text{-}Status]_f\ arena\text{-}el\text{-}rel \to status\text{-}rel$ **by** (*auto simp*: *is-Status-def*)
sepref-def *xarena-status-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda eli.\ eli\ AND\ 0b11)$ :: $uint32\text{-}nat\text{-}assn^k \to_a uint32\text{-}nat\text{-}assn$
  apply (*annot-unat-const* $TYPE(32)$)
  by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-status-impl.refine*[*FCOMP xarena-status-refine1*]

lemma *xarena-used-refine1*: $(\lambda eli.\ eli\ AND\ 0b100 \neq 0, xarena\text{-}used) \in [is\text{-}Status]_f\ arena\text{-}el\text{-}rel \to bool\text{-}rel$
  by (*auto simp*: *is-Status-def status-rel-def bitfield-rel-def*)

sepref-def *xarena-used-impl* [*llvm-inline*] **is** [] *RETURN o* $(\lambda eli.\ eli\ AND\ 0b100 \neq 0)$ :: $uint32\text{-}nat\text{-}assn^k \to_a bool1\text{-}assn$
  apply (*annot-unat-const* $TYPE(32)$)
  by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-used-impl.refine*[*FCOMP xarena-used-refine1*]

lemma *status-eq-refine1*: $((=),(=)) \in status\text{-}rel \to status\text{-}rel \to bool\text{-}rel$
  by (*auto simp*: *status-rel-def*)

sepref-def *status-eq-impl* [*llvm-inline*] **is** [] *uncurry* (*RETURN oo* $(=)$)
  :: $(unat\text{-}assn'\ TYPE(32))^k *_a (unat\text{-}assn'\ TYPE(32))^k \to_a bool1\text{-}assn$

by *sepref*

lemmas [*sepref-fr-rules*] = *status-eq-impl.refine*[*FCOMP status-eq-refine1*]

definition *AStatus-impl1 cs used* ≡ (*cs AND unat-const TYPE*(*32*) *0b11*) + (*if used then unat-const TYPE*(*32*) *0b100 else unat-const TYPE*(*32*) *0b0*)
lemma *AStatus-refine1*: (*AStatus-impl1*, *AStatus*) ∈ *status-rel* → *bool-rel* → *arena-el-rel*
  by (*auto simp*: *status-rel-def bitfield-rel-def AStatus-impl1-def split*: *if-splits*)
sepref-def *AStatus-impl* [*llvm-inline*] **is** [] *uncurry* (*RETURN oo AStatus-impl1*) :: $uint32\text{-}nat\text{-}assn^k *_a bool1\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$
  unfolding *AStatus-impl1-def*
  supply [*split*] = *if-splits*
  by *sepref*
lemmas [*sepref-fr-rules*] = *AStatus-impl.refine*[*FCOMP AStatus-refine1*]

Arena Operations

Length **abbreviation** *arena-fast-assn* ≡ *al-assn′ TYPE*(*64*) *arena-el-impl-assn*

lemma *arena-lengthI*:
  assumes *arena-is-valid-clause-idx a b*
  shows *Suc 0* ≤ *b*
  and *b* < *length a*
  and *is-Size* (*a* ! (*b* − *Suc 0*))
  using *SIZE-SHIFT-def assms*
  by (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)

lemma *arena-length-alt*:
  ‹*arena-length arena i* = (
    let l = *xarena-length* (*arena*!(*i* − *snat-const TYPE*(*64*) *1*))
    in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
  by (*simp add*: *arena-length-def SIZE-SHIFT-def*)

sepref-register *arena-length*
sepref-def *arena-length-impl*
  is *uncurry* (*RETURN oo arena-length*)
    :: $[uncurry\ arena\text{-}is\text{-}valid\text{-}clause\text{-}idx]_a\ arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow snat\text{-}assn'\ TYPE(64)$
  unfolding *arena-length-alt*
  supply [*dest*] = *arena-lengthI*
  by *sepref*

Literal at given position **lemma** *arena-lit-implI*:
  assumes *arena-lit-pre a b*
  shows *b* < *length a is-Lit* (*a* ! *b*)
  using *assms* **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
  by (*fastforce dest*: *arena-lifting*)+

sepref-register *arena-lit xarena-lit*
sepref-def *arena-lit-impl*
  is *uncurry* (*RETURN oo arena-lit*)
    :: $[uncurry\ arena\text{-}lit\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow unat\text{-}lit\text{-}assn$
  supply [*intro*] = *arena-lit-implI*
  unfolding *arena-lit-def*

by *sepref*

sepref-register *mop-arena-lit mop-arena-lit2*
sepref-def *mop-arena-lit-impl*
is *uncurry* (*mop-arena-lit*)
:: *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*
supply [*intro*] = *arena-lit-implI*
unfolding *mop-arena-lit-def*
by *sepref*

sepref-def *mop-arena-lit2-impl*
is *uncurry2* (*mop-arena-lit2*)
:: $[\lambda((N, -), -).$ *length* $N \leq$ *sint64-max*$]_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$ *unat-lit-assn*
supply [*intro*] = *arena-lit-implI*
supply [*dest*] = *arena-lit-pre-le-lengthD*
unfolding *mop-arena-lit2-def*
by *sepref*

Status of the clause **lemma** *arena-status-implI*:
assumes *arena-is-valid-clause-vdom a b*
shows $4 \leq b$ $b - 4 <$ *length a is-Status* $(a ! (b-4))$
using *assms STATUS-SHIFT-def arena-dom-status-iff*
unfolding *arena-is-valid-clause-vdom-def*
by (*auto dest*: *valid-arena-in-vdom-le-arena*)

sepref-register *arena-status xarena-status*
sepref-def *arena-status-impl*
is *uncurry* (*RETURN oo arena-status*)
:: [*uncurry arena-is-valid-clause-vdom*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$ *status-impl-assn*
supply [*intro*] = *arena-status-implI*
unfolding *arena-status-def STATUS-SHIFT-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Swap literals **sepref-register** *swap-lits*
sepref-def *swap-lits-impl* **is** *uncurry3* (*RETURN oooo swap-lits*)
:: $[\lambda(((C,i),j),arena).$ $C + i <$ *length arena* $\wedge$ $C + j <$ *length arena*$]_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow$ *arena-fast-assn*
unfolding *swap-lits-def convert-swap*
unfolding *gen-swap*
by *sepref*

Get LBD **lemma** *get-clause-LBD-preI*:
assumes *get-clause-LBD-pre a b*
shows $2 \leq b$
and $b <$ *length a*
and *is-LBD* $(a ! (b - 2))$
using *LBD-SHIFT-def assms*
by (*auto simp*: *get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-lifting*)

sepref-register *arena-lbd*
sepref-def *arena-lbd-impl*
is *uncurry* (*RETURN oo arena-lbd*)
:: [*uncurry get-clause-LBD-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$*uint32-nat-assn*

unfolding *arena-lbd-def LBD-SHIFT-def*
supply [*dest*] = *get-clause-LBD-preI*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Get Saved Position **lemma** *arena-posI*:
assumes *get-saved-pos-pre a b*
shows $5 \le b$
and $b <$ *length a*
and *is-Pos* (*a* ! ($b - 5$))
using *POS-SHIFT-def assms is-short-clause-def*[*of* ‹- ∝ *b*›]
apply (*auto simp*: *get-saved-pos-pre-def arena-is-valid-clause-idx-def arena-lifting*
MAX-LENGTH-SHORT-CLAUSE-def[*symmetric*] *simp del*: *MAX-LENGTH-SHORT-CLAUSE-def*)
using *arena-lifting*(*1*) *arena-lifting*(*4*) *header-size-def* **apply** *fastforce*
done

lemma *arena-pos-alt*:
‹*arena-pos arena i* = (
let l = *xarena-pos* (*arena*!($i -$ *snat-const TYPE*(*64*) *5*))
in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
by (*simp add*: *arena-pos-def POS-SHIFT-def*)

sepref-register *arena-pos*
sepref-def *arena-pos-impl*
is *uncurry* (*RETURN oo arena-pos*)
:: [*uncurry get-saved-pos-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$ *snat-assn*$'$ *TYPE*(*64*)
unfolding *arena-pos-alt*
supply [*dest*] = *arena-posI*
by *sepref*

Update LBD **lemma** *update-lbdI*:
assumes *update-lbd-pre* ((*b*, *lbd*), *a*)
shows $2 \le b$
and $b - 2 <$ *length a*
using *LBD-SHIFT-def assms*
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting update-lbd-pre-def*
dest: *arena-lifting*(*10*))
by (*simp add*: *less-imp-diff-less valid-arena-def*)

sepref-register *update-lbd*
sepref-def *update-lbd-impl*
is *uncurry2* (*RETURN ooo update-lbd*)
:: [*update-lbd-pre*]$_a$ *sint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow$ *arena-fast-assn*
unfolding *update-lbd-def LBD-SHIFT-def*
supply [*simp*] = *update-lbdI*
and [*dest*] = *arena-posI*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Update Saved Position **lemma** *update-posI*:
assumes *isa-update-pos-pre* ((*b*, *pos*), *a*)
shows $5 \le b$ $2 \le$ *pos* $b{-}5 <$ *length a*
using *assms POS-SHIFT-def*
unfolding *isa-update-pos-pre-def*
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)

```
 apply (metis (full-types) MAX-LENGTH-SHORT-CLAUSE-def arena-is-valid-clause-idx-def arena-posI(1)
get-saved-pos-pre-def)
  by (simp add: less-imp-diff-less valid-arena-def)

lemma update-posI2:
  assumes isa-update-pos-pre ((b, pos), a)
  assumes rdomp (al-assn arena-el-impl-assn :: - ⇒ (32 word, 64) array-list ⇒ assn) a
  shows pos − 2 < max-unat 32
proof −
  obtain N vdom where
    ‹valid-arena a N vdom› and
    ‹b ∈# dom-m N›
    using assms(1) unfolding isa-update-pos-pre-def arena-is-valid-clause-idx-def
    by auto
  then have eq: ‹length (N ∝ b) = arena-length a b› and
    le: ‹b < length a› and
    size: ‹is-Size (a ! (b − SIZE-SHIFT))›
    by (auto simp: arena-lifting)

  have ‹i<length a ⟹ rdomp arena-el-impl-assn (a ! i)› for i
    using rdomp-al-dest'[OF assms(2)]
    by auto
  from this[of ‹b − SIZE-SHIFT›] have ‹rdomp arena-el-impl-assn (a ! (b − SIZE-SHIFT))›
    using le by auto
  then have ‹length (N ∝ b) ≤ uint32-max + 2›
    using size eq unfolding rdomp-pure
    apply (auto simp: rdomp-def arena-el-impl-rel-def is-Size-def
       comp-def pure-def unat-rel-def unat.rel-def br-def
       arena-length-def uint32-max-def)
     subgoal for x
       using unat-lt-max-unat[of x]
       apply (auto simp: max-unat-def)
       done
    done
  then show ?thesis
    using assms POS-SHIFT-def
    unfolding isa-update-pos-pre-def
    by (auto simp: arena-is-valid-clause-idx-def arena-lifting eq
       uint32-max-def max-unat-def)
qed

sepref-register arena-update-pos
sepref-def update-pos-impl
  is uncurry2 (RETURN ooo arena-update-pos)
    :: [isa-update-pos-pre]a sint64-nat-assn^k *a sint64-nat-assn^k *a arena-fast-assn^d → arena-fast-assn
  unfolding arena-update-pos-def POS-SHIFT-def
  apply (annot-snat-const TYPE(64))
  apply (rewrite at APos ⌑ annot-snat-unat-downcast[where 'l=32])
  supply [simp] = update-posI and [dest] = update-posI2
  by sepref


sepref-register IRRED LEARNED DELETED
lemma IRRED-impl[sepref-import-param]: (0,IRRED) ∈ status-impl-rel
  unfolding status-impl-rel-def status-rel-def unat-rel-def unat.rel-def
```

by (*auto simp*: *in-br-conv*)

lemma *LEARNED-impl*[*sepref-import-param*]: (*1*,*LEARNED*) ∈ *status-impl-rel*
unfolding *status-impl-rel-def status-rel-def unat-rel-def unat.rel-def*
by (*auto simp*: *in-br-conv*)

lemma *DELETED-impl*[*sepref-import-param*]: (*3*,*DELETED*) ∈ *status-impl-rel*
unfolding *status-impl-rel-def status-rel-def unat-rel-def unat.rel-def*
by (*auto simp*: *in-br-conv*)

lemma *mark-garbageI*:
assumes *mark-garbage-pre* (*a*, *b*)
shows $4 \leq b$ $b-4 <$ *length a*
using *assms STATUS-SHIFT-def*
unfolding *mark-garbage-pre-def*
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)
apply (*metis* (*full-types*) *arena-dom-status-iff*(*5*) *insertCI valid-arena-extra-information-mark-to-delete*)
by (*simp add*: *less-imp-diff-less valid-arena-def*)

sepref-register *extra-information-mark-to-delete*
sepref-def *mark-garbage-impl* **is** *uncurry* (*RETURN oo extra-information-mark-to-delete*)
:: $[mark\text{-}garbage\text{-}pre]_a$ $arena\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$
unfolding *extra-information-mark-to-delete-def STATUS-SHIFT-def*
apply (*annot-snat-const TYPE*(*64*))
supply [*simp*] = *mark-garbageI*
by *sepref*

Activity **lemma** *arena-act-implI*:
assumes *arena-act-pre a b*
shows $3 \leq b$ $b - 3 <$ *length a* *is-Act* (*a* ! (*b*−*3*))
using *assms ACTIVITY-SHIFT-def*
apply (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def arena-lifting*)
by (*simp add*: *less-imp-diff-less valid-arena-def*)

sepref-register *arena-act*
sepref-def *arena-act-impl*
is *uncurry* (*RETURN oo arena-act*)
:: $[uncurry\ arena\text{-}act\text{-}pre]_a$ $arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow uint32\text{-}nat\text{-}assn$
supply [*intro*] = *arena-act-implI*
unfolding *arena-act-def ACTIVITY-SHIFT-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Increment Activity **context begin**

interpretation *llvm-prim-arith-setup* **.**

sepref-register *op-incr-mod32*
lemma *op-incr-mod32-hnr*[*sepref-fr-rules*]:
($\lambda x.$ *ll-add x 1*, *RETURN o op-incr-mod32*) ∈ $uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$
unfolding *unat-rel-def unat.rel-def*
apply *sepref-to-hoare*
apply (*simp add*: *in-br-conv*)
apply *vcg'*

unfolding *op-incr-mod32-def*
by (*simp add*: *unat-word-ariths*)

end

sepref-register *arena-incr-act*
sepref-def *arena-incr-act-impl* **is** *uncurry* (*RETURN oo arena-incr-act*)
:: $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$
unfolding *arena-incr-act-def ACTIVITY-SHIFT-def*
supply [*intro*] = *arena-act-implI*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *arena-decr-act*
sepref-def *arena-decr-act-impl* **is** *uncurry* (*RETURN oo arena-decr-act*)
:: $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$
unfolding *arena-decr-act-def ACTIVITY-SHIFT-def*
supply [*intro*] = *arena-act-implI*
apply (*rewrite at - div* ¤ *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Mark used **term** *mark-used*

lemma *arena-mark-used-implI*:
assumes *arena-act-pre a b*
shows $4 \leq b$ $b - 4 < length\ a$ *is-Status* $(a\ !\ (b-4))$
using *assms STATUS-SHIFT-def*
apply (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def arena-lifting*)
subgoal by (*metis* (*full-types*) *arena-is-valid-clause-vdom-def arena-status-implI*(*1*) *insertCI valid-arena-extra-informat*
subgoal by (*simp add*: *less-imp-diff-less valid-arena-def*)
done

sepref-register *mark-used*
sepref-def *mark-used-impl* **is** *uncurry* (*RETURN oo mark-used*)
:: $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$
unfolding *mark-used-def STATUS-SHIFT-def*
supply [*intro*] = *arena-mark-used-implI*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *mark-unused*
sepref-def *mark-unused-impl* **is** *uncurry* (*RETURN oo mark-unused*)
:: $[uncurry\ arena\text{-}act\text{-}pre]_a\ arena\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arena\text{-}fast\text{-}assn$
unfolding *mark-unused-def STATUS-SHIFT-def*
supply [*intro*] = *arena-mark-used-implI*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

Marked as used? **lemma** *arena-marked-as-used-implI*:
assumes *marked-as-used-pre a b*
shows $4 \leq b$ $b - 4 < length\ a$ *is-Status* $(a\ !\ (b-4))$
using *assms STATUS-SHIFT-def*
apply (*auto simp*: *marked-as-used-pre-def arena-is-valid-clause-idx-def arena-lifting*)

subgoal by (*metis* (*full-types*) *arena-is-valid-clause-vdom-def arena-status-implI*(*1*) *insertCI valid-arena-extra-informat*
subgoal by (*simp add*: *less-imp-diff-less valid-arena-def*)
done

sepref-register *marked-as-used*
sepref-def *marked-as-used-impl*
is *uncurry* (*RETURN oo marked-as-used*)
:: $[uncurry\ marked\text{-}as\text{-}used\text{-}pre]_a\ arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow bool1\text{-}assn$
supply [*intro*] = *arena-marked-as-used-implI*
unfolding *marked-as-used-def STATUS-SHIFT-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *MAX-LENGTH-SHORT-CLAUSE*
sepref-def *MAX-LENGTH-SHORT-CLAUSE-impl* **is** *uncurry0* (*RETURN MAX-LENGTH-SHORT-CLAUSE*)
:: $unit\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$
unfolding *MAX-LENGTH-SHORT-CLAUSE-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

definition *arena-other-watched-as-swap* :: ‹*nat list* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat nres*› **where**
‹*arena-other-watched-as-swap S L C i* = *do* {
ASSERT(*i* < *2* ∧
C + *i* < *length S* ∧
C < *length S* ∧
(*C* + *1*) < *length S*);
K ← *RETURN* (*S* ! *C*);
K' ← *RETURN* (*S* ! (*1* + *C*));
RETURN (*L XOR K XOR K'*)
}›

lemma *arena-other-watched-as-swap-arena-other-watched*:
assumes
N: ‹(*N*, *N'*) ∈ ⟨*arena-el-rel*⟩*list-rel*› **and**
L: ‹(*L*, *L'*) ∈ *nat-lit-rel*› **and**
C: ‹(*C*, *C'*) ∈ *nat-rel*› **and**
i: ‹(*i*, *i'*) ∈ *nat-rel*›
shows
‹*arena-other-watched-as-swap N L C i* ≤ ⇓*nat-lit-rel*
(*arena-other-watched N' L' C' i'*)›
proof −
have *eq*: ‹*i* =*i'*› ‹*C*=*C'*›
using *assms* **by** *auto*
have *A*: ‹*Pos* (*L div 2*) = *A* ⟹ *even L* ⟹ *L* = *2* ∗ *atm-of A*› **for** *A* :: ‹*nat literal*›
by (*cases A*)
auto
have *Ci*: ‹(*C'* + *i'*, *C'* + *i'*) ∈ *nat-rel*›
unfolding *eq* **by** *auto*
have [*simp*]: ‹*L* = *N* ! (*C*+*i*)› **if** ‹*L'* = *arena-lit N'* (*C'* + *i'*)› ‹*C'* + *i'* < *length N'*›
‹*arena-lit-pre2 N' C i*›
using *that param-nth*[*OF that*(*2*) *Ci N*] *C i L*
unfolding *arena-lit-pre2-def*
apply − **apply** *normalize-goal*+
subgoal for *N'' vdom*
using *arena-lifting*(*6*)[*of N' N'' vdom C i*] *A*[*of* ‹*arena-lit N'* (*C'* + *i'*)›]

apply (*simp only*: *list-rel-imp-same-length*[*of N*] *eq*)
apply (*cases* ‹N' ! (C' + i')›; *cases* ‹*arena-lit* N' (C' + i')›)
apply (*simp-all add*: *eq nat-lit-rel-def br-def*)
apply (*auto split*: *if-splits simp*: *eq-commute*[*of* - ‹*Pos* (*L div 2*)›]
eq-commute[*of* - ‹*ALit* (*Pos* (- *div 2*))›] *arena-lit-def*)
using *div2-even-ext-nat* **by** *blast*
done
have [*simp*]: ‹N ! (C' + i') *XOR* N ! C' *XOR* N ! *Suc* C' = N ! (C' + (*Suc 0* − *i*))› **if** ‹$i < 2$›
using *that i*
by (*cases i*; *cases* ‹*i*−*1*›)
(*auto simp*: *bin-pos-same-XOR3-nat*)
have *Ci'*: ‹(C' + (*1* − i'), C' + (*1* − i')) ∈ *nat-rel*›
unfolding *eq* **by** *auto*

have [*intro*!]: ‹(N ! (*Suc* C' − i'), *arena-lit* N' (*Suc* C' − i')) ∈ *nat-lit-rel*›
if ‹*arena-lit-pre2* N' *C i*› ‹$i < 2$›
using *that param-nth*[*OF* - *Ci'* *N*]
unfolding *arena-lit-pre2-def*
apply − **apply** *normalize-goal*+
apply (*subgoal-tac* ‹C' + (*Suc 0* − i') < *length* N'›)
defer
subgoal for N'' *vdom*
using
arena-lifting(*7*)[*of* N' N'' *vdom C i*]
apply (*auto simp*: *arena-lit-pre2-def list-rel-imp-same-length*[*of N*]
simp del: *arena-el-rel-def*)
by (*metis add.right-neutral add-Suc add-diff-cancel-left' arena-lifting*(*22*) *arena-lifting*(*4*) *arena-lifting*(*7*) *eq*(*1*) *eq*(*2*) *less-2-cases less-Suc-eq not-less-eq plus-1-eq-Suc*)
apply (*subgoal-tac* ‹(*Suc 0* − i') < *length* ($x \propto C$)›)
defer
subgoal for N'' *vdom*
using
arena-lifting(*7*)[*of* N' N'' *vdom C i*]
by (*auto simp*: *arena-lit-pre2-def list-rel-imp-same-length*[*of N*]
arena-lifting(*22*) *arena-lifting*(*4*) *less-imp-diff-less*
simp del: *arena-el-rel-def*)
subgoal for N'' *vdom*
using
arena-lifting(*6*)[*of* N' N'' *vdom C* ‹*Suc 0* − *i*›]
by (*cases* ‹N' ! (C' + (*Suc 0* − i'))›)
(*auto simp*: *arena-lit-pre2-def list-rel-imp-same-length*[*of N*] *eq*
arena-lit-def arena-lifting)
done
show *?thesis*
using *assms*
unfolding *arena-other-watched-as-swap-def arena-other-watched-def*
le-ASSERT-iff mop-arena-lit2-def
apply (*refine-vcg*)
apply (*auto simp*: *le-ASSERT-iff list-rel-imp-same-length arena-lit-pre2-def*
arena-lifting
bin-pos-same-XOR3-nat)
using *arena-lifting*(*22*) *arena-lifting*(*4*) *arena-lifting*(*7*) **apply** *fastforce*
using *arena-lifting*(*22*) *arena-lifting*(*4*) *arena-lifting*(*7*) *arena-lit-pre2-def* **apply** *fastforce*
done
qed

sepref-def *arena-other-watched-as-swap-impl*
is ‹*uncurry3 arena-other-watched-as-swap*›
:: ‹$(al\text{-}assn'\ (TYPE(64))\ uint32\text{-}nat\text{-}assn)^k *_a uint32\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
supply[[*goals-limit=1*]]
unfolding *arena-other-watched-as-swap-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *arena-other-watched-as-swap-arena-other-watched′*:
‹(*arena-other-watched-as-swap*, *arena-other-watched*) ∈
⟨*arena-el-rel*⟩*list-rel* → *nat-lit-rel* → *nat-rel* → *nat-rel* →
⟨*nat-lit-rel*⟩*nres-rel*›
apply (*intro fun-relI nres-relI*)
using *arena-other-watched-as-swap-arena-other-watched*
by *blast*

lemma *arena-fast-al-unat-assn*:
‹*hr-comp* (*al-assn unat-assn*) (⟨*arena-el-rel*⟩*list-rel*) = *arena-fast-assn*›
unfolding *al-assn-def hr-comp-assoc*
by (*auto simp*: *arena-el-impl-rel-def list-rel-compp*)

lemmas [*sepref-fr-rules*] =
arena-other-watched-as-swap-impl.refine[*FCOMP arena-other-watched-as-swap-arena-other-watched′*,
unfolded arena-fast-al-unat-assn]

end

sepref-def *mop-arena-length-impl*
is ‹*uncurry mop-arena-length*›
:: ‹$arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
unfolding *mop-arena-length-def*
by *sepref*

experiment begin
export-llvm
arena-length-impl
arena-lit-impl
arena-status-impl
swap-lits-impl
arena-lbd-impl
arena-pos-impl
update-lbd-impl
update-pos-impl
mark-garbage-impl
arena-act-impl
arena-incr-act-impl
arena-decr-act-impl
mark-used-impl
mark-unused-impl
marked-as-used-impl
MAX-LENGTH-SHORT-CLAUSE-impl

end

end
theory *IsaSAT-Clauses*
imports *IsaSAT-Arena*
begin

Chapter 3

The memory representation: Manipulation of all clauses

Representation of Clauses

named-theorems *isasat-codegen* ‹*lemmas that should be unfolded to generate (efficient) code*›

type-synonym *clause-annot* = ‹*clause-status* × *nat* × *nat*›

type-synonym *clause-annots* = ‹*clause-annot list*›

definition *list-fmap-rel* :: ‹- ⇒ (*arena* × *nat clauses-l*) *set*› **where**
‹*list-fmap-rel vdom* = {(*arena*, *N*). *valid-arena arena N vdom*}›

lemma *nth-clauses-l*:
‹(*uncurry2* (*RETURN ooo* (λ*N i j*. *arena-lit N* (*i*+*j*))),
uncurry2 (*RETURN ooo* (λ*N i j*. *N* ∝ *i* ! *j*)))
∈ [λ((*N*, *i*), *j*). *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*)]$_f$
list-fmap-rel vdom ×$_f$ *nat-rel* ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *list-fmap-rel-def arena-lifting*)

abbreviation *clauses-l-fmat* **where**
‹*clauses-l-fmat* ≡ *list-fmap-rel*›

type-synonym *vdom* = ‹*nat set*›

definition *fmap-rll* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll l i j* = *l* ∝ *i* ! *j*›

definition *fmap-rll-u* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll-u* = *fmap-rll*›

definition *fmap-rll-u64* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ ′*a literal* **where**
[*simp*]: ‹*fmap-rll-u64* = *fmap-rll*›

definition *fmap-length-rll-u* :: (*nat*, ′*a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
‹*fmap-length-rll-u l i* = *length-uint32-nat* (*l* ∝ *i*)›

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll-u64* :: (*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
⟨*fmap-length-rll-u64 l i* = *length-uint32-nat* (*l* ∝ *i*)⟩

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll* :: (*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* **where**
[*simp*]: ⟨*fmap-length-rll l i* = *length* (*l* ∝ *i*)⟩

definition *fmap-swap-ll* **where**
[*simp*]: ⟨*fmap-swap-ll N i j f* = (*N*(*i* ↪ *swap* (*N* ∝ *i*) *j f*))⟩

From a performance point of view, appending several time a single element is less efficient than reserving a space that is large enough directly. However, in this case the list of clauses N is so large that there should not be any difference

definition *fm-add-new* **where**
⟨*fm-add-new b C N0* = *do* {
  let st = (*if b then AStatus IRRED False else AStatus LEARNED False*);
  let l = *length N0*;
  let s = *length C* − *2*;
  let N = (*if is-short-clause C then*
      (((*N0* @ [*st*]) @ [*AActivity 0*]) @ [*ALBD s*]) @ [*ASize s*]
      else ((((*N0* @ [*APos 0*]) @ [*st*]) @ [*AActivity 0*]) @ [*ALBD s*]) @ [*ASize* (*s*)]);
  (*i*, *N*) ← $WHILE_T$ $^{\lambda(i,\ N).\ i\ <\ length\ C\ \longrightarrow\ length\ N\ <\ header\text{-}size\ C\ +\ length\ N0\ +\ length\ C}$
   (λ(*i*, *N*). *i* < *length C*)
   (λ(*i*, *N*). *do* {
    ASSERT(*i* < *length C*);
    RETURN (*i*+*1*, *N* @ [*ALit* (*C* ! *i*)])
   })
   (*0*, *N*);
  RETURN (*N*, *l* + *header-size C*)
}⟩

lemma *header-size-Suc-def*:
⟨*header-size C* =
  (*if is-short-clause C then Suc* (*Suc* (*Suc* (*Suc 0*))) *else Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))⟩
unfolding *header-size-def*
by *auto*

lemma *nth-append-clause*:
⟨*a* < *length C* ⟹ *append-clause b C N* ! (*length N* + *header-size C* + *a*) = *ALit* (*C* ! *a*)⟩
unfolding *append-clause-def header-size-Suc-def append-clause-skeleton-def*
by (*auto simp*: *nth-Cons nth-append*)

lemma *fm-add-new-append-clause*:
⟨*fm-add-new b C N* ≤ *RETURN* (*append-clause b C N*, *length N* + *header-size C*)⟩
unfolding *fm-add-new-def*
apply (*rewrite at* ⟨*let* - = *length* - *in* -⟩ *Let-def*)
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ⟨*measure* (λ(*i*, -). *Suc* (*length C*) − *i*)⟩ **and**
 I' = ⟨λ(*i*, *N'*). *N'* = *take* (*length N* + *header-size C* + *i*) (*append-clause b C N*) ∧
  i ≤ *length C*⟩])
subgoal by *auto*
subgoal by (*auto simp*: *append-clause-def header-size-def*
 append-clause-skeleton-def split: *if-splits*)

subgoal by (*auto simp*: *append-clause-def header-size-def*
append-clause-skeleton-def split: *if-splits*)
subgoal by *simp*
subgoal by *simp*
subgoal by *auto*
subgoal by (*auto simp*: *take-Suc-conv-app-nth nth-append-clause*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

definition *fm-add-new-at-position*
:: ‹$bool \Rightarrow nat \Rightarrow 'v$ *clause-l* $\Rightarrow 'v$ *clauses-l* $\Rightarrow 'v$ *clauses-l*›
where
‹*fm-add-new-at-position* $b\ i\ C\ N$ = *fmupd* $i\ (C,\ b)\ N$›

definition *AStatus-IRRED* **where**
‹*AStatus-IRRED* = *AStatus IRRED False*›

definition *AStatus-IRRED2* **where**
‹*AStatus-IRRED2* = *AStatus IRRED True*›

definition *AStatus-LEARNED* **where**
‹*AStatus-LEARNED* = *AStatus LEARNED True*›

definition *AStatus-LEARNED2* **where**
‹*AStatus-LEARNED2* = *AStatus LEARNED False*›

definition (**in** −)*fm-add-new-fast* **where**
[*simp*]: ‹*fm-add-new-fast* = *fm-add-new*›

lemma (**in** −)*append-and-length-code-fast*:
‹*length ba* $\leq$ *Suc* (*Suc uint32-max*) $\Longrightarrow$
$2 \leq$ *length ba* $\Longrightarrow$
length b $\leq$ *uint64-max* − (*uint32-max* + 5) $\Longrightarrow$
(*aa*, *header-size ba*) $\in$ *uint64-nat-rel* $\Longrightarrow$
(*ab*, *length b*) $\in$ *uint64-nat-rel* $\Longrightarrow$
length b + *header-size ba* $\leq$ *uint64-max*›
by (*auto simp*: *uint64-max-def uint32-max-def header-size-def*)

definition (**in** −)*four-uint64-nat* **where**
[*simp*]: ‹*four-uint64-nat* = (4 :: *nat*)›
definition (**in** −)*five-uint64-nat* **where**
[*simp*]: ‹*five-uint64-nat* = (5 :: *nat*)›

definition *append-and-length-fast-code-pre* **where**
‹*append-and-length-fast-code-pre* $\equiv \lambda((b,\ C),\ N).$ *length C* $\leq$ *uint32-max*+2 $\wedge$ *length C* $\geq 2\ \wedge$
length N + *length C* + 5 $\leq$ *sint64-max*›

lemma *fm-add-new-alt-def*:
‹*fm-add-new b C N0* = *do* {

let st = (*if b then AStatus-IRRED else AStatus-LEARNED2*);
let l = *length N0*;
let s = *length C* − *2*;
let N =
(*if is-short-clause C*
then (((*N0* @ [*st*]) @ [*AActivity 0*]) @ [*ALBD s*]) @
[*ASize s*]
else ((((*N0* @ [*APos 0*]) @ [*st*]) @
[*AActivity 0*]) @
[*ALBD s*]) @
[*ASize s*]);
(*i*, *N*) ←
$WHILE_T$ λ(*i*, *N*). *i* < *length C* ⟶ *length N* < *header-size C* + *length N0* + *length C*
(λ(*i*, *N*). *i* < *length C*)
(λ(*i*, *N*). *do* {
- ← *ASSERT* (*i* < *length C*);
RETURN (*i* + *1*, *N* @ [*ALit* (*C* ! *i*)])
})
(*0*, *N*);
RETURN (*N*, *l* + *header-size C*)
}›
unfolding *fm-add-new-def Let-def AStatus-LEARNED2-def AStatus-IRRED2-def AStatus-LEARNED-def AStatus-IRRED-def*
by *auto*

definition *fmap-swap-ll-u64* **where**
[*simp*]: ‹*fmap-swap-ll-u64* = *fmap-swap-ll*›

definition *fm-mv-clause-to-new-arena* **where**
‹*fm-mv-clause-to-new-arena C old-arena new-arena0* = *do* {
ASSERT(*arena-is-valid-clause-idx old-arena C*);
ASSERT(*C* ≥ (*if* (*arena-length old-arena C*) ≤ *4 then 4 else 5*));
let st = *C* − (*if* (*arena-length old-arena C*) ≤ *4 then 4 else 5*);
ASSERT(*C* + (*arena-length old-arena C*) ≤ *length old-arena*);
let en = *C* + (*arena-length old-arena C*);
(*i*, *new-arena*) ←
$WHILE_T$ λ(*i*, *new-arena*). *i* < *en* ⟶ *length new-arena* < *length new-arena0* + (*arena-length old-arena C*) + (*if* (*arena-l*
(λ(*i*, *new-arena*). *i* < *en*)
(λ(*i*, *new-arena*). *do* {
ASSERT (*i* < *length old-arena* ∧ *i* < *en*);
RETURN (*i* + *1*, *new-arena* @ [*old-arena* ! *i*])
})
(*st*, *new-arena0*);
RETURN (*new-arena*)
}›

lemma *valid-arena-append-clause-slice*:
assumes
‹*valid-arena old-arena N vd*› **and**
‹*valid-arena new-arena N′ vd′*› **and**
‹*C* ∈# *dom-m N*›
shows ‹*valid-arena* (*new-arena* @ *clause-slice old-arena N C*)
(*fmupd* (*length new-arena* + *header-size* (*N* ∝ *C*)) (*N* ∝ *C*, *irred N C*) *N′*)
(*insert* (*length new-arena* + *header-size* (*N* ∝ *C*)) *vd′*)›
proof −

define *pos st lbd act used* **where**
⟨*pos* = (*if is-long-clause* ($N \propto C$) *then arena-pos old-arena* $C - 2$ *else 0*)⟩ **and**
⟨*st* = *arena-status old-arena C*⟩ **and**
⟨*lbd* = *arena-lbd old-arena C*⟩ **and**
⟨*act* = *arena-act old-arena C*⟩ **and**
⟨*used* = *arena-used old-arena C*⟩
have ⟨$2 \le$ *length* ($N \propto C$)⟩
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def
arena-lbd-def xarena-lbd-def
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def pos-def arena-pos-def
xarena-pos-def
arena-lbd-def xarena-lbd-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def is-LBD-def*
is-Act-def)
have
45: ⟨*4* = (*Suc* (*Suc* (*Suc* (*Suc 0*))))⟩
⟨*5* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))⟩
by *auto*
have *sl*: ⟨*clause-slice old-arena N C* =
(*if is-long-clause* ($N \propto C$) *then* [*APos pos*]
else []) @
[*AStatus st used*, *AActivity act*, *ALBD lbd*, *ASize* (*length* ($N \propto C$) − *2*)] @
map ALit ($N \propto C$) ⟩
unfolding *st-def used-def act-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
arena-act-def xarena-used-def
xarena-act-def pos-def arena-pos-def
xarena-pos-def
arena-lbd-def xarena-lbd-def
arena-length-def xarena-length-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def is-LBD-def*
is-Act-def header-size-def 45
slice-Suc-nth[*of* ⟨*C* − *Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*))))⟩]
slice-Suc-nth[*of* ⟨*C* − *Suc* (*Suc* (*Suc* (*Suc 0*)))⟩]
slice-Suc-nth[*of* ⟨*C* − *Suc* (*Suc* (*Suc 0*))⟩]
slice-Suc-nth[*of* ⟨*C* − *Suc* (*Suc 0*)⟩]
slice-Suc-nth[*of* ⟨*C* − *Suc 0*⟩]
SHIFTS-alt-def arena-length-def
arena-pos-def xarena-pos-def
arena-status-def xarena-status-def)

have ⟨$2 \le$ *length* ($N \propto C$)⟩ **and**
⟨*pos* $\le$ *length* ($N \propto C$) − *2*⟩ **and**
⟨*st* = *IRRED* $\longleftrightarrow$ *irred N C*⟩ **and**
⟨*st* $\ne$ *DELETED*⟩

unfolding *st-def used-def act-def lbd-def pos-def*
append-clause-skeleton-def st-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*cases* ‹*is-short-clause* ($N \propto C$)›;
auto split: *arena-el.splits if-splits*
simp: *header-size-def arena-pos-def*; *fail*)+

then have ‹*valid-arena* (*append-clause-skeleton pos st used act lbd* ($N \propto C$) *new-arena*)
(*fmupd* (*length new-arena* + *header-size* ($N \propto C$)) ($N \propto C$, *irred N C*) N')
(*insert* (*length new-arena* + *header-size* ($N \propto C$)) vd')›
apply −
by (*rule valid-arena-append-clause-skeleton*[*OF assms*(*2*), *of* ‹$N \propto C$› - *st*
pos used act lbd]) *auto*
moreover have
‹*append-clause-skeleton pos st used act lbd* ($N \propto C$) *new-arena* =
new-arena @ *clause-slice old-arena N C*›
by (*auto simp*: *append-clause-skeleton-def sl*)
ultimately show *?thesis*
by *auto*
qed

lemma *fm-mv-clause-to-new-arena*:
assumes ‹*valid-arena old-arena N vd*› **and**
‹*valid-arena new-arena* N' vd'› **and**
‹$C \in\#$ *dom-m N*›
shows ‹*fm-mv-clause-to-new-arena C old-arena new-arena* ≤
SPEC(λ*new-arena*′.
new-arena′ = *new-arena* @ *clause-slice old-arena N C* ∧
valid-arena (*new-arena* @ *clause-slice old-arena N C*)
(*fmupd* (*length new-arena* + *header-size* ($N \propto C$)) ($N \propto C$, *irred N C*) N')
(*insert* (*length new-arena* + *header-size* ($N \propto C$)) vd'))›
proof −
define *st* **and** *en* **where**
‹*st* = *C* − (*if arena-length old-arena C* ≤ *4 then 4 else 5*)› **and**
‹*en* = *C* + *arena-length old-arena C*›
have *st*:
‹*st* = *C* − *header-size* ($N \propto C$)›
using *assms*
unfolding *st-def*
by (*auto simp*: *st-def header-size-def*
arena-lifting)
show *?thesis*
using *assms*
unfolding *fm-mv-clause-to-new-arena-def st-def*[*symmetric*]
en-def[*symmetric*] *Let-def*
apply (*refine-vcg*
WHILEIT-rule-stronger-inv[**where** *R* = ‹*measure* (λ(*i*, *N*). *en* − *i*)› **and**
I′ = ‹λ(*i*, *new-arena*′). *i* ≤ *C* + *length* ($N \propto C$) ∧ *i* ≥ *st* ∧
new-arena′ = *new-arena* @
Misc.slice (*C* − *header-size* ($N \propto C$)) *i old-arena*›])
subgoal
unfolding *arena-is-valid-clause-idx-def*
by *auto*
subgoal using *arena-lifting*(*4*)[*OF assms*(*1*)] **by** (*auto*
dest!: *arena-lifting*(*1*)[*of* - *N* - *C*] *simp*: *header-size-def split*: *if-splits*)
subgoal using *arena-lifting*(*10*, *4*) *en-def* **by** *auto*

subgoal
  by *auto*
subgoal by *auto*
subgoal
  using *arena-lifting*[*OF assms*(*1*,*3*)]
  by (*auto simp*: *st*)
subgoal
  by (*auto simp*: *st arena-lifting*)
subgoal
  using *arena-lifting*[*OF assms*(*1*,*3*)]
  by (*auto simp*: *st en-def*)
subgoal
  using *arena-lifting*[*OF assms*(*1*,*3*)]
  by (*auto simp*: *st en-def*)
subgoal by *auto*
subgoal using *arena-lifting*[*OF assms*(*1*,*3*)]
    by (*auto simp*: *slice-len-min-If en-def st-def header-size-def*)
subgoal
  using *arena-lifting*[*OF assms*(*1*,*3*)]
  by (*auto simp*: *st en-def*)
subgoal
  using *arena-lifting*[*OF assms*(*1*,*3*)]
  by (*auto simp*: *st*)
subgoal
  by (*auto simp*: *st en-def arena-lifting*[*OF assms*(*1*,*3*)]
    slice-append-nth)
subgoal by *auto*
subgoal by (*auto simp*: *en-def arena-lifting*)
subgoal
  using *valid-arena-append-clause-slice*[*OF assms*]
  by *auto*
done
qed

lemma *size-learned-clss-dom-m*: ‹*size* (*learned-clss-l N*) $\leq$ *size* (*dom-m N*)›
  unfolding *ran-m-def*
  apply (*rule order-trans*[*OF size-filter-mset-lesseq*])
  by (*auto simp*: *ran-m-def*)

lemma *valid-arena-ge-length-clauses*:
  assumes ‹*valid-arena arena N vdom*›
  shows ‹*length arena* $\geq$ ($\sum$ *C* $\in\#$ *dom-m N*. *length* (*N* $\propto$ *C*) + *header-size* (*N* $\propto$ *C*))›
proof −
  obtain *xs* **where**
    mset-xs: ‹*mset xs* = *dom-m N*› **and** *sorted*: ‹*sorted xs*› **and** *dist*[*simp*]: ‹*distinct xs*› **and** *set-xs*: ‹*set xs* = *set-mset* (*dom-m N*)›
    using *distinct-mset-dom distinct-mset-mset-distinct mset-sorted-list-of-multiset* **by** *fastforce*
  then have *1*: ‹*set-mset* (*mset xs*) = *set xs*› **by** (*meson set-mset-mset*)

  have *diff*: ‹*xs* $\neq$ [] $\Longrightarrow$ *a* $\in$ *set xs* $\Longrightarrow$ *a* < *last xs* $\Longrightarrow$ *a* + *length* (*N* $\propto$ *a*) $\leq$ *last xs*› **for** *a*
    using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a* ‹*last xs*›]
    mset-xs[*symmetric*] *sorted* **by** (*cases xs rule*: *rev-cases*; *auto simp*: *sorted-append*)
  have ‹*set xs* $\subseteq$ *set-mset* (*dom-m N*)›
    using *mset-xs*[*symmetric*] **by** *auto*
  then have ‹($\sum$ *A*$\in$*set xs*. *length* (*N* $\propto$ *A*) + *header-size* (*N* $\propto$ *A*)) $\leq$ *Max* (*insert 0* ((*λA*. *A* + *length*

(N ∝ A)) ‘ (set xs)))›
(**is** ‹?P xs ≤ ?Q xs›)
using *sorted dist*
proof (*induction xs rule*: *rev-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*snoc x xs*)
then have *IH*: ‹(∑ A∈set xs. length (N ∝ A) + header-size (N ∝ A))
≤ Max (insert 0 ((λA. A + length (N ∝ A)) ‘ set xs))› **and**
x-dom: ‹x ∈# dom-m N› **and**
x-max: ‹⋀a. a ∈ set xs ⟹ x > a› **and**
xs-N: ‹set xs ⊆ set-mset (dom-m N)›
by (*auto simp*: *sorted-append order.order-iff-strict dest*!: *bspec*)
have *x-ge*: ‹header-size (N ∝ x) ≤ x›
using *assms* ‹x ∈# dom-m N› *arena-lifting*(*1*) **by** *blast*
have *diff*: ‹a ∈ set xs ⟹ a + length (N ∝ a) + header-size (N ∝ x) ≤ x›
‹a ∈ set xs ⟹ a + length (N ∝ a) ≤ x› **for** *a*
using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a x*]
x-max[*of a*] *xs-N x-dom* **by** *auto*

have ‹?P (xs @ [x]) ≤ ?P xs + length (N ∝ x) + header-size (N ∝ x)›
using *snoc* **by** *auto*
also have ‹... ≤ ?Q xs + (length (N ∝ x) + header-size (N ∝ x))›
using *IH* **by** *auto*
also have ‹... ≤ (length (N ∝ x) + x)›
by (*subst linordered-ab-semigroup-add-class.Max-add-commute2*[*symmetric*]; *auto intro*: *diff x-ge*)
also have ‹... = Max (insert (x + length (N ∝ x)) ((λx. x + length (N ∝ x)) ‘ set xs))›
by (*subst eq-commute*)
(*auto intro*!: *linorder-class.Max-eqI intro*: *order-trans*[*OF diff*(*2*)])
finally show *?case* **by** *auto*
qed
also have ‹... ≤ (if xs = [] then 0 else last xs + length (N ∝ last xs))›
using *sorted distinct-sorted-append*[*of* ‹butlast xs› ‹last xs›] *dist*
by (*cases* ‹xs› *rule*: *rev-cases*)
(*auto intro*: *order-trans*[*OF diff*])
also have ‹... ≤ length arena›
using *arena-lifting*(*7*)[*OF assms*, *of* ‹last xs› ‹length (N ∝ last xs) − 1›] *mset-xs*[*symmetric*] *assms*
by (*cases* ‹xs› *rule*: *rev-cases*) (*auto simp*: *arena-lifting*)
finally show *?thesis*
unfolding *mset-xs*[*symmetric*]
by (*subst distinct-sum-mset-sum*) *auto*
qed

lemma *valid-arena-size-dom-m-le-arena*: ‹valid-arena arena N vdom ⟹ size (dom-m N) ≤ length arena›
using *valid-arena-ge-length-clauses*[*of arena N vdom*]
ordered-comm-monoid-add-class.sum-mset-mono[*of* ‹dom-m N› ‹λ-. 1›
‹λC. length (N ∝ C) + header-size (N ∝ C)›]
by (*fastforce simp*: *header-size-def split*: *if-splits*)

end
theory *IsaSAT-Clauses-LLVM*
imports *IsaSAT-Clauses IsaSAT-Arena-LLVM*
begin

sepref-register *is-short-clause header-size fm-add-new-fast fm-mv-clause-to-new-arena*

abbreviation *clause-ll-assn* :: ‹*nat clause-l* ⇒ - ⇒ *assn*› **where**
‹*clause-ll-assn* ≡ *larray64-assn unat-lit-assn*›

sepref-def *is-short-clause-code*
is ‹*RETURN o is-short-clause*›
:: ‹ $clause\text{-}ll\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
unfolding *is-short-clause-def*
by *sepref*

sepref-def *header-size-code*
is ‹*RETURN o header-size*›
:: ‹$clause\text{-}ll\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
unfolding *header-size-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *header-size-bound*: *header-size x* ≤ *5* **by** (*auto simp*: *header-size-def*)

lemma *fm-add-new-bounds1*: ⟦
length a2' < *header-size baa* + *length b* + *length baa*;
length b + *length baa* + *5* ≤ *sint64-max* ⟧
⟹ *Suc* (*length a2'*) < *max-snat 64*

length b + *length baa* + *5* ≤ *sint64-max* ⟹ *length b* + *header-size baa* < *max-snat 64*
using *header-size-bound*[*of baa*]
by (*auto simp*: *max-snat-def sint64-max-def*)

sepref-def *append-and-length-fast-code*
is ‹*uncurry2 fm-add-new-fast*›
:: ‹$[append\text{-}and\text{-}length\text{-}fast\text{-}code\text{-}pre]_a$
$bool1\text{-}assn^k *_a clause\text{-}ll\text{-}assn^k *_a (arena\text{-}fast\text{-}assn)^d \rightarrow$
$arena\text{-}fast\text{-}assn \times_a sint64\text{-}nat\text{-}assn$›
unfolding *fm-add-new-fast-def fm-add-new-def append-and-length-fast-code-pre-def*

apply (*rewrite at AActivity* ⌑ *unat-const-fold*[**where** *'a=32*])+
apply (*rewrite at APos* ⌑ *unat-const-fold*[**where** *'a=32*])+
apply (*rewrite at length - − 2 annot-snat-unat-downcast*[**where** *'l=32*])

supply [*simp*] = *fm-add-new-bounds1*[*simplified*]

apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *fm-mv-clause-to-new-arena-fast-code*
is ‹*uncurry2 fm-mv-clause-to-new-arena*›
:: ‹$[\lambda((n, arena_o), arena).\ length\ arena_o \leq sint64\text{-}max \wedge length\ arena + arena\text{-}length\ arena_o\ n +$
$(if\ arena\text{-}length\ arena_o\ \ n \leq 4\ then\ 4\ else\ 5) \leq sint64\text{-}max]_a$
$sint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow arena\text{-}fast\text{-}assn$›
supply [[*goals-limit=1*]] *if-splits*[*split*]
unfolding *fm-mv-clause-to-new-arena-def*

apply (*annot-snat-const* $TYPE(64)$)
by *sepref*

experiment begin
export-llvm
is-short-clause-code
header-size-code
append-and-length-fast-code
fm-mv-clause-to-new-arena-fast-code
end

end
theory *IsaSAT-Trail*
imports *IsaSAT-Literals*

begin

Chapter 4

Efficient Trail

Our trail contains several additional information compared to the simple trail:

- the (reversed) trail in an array (i.e., the trail in the same order as presented in “Automated Reasoning”);
- the mapping from any *literal* (and not an atom) to its polarity;
- the mapping from a *atom* to its level or reason (in two different arrays);
- the current level of the state;
- the control stack.

We copied the idea from the mapping from a literals to it polarity instead of an atom to its polarity from a comment by Armin Biere in CaDiCal. We only observed a (at best) faint performance increase, but as it seemed slightly faster and does not increase the length of the formalisation, we kept it.

The control stack is the latest addition: it contains the positions of the decisions in the trail. It is mostly to enable fast restarts (since it allows to directly iterate over all decision of the trail), but might also slightly speed up backjumping (since we know how far we are going back in the trail). Remark that the control stack contains is not updated during the backjumping, but only *after* doing it (as we keep only the the beginning of it).

4.1 Polarities

type-synonym *tri-bool* = ⟨*bool option*⟩

definition *UNSET* :: ⟨*tri-bool*⟩ **where**
[*simp*]: ⟨*UNSET* = *None*⟩

definition *SET-FALSE* :: ⟨*tri-bool*⟩ **where**
[*simp*]: ⟨*SET-FALSE* = *Some False*⟩

definition *SET-TRUE* :: ⟨*tri-bool*⟩ **where**
[*simp*]: ⟨*SET-TRUE* = *Some True*⟩

definition (**in** −) *tri-bool-eq* :: ⟨*tri-bool* ⇒ *tri-bool* ⇒ *bool*⟩ **where**
⟨*tri-bool-eq* = (=)⟩

4.2 Types

type-synonym *trail-pol* =
  ‹*nat literal list* × *tri-bool list* × *nat list* × *nat list* × *nat* × *nat list*›

definition *get-level-atm* **where**
  ‹*get-level-atm M L* = *get-level M* (*Pos L*)›

definition *polarity-atm* **where**
  ‹*polarity-atm M L* =
    (*if Pos L* ∈ *lits-of-l M then SET-TRUE*
    else if Neg L ∈ *lits-of-l M then SET-FALSE*
    else None)›

definition *defined-atm* :: ‹(′*v*, *nat*) *ann-lits* ⇒ ′*v* ⇒ *bool*› **where**
‹*defined-atm M L* = *defined-lit M* (*Pos L*)›

abbreviation *undefined-atm* **where**
  ‹*undefined-atm M L* ≡ ¬*defined-atm M L*›

4.3 Control Stack

inductive *control-stack* **where**
empty:
  ‹*control-stack* [] []› |
cons-prop:
  ‹*control-stack cs M* ⟹ *control-stack cs* (*Propagated L C* # *M*)› |
cons-dec:
  ‹*control-stack cs M* ⟹ *n* = *length M* ⟹ *control-stack* (*cs* @ [*n*]) (*Decided L* # *M*)›

inductive-cases *control-stackE*: ‹*control-stack cs M*›

lemma *control-stack-length-count-dec*:
  ‹*control-stack cs M* ⟹ *length cs* = *count-decided M*›
  by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-le-length-M*:
  ‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *c* < *length M*›
  by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-propa*[*simp*]:
  ‹*control-stack cs* (*Propagated x21 x22* # *list*) ⟷ *control-stack cs list*›
  by (*auto simp*: *control-stack.intros elim*: *control-stackE*)

lemma *control-stack-filter-map-nth*:
  ‹*control-stack cs M* ⟹ *filter is-decided* (*rev M*) = *map* (*nth* (*rev M*)) *cs*›
  apply (*induction rule*: *control-stack.induct*)
  subgoal by *auto*
  subgoal for *cs M L C*
    using *control-stack-le-length-M*[*of cs M*]
    by (*auto simp*: *nth-append*)
  subgoal for *cs M L*
    using *control-stack-le-length-M*[*of cs M*]
    by (*auto simp*: *nth-append*)
  done

lemma *control-stack-empty-cs*[*simp*]: ‹*control-stack* [] *M* ⟷ *count-decided M* = *0*›
  by (*induction M rule*:*ann-lit-list-induct*)
    (*auto simp*: *control-stack.empty control-stack.cons-prop elim*: *control-stackE*)

This is an other possible definition. It is not inductive, which makes it easier to reason about appending (or removing) some literals from the trail. It is however much less clear if the definition is correct.

definition *control-stack'* **where**
  ‹*control-stack' cs M* ⟷
    (*length cs* = *count-decided M* ∧
      (∀ *L*∈*set M*. *is-decided L* ⟶ (*cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧
        rev M!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*)))›

lemma *control-stack-rev-get-lev*:
  ‹*control-stack cs M* ⟹
    no-dup M ⟹ *L*∈*set M* ⟹ *is-decided L* ⟹ *rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
  apply (*induction arbitrary*: *L rule*: *control-stack.induct*)
  subgoal by *auto*
  subgoal for *cs M L C La*
    using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
      count-decided-ge-get-level[*of M* ‹*lit-of La*›]
    apply (*auto simp*: *get-level-cons-if nth-append atm-of-eq-atm-of undefined-notin*)
    by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
        le-SucI le-refl neq0-conv nth-mem)
  subgoal for *cs M L*
    using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
    apply (*auto simp*: *nth-append get-level-cons-if atm-of-eq-atm-of undefined-notin*)
    by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
        le-SucI le-refl neq0-conv)+
  done

lemma *control-stack-alt-def-imp*:
  ‹*no-dup M* ⟹ (⋀*L*. *L* ∈*set M* ⟹ *is-decided L* ⟹ *cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧
        rev M!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*) ⟹
    length cs = *count-decided M* ⟹
    control-stack cs M›
proof (*induction M arbitrary*: *cs rule*:*ann-lit-list-induct*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Decided L M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* = *this*(*4*)
  from *length* **obtain** *cs' n* **where** *cs*[*simp*]: ‹*cs* = *cs'* @ [*n*]›
    using *length* **by** (*cases cs rule*: *rev-cases*) *auto*
  have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*›
    by (*auto simp*: *count-decided-0-iff*)
  have *dec'*: ‹*L'*∈*set M* ⟹ *is-decided L'* ⟹ *cs'* ! (*get-level M* (*lit-of L'*) − *1*) < *length M* ∧
        rev M ! (*cs'* ! (*get-level M* (*lit-of L'*) − *1*)) = *L'*› **for** *L'*
    using *dec*[*of L'*] *n-d length*
    count-decided-ge-get-level[*of M* ‹*lit-of L'*›]
    apply (*auto simp*: *get-level-cons-if atm-of-eq-atm-of undefined-notin*
        split: *if-splits*)
    apply (*auto simp*: *nth-append split*: *if-splits*)
    done
  have *le*: ‹*length cs'* = *count-decided M*›

using *length* **by** *auto*
have [*simp*]: ‹*n* = *length M*›
using *n-d dec*[*of* ‹*Decided L*›] *le undefined-notin*[*of M* ‹*rev M* ! *n*›] *nth-mem*[*of n* ‹*rev M*›]
by (*auto simp*: *nth-append split*: *if-splits*)
show *?case*
unfolding *cs*
apply (*rule control-stack.cons-dec*)
subgoal
apply (*rule IH*)
using *n-d dec′ le* **by** *auto*
subgoal by *auto*
done
next
case (*Propagated L m M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* = *this*(*4*)
have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*› **for** *n*
by (*auto simp*: *count-decided-0-iff*)
have *dec′*: ‹*L′*∈*set M* ⟹ *is-decided L′* ⟹ *cs* ! (*get-level M* (*lit-of L′*) − *1*) < *length M* ∧
rev M ! (*cs* ! (*get-level M* (*lit-of L′*) − *1*)) = *L′*› **for** *L′*
using *dec*[*of L′*] *n-d length*
count-decided-ge-get-level[*of M* ‹*lit-of L′*›]
apply (*cases L′*)
apply (*auto simp*: *get-level-cons-if atm-of-eq-atm-of undefined-notin*
split: *if-splits*)
apply (*auto simp*: *nth-append split*: *if-splits*)
done
show *?case*
apply (*rule control-stack.cons-prop*)
apply (*rule IH*)
subgoal using *n-d* **by** *auto*
subgoal using *dec′* **by** *auto*
subgoal using *length* **by** *auto*
done
qed

lemma *control-stack-alt-def*: ‹*no-dup M* ⟹ *control-stack′ cs M* ⟷ *control-stack cs M*›
using *control-stack-alt-def-imp*[*of M cs*] *control-stack-rev-get-lev*[*of cs M*]
control-stack-length-count-dec[*of cs M*] *control-stack-le-length-M*[*of cs M*]
unfolding *control-stack′-def* **apply** −
apply (*rule iffI*)
subgoal by *blast*
subgoal
using *count-decided-ge-get-level*[*of M*]
by (*metis One-nat-def Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff*
less-imp-diff-less neq0-conv nth-mem)
done

lemma *control-stack-decomp*:
assumes
decomp: ‹(*Decided L* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
cs: ‹*control-stack cs M*› **and**
n-d: ‹*no-dup M*›
shows ‹*control-stack* (*take* (*count-decided M1*) *cs*) *M1*›
proof −
obtain *M3* **where** *M*: ‹*M* = *M3* @ *M2* @ *Decided L* # *M1*›
using *decomp* **by** *auto*

define *M2'* **where** ‹*M2'* = *M3* @ *M2*›
have *M*: ‹*M* = *M2'* @ *Decided L* # *M1*›
unfolding *M M2'-def* **by** *auto*
have *n-d1*: ‹*no-dup M1*›
using *n-d no-dup-appendD* **unfolding** *M* **by** *auto*
have ‹*control-stack' cs M*›
using *cs*
apply (*subst* (*asm*) *control-stack-alt-def*[*symmetric*])
apply (*rule n-d*)
apply *assumption*
done
then have
cs-M: ‹*length cs* = *count-decided M*› **and**
L: ‹$\bigwedge$*L*. *L*∈*set M* ⟹ *is-decided L* ⟹
cs ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧ *rev M* ! (*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
unfolding *control-stack'-def* **by** *auto*
have *H*: ‹*L'* ∈ *set M1* ⟹ *undefined-lit M2'* (*lit-of L'*) ∧ *atm-of* (*lit-of L'*) ≠ *atm-of L*› **for** *L'*
using *n-d* **unfolding** *M*
by (*metis atm-of-eq-atm-of defined-lit-no-dupD*(*1*) *defined-lit-uminus lit-of.simps*(*1*)
no-dup-appendD no-dup-append-cons no-dup-cons undefined-notin)
have ‹*distinct M*›
using *no-dup-imp-distinct*[*OF n-d*] **.**
then have *K*: ‹*L'* ∈ *set M1* ⟹ *x* < *length M* ⟹ *rev M* ! *x* = *L'* ⟹ *x* < *length M1*› **for** *x L'*
unfolding *M* **apply** (*auto simp*: *nth-append nth-Cons split*: *if-splits nat.splits*)
by (*metis length-rev less-diff-conv local.H not-less-eq nth-mem set-rev undefined-notin*)
have *I*: ‹*L* ∈ *set M1* ⟹ *is-decided L* ⟹ *get-level M1* (*lit-of L*) > *0*› **for** *L*
using *n-d* **unfolding** *M* **by** (*auto dest*!: *split-list*)
have *cs'*: ‹*control-stack'* (*take* (*count-decided M1*) *cs*) *M1*›
unfolding *control-stack'-def*
apply (*intro conjI ballI impI*)
subgoal using *cs-M* **unfolding** *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ‹*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)›]
count-decided-ge-get-level[*of* ‹*M1*› ‹*lit-of L*›] *I*[*of L*]
unfolding *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ‹*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)›]
count-decided-ge-get-level[*of* ‹*M1*› ‹*lit-of L*›] *I*[*of L*]
unfolding *M* **by** *auto*
done
show *?thesis*
apply (*subst control-stack-alt-def*[*symmetric*])
apply (*rule n-d1*)
apply (*rule cs'*)
done
qed

4.4 Encoding of the reasons

definition *DECISION-REASON* :: *nat* **where**
‹*DECISION-REASON* = *1*›

definition *ann-lits-split-reasons* **where**
‹*ann-lits-split-reasons* $\mathcal{A}$ = {((*M*, *reasons*), *M'*). *M* = *map lit-of* (*rev M'*) ∧
(∀ *L* ∈ *set M'*. *is-proped L* ⟶
reasons ! (*atm-of* (*lit-of L*)) = *mark-of L* ∧ *mark-of L* ≠ *DECISION-REASON*) ∧
(∀ *L* ∈ *set M'*. *is-decided L* ⟶ *reasons* ! (*atm-of* (*lit-of L*)) = *DECISION-REASON*) ∧

$(\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ reasons)$
}›

definition *trail-pol* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*› **where**
‹*trail-pol* $\mathcal{A}$ =
$\{((M', xs, lvls, reasons, k, cs), M).\ ((M', reasons), M) \in ann\text{-}lits\text{-}split\text{-}reasons\ \mathcal{A} \wedge$
$no\text{-}dup\ M \wedge$
$(\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}.\ nat\text{-}of\text{-}lit\ L < length\ xs \wedge xs\ !\ (nat\text{-}of\text{-}lit\ L) = polarity\ M\ L) \wedge$
$(\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ lvls \wedge lvls\ !\ (atm\text{-}of\ L) = get\text{-}level\ M\ L) \wedge$
$k = count\text{-}decided\ M \wedge$
$(\forall L{\in}set\ M.\ lit\text{-}of\ L \in\# \mathcal{L}_{all}\ \mathcal{A}) \wedge$
$control\text{-}stack\ cs\ M \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A}\}$›

4.5 Definition of the full trail

lemma *trail-pol-alt-def*:
‹$trail\text{-}pol\ \mathcal{A} = \{((M', xs, lvls, reasons, k, cs), M).$
$((M', reasons), M) \in ann\text{-}lits\text{-}split\text{-}reasons\ \mathcal{A} \wedge$
$no\text{-}dup\ M \wedge$
$(\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}.\ nat\text{-}of\text{-}lit\ L < length\ xs \wedge xs\ !\ (nat\text{-}of\text{-}lit\ L) = polarity\ M\ L) \wedge$
$(\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}.\ atm\text{-}of\ L < length\ lvls \wedge lvls\ !\ (atm\text{-}of\ L) = get\text{-}level\ M\ L) \wedge$
$k = count\text{-}decided\ M \wedge$
$(\forall L{\in}set\ M.\ lit\text{-}of\ L \in\# \mathcal{L}_{all}\ \mathcal{A}) \wedge$
$control\text{-}stack\ cs\ M \wedge literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}trail\ \mathcal{A}\ M \wedge$
$length\ M < uint32\text{-}max \wedge$
$length\ M \leq uint32\text{-}max\ div\ 2 + 1 \wedge$
$count\text{-}decided\ M < uint32\text{-}max \wedge$
$length\ M' = length\ M \wedge$
$M' = map\ lit\text{-}of\ (rev\ M) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A}$
}›
proof −
have [*intro*!]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint32-max-def count-decided-def*
simp del: *length-filter-le*
dest: *length-trail-uint32-max-div2*)
show *?thesis*
unfolding *trail-pol-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint32-max-def ann-lits-split-reasons-def*
dest: *length-trail-uint32-max-div2*
simp del: *isasat-input-bounded-def*)
qed

4.6 Code generation

4.6.1 Conversion between incomplete and complete mode

definition *trail-fast-of-slow* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-fast-of-slow* = *id*›

definition *trail-pol-slow-of-fast* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-slow-of-fast* =
$(\lambda(M, val, lvls, reason, k, cs).\ (M, val, lvls, reason, k, cs))$›

definition *trail-slow-of-fast* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-slow-of-fast* = *id*›

definition *trail-pol-fast-of-slow* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-fast-of-slow* =
(λ(*M*, *val*, *lvls*, *reason*, *k*, *cs*). (*M*, *val*, *lvls*, *reason*, *k*, *cs*))›

lemma *trail-pol-slow-of-fast-alt-def*:
‹*trail-pol-slow-of-fast M* = *M*›
by (*cases M*)
(*auto simp*: *trail-pol-slow-of-fast-def*)

lemma *trail-pol-fast-of-slow-trail-fast-of-slow*:
‹(*RETURN o trail-pol-fast-of-slow*, *RETURN o trail-fast-of-slow*)
$\in [\lambda M.\ (\forall C\ L.\ Propagated\ L\ C \in set\ M \longrightarrow C < uint64\text{-}max)]_f$
$trail\text{-}pol\ \mathcal{A} \rightarrow \langle trail\text{-}pol\ \mathcal{A}\rangle\ nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def*
trail-fast-of-slow-def)

lemma *trail-pol-slow-of-fast-trail-slow-of-fast*:
‹(*RETURN o trail-pol-slow-of-fast*, *RETURN o trail-slow-of-fast*)
$\in trail\text{-}pol\ \mathcal{A} \rightarrow_f \langle trail\text{-}pol\ \mathcal{A}\rangle\ nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def*
trail-fast-of-slow-def trail-slow-of-fast-def
trail-pol-slow-of-fast-def)

lemma *trail-pol-same-length*[*simp*]: ‹$(M', M) \in trail\text{-}pol\ \mathcal{A} \Longrightarrow length\ (fst\ M') = length\ M$›
by (*auto simp*: *trail-pol-alt-def*)

definition *counts-maximum-level* **where**
‹$counts\text{-}maximum\text{-}level\ M\ C = \{i.\ C \neq None \longrightarrow i = card\text{-}max\text{-}lvl\ M\ (the\ C)\}$›

lemma *counts-maximum-level-None*[*simp*]: ‹*counts-maximum-level M None* = *Collect* (λ-. *True*)›
by (*auto simp*: *counts-maximum-level-def*)

4.6.2 Level of a literal

definition *get-level-atm-pol-pre* **where**
‹$get\text{-}level\text{-}atm\text{-}pol\text{-}pre = (\lambda((M, xs, lvls, k), L).\ L < length\ lvls)$›

definition *get-level-atm-pol* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat*› **where**
‹*get-level-atm-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *lvls* ! *L*)›

lemma *get-level-atm-pol-pre*:
assumes
‹$Pos\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$› **and**
‹$(M', M) \in trail\text{-}pol\ \mathcal{A}$›
shows ‹$get\text{-}level\text{-}atm\text{-}pol\text{-}pre\ (M', L)$›
using *assms*
by (*auto 5 5 simp*: *trail-pol-def nat-lit-rel-def*
br-def get-level-atm-pol-pre-def intro!: *ext*)

lemma (**in** −) *get-level-get-level-atm*: ‹*get-level M L* = *get-level-atm M* (*atm-of L*)›

unfolding *get-level-atm-def*
by (*cases L*) (*auto simp*: *get-level-Neg-Pos*)

definition *get-level-pol* **where**
‹*get-level-pol M L* = *get-level-atm-pol M* (*atm-of L*)›

definition *get-level-pol-pre* **where**
‹*get-level-pol-pre* = (λ((*M*, *xs*, *lvls*, *k*), *L*). *atm-of L* < *length lvls*)›

lemma *get-level-pol-pre*:
assumes
‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and**
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$›
shows ‹*get-level-pol-pre* (M', *L*)›
using *assms*
by (*auto 5 5 simp*: *trail-pol-def nat-lit-rel-def*
br-def get-level-pol-pre-def intro!: *ext*)

lemma *get-level-get-level-pol*:
assumes
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and** ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$›
shows ‹*get-level M L* = *get-level-pol* M' *L*›
using *assms*
by (*auto simp*: *get-level-pol-def get-level-atm-pol-def trail-pol-def*)

4.6.3 Current level

definition (**in** −) *count-decided-pol* **where**
‹*count-decided-pol* = (λ(-, -, -, -, *k*, -). *k*)›

lemma *count-decided-trail-ref*:
‹(*RETURN o count-decided-pol*, *RETURN o count-decided*) $\in$ *trail-pol* $\mathcal{A} \rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *trail-pol-def count-decided-pol-def*)

4.6.4 Polarity

definition (**in** −) *polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option*› **where**
‹*polarity-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *do* {
xs ! (*nat-of-lit L*)
})›

definition *polarity-pol-pre* **where**
‹*polarity-pol-pre* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *nat-of-lit L* < *length xs*)›

lemma *polarity-pol-polarity*:
‹(*uncurry* (*RETURN oo polarity-pol*), *uncurry* (*RETURN oo polarity*)) $\in$
[λ(*M*, *L*). $L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_f$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
by (*intro nres-relI frefI*)
(*auto simp*: *trail-pol-def polarity-def polarity-pol-def*
dest!: *multi-member-split*)

lemma *polarity-pol-pre*:
‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$ *polarity-pol-pre* M' *L*›
by (*auto simp*: *trail-pol-def polarity-def polarity-pol-def polarity-pol-pre-def*
dest!: *multi-member-split*)

4.6.5 Length of the trail

definition (**in** −) *isa-length-trail-pre* **where**
‹$isa\text{-}length\text{-}trail\text{-}pre = (\lambda\ (M', xs, lvls, reasons, k, cs).\ length\ M' \le uint32\text{-}max)$›

definition (**in** −) *isa-length-trail* **where**
‹$isa\text{-}length\text{-}trail = (\lambda\ (M', xs, lvls, reasons, k, cs).\ length\text{-}uint32\text{-}nat\ M')$›

lemma *isa-length-trail-pre*:
‹$(M, M') \in trail\text{-}pol\ \mathcal{A} \Longrightarrow isa\text{-}length\text{-}trail\text{-}pre\ M$›
by (*auto simp*: *isa-length-trail-def trail-pol-alt-def isa-length-trail-pre-def*)

lemma *isa-length-trail-length-u*:
‹$(RETURN\ o\ isa\text{-}length\text{-}trail, RETURN\ o\ length\text{-}uint32\text{-}nat) \in trail\text{-}pol\ \mathcal{A} \rightarrow_f \langle nat\text{-}rel\rangle nres\text{-}rel$›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-length-trail-def trail-pol-alt-def*
intro!: *ASSERT-leI*)

4.6.6 Consing elements

definition *cons-trail-Propagated-tr-pre* **where**
‹$cons\text{-}trail\text{-}Propagated\text{-}tr\text{-}pre = (\lambda((L, C), (M, xs, lvls, reasons, k)).\ nat\text{-}of\text{-}lit\ L < length\ xs \land$
$nat\text{-}of\text{-}lit\ (-L) < length\ xs \land atm\text{-}of\ L < length\ lvls \land atm\text{-}of\ L < length\ reasons \land length\ M < uint32\text{-}max)$›

definition *cons-trail-Propagated-tr* :: ‹$nat\ literal \Rightarrow nat \Rightarrow trail\text{-}pol \Rightarrow trail\text{-}pol\ nres$› **where**
‹$cons\text{-}trail\text{-}Propagated\text{-}tr = (\lambda L\ C\ (M', xs, lvls, reasons, k, cs).\ do\ \{$
$ASSERT(cons\text{-}trail\text{-}Propagated\text{-}tr\text{-}pre\ ((L, C), (M', xs, lvls, reasons, k, cs)));$
$RETURN\ (M' @ [L], let\ xs = xs[nat\text{-}of\text{-}lit\ L := SET\text{-}TRUE]\ in\ xs[nat\text{-}of\text{-}lit\ (-L) := SET\text{-}FALSE],$
$lvls[atm\text{-}of\ L := k], reasons[atm\text{-}of\ L := C], k, cs)\})$›

lemma *in-list-pos-neg-notD*: ‹$Pos\ (atm\text{-}of\ (lit\text{-}of\ La)) \notin lits\text{-}of\text{-}l\ bc \Longrightarrow$
$Neg\ (atm\text{-}of\ (lit\text{-}of\ La)) \notin lits\text{-}of\text{-}l\ bc \Longrightarrow$
$La \in set\ bc \Longrightarrow False$›
by (*metis Neg-atm-of-iff Pos-atm-of-iff lits-of-def rev-image-eqI*)

lemma *cons-trail-Propagated-tr-pre*:
assumes ‹$(M', M) \in trail\text{-}pol\ \mathcal{A}$› **and**
‹$undefined\text{-}lit\ M\ L$› **and**
‹$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$› **and**
‹$C \ne DECISION\text{-}REASON$›
shows ‹$cons\text{-}trail\text{-}Propagated\text{-}tr\text{-}pre\ ((L, C), M')$›
using *assms*
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
cons-trail-Propagated-tr-pre-def
intro!: *ext*)

lemma *cons-trail-Propagated-tr*:
‹$(uncurry2\ (cons\text{-}trail\text{-}Propagated\text{-}tr), uncurry2\ (cons\text{-}trail\text{-}propagate\text{-}l)) \in$
$[\lambda((L, C), M).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \land C \ne DECISION\text{-}REASON]_f$
$Id \times_f nat\text{-}rel \times_f trail\text{-}pol\ \mathcal{A} \rightarrow \langle trail\text{-}pol\ \mathcal{A}\rangle nres\text{-}rel$›
unfolding *cons-trail-Propagated-tr-def cons-trail-propagate-l-def*
apply (*intro frefI nres-relI*)
subgoal for *x y*

using *cons-trail-Propagated-tr-pre*[*of* ‹*snd* (*x*)› ‹*snd* (*y*)› $\mathcal{A}$ ‹*fst* (*fst y*)› ‹*snd* (*fst y*)›]
unfolding *uncurry-def*
apply *refine-vcg*
subgoal by *auto*
subgoal
by (*cases* ‹*fst* (*fst y*)›)
(*auto simp add*: *trail-pol-def polarity-def uminus-lit-swap*
cons-trail-Propagated-tr-def Decided-Propagated-in-iff-in-lits-of-l nth-list-update′
ann-lits-split-reasons-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
uminus-$\mathcal{A}_{in}$*-iff atm-of-eq-atm-of*
intro!: *ASSERT-refine-right*
dest!: *in-list-pos-neg-notD dest*: *pos-lit-in-atms-of neg-lit-in-atms-of dest*!: *multi-member-split*
simp del: *nat-of-lit.simps*)
done
done

lemma *cons-trail-Propagated-tr2*:
‹$(((L, C), M), ((L', C'), M')) \in Id \times_f Id \times_f$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$
$C \neq$ *DECISION-REASON* $\Longrightarrow$
cons-trail-Propagated-tr $L\ C\ M$
$\leq \Downarrow (\{(M'', M''').\ (M'', M''') \in$ *trail-pol* $\mathcal{A} \wedge M''' =$ *Propagated* $L\ C\ \#\ M' \wedge$ *no-dup* $M'''\})$
(*cons-trail-propagate-l* $L'\ C'\ M'$)›
using *cons-trail-Propagated-tr*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}\ L'\ C'\ M'\ L\ C\ M$]
unfolding *cons-trail-Propagated-tr-def cons-trail-propagate-l-def*
using *cons-trail-Propagated-tr-pre*[*of* $M\ M'\ \mathcal{A}\ L\ C$]
unfolding *uncurry-def*
apply *refine-vcg*
subgoal by *auto*
subgoal
by (*auto simp*: *trail-pol-def*)
done

lemma *undefined-lit-count-decided-uint32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹$\forall L \in$*set* M. *lit-of* $L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** *n-d*: ‹*no-dup* M› **and**
‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹*undefined-lit* $M\ L$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*Suc* (*count-decided* M) $\leq$ *uint32-max*›
proof –
have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of* x). $x \in\#$ *mset* M#}›
using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
have *incl*: ‹*atm-of* ‘# *lit-of* ‘# *mset* (*Decided* L # M) $\subseteq\#$ *remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}\ \mathcal{A}$)›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *assms dist-atm-M*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*
atm-of-eq-atm-of)
from *size-mset-mono*[*OF this*] **have** *1*: ‹*count-decided* $M + 1 \leq$ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$
$\mathcal{A}$))›
using *length-filter-le*[*of is-decided M*] **unfolding** *uint32-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}\ \mathcal{A}$)))›
by (*auto simp*: *inj-on-def*)
have *H*: ‹$xa \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$ *atm-of xa* $\leq$ *uint32-max div 2*› **for** *xa*
using *bounded*
by (*cases xa*) (*auto simp*: *uint32-max-def*)

have ‹*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$) ⊆# *mset* [*0*..< *1* + (*uint32-max div 2*)]›
  apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
  using *H distinct-image-mset-inj*[*OF inj-on*]
  by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
    dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
  by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}$))) ≤ *1* + *uint32-max div 2*›
  by *auto*

show *?thesis*
  using *1 2* **by** (*auto simp*: *uint32-max-def*)

from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* + *1* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
  unfolding *uint32-max-def count-decided-def*
  by (*auto simp del*: *length-filter-le*)
with *2* **have** ‹*length M* ≤ *uint32-max*›
  by *auto*
qed

lemma *length-trail-uint32-max*:
  assumes
    M-$\mathcal{L}_{all}$: ‹∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**
    bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
  shows ‹*length M* ≤ *uint32-max*›
proof −
  have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
    using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
  have *incl*: ‹*atm-of* ‘# *lit-of* ‘# *mset M* ⊆# *remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$)›
    apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
    using *assms dist-atm-M*
    by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*
      atm-of-eq-atm-of)

  have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)))›
    by (*auto simp*: *inj-on-def*)
  have *H*: ‹*xa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of xa* ≤ *uint32-max div 2*› **for** *xa*
    using *bounded*
    by (*cases xa*) (*auto simp*: *uint32-max-def*)
  have ‹*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$) ⊆# *mset* [*0*..< *1* + (*uint32-max div 2*)]›
    apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
    using *H distinct-image-mset-inj*[*OF inj-on*]
    by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
      dest: *le-neq-implies-less*)+
  note - = *size-mset-mono*[*OF this*]
  moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
    by *simp*
  ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$)) ≤ *1* + *uint32-max div 2*›
    by *auto*
  from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
    unfolding *uint32-max-def count-decided-def*
    by (*auto simp del*: *length-filter-le*)
  with *2* **show** *?thesis*
    by (*auto simp*: *uint32-max-def*)
qed

definition *last-trail-pol-pre* **where**
⟨*last-trail-pol-pre* = (λ(*M*, *xs*, *lvls*, *reasons*, *k*). *atm-of* (*last M*) < *length reasons* ∧ *M* ≠ [])⟩

definition (**in** −) *last-trail-pol* :: ⟨*trail-pol* ⇒ (*nat literal* × *nat option*)⟩ **where**
⟨*last-trail-pol* = (λ(*M*, *xs*, *lvls*, *reasons*, *k*).
    let r = *reasons* ! (*atm-of* (*last M*)) *in*
    (*last M*, *if r* = *DECISION-REASON then None else Some r*))⟩

definition *tl-trailt-tr* :: ⟨*trail-pol* ⇒ *trail-pol*⟩ **where**
⟨*tl-trailt-tr* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*).
  let L = *last M' in*
  (*butlast M'*,
  let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* (−*L*) := *None*],
  lvls[*atm-of L* := *0*],
  reasons, *if reasons* ! *atm-of L* = *DECISION-REASON then k*−*1 else k*,
    if reasons ! *atm-of L* = *DECISION-REASON then butlast cs else cs*))⟩

definition *tl-trailt-tr-pre* **where**
⟨*tl-trailt-tr-pre* = (λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). *M* ≠ [] ∧ *nat-of-lit*(*last M*) < *length xs* ∧
    nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
    atm-of (*last M*) < *length reason* ∧
    (*reason* ! *atm-of* (*last M*) = *DECISION-REASON* ⟶ *k* ≥ *1* ∧ *cs* ≠ []))⟩

lemma *ann-lits-split-reasons-map-lit-of*:
⟨((*M*, *reasons*), *M'*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ⟹ *M* = *map lit-of* (*rev M'*)⟩
by (*auto simp*: *ann-lits-split-reasons-def*)

lemma *control-stack-dec-butlast*:
⟨*control-stack b* (*Decided x1* # *M's*) ⟹ *control-stack* (*butlast b*) *M's*⟩
by (*cases b rule*: *rev-cases*) (*auto dest*: *control-stackE*)

lemma *tl-trail-tr*:
⟨((*RETURN o tl-trailt-tr*), (*RETURN o tl*)) ∈
  [λ*M*. *M* ≠ []]$_f$ *trail-pol* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*⟩
proof −
 show *?thesis*
  apply (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ⟨*y*⟩)
  subgoal by *fast*
  subgoal for *M M' L M's*
   unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def*
   apply *clarify*
   apply (*intro conjI*; *clarify?*; (*intro conjI*)*?*)
   subgoal
    by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
      ann-lits-split-reasons-def Let-def)
   subgoal by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*)
   subgoal by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*)
   subgoal
    by (*cases* ⟨*lit-of L*⟩)
     (*auto simp*: *polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l*
      uminus-lit-swap Let-def
      dest: *ann-lits-split-reasons-map-lit-of*)
   subgoal

by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*
atm-of-eq-atm-of get-level-cons-if)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
dest: *no-dup-consistentD*)
subgoal
by (*auto simp*: *tl-trailt-tr-def*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
done
done
qed

lemma *tl-trailt-tr-pre*:
assumes ‹$M \neq []$›
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$›
shows ‹*tl-trailt-tr-pre* M'›
proof −
have [*simp*]: ‹$x \neq [] \Longrightarrow$ *is-decided* (*last* x) $\Longrightarrow$ *Suc 0* $\leq$ *count-decided* x› **for** x
by (*cases x rule*: *rev-cases*) *auto*
show *?thesis*
using *assms*
by (*cases M*; *cases* ‹*hd M*›)
(*auto simp*: *trail-pol-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map tl-trailt-tr-pre-def simp del*: *rev-map*
intro!: *ext*)
qed

definition *tl-trail-propedt-tr* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trail-propedt-tr* = (λ(M', *xs*, *lvls*, *reasons*, *k*, *cs*).
let L = *last* M' *in*
(*butlast* M',
let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* ($-L$) := *None*],
lvls[*atm-of L* := *0*],
reasons, *k*, *cs*))›

definition *tl-trail-propedt-tr-pre* **where**
‹*tl-trail-propedt-tr-pre* =
(λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). $M \neq []$ ∧ *nat-of-lit*(*last M*) < *length xs* ∧
nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
atm-of (*last M*) < *length reason*)›

lemma *tl-trail-propedt-tr-pre*:
assumes ‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and**
‹$M \neq []$›
shows ‹*tl-trail-propedt-tr-pre* M'›
using *assms*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def*

tl-trail-propedt-tr-def tl-trail-propedt-tr-pre-def
apply *clarify*
apply (*cases M*; *intro conjI*; *clarify?*; (*intro conjI*)*?*)
subgoal
by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
ann-lits-split-reasons-def Let-def)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
dest: *no-dup-consistentD*)
subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
done

definition (**in** −) *lit-of-hd-trail* **where**
‹*lit-of-hd-trail M* = *lit-of* (*hd M*)›

definition (**in** −) *lit-of-last-trail-pol* **where**
‹*lit-of-last-trail-pol* = (λ(*M*, -). *last M*)›

lemma *lit-of-last-trail-pol-lit-of-last-trail*:
‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) ∈
[λ*S*. *S* ≠ []]$_f$ *trail-pol* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
by (*auto simp*: *lit-of-hd-trail-def trail-pol-def lit-of-last-trail-pol-def*
ann-lits-split-reasons-def hd-map rev-map[*symmetric*] *last-rev*
intro!: *frefI nres-relI*)

4.6.7 Setting a new literal

definition *cons-trail-Decided* :: ‹*nat literal* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*cons-trail-Decided L M′* = *Decided L* # *M′*›

definition *cons-trail-Decided-tr* :: ‹*nat literal* ⇒ *trail-pol* ⇒ *trail-pol*› **where**
‹*cons-trail-Decided-tr* = (λ*L* (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*). *do*{
let n = *length M′ in*
(*M′* @ [*L*], *let xs* = *xs*[*nat-of-lit L* := *SET-TRUE*] *in xs*[*nat-of-lit* (−*L*) := *SET-FALSE*],
lvls[*atm-of L* := *k+1*], *reasons*[*atm-of L* := *DECISION-REASON*], *k+1*, *cs* @ [*n*])})›

definition *cons-trail-Decided-tr-pre* **where**
‹*cons-trail-Decided-tr-pre* =
(λ(*L*, (*M*, *xs*, *lvls*, *reason*, *k*, *cs*)). *nat-of-lit L* < *length xs* ∧ *nat-of-lit* (−*L*) < *length xs* ∧
atm-of L < *length lvls* ∧ *atm-of L* < *length reason* ∧ *length cs* < *uint32-max* ∧
Suc k ≤ *uint32-max* ∧ *length M* < *uint32-max*)›

lemma *length-cons-trail-Decided*[*simp*]:
⟨*length* (*cons-trail-Decided L M*) = *Suc* (*length M*)⟩
by (*auto simp*: *cons-trail-Decided-def*)

lemma *cons-trail-Decided-tr*:
⟨(*uncurry* (*RETURN oo cons-trail-Decided-tr*), *uncurry* (*RETURN oo cons-trail-Decided*)) ∈
$[\lambda(L, M).$ *undefined-lit M L* $\wedge L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *Id* $\times_f$ *trail-pol* $\mathcal{A} \rightarrow \langle$*trail-pol* $\mathcal{A}\rangle$*nres-rel*⟩
by (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ⟨*fst x*⟩)
(*auto simp*: *trail-pol-def polarity-def cons-trail-Decided-def uminus-lit-swap*
Decided-Propagated-in-iff-in-lits-of-l
cons-trail-Decided-tr-def nth-list-update′ ann-lits-split-reasons-def
dest!: *in-list-pos-neg-notD multi-member-split*
intro: *control-stack.cons-dec*
simp del: *nat-of-lit.simps*)

lemma *cons-trail-Decided-tr-pre*:
assumes ⟨(M', M) ∈ *trail-pol* $\mathcal{A}$⟩ **and**
⟨$L \in\# \mathcal{L}_{all}\ \mathcal{A}$⟩ **and** ⟨*undefined-lit M L*⟩
shows ⟨*cons-trail-Decided-tr-pre* (L, M')⟩
using *assms*
by (*auto simp*: *trail-pol-alt-def image-image ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
cons-trail-Decided-tr-pre-def control-stack-length-count-dec
intro!: *ext undefined-lit-count-decided-uint32-max length-trail-uint32-max*)

4.6.8 Polarity: Defined or Undefined

definition (**in** −) *defined-atm-pol-pre* **where**
⟨*defined-atm-pol-pre* = (λ(M, *xs*, *lvls*, k) L. $2{*}L <$ *length xs* $\wedge$
$2{*}L \leq$ *uint32-max*)⟩

definition (**in** −) *defined-atm-pol* **where**
⟨*defined-atm-pol* = (λ(M, *xs*, *lvls*, k) L. $\neg$((*xs*!($2{*}L$)) = *None*))⟩

lemma *undefined-atm-code*:
⟨(*uncurry* (*RETURN oo defined-atm-pol*), *uncurry* (*RETURN oo defined-atm*)) ∈
$[\lambda(M, L).$ *Pos* $L \in\# \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨*bool-rel*⟩ *nres-rel*⟩ (**is** *?A*) **and**
defined-atm-pol-pre:
⟨(M', M) ∈ *trail-pol* $\mathcal{A} \Longrightarrow L \in\#\ \ \mathcal{A} \Longrightarrow$ *defined-atm-pol-pre* M' L⟩
proof −
have H: ⟨$2{*}L <$ *length xs*⟩ (**is** ⟨*?length*⟩) **and**
none: ⟨*defined-atm M L* ⟷ *xs* ! ($2{*}L$) ≠ *None*⟩ (**is** *?undef*) **and**
le: ⟨$2{*}L \leq$ *uint32-max*⟩ (**is** *?le*)
if *L-N*: ⟨*Pos* $L \in\# \mathcal{L}_{all}\ \mathcal{A}$⟩ **and** *tr*: ⟨(($M'$, *xs*, *lvls*, k), M) ∈ *trail-pol* $\mathcal{A}$⟩
for *M xs lvls k M′ L*
proof −
have
⟨M' = *map lit-of* (*rev M*)⟩ **and**
⟨$\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! *nat-of-lit L* = *polarity M L*⟩
using *tr* **unfolding** *trail-pol-def ann-lits-split-reasons-def* **by** *fast+*
then have L: ⟨*nat-of-lit* (*Pos L*) < *length xs*⟩ **and**
xsL: ⟨*xs* ! (*nat-of-lit* (*Pos L*)) = *polarity M* (*Pos L*)⟩
using *L-N* **by** (*auto dest*!: *multi-member-split*)
show *?length*
using L **by** *simp*
show *?undef*

using *xsL* **by** (*auto simp*: *polarity-def defined-atm-def*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits*)
show ‹$2*L \leq$ *uint32-max*›
using *tr L-N* **unfolding** *trail-pol-def* **by** *auto*
qed
show *?A*
unfolding *defined-atm-pol-def*
by (*intro frefI nres-relI*) (*auto 5 5 simp*: *none H le intro*!: *ASSERT-leI*)
show ‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\# \mathcal{A} \Longrightarrow$ *defined-atm-pol-pre* M' L›
using *H le* **by** (*auto simp*: *defined-atm-pol-pre-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
qed

4.6.9 Reasons

definition *get-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-propagation-reason-pol* = (λ(-, -, -, *reasons*, -) *L*. *do* {
ASSERT(*atm-of L* < *length reasons*);
let r = *reasons* ! *atm-of L*;
RETURN (*if r* = *DECISION-REASON then None else Some r*)})›

lemma *get-propagation-reason-pol*:
‹(*uncurry get-propagation-reason-pol*, *uncurry get-propagation-reason*) ∈
$[\lambda(M, L).\ L \in$ *lits-of-l* $M]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *lits-of-def*
apply *clarify*
apply (*rename-tac a aa ab ac b ba ad bb x*, *case-tac x*)
by (*auto simp*: *get-propagation-reason-def get-propagation-reason-pol-def*
trail-pol-def ann-lits-split-reasons-def lits-of-def assert-bind-spec-conv)

definition *get-propagation-reason-raw-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*get-propagation-reason-raw-pol* = (λ(-, -, -, *reasons*, -) *L*. *do* {
ASSERT(*atm-of L* < *length reasons*);
let r = *reasons* ! *atm-of L*;
RETURN r})›

The version *get-propagation-reason* can return the reason, but does not have to: it can be more suitable for specification (like for the conflict minimisation, where finding the reason is not mandatory).

The following version *always* returns the reasons if there is one. Remark that both functions are linked to the same code (but *get-propagation-reason* can be called first with some additional filtering later).

definition (**in** −) *get-the-propagation-reason*
:: ‹(*'v*, *'mark*) *ann-lits* ⇒ *'v literal* ⇒ *'mark option nres*›
where
‹*get-the-propagation-reason M L* = *SPEC*(λ*C*.
(*C* ≠ *None* ⟷ *Propagated L* (*the C*) ∈ *set M*) ∧
(*C* = *None* ⟷ *Decided L* ∈ *set M* ∨ *L* ∉ *lits-of-l M*))›

lemma *no-dup-Decided-PropedD*:
‹*no-dup ad* ⟹ *Decided L* ∈ *set ad* ⟹ *Propagated L C* ∈ *set ad* ⟹ *False*›
by (*metis annotated-lit.distinct*(*1*) *in-set-conv-decomp lit-of.simps*(*1*) *lit-of.simps*(*2*)
no-dup-appendD no-dup-cons undefined-notin xy-in-set-cases)

definition *get-the-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-the-propagation-reason-pol* = (λ(-, *xs*, -, *reasons*, -) *L*. *do* {
  ASSERT(*atm-of L* < *length reasons*);
  ASSERT(*nat-of-lit L* < *length xs*);
  let r = *reasons* ! *atm-of L*;
  RETURN (*if xs* ! *nat-of-lit L* = *SET-TRUE* ∧ *r* ≠ *DECISION-REASON then Some r else None*)})›

lemma *get-the-propagation-reason-pol*:
‹(*uncurry get-the-propagation-reason-pol*, *uncurry get-the-propagation-reason*) ∈
  $[\lambda(M,\ L).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
proof −
 have [*dest*]: ‹*no-dup bb* ⟹
  SET-TRUE = *polarity bb* (*Pos x1*) ⟹ *Pos x1* ∈ *lits-of-l bb* ∧ *Neg x1* ∉ *lits-of-l bb*› **for** *bb x1*
  by (*auto simp*: *polarity-def split*: *if-splits dest*: *no-dup-consistentD*)
 show *?thesis*
  apply (*intro frefI nres-relI*)
  unfolding *lits-of-def get-the-propagation-reason-def uncurry-def get-the-propagation-reason-pol-def*
  apply *clarify*
  apply (*refine-vcg*)
  subgoal
   by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
    trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv
    dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
  subgoal
   by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
    trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv
    dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
  subgoal for *a aa ab ac ad b ba ae bb*
   apply (*cases* ‹*aa* ! *nat-of-lit ba* ≠ *SET-TRUE*›)
   apply (*subgoal-tac* ‹*ba* ∉ *lits-of-l ae*›)
   prefer *2*
   subgoal
    by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
     trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec′(*2*)
     dest: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
   subgoal
    by (*auto simp*: *lits-of-def dest*: *imageI*[*of* - - *lit-of*])

   apply (*subgoal-tac* ‹*ba* ∈ *lits-of-l ae*›)
   prefer *2*
   subgoal
    by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
     trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec′(*2*)
     dest: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
   subgoal
    apply (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def*
    trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv lits-of-def
    dest!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
    apply (*case-tac x*; *auto*)
    apply (*case-tac x*; *auto*)
    done
   done
  done
qed

4.7 Direct access to elements in the trail

definition (**in** −) *rev-trail-nth* **where**
‹*rev-trail-nth M i* = *lit-of* (*rev M* ! *i*)›

definition (**in** −) *isa-trail-nth* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isa-trail-nth* = (λ(*M*, -) *i*. *do* {
  ASSERT(*i* < *length M*);
  RETURN (*M* ! *i*)
})›

lemma *isa-trail-nth-rev-trail-nth*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) ∈
  $[\lambda(M, i).\ i < length\ M]_f$ *trail-pol* $\mathcal{A} \times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
  (*auto simp*: *isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def*
  intro!: *ASSERT-leI*)

We here define a variant of the trail representation, where the the control stack is out of sync of the trail (i.e., there are some leftovers at the end). This might make backtracking a little faster.

definition *trail-pol-no-CS* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*›
where
‹*trail-pol-no-CS* $\mathcal{A}$ =
 {((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*), *M*). ((*M'*, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
  no-dup M ∧
  (∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
  (∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
  (∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
  isasat-input-bounded $\mathcal{A}$ ∧
  control-stack (*take* (*count-decided M*) *cs*) *M*
}›

definition *tl-trailt-tr-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trailt-tr-no-CS* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*).
  let L = *last M'* *in*
  (*butlast M'*,
  let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* (−*L*) := *None*],
  lvls[*atm-of L* := *0*],
  reasons, *k*, *cs*))›

definition *tl-trailt-tr-no-CS-pre* **where**
‹*tl-trailt-tr-no-CS-pre* = (λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). *M* ≠ [] ∧ *nat-of-lit*(*last M*) < *length xs* ∧
    nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
    atm-of (*last M*) < *length reason*)›

lemma *control-stack-take-Suc-count-dec-unstack*:
‹*control-stack* (*take* (*Suc* (*count-decided M's*)) *cs*) (*Decided x1* # *M's*) ⟹
  control-stack (*take* (*count-decided M's*) *cs*) *M's*›
using *control-stack-length-count-dec*[*of* ‹*take* (*Suc* (*count-decided M's*)) *cs*› ‹*Decided x1* # *M's*›]
by (*auto simp*: *min-def take-Suc-conv-app-nth split*: *if-splits elim*: *control-stackE*)

lemma *tl-trailt-tr-no-CS-pre*:
assumes ‹(*M'*, *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$› **and** ‹*M* ≠ []›
shows ‹*tl-trailt-tr-no-CS-pre M'*›
proof −
have [*simp*]: ‹*x* ≠ [] ⟹ *is-decided* (*last x*) ⟹ *Suc 0* ≤ *count-decided x*› **for** *x*

by (*cases x rule*: *rev-cases*) *auto*
show *?thesis*
using *assms*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def*
tl-trailt-tr-no-CS-def tl-trailt-tr-no-CS-pre-def
by (*cases M*; *cases* ‹*hd M*›)
(*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map simp del*: *rev-map*
intro!: *ext*)
qed

lemma *tl-trail-tr-no-CS*:
‹((*RETURN o tl-trailt-tr-no-CS*), (*RETURN o tl*)) ∈
[$\lambda M.\ M \neq []]_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ‹*y*›)
subgoal by *fast*
subgoal for *M M' L M's*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def*
tl-trailt-tr-no-CS-def
apply *clarify*
apply (*intro conjI*; *clarify?*; (*intro conjI*)*?*)
subgoal
by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
ann-lits-split-reasons-def Let-def)
subgoal by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*)
subgoal by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*)
subgoal
by (*cases* ‹*lit-of L*›)
(*auto simp*: *polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l*
uminus-lit-swap Let-def
dest: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*
atm-of-eq-atm-of get-level-cons-if)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast control-stack-take-Suc-count-dec-unstack
dest: *no-dup-consistentD ann-lits-split-reasons-map-lit-of*)
done
done

definition *trail-conv-to-no-CS* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-to-no-CS M* = *M*›

definition *trail-pol-conv-to-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-conv-to-no-CS M* = *M*›

lemma *id-trail-conv-to-no-CS*:
‹(*RETURN o trail-pol-conv-to-no-CS*, *RETURN o trail-conv-to-no-CS*) $\in$ *trail-pol* $\mathcal{A} \rightarrow_f$ ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def control-stack-length-count-dec trail-pol-conv-to-no-CS-def intro*: *ext*)

definition *trail-conv-back* :: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-back j M* = *M*›

definition (**in** −) *trail-conv-back-imp* :: ‹*nat* ⇒ *trail-pol* ⇒ *trail-pol nres*› **where**
‹*trail-conv-back-imp j* = (λ(*M*, *xs*, *lvls*, *reason*, -, *cs*). *do* {
ASSERT($j \leq$ *length cs*); *RETURN* (*M*, *xs*, *lvls*, *reason*, *j*, *take* (*j*) *cs*)})›

lemma *trail-conv-back*:
‹(*uncurry trail-conv-back-imp*, *uncurry* (*RETURN oo trail-conv-back*))
$\in$ [λ(*k*, *M*). *k* = *count-decided M*]$_f$ *nat-rel* $\times_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def control-stack-length-count-dec trail-conv-back-def trail-conv-back-imp-def intro*: *ext intro*!: *ASSERT-refine-left dest*: *control-stack-length-count-dec multi-member-split*)

definition (**in** −)*take-arl* **where**
‹*take-arl* = (λ*i* (*xs*, *j*). (*xs*, *i*))›

lemma *isa-trail-nth-rev-trail-nth-no-CS*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) $\in$
[λ(*M*, *i*). *i* < *length M*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ $\times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def trail-pol-no-CS-def*
intro!: *ASSERT-leI*)

lemma *trail-pol-no-CS-alt-def*:
‹*trail-pol-no-CS* $\mathcal{A}$ =
{((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*), *M*). ((*M'*, *reasons*), *M*) $\in$ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack (*take* (*count-decided M*) *cs*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M* ∧
length M < *uint32-max* ∧
length M ≤ *uint32-max div 2* + *1* ∧
count-decided M < *uint32-max* ∧
length M' = *length M* ∧
isasat-input-bounded $\mathcal{A}$ ∧
M' = *map lit-of* (*rev M*)
}›
proof −
have [*intro*!]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint32-max-def count-decided-def*

simp del: *length-filter-le*
dest: *length-trail-uint32-max-div2*)
show *?thesis*
unfolding *trail-pol-no-CS-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def uint32-max-def ann-lits-split-reasons-def*
dest: *length-trail-uint32-max-div2*
simp del: *isasat-input-bounded-def*)
qed

lemma *isa-length-trail-length-u-no-CS*:
‹(*RETURN o isa-length-trail*, *RETURN o length-uint32-nat*) ∈ *trail-pol-no-CS* $\mathcal{A} \to_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-length-trail-def trail-pol-no-CS-alt-def ann-lits-split-reasons-def*
intro!: *ASSERT-leI*)

lemma *control-stack-is-decided*:
‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *is-decided* ((*rev M*)!*c*)›
by (*induction arbitrary*: *c rule*: *control-stack.induct*) (*auto simp*: *nth-append*
dest: *control-stack-le-length-M*)

lemma *control-stack-distinct*:
‹*control-stack cs M* ⟹ *distinct cs*›
by (*induction rule*: *control-stack.induct*) (*auto simp*: *nth-append*
dest: *control-stack-le-length-M*)

lemma *control-stack-level-control-stack*:
assumes
cs: ‹*control-stack cs M*› **and**
n-d: ‹*no-dup M*› **and**
i: ‹*i* < *length cs*›
shows ‹*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) = *Suc i*›
proof −
define *L* **where** ‹*L* = *rev M* ! (*cs* ! *i*)›
have *csi*: ‹*cs* ! *i* < *length M*›
using *cs i* **by** (*auto intro*: *control-stack-le-length-M*)
then have *L-M*: ‹*L* ∈ *set M*›
using *nth-mem*[*of* ‹*cs* !*i*› ‹*rev M*›] **unfolding** *L-def* **by** (*auto simp del*: *nth-mem*)
have *dec-L*: ‹*is-decided L*›
using *control-stack-is-decided*[*OF cs*] *i* **unfolding** *L-def* **by** *auto*
then have ‹*rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
using *control-stack-rev-get-lev*[*OF cs n-d L-M*] **by** *auto*
moreover have ‹*distinct M*›
using *no-dup-distinct*[*OF n-d*] **unfolding** *mset-map*[*symmetric*] *distinct-mset-mset-distinct*
by (*rule distinct-mapI*)

moreover have *lev0*: ‹*get-level M* (*lit-of L*) ≥ *1*›
using *split-list*[*OF L-M*] *n-d dec-L* **by** (*auto simp*: *get-level-append-if*)
moreover have ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*) < *length M*›
using *control-stack-le-length-M*[*OF cs*,
of ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*)›, *OF nth-mem*]
control-stack-length-count-dec[*OF cs*] *count-decided-ge-get-level*[*of M*
‹*lit-of* (*rev M* ! (*cs* ! *i*))›] *lev0*
by (*auto simp*: *L-def*)

ultimately have ‹cs ! ($get\text{-}level\ M$ ($lit\text{-}of\ L$) − 1) = cs ! i›
using *nth-eq-iff-index-eq*[*of* ‹*rev M*›] *csi* **unfolding** *L-def* **by** *auto*
then have ‹i = $get\text{-}level\ M$ ($lit\text{-}of\ L$) − 1›
using *nth-eq-iff-index-eq*[*OF control-stack-distinct*[*OF cs*], *of i* ‹*get-level M* (*lit-of L*) − *1*›]
i lev0 count-decided-ge-get-level[*of M* ‹*lit-of* (*rev M* ! (*cs* ! *i*))›]
control-stack-length-count-dec[*OF cs*]
by (*auto simp*: *L-def*)
then show *?thesis* **using** *lev0* **unfolding** *L-def*[*symmetric*] **by** *auto*
qed

definition *get-pos-of-level-in-trail* **where**
‹*get-pos-of-level-in-trail* M_0 *lev* =
$SPEC(\lambda i.\ i < length\ M_0 \wedge is\text{-}decided\ (rev\ M_0!i) \wedge get\text{-}level\ M_0\ (lit\text{-}of\ (rev\ M_0!i)) = lev{+}1)$›

definition (**in** −) *get-pos-of-level-in-trail-imp* **where**
‹*get-pos-of-level-in-trail-imp* = (λ(M', *xs*, *lvls*, *reasons*, *k*, *cs*) *lev*. *do* {
ASSERT(*lev* < *length cs*);
RETURN (*cs* ! *lev*)
})›
definition *get-pos-of-level-in-trail-pre* **where**
‹*get-pos-of-level-in-trail-pre* = (λ(*M*, *lev*). *lev* < *count-decided M*)›

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
$[get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}pre]_f$ *trail-pol-no-CS* $\mathcal{A} \times_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-no-CS-alt-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-no-CS-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
$[get\text{-}pos\text{-}of\text{-}level\text{-}in\text{-}trail\text{-}pre]_f$ *trail-pol* $\mathcal{A} \times_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *lit-of-last-trail-pol-lit-of-last-trail-no-CS*:

‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) ∈
    $[\lambda S.\ S \neq []]_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
by (*auto simp*: *lit-of-hd-trail-def trail-pol-no-CS-def lit-of-last-trail-pol-def*
  ann-lits-split-reasons-def hd-map rev-map[*symmetric*] *last-rev*
  intro!: *frefI nres-relI*)

end
theory *Watched-Literals-VMTF*
  imports *IsaSAT-Literals*
begin

4.7.1 Variable-Move-to-Front

Variants around head and last

definition *option-hd* :: ‹*'a list* ⇒ *'a option*› **where**
  ‹*option-hd xs* = (*if xs* = [] *then None else Some* (*hd xs*))›

lemma *option-hd-None-iff*[*iff*]: ‹*option-hd zs* = *None* ⟷ *zs* = []› ‹*None* = *option-hd zs* ⟷ *zs* = []›
  by (*auto simp*: *option-hd-def*)

lemma *option-hd-Some-iff*[*iff*]: ‹*option-hd zs* = *Some y* ⟷ (*zs* ≠ [] ∧ *y* = *hd zs*)›
  ‹*Some y* = *option-hd zs* ⟷ (*zs* ≠ [] ∧ *y* = *hd zs*)›
  by (*auto simp*: *option-hd-def*)

lemma *option-hd-Some-hd*[*simp*]: ‹*zs* ≠ [] ⟹ *option-hd zs* = *Some* (*hd zs*)›
  by (*auto simp*: *option-hd-def*)

lemma *option-hd-Nil*[*simp*]: ‹*option-hd* [] = *None*›
  by (*auto simp*: *option-hd-def*)

definition *option-last* **where**
  ‹*option-last l* = (*if l* = [] *then None else Some* (*last l*))›

lemma
  option-last-None-iff[*iff*]: ‹*option-last l* = *None* ⟷ *l* = []› ‹*None* = *option-last l* ⟷ *l* = []› **and**
  option-last-Some-iff[*iff*]:
    ‹*option-last l* = *Some a* ⟷ *l* ≠ [] ∧ *a* = *last l*›
    ‹*Some a* = *option-last l* ⟷ *l* ≠ [] ∧ *a* = *last l*›
  by (*auto simp*: *option-last-def*)

lemma *option-last-Some*[*simp*]: ‹*l* ≠ [] ⟹ *option-last l* = *Some* (*last l*)›
  by (*auto simp*: *option-last-def*)

lemma *option-last-Nil*[*simp*]: ‹*option-last* [] = *None*›
  by (*auto simp*: *option-last-def*)

lemma *option-last-remove1-not-last*:
  ‹*x* ≠ *last xs* ⟹ *option-last xs* = *option-last* (*remove1 x xs*)›
  by (*cases xs rule*: *rev-cases*)
    (*auto simp*: *option-last-def remove1-Nil-iff remove1-append*)

lemma *option-hd-rev*: ‹*option-hd* (*rev xs*) = *option-last xs*›
  by (*cases xs rule*: *rev-cases*) *auto*

lemma *map-option-option-last*:

‹map-option f (option-last xs) = option-last (map f xs)›
  by (*cases xs rule*: *rev-cases*) *auto*

Specification

type-synonym *'v abs-vmtf-ns* = ‹*'v set* × *'v set*›
type-synonym *'v abs-vmtf-ns-remove* = ‹*'v abs-vmtf-ns* × *'v set*›

datatype (*'v*, *'n*) *vmtf-node* = *VMTF-Node* (*stamp* : *'n*) (*get-prev*: ‹*'v option*›) (*get-next*: ‹*'v option*›)
type-synonym *nat-vmtf-node* = ‹(*nat*, *nat*) *vmtf-node*›

inductive *vmtf-ns* :: ‹*nat list* ⇒ *nat* ⇒ *nat-vmtf-node list* ⇒ *bool*› **where**
Nil: ‹*vmtf-ns* [] *st xs*› |
Cons1: ‹$a < length\ xs \Longrightarrow m \geq n \Longrightarrow xs\ !\ a = VMTF\text{-}Node\ (n{::}nat)\ None\ None \Longrightarrow vmtf\text{-}ns\ [a]\ m\ xs$›
|
Cons: ‹$vmtf\text{-}ns\ (b\ \#\ l)\ m\ xs \Longrightarrow a < length\ xs \Longrightarrow xs\ !\ a = VMTF\text{-}Node\ n\ None\ (Some\ b) \Longrightarrow$
  $a \neq b \Longrightarrow a \notin set\ l \Longrightarrow n > m \Longrightarrow$
  $xs' = xs[b := VMTF\text{-}Node\ (stamp\ (xs!b))\ (Some\ a)\ (get\text{-}next\ (xs!b))] \Longrightarrow n' \geq n \Longrightarrow$
  $vmtf\text{-}ns\ (a\ \#\ b\ \#\ l)\ n'\ xs'$›

inductive-cases *vmtf-nsE*: ‹*vmtf-ns xs st zs*›

lemma *vmtf-ns-le-length*: ‹$vmtf\text{-}ns\ l\ m\ xs \Longrightarrow i \in set\ l \Longrightarrow i < length\ xs$›
  apply (*induction rule*: *vmtf-ns.induct*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  done

lemma *vmtf-ns-distinct*: ‹$vmtf\text{-}ns\ l\ m\ xs \Longrightarrow distinct\ l$›
  apply (*induction rule*: *vmtf-ns.induct*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  subgoal by (*auto intro*: *vmtf-ns.intros*)
  done

lemma *vmtf-ns-eq-iff*:
  assumes
    ‹$\forall i \in set\ l.\ xs\ !\ i = zs\ !\ i$› **and**
    ‹$\forall i \in set\ l.\ i < length\ xs \wedge i < length\ zs$›
  shows ‹$vmtf\text{-}ns\ l\ m\ zs \longleftrightarrow vmtf\text{-}ns\ l\ m\ xs$› (**is** ‹$?A \longleftrightarrow ?B$›)
proof −
  have ‹*vmtf-ns l m xs*›
    if
      ‹*vmtf-ns l m zs*› **and**
      ‹$(\forall i \in set\ l.\ xs\ !\ i = zs\ !\ i)$› **and**
      ‹$(\forall i \in set\ l.\ i < length\ xs \wedge i < length\ zs)$›
    for *xs zs*
    using *that*
  proof (*induction arbitrary*: *xs rule*: *vmtf-ns.induct*)
    case (*Nil st xs zs*)
    then show *?case* **by** (*auto intro*: *vmtf-ns.intros*)
  next
    case (*Cons1 a xs n zs*)
    show *?case* **by** (*rule vmtf-ns.Cons1*) (*use Cons1* **in** ‹*auto intro*: *vmtf-ns.intros*›)
  next

case (*Cons b l m xs c n zs n′ zs′*) **note** *vmtf-ns* = *this*(*1*) **and** *a-le-y* = *this*(*2*) **and** *zs-a* = *this*(*3*)
and *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs′* = *this*(*7*) **and** *nn′* = *this*(*8*) **and**
IH = *this*(*9*) **and** *H* = *this*(*10*−)
have ‹*vmtf-ns* (*c* # *b* # *l*) *n′ zs*›
by (*rule vmtf-ns.Cons*[*OF Cons.hyps*])
have [*simp*]: ‹*b* < *length xs*› ‹*b* < *length zs*›
using *H xs′* **by** *auto*
have [*simp*]: ‹*b* ∉ *set l*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*
then have *K*: ‹∀ *i*∈*set l*. *zs* ! *i* = (*if b* = *i then x else xs* ! *i*) =
(∀ *i*∈*set l*. *zs* ! *i* = *xs* ! *i*)› **for** *x*
using *H*(*2*)
by (*simp add*: *H*(*1*) *xs′*)
have *next-xs-b*: ‹*get-next* (*xs* ! *b*) = *None*› **if** ‹*l* = []›
using *vmtf-ns* **unfolding** *that* **by** (*auto simp*: *elim*!: *vmtf-nsE*)
have *prev-xs-b*: ‹*get-prev* (*xs* ! *b*) = *None*›
using *vmtf-ns* **by** (*auto elim*: *vmtf-nsE*)
have *vmtf-ns-zs*: ‹*vmtf-ns* (*b* # *l*) *m* (*zs′*[*b* := *xs*!*b*])›
apply (*rule IH*)
subgoal using *H*(*1*) *ab next-xs-b prev-xs-b H* **unfolding** *xs′* **by** (*auto simp*: *K*)
subgoal using *H*(*2*) *ab next-xs-b prev-xs-b* **unfolding** *xs′* **by** (*auto simp*: *K*)
done
have ‹*zs′* ! *b* = *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some c*) (*get-next* (*xs* ! *b*))›
using *H*(*1*) **unfolding** *xs′* **by** *auto*
show *?case*
apply (*rule vmtf-ns.Cons*[*OF vmtf-ns-zs*, *of* - *n*])
subgoal using *a-le-y xs′ H*(*2*) **by** *auto*
subgoal using *ab zs-a xs′ H*(*1*) **by** (*auto simp*: *K*)
subgoal using *ab* **.**
subgoal using *a-l* **.**
subgoal using *mn* **.**
subgoal using *ab xs′ H*(*1*) **by** (*metis H*(*2*) *insert-iff list.set*(*2*) *list-update-id*
list-update-overwrite nth-list-update-eq)
subgoal using *nn′* **.**
done
qed
then show *?thesis*
using *assms* **by** *metis*
qed

lemmas *vmtf-ns-eq-iffI* = *vmtf-ns-eq-iff*[*THEN iffD1*]

lemma *vmtf-ns-stamp-increase*: ‹*vmtf-ns xs p zs* ⟹ *p* ≤ *p′* ⟹ *vmtf-ns xs p′ zs*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*rule vmtf-ns.Cons1*) (*auto intro*!: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

lemma *vmtf-ns-single-iff*: ‹*vmtf-ns* [*a*] *m xs* ⟷ (*a* < *length xs* ∧ *m* ≥ *stamp* (*xs* ! *a*) ∧
xs ! *a* = *VMTF-Node* (*stamp* (*xs* ! *a*)) *None None*)›
by (*auto 5 5 elim*!: *vmtf-nsE intro*: *vmtf-ns.intros*)

lemma *vmtf-ns-append-decomp*:
assumes ‹*vmtf-ns* (*axs* @ [*ax*, *ay*] @ *azs*) *an ns*›
shows ‹(*vmtf-ns* (*axs* @ [*ax*]) *an* (*ns*[*ax*:= *VMTF-Node* (*stamp* (*ns*!*ax*)) (*get-prev* (*ns*!*ax*)) *None*]) ∧

```
    vmtf-ns (ay # azs) (stamp (ns!ay)) (ns[ay:= VMTF-Node (stamp (ns!ay)) None (get-next (ns!ay))])
∧
    stamp (ns!ax) > stamp (ns!ay))›
  using assms
proof (induction ‹axs @ [ax, ay] @ azs› an ns arbitrary: axs ax ay azs rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by simp
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10−)
  have b-le-xs: ‹b < length xs›
    using vmtf-ns by (auto intro: vmtf-ns-le-length simp: xs')
  show ?case
  proof (cases ‹axs›)
    case [simp]: Nil
    then have [simp]: ‹ax = a› ‹ay = b› ‹azs = l›
      using decomp by auto
    show ?thesis
    proof (cases l)
      case Nil
      then show ?thesis
        using vmtf-ns xs' a-le-y zs-a ab a-l mn nn' by (cases ‹xs ! b›)
          (auto simp: vmtf-ns-single-iff)
    next
      case (Cons al als) note l = this
       have vmtf-ns-b: ‹vmtf-ns [b] m (xs[b := VMTF-Node (stamp (xs ! b)) (get-prev (xs ! b)) None])›
and
        vmtf-ns-l: ‹vmtf-ns (al # als) (stamp (xs ! al))
           (xs[al := VMTF-Node (stamp (xs ! al)) None (get-next (xs ! al))])› and
        stamp-al-b: ‹stamp (xs ! al) < stamp (xs ! b)›
        using IH[of Nil b al als] unfolding l by auto
      have ‹vmtf-ns [a] n' (xs'[a := VMTF-Node (stamp (xs' ! a)) (get-prev (xs' ! a)) None])›
          using a-le-y xs' ab mn nn' zs-a by (auto simp: vmtf-ns-single-iff)
      have al-b[simp]: ‹al ≠ b› and b-als: ‹b ∉ set als›
        using vmtf-ns unfolding l by (auto dest: vmtf-ns-distinct)
      have al-le-xs: ‹al < length xs›
        using vmtf-ns vmtf-ns-l by (auto intro: vmtf-ns-le-length simp: l xs')
      have xs-al: ‹xs ! al = VMTF-Node (stamp (xs ! al)) (Some b) (get-next (xs ! al))›
        using vmtf-ns unfolding l by (auto 5 5 elim: vmtf-nsE)
      have xs-b: ‹xs ! b = VMTF-Node (stamp (xs ! b)) None (get-next (xs ! b))›
        using vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›) (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)

      have ‹vmtf-ns (b # al # als) (stamp (xs' ! b))
          (xs'[b := VMTF-Node (stamp (xs' ! b)) None (get-next (xs' ! b))])›
        apply (rule vmtf-ns.Cons[OF vmtf-ns-l, of - ‹stamp (xs' ! b)›])
        subgoal using b-le-xs by auto
        subgoal using xs-b vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›)
            (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)
        subgoal using al-b by blast
        subgoal using b-als .
        subgoal using xs' b-le-xs stamp-al-b by (simp add:)
        subgoal using ab unfolding xs' by (simp add: b-le-xs al-le-xs xs-al[symmetric]
```

```
          xs-b[symmetric])
      subgoal by simp
      done
    moreover have ‹vmtf-ns [a] n′
        (xs′[a := VMTF-Node (stamp (xs′ ! a)) (get-prev (xs′ ! a)) None])›
      using ab a-le-y mn nn′ zs-a by (auto simp: vmtf-ns-single-iff xs′)
    moreover have ‹stamp (xs′ ! b) < stamp (xs′ ! a)›
      using b-le-xs ab mn vmtf-ns-b zs-a by (auto simp add: xs′ vmtf-ns-single-iff)
    ultimately show ?thesis
      unfolding l by (simp add: l)
  qed
next
  case (Cons aaxs axs′) note axs = this
  have [simp]: ‹aaxs = a› and bl: ‹b # l = axs′ @ [ax, ay] @ azs›
    using decomp unfolding axs by simp-all
  have
    vmtf-ns-axs′: ‹vmtf-ns (axs′ @ [ax]) m
      (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) None])› and
    vmtf-ns-ay: ‹vmtf-ns (ay # azs) (stamp (xs ! ay))
      (xs[ay := VMTF-Node (stamp (xs ! ay)) None (get-next (xs ! ay))])› and
    stamp: ‹stamp (xs ! ay) < stamp (xs ! ax)›
    using IH[OF bl] by fast+
  have b-ay: ‹b ≠ ay›
    using bl vmtf-ns-distinct[OF vmtf-ns] by (cases axs′) auto
  have vmtf-ns-ay′: ‹vmtf-ns (ay # azs) (stamp (xs′ ! ay))
      (xs[ay := VMTF-Node (stamp (xs ! ay)) None (get-next (xs ! ay))])›
    using vmtf-ns-ay xs′ b-ay by (auto)
  have [simp]: ‹ay < length xs›
      using vmtf-ns by (auto intro: vmtf-ns-le-length simp: bl xs′)
  have in-azs-noteq-b: ‹i ∈ set azs ⟹ i ≠ b› for i
    using vmtf-ns-distinct[OF vmtf-ns] bl by (cases axs′) (auto simp: xs′ b-ay)
  have a-ax[simp]: ‹a ≠ ax›
    using ab a-l bl by (cases axs′) (auto simp: xs′ b-ay)
  have ‹vmtf-ns (axs @ [ax]) n′
     (xs′[ax := VMTF-Node (stamp (xs′ ! ax)) (get-prev (xs′ ! ax)) None])›
  proof (cases axs′)
    case Nil
    then have [simp]: ‹ax = b›
      using bl by auto
    have ‹vmtf-ns [ax] m (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) None])›
      using vmtf-ns-axs′ unfolding axs Nil by simp
    then have ‹vmtf-ns (aaxs # ax # []) n′
        (xs′[ax := VMTF-Node (stamp (xs′ ! ax)) (get-prev (xs′ ! ax)) None])›
      apply (rule vmtf-ns.Cons[of - - - - - n])
      subgoal using a-le-y by auto
      subgoal using zs-a a-le-y ab by auto
      subgoal using ab by auto
      subgoal by simp
      subgoal using mn .
      subgoal using zs-a a-le-y ab xs′ b-le-xs by auto
      subgoal using nn′ .
      done
    then show ?thesis
      using vmtf-ns-axs′ unfolding axs Nil by simp
  next
    case (Cons aaaxs′ axs′′)
```

have [*simp*]: ‹*aaaxs'* = *b*›
  using *bl* **unfolding** *Cons* **by** *auto*
have ‹*vmtf-ns* (*aaaxs'* # *axs''* @ [*ax*]) *m*
    (*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) *None*])›
  using *vmtf-ns-axs'* **unfolding** *axs Cons* **by** *simp*
then have ‹*vmtf-ns* (*a* # *aaaxs'* # *axs''* @ [*ax*]) *n'*
    (*xs'*[*ax* := *VMTF-Node* (*stamp* (*xs'* ! *ax*)) (*get-prev* (*xs'* ! *ax*)) *None*])›
  apply (*rule vmtf-ns.Cons*[*of* - - - - - *n*])
  subgoal using *a-le-y* **by** *auto*
  subgoal using *zs-a a-le-y a-ax ab* **by** (*auto simp del*: ‹*a* ≠ *ax*›)
  subgoal using *ab* **by** *auto*
  subgoal using *a-l bl* **unfolding** *Cons* **by** *simp*
  subgoal using *mn* **.**
  subgoal using *zs-a a-le-y ab xs' b-le-xs* **by** (*auto simp*: *list-update-swap*)
  subgoal using *nn'* **.**
  done
then show *?thesis*
  unfolding *axs Cons* **by** *simp*
qed
moreover have ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs'* ! *ay*))
    (*xs'*[*ay* := *VMTF-Node* (*stamp* (*xs'* ! *ay*)) *None* (*get-next* (*xs'* ! *ay*))])›
  apply (*rule vmtf-ns-eq-iffI*[*OF* - - *vmtf-ns-ay'*])
  subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *bl b-le-xs in-azs-noteq-b* **by** (*auto simp*: *xs' b-ay*)
  subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] *bl* **unfolding** *xs'* **by** *auto*
  done
moreover have ‹*stamp* (*xs'* ! *ay*) < *stamp* (*xs'* ! *ax*)›
  using *stamp* **unfolding** *axs xs'* **by** (*auto simp*: *b-le-xs b-ay*)
ultimately show *?thesis*
  unfolding *axs xs'* **by** *fast*
qed
qed

lemma *vmtf-ns-append-rebuild*:
  assumes
    ‹(*vmtf-ns* (*axs* @ [*ax*]) *an ns*) › **and**
    ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*ns*!*ay*)) *ns*› **and**
    ‹*stamp* (*ns*!*ax*) > *stamp* (*ns*!*ay*)› **and**
    ‹*distinct* (*axs* @ [*ax*, *ay*] @ *azs*)›
  shows ‹*vmtf-ns* (*axs* @ [*ax*, *ay*] @ *azs*) *an*
    (*ns*[*ax* := *VMTF-Node* (*stamp* (*ns*!*ax*)) (*get-prev* (*ns*!*ax*)) (*Some ay*) ,
      ay := *VMTF-Node* (*stamp* (*ns*!*ay*)) (*Some ax*) (*get-next* (*ns*!*ay*))])›
  using *assms*
proof (*induction* ‹*axs* @ [*ax*]› *an ns arbitrary*: *axs ax ay azs rule*: *vmtf-ns.induct*)
  case (*Nil st xs*)
  then show *?case* **by** *simp*
next
  case (*Cons1 a xs m n*) **note** *a-le-xs* = *this*(*1*) **and** *nm* = *this*(*2*) **and** *xs-a* = *this*(*3*) **and** *a* = *this*(*4*)
    and *vmtf-ns* = *this*(*5*) **and** *stamp* = *this*(*6*) **and** *dist* = *this*(*7*)
  have *a-ax*: ‹*ax* = *a*›
    using *a* **by** *simp*

  have *vmtf-ns-ay'*: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs* ! *ay*)) (*xs*[*ax* := *VMTF-Node n None* (*Some ay*)])›
    apply (*rule vmtf-ns-eq-iffI*[*OF* - - *vmtf-ns*])
    subgoal using *dist a-ax a-le-xs* **by** *auto*
    subgoal using *vmtf-ns vmtf-ns-le-length* **by** *auto*
    done

then have ‹*vmtf-ns* (*ax* # *ay* # *azs*) *m* (*xs*[*ax* := *VMTF-Node n None* (*Some ay*),
ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
apply (*rule vmtf-ns.Cons*[*of* - - - - - ‹*stamp* (*xs* ! *a*)›])
subgoal using *a-le-xs* **unfolding** *a-ax* **by** *auto*
subgoal using *xs-a a-ax a-le-xs* **by** *auto*
subgoal using *dist* **by** *auto*
subgoal using *dist* **by** *auto*
subgoal using *stamp* **by** (*simp add*: *a-ax*)
subgoal using *a-ax a-le-xs dist* **by** *auto*
subgoal by (*simp add*: *nm xs-a*)
done
then show *?case*
using *a-ax a xs-a* **by** *auto*
next
case (*Cons b l m xs a n xs′ n′*) **note** *vmtf-ns* = *this*(*1*) **and** *IH* = *this*(*2*) **and** *a-le-y* = *this*(*3*) **and** *zs-a* = *this*(*4*) **and** *ab* = *this*(*5*) **and** *a-l* = *this*(*6*) **and** *mn* = *this*(*7*) **and** *xs′* = *this*(*8*) **and** *nn′* = *this*(*9*) **and** *decomp* = *this*(*10*) **and** *vmtf-ns-ay* = *this*(*11*) **and** *stamp* = *this*(*12*) **and** *dist* = *this*(*13*)

have *dist-b*: ‹*distinct* ((*a* # *b* # *l*) @ *ay* # *azs*)›
using *dist* **unfolding** *decomp* **by** *auto*
then have *b-ay*: ‹*b* ≠ *ay*›
by *auto*
have *b-le-xs*: ‹*b* < *length xs*›
using *vmtf-ns vmtf-ns-le-length* **by** *auto*
have *a-ax*: ‹*a* ≠ *ax*› **and** *a-ay*: ‹*a* ≠ *ay*›
using *dist-b decomp dist* **by** (*cases axs*; *auto*)+
have *vmtf-ns-ay′*: ‹*vmtf-ns* (*ay* # *azs*) (*stamp* (*xs* ! *ay*)) *xs*›
apply (*rule vmtf-ns-eq-iffI*[*of* - - *xs′*])
subgoal using *xs′ b-ay dist-b b-le-xs* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-ay*] *xs′* **by** *auto*
subgoal using *xs′ b-ay dist-b b-le-xs vmtf-ns-ay xs′* **by** *auto*
done

have ‹*vmtf-ns* (*tl axs* @ [*ax*, *ay*] @ *azs*) *m*
(*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) (*Some ay*),
ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
apply (*rule IH*)
subgoal using *decomp* **by** (*cases axs*) *auto*
subgoal using *vmtf-ns-ay′* **.**
subgoal using *stamp xs′ b-ay b-le-xs* **by** (*cases* ‹*ax* = *b*›) *auto*
subgoal using *dist* **by** (*cases axs*) *auto*
done
moreover have ‹*tl axs* @ [*ax*, *ay*] @ *azs* = *b* # *l* @ *ay* # *azs*›
using *decomp* **by** (*cases axs*) *auto*
ultimately have *vmtf-ns-tl-axs*: ‹*vmtf-ns* (*b* # *l* @ *ay* # *azs*) *m*
(*xs*[*ax* := *VMTF-Node* (*stamp* (*xs* ! *ax*)) (*get-prev* (*xs* ! *ax*)) (*Some ay*),
ay := *VMTF-Node* (*stamp* (*xs* ! *ay*)) (*Some ax*) (*get-next* (*xs* ! *ay*))])›
by *metis*

then have ‹*vmtf-ns* (*a* # *b* # *l* @ *ay* # *azs*) *n′*
(*xs′*[*ax* := *VMTF-Node* (*stamp* (*xs′* ! *ax*)) (*get-prev* (*xs′* ! *ax*)) (*Some ay*),
ay := *VMTF-Node* (*stamp* (*xs′* ! *ay*)) (*Some ax*) (*get-next* (*xs′* ! *ay*))])›
apply (*rule vmtf-ns.Cons*[*of* - - - - - ‹*stamp* (*xs* ! *a*)›])
subgoal using *a-le-y* **by** *simp*

    subgoal using *zs-a a-le-y a-ax a-ay* **by** *auto*
    subgoal using *ab* **.**
    subgoal using *dist-b* **by** *auto*
    subgoal using *mn* **by** (*simp add*: *zs-a*)
    subgoal using *zs-a a-le-y a-ax a-ay b-ay b-le-xs* **unfolding** *xs'*
      by (*auto simp*: *list-update-swap*)
    subgoal using *stamp xs' nn' b-ay b-le-xs zs-a* **by** *auto*
    done
  then show *?case*
    by (*metis append.assoc append-Cons append-Nil decomp*)
qed

It is tempting to remove the *update-x*. However, it leads to more complicated reasoning later: What happens if x is not in the list, but its successor is? Moreover, it is unlikely to really make a big difference (performance-wise).

definition *ns-vmtf-dequeue* :: ‹*nat* ⇒ *nat-vmtf-node list* ⇒ *nat-vmtf-node list*› **where**
‹*ns-vmtf-dequeue y xs* =
  (*let x* = *xs* ! *y*;
   u-prev =
      (*case get-prev x of None* ⇒ *xs*
      | *Some a* ⇒ *xs*[*a*:= *VMTF-Node* (*stamp* (*xs*!*a*)) (*get-prev* (*xs*!*a*)) (*get-next x*)]);
   u-next =
      (*case get-next x of None* ⇒ *u-prev*
      | *Some a* ⇒ *u-prev*[*a*:= *VMTF-Node* (*stamp* (*u-prev*!*a*)) (*get-prev x*) (*get-next* (*u-prev*!*a*))]);
    u-x = *u-next*[*y*:= *VMTF-Node* (*stamp* (*u-next*!*y*)) *None None*]
    in
   u-x)
  ›

lemma *vmtf-ns-different-same-neq*: ‹*vmtf-ns* (*b* # *c* # *l'*) *m xs* ⟹ *vmtf-ns* (*c* # *l'*) *m xs* ⟹ *False*›
  apply (*cases l'*)
  subgoal by (*force elim*: *vmtf-nsE*)
  subgoal for *x xs*
    apply (*subst* (*asm*) *vmtf-ns.simps*)
    apply (*subst* (*asm*)(*2*) *vmtf-ns.simps*)
    by (*metis* (*no-types*, *lifting*) *vmtf-node.inject length-list-update list.discI list-tail-coinc*
        nth-list-update-eq nth-list-update-neq option.discI)
  done

lemma *vmtf-ns-last-next*:
  ‹*vmtf-ns* (*xs* @ [*x*]) *m ns* ⟹ *get-next* (*ns* ! *x*) = *None*›
  apply (*induction xs* @ [*x*] *m ns arbitrary*: *xs x rule*: *vmtf-ns.induct*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *b l m xs a n xs' n' xsa x*
    by (*cases* ‹*xs* ! *b*›; *cases* ‹*x* = *b*›; *cases xsa*)
       (*force simp*: *vmtf-ns-le-length*)+
  done

lemma *vmtf-ns-hd-prev*:
  ‹*vmtf-ns* (*x* # *xs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *None*›
  apply (*induction x* # *xs m ns arbitrary*: *xs x rule*: *vmtf-ns.induct*)
  subgoal by *auto*
  subgoal by *auto*
  done

```
lemma vmtf-ns-last-mid-get-next:
  ‹vmtf-ns (xs @ [x, y] @ zs) m ns ⟹ get-next (ns ! x) = Some y›
  apply (induction xs @ [x, y] @ zs m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  subgoal for b l m xs a n xs' n' xsa x
    by (cases ‹xs ! b›; cases ‹x = b›; cases xsa)
       (force simp: vmtf-ns-le-length)+
  done

lemma vmtf-ns-last-mid-get-next-option-hd:
  ‹vmtf-ns (xs @ x # zs) m ns ⟹ get-next (ns ! x) = option-hd zs›
  using vmtf-ns-last-mid-get-next[of xs x ‹hd zs› ‹tl zs› m ns]
  vmtf-ns-last-next[of xs x]
  by (cases zs) auto

lemma vmtf-ns-last-mid-get-prev:
  assumes ‹vmtf-ns (xs @ [x, y] @ zs) m ns›
  shows ‹get-prev (ns ! y) = Some x›
    using assms
proof (induction xs @ [x, y] @ zs m ns arbitrary: xs x rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by auto
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xxs a n xxs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10)
  show ?case
  proof (cases xs)
    case Nil
    then show ?thesis using Cons vmtf-ns-le-length by auto
  next
    case (Cons aaxs axs')
    then have b-l: ‹b # l = tl xs @ [x, y] @ zs›
      using decomp by auto
    then have ‹get-prev (xxs ! y) = Some x›
      by (rule IH)
    moreover have ‹x ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] b-l by auto
    moreover have ‹b ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] decomp by (cases axs') (auto simp add: Cons)
    moreover have ‹y < length xxs› ‹b < length xxs›
      using vmtf-ns-le-length[OF vmtf-ns, unfolded b-l] vmtf-ns-le-length[OF vmtf-ns] by auto
    ultimately show ?thesis
      unfolding xs' by auto
  qed
qed

lemma vmtf-ns-last-mid-get-prev-option-last:
  ‹vmtf-ns (xs @ x # zs) m ns ⟹ get-prev (ns ! x) = option-last xs›
  using vmtf-ns-last-mid-get-prev[of ‹butlast xs› ‹last xs› ‹x› ‹zs› m ns]
  by (cases xs rule: rev-cases) (auto elim: vmtf-nsE)
```

lemma *length-ns-vmtf-dequeue*[*simp*]: ‹*length* (*ns-vmtf-dequeue x ns*) = *length ns*›
  unfolding *ns-vmtf-dequeue-def* **by** (*auto simp*: *Let-def split*: *option.splits*)

lemma *vmtf-ns-skip-fst*:
  assumes *vmtf-ns*: ‹*vmtf-ns* (x # y' # zs') *m ns*›
  shows ‹$\exists$ *n*. *vmtf-ns* (y' # zs') *n* (*ns*[y' := *VMTF-Node* (*stamp* (*ns* ! y')) *None* (*get-next* (*ns* ! y'))]) $\wedge$
    $m \geq n$›
  using *assms*
proof (*rule vmtf-nsE*, *goal-cases*)
  case *1*
  then show *?case* **by** *simp*
next
  case (*2 a n*)
  then show *?case* **by** *simp*
next
  case (*3 b l m xs a n*)
  moreover have ‹*get-prev* (*xs* ! *b*) = *None*›
    using *3*(*3*) **by** (*fastforce elim*: *vmtf-nsE*)
  moreover have ‹$b <$ *length xs*›
    using *3*(*3*) *vmtf-ns-le-length* **by** *auto*
  ultimately show *?case*
    by (*cases* ‹*xs* ! *b*›) (*auto simp*: *eq-commute*[*of* ‹*xs* ! *b*›])
qed

definition *vmtf-ns-notin* **where**
  ‹*vmtf-ns-notin l m xs* $\longleftrightarrow$ ($\forall$ *i*<*length xs*. *i*$\notin$*set l* $\longrightarrow$ (*get-prev* (*xs* ! *i*) = *None* $\wedge$
    get-next (*xs* ! *i*) = *None*))›

lemma *vmtf-ns-notinI*:
  ‹($\bigwedge$*i*. *i* <*length xs* $\Longrightarrow$ *i*$\notin$*set l* $\Longrightarrow$ *get-prev* (*xs* ! *i*) = *None* $\wedge$
    get-next (*xs* ! *i*) = *None*) $\Longrightarrow$ *vmtf-ns-notin l m xs*›
  by (*auto simp*: *vmtf-ns-notin-def*)

lemma *stamp-ns-vmtf-dequeue*:
  ‹*axs* < *length zs* $\Longrightarrow$ *stamp* (*ns-vmtf-dequeue x zs* ! *axs*) = *stamp* (*zs* ! *axs*)›
  by (*cases* ‹*zs* ! (*the* (*get-next* (*zs* ! *x*)))›; *cases* ‹(*the* (*get-next* (*zs* ! *x*))) = *axs*›;
    cases ‹(*the* (*get-prev* (*zs* ! *x*))) = *axs*›; *cases* ‹*zs* ! *x*›)
   (*auto simp*: *nth-list-update'* *ns-vmtf-dequeue-def Let-def split*: *option.splits*)

lemma *sorted-many-eq-append*: ‹*sorted* (*xs* @ [*x*, *y*]) $\longleftrightarrow$ *sorted* (*xs* @ [*x*]) $\wedge$ $x \leq y$›
  using *sorted-append*[*of* ‹*xs* @ [*x*]› ‹[*y*]›] *sorted-append*[*of xs* ‹[*x*]›]
  by *force*

lemma *vmtf-ns-stamp-sorted*:
  assumes ‹*vmtf-ns l m ns*›
  shows ‹*sorted* (*map* (λ*a*. *stamp* (*ns*!*a*)) (*rev l*)) $\wedge$ ($\forall$ *a* $\in$ *set l*. *stamp* (*ns*!*a*) $\leq$ *m*)›
  using *assms*
proof (*induction rule*: *vmtf-ns.induct*)
  case (*Cons b l m xs a n xs' n'*) **note** *vmtf-ns* = *this*(*1*) **and** *IH* = *this*(*9*) **and** *a-le-y* = *this*(*2*) **and**
   zs-a = *this*(*3*) **and** *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and**
   nn' = *this*(*8*)
  have *H*:
  ‹*map* (λ*aa*. *stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *aa*)) (*rev l*) =
    map (λ*a*. *stamp* (*xs* ! *a*)) (*rev l*)›
    apply (*rule map-cong*)

  subgoal by *auto*
  subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] by *auto*
  done
 have [*simp*]: ‹*stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *b*) = *stamp* (*xs* ! *b*)›
  using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] by (*cases* ‹*xs* ! *b*›) *auto*
 have ‹*stamp* (*xs*[*b* := *VMTF-Node* (*stamp* (*xs* ! *b*)) (*Some a*) (*get-next* (*xs* ! *b*))] ! *aa*) ≤ *n′*›
  if ‹*aa* ∈ *set l*› for *aa*
  apply (*cases* ‹*aa* = *b*›)
  subgoal using *Cons* by *auto*
  subgoal using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] *IH nn′ mn that* by *auto*
  done
 then show *?case*
  using *Cons* by (*auto simp*: *H sorted-many-eq-append*)
qed *auto*

lemma *vmtf-ns-ns-vmtf-dequeue*:
 assumes *vmtf-ns*: ‹*vmtf-ns l m ns*› and *notin*: ‹*vmtf-ns-notin l m ns*› and *valid*: ‹*x* < *length ns*›
 shows ‹*vmtf-ns* (*remove1 x l*) *m* (*ns-vmtf-dequeue x ns*)›
proof (*cases* ‹*x* ∈ *set l*›)
 case *False*
 then have *H*: ‹*remove1 x l* = *l*›
  by (*simp add*: *remove1-idem*)
 have *simp-is-stupid*[*simp*]: ‹*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *a* ≠ *x*› ‹*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *x* ≠ *a*›
for *a x*
  by *auto*
 have
   ‹*get-prev* (*ns* ! *x*) = *None* › and
   ‹*get-next* (*ns* ! *x*) = *None*›
  using *notin False valid* unfolding *vmtf-ns-notin-def* by *auto*
 then have *vmtf-ns-eq*: ‹(*ns-vmtf-dequeue x ns*) ! *a* = *ns* ! *a*› if ‹*a* ∈ *set l*› for *a*
  using *that False valid* unfolding *vmtf-ns-notin-def ns-vmtf-dequeue-def*
  by (*cases* ‹*ns* ! (*the* (*get-prev* (*ns* ! *x*)))›; *cases* ‹*ns* ! (*the* (*get-next* (*ns* ! *x*)))›)
   (*auto simp*: *Let-def split*: *option.splits*)
 show *?thesis*
  unfolding *H*
  apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns*])
  subgoal using *vmtf-ns-eq* by *blast*
  subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] by *auto*
  done
next
 case *True*
 then obtain *xs zs* where
  l: ‹*l* = *xs* @ *x* # *zs*›
  by (*meson split-list*)
 have *r-l*: ‹*remove1 x l* = *xs* @ *zs*›
  using *vmtf-ns-distinct*[*OF vmtf-ns*] unfolding *l* by (*simp add*: *remove1-append*)
 have *dist*: ‹*distinct l*›
  using *vmtf-ns-distinct*[*OF vmtf-ns*] .
 have *le-length*: ‹*i* ∈ *set l* ⟹ *i* < *length ns*› for *i*
  using *vmtf-ns-le-length*[*OF vmtf-ns*] .
 consider
  (*xs-zs-empty*) ‹*xs* = []› and ‹*zs* = []› |
  (*xs-nempty-zs-empty*) *x′ xs′* where ‹*xs* = *xs′* @ [*x′*]› and ‹*zs* = []› |
  (*xs-empty-zs-nempty*) *y′ zs′* where ‹*xs* = []› and ‹*zs* = *y′* # *zs′*› |
  (*xs-zs-nempty*) *x′ y′ xs′ zs′* where ‹*xs* = *xs′* @ [*x′*]› and ‹*zs* = *y′* # *zs′*›

by (*cases xs rule*: *rev-cases*; *cases zs*)
then show *?thesis*
proof *cases*
case *xs-zs-empty*
then show *?thesis*
using *vmtf-ns* **by** (*auto simp*: *r-l intro*: *vmtf-ns.intros*)
next
case *xs-empty-zs-nempty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*2*)
have [*simp*]: ‹$x \neq y'$› ‹$y' \neq x$› ‹$x \notin$ *set zs'*›
using *dist* **unfolding** *l xs zs* **by** *auto*
have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *None*› ‹*get-next* (*ns* ! *x*) = *option-hd zs*›
using *vmtf-ns* **unfolding** *l xs zs*
by (*cases zs*; *auto 5 5 simp*: *option-hd-def elim*: *vmtf-nsE*; *fail*)+
then have *vmtf'*: ‹*vmtf-ns* (*y'* # *zs'*) *m* (*ns*[*y'* := *VMTF-Node* (*stamp* (*ns* ! *y'*)) *None* (*get-next* (*ns* ! *y'*))])›
using *vmtf-ns* **unfolding** *r-l* **unfolding** *l xs zs*
by (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update' zs*
split: *option.splits*
intro: *vmtf-ns.intros vmtf-ns-stamp-increase dest*: *vmtf-ns-skip-fst*)
show *?thesis*
apply (*rule vmtf-ns-eq-iffI*[*of* - -
‹(*ns*[*y'* := *VMTF-Node* (*stamp* (*ns* ! *y'*)) *None* (*get-next* (*ns* ! *y'*))])› *m*])
subgoal
using *prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
by (*cases* ‹*ns* ! *x*›) (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update'*)
subgoal
using *prev-next le-length* **unfolding** *r-l* **unfolding** *l xs zs*
by (*cases* ‹*ns* ! *x*›) *auto*
subgoal
using *vmtf'* **unfolding** *r-l* **unfolding** *l xs zs* **by** *auto*
done
next
case *xs-nempty-zs-empty* **note** *xs* = *this*(*1*) **and** *zs* = *this*(*2*)
have [*simp*]: ‹$x \neq x'$› ‹$x' \neq x$› ‹$x' \notin$ *set xs'*› ‹$x \notin$ *set xs'*›
using *dist* **unfolding** *l xs zs* **by** *auto*
have *prev-next*: ‹*get-prev* (*ns* ! *x*) = *Some x'*› ‹*get-next* (*ns* ! *x*) = *None*›
using *vmtf-ns vmtf-ns-append-decomp*[*of xs' x' x zs m ns*] **unfolding** *l xs zs*
by (*auto simp*: *vmtf-ns-single-iff intro*: *vmtf-ns-last-mid-get-prev*)
then have *vmtf'*: ‹*vmtf-ns* (*xs'* @ [*x'*]) *m* (*ns*[*x'* := *VMTF-Node* (*stamp* (*ns* ! *x'*)) (*get-prev* (*ns* ! *x'*)) *None*])›
using *vmtf-ns* **unfolding** *r-l* **unfolding** *l xs zs*
by (*auto simp*: *ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp split*: *option.splits*
intro: *vmtf-ns.intros*)
show *?thesis*
apply (*rule vmtf-ns-eq-iffI*[*of* - -
‹(*ns*[*x'* := *VMTF-Node* (*stamp* (*ns* ! *x'*)) (*get-prev* (*ns* ! *x'*)) *None*])› *m*])
subgoal
using *prev-next* **unfolding** *r-l* **unfolding** *l xs zs*
by (*cases* ‹*ns* ! *x'*›) (*auto simp*: *ns-vmtf-dequeue-def Let-def nth-list-update'*)
subgoal
using *prev-next le-length* **unfolding** *r-l* **unfolding** *l xs zs*
by (*cases* ‹*ns* ! *x*›) *auto*
subgoal
using *vmtf'* **unfolding** *r-l* **unfolding** *l xs zs* **by** *auto*
done
next

case *xs-zs-nempty* **note** $xs = this(1)$ **and** $zs = this(2)$
have *vmtf-ns-x′-x*: ‹*vmtf-ns* (xs' @ [x', x] @ (y' # zs')) m ns› **and**
vmtf-ns-x-y: ‹*vmtf-ns* ((xs' @ [x']) @ [x, y'] @ zs') m ns›
using *vmtf-ns* **unfolding** l xs zs **by** *simp-all*
from *vmtf-ns-append-decomp*[*OF vmtf-ns-x′-x*] **have**
vmtf-ns-xs: ‹*vmtf-ns* (xs' @ [x']) m (ns[x' := *VMTF-Node* (*stamp* (ns ! x')) (*get-prev* (ns ! x')) *None*])› **and**
vmtf-ns-zs: ‹*vmtf-ns* (x # y' # zs') (*stamp* (ns ! x)) (ns[x := *VMTF-Node* (*stamp* (ns ! x)) *None* (*get-next* (ns ! x))])› **and**
stamp: ‹*stamp* (ns ! x) < *stamp* (ns ! x')›
by *fast+*
have [*simp*]: ‹y' < *length* ns› ‹x < *length* ns› ‹$x \neq y'$› ‹$x' \neq y'$› ‹x' < *length* ns› ‹$y' \neq x'$›
‹$x' \neq x$› ‹$x \neq x'$› ‹$y' \neq x$›
and *x-zs′*: ‹$x \notin$ *set* zs'› ‹$x \notin$ *set* xs'› **and** *x′-zs′*: ‹$x' \notin$ *set* zs'› **and** *y′-xs′*: ‹$y' \notin$ *set* xs'›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** l xs zs
by *auto*
obtain n **where**
vmtf-ns-zs′: ‹*vmtf-ns* (y' # zs') n (ns[x := *VMTF-Node* (*stamp* (ns ! x)) *None* (*get-next* (ns ! x)),
y' := *VMTF-Node* (*stamp* (ns[x := *VMTF-Node* (*stamp* (ns ! x)) *None* (*get-next* (ns ! x))] ! y')) *None*
(*get-next* (ns[x := *VMTF-Node* (*stamp* (ns ! x)) *None* (*get-next* (ns ! x))] ! y'))])› **and**
‹$n \leq$ *stamp* (ns ! x)›
using *vmtf-ns-skip-fst*[*OF vmtf-ns-zs*] **by** *blast*
then have *vmtf-ns-y′-zs′-x-y′*: ‹*vmtf-ns* (y' # zs') n (ns[x := *VMTF-Node* (*stamp* (ns ! x)) *None* (*get-next* (ns ! x)),
y' := *VMTF-Node* (*stamp* (ns ! y')) *None* (*get-next* (ns ! y'))])›
by *auto*

define ns' **where** ‹ns' = ns[x' := *VMTF-Node* (*stamp* (ns ! x')) (*get-prev* (ns ! x')) *None*,
y' := *VMTF-Node* (*stamp* (ns ! y')) *None* (*get-next* (ns ! y'))]›
have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (y' # zs') n (ns[y' := *VMTF-Node* (*stamp* (ns ! y')) *None* (*get-next* (ns ! y'))])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-x-y′*])
subgoal using *x-zs′* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** l xs zs **by** *auto*
done
moreover have *stamp-y′-n*: ‹*stamp* (ns[x' := *VMTF-Node* (*stamp* (ns ! x')) (*get-prev* (ns ! x')) *None*] ! y') $\leq n$›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns-y′-zs′-y′*] *stamp* **unfolding** l xs zs
by (*auto simp*: *sorted-append*)
ultimately have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (y' # zs') (*stamp* (ns' ! y'))
(ns[y' := *VMTF-Node* (*stamp* (ns ! y')) *None* (*get-next* (ns ! y'))])›
using l *vmtf-ns vmtf-ns-append-decomp xs-zs-nempty*(2) *ns′-def* **by** *auto*
have *vmtf-ns-y′-zs′-x′-y′*: ‹*vmtf-ns* (y' # zs') (*stamp* (ns' ! y')) ns'›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-y′*])
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** l xs zs **by** *auto*
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** l xs zs **by** *auto*
done
have *vmtf-ns-xs′*: ‹*vmtf-ns* (xs' @ [x']) m ns'›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-xs*])
subgoal using *y′-xs′ ns′-def* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-xs*] *ns′-def* **by** *auto*
done
have *vmtf-x′-y′*: ‹*vmtf-ns* (xs' @ [x', y'] @ zs') m
(ns'[x' := *VMTF-Node* (*stamp* (ns' ! x')) (*get-prev* (ns' ! x')) (*Some* y'),
y' := *VMTF-Node* (*stamp* (ns' ! y')) (*Some* x') (*get-next* (ns' ! y'))])›

apply (*rule vmtf-ns-append-rebuild*[*OF vmtf-ns-xs′ vmtf-ns-y′-zs′-x′-y′*])
subgoal using *stamp-y′-n vmtf-ns-xs vmtf-ns-zs stamp* ‹$n \leq stamp\ (ns\ !\ x)$›
unfolding *ns′-def* **by** *auto*
subgoal by (*metis append.assoc append-Cons distinct-remove1 r-l self-append-conv2 vmtf-ns vmtf-ns-distinct xs zs*)
done
have ‹*vmtf-ns* (*xs′* @ [*x′*, *y′*] @ *zs′*) *m*
(*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*)),
x := *VMTF-Node* (*stamp* (*ns′* ! *x*)) *None None*])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-x′-y′*])
subgoal
using *vmtf-ns-last-mid-get-next*[*OF vmtf-ns-x-y*] *vmtf-ns-last-mid-get-prev*[*OF vmtf-ns-x′-x*] *x-zs′*
by (*cases* ‹*ns*!*x*›; *auto simp*: *nth-list-update′ ns′-def*)
subgoal using *le-length* **unfolding** *l xs zs ns′-def* **by** *auto*
done
moreover have ‹*xs′* @ [*x′*, *y′*] @ *zs′* = *remove1 x l*›
unfolding *r-l xs zs* **by** *auto*
moreover have ‹*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*)),
x := *VMTF-Node* (*stamp* (*ns′* ! *x*)) *None None*] = *ns-vmtf-dequeue x ns*›
using *vmtf-ns-last-mid-get-next*[*OF vmtf-ns-x-y*] *vmtf-ns-last-mid-get-prev*[*OF vmtf-ns-x′-x*]
list-update-swap[*of x′ y′* - ‹- :: *nat-vmtf-node*›]
unfolding *ns′-def ns-vmtf-dequeue-def*
by (*auto simp*: *Let-def*)
ultimately show *?thesis*
by *force*
qed
qed

lemma *vmtf-ns-hd-next*:
‹*vmtf-ns* (*x* # *a* # *list*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *Some a*›
by (*auto 5 5 elim*: *vmtf-nsE*)

lemma *vmtf-ns-notin-dequeue*:
assumes *vmtf-ns*: ‹*vmtf-ns l m ns*› **and** *notin*: ‹*vmtf-ns-notin l m ns*› **and** *valid*: ‹$x < length\ ns$›
shows ‹*vmtf-ns-notin* (*remove1 x l*) *m* (*ns-vmtf-dequeue x ns*)›
proof (*cases* ‹$x \in set\ l$›)
case *False*
then have *H*: ‹*remove1 x l* = *l*›
by (*simp add*: *remove1-idem*)
have *simp-is-stupid*[*simp*]: ‹$a \in set\ l \Longrightarrow x \notin set\ l \Longrightarrow a \neq x$› ‹$a \in set\ l \Longrightarrow x \notin set\ l \Longrightarrow x \neq a$›
for *a x*
by *auto*
have
‹*get-prev* (*ns* ! *x*) = *None*› **and**
‹*get-next* (*ns* ! *x*) = *None*›
using *notin False valid* **unfolding** *vmtf-ns-notin-def* **by** *auto*
show *?thesis*
using *notin valid False* **unfolding** *vmtf-ns-notin-def*
by (*auto simp*: *vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def H split*: *option.splits*)
next
case *True*
then obtain *xs zs* **where**
l: ‹*l* = *xs* @ *x* # *zs*›
by (*meson split-list*)

have r-l: ‹remove1 x l = xs @ zs›
  using vmtf-ns-distinct[OF vmtf-ns] **unfolding** l **by** (simp add: remove1-append)

consider
  (xs-zs-empty) ‹xs = []› **and** ‹zs = []› |
  (xs-nempty-zs-empty) x′ xs′ **where** ‹xs = xs′ @ [x′]› **and** ‹zs = []› |
  (xs-empty-zs-nempty) y′ zs′ **where** ‹xs = []› **and** ‹zs = y′ # zs′› |
  (xs-zs-nempty) x′ y′ xs′ zs′ **where** ‹xs = xs′ @ [x′]› **and** ‹zs = y′ # zs′›
  by (cases xs rule: rev-cases; cases zs)
then show ?thesis
proof cases
  case xs-zs-empty
  then show ?thesis
    using notin vmtf-ns **unfolding** l **apply** (cases ‹ns ! x›)
      by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def vmtf-ns-single-iff
        split: option.splits)
next
  case xs-empty-zs-nempty **note** xs = this(1) **and** zs = this(1)
  have prev-next: ‹get-prev (ns ! x) = None› ‹get-next (ns ! x) = option-hd zs›
    using vmtf-ns **unfolding** l xs zs
    by (cases zs; auto simp: option-hd-def elim: vmtf-nsE dest: vmtf-ns-hd-next)+
  show ?thesis
    apply (rule vmtf-ns-notinI)
    apply (case-tac ‹i = x›)
    subgoal
      using vmtf-ns prev-next **unfolding** r-l **unfolding** l xs zs
      by (cases zs) (auto simp: ns-vmtf-dequeue-def Let-def
          vmtf-ns-notin-def vmtf-ns-single-iff
          split: option.splits)
    subgoal
      using vmtf-ns notin prev-next **unfolding** r-l **unfolding** l xs zs
      by (auto simp: ns-vmtf-dequeue-def Let-def
          vmtf-ns-notin-def vmtf-ns-single-iff
          split: option.splits
          intro: vmtf-ns.intros vmtf-ns-stamp-increase dest: vmtf-ns-skip-fst)
    done
next
  case xs-nempty-zs-empty **note** xs = this(1) **and** zs = this(2)
  have prev-next: ‹get-prev (ns ! x) = Some x′› ‹get-next (ns ! x) = None›
    using vmtf-ns vmtf-ns-append-decomp[of xs′ x′ x zs m ns] **unfolding** l xs zs
    by (auto simp: vmtf-ns-single-iff intro: vmtf-ns-last-mid-get-prev)
  then show ?thesis
    using vmtf-ns notin **unfolding** r-l **unfolding** l xs zs
    by (auto simp: ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp vmtf-ns-notin-def
        split: option.splits
        intro: vmtf-ns.intros)
next
  case xs-zs-nempty **note** xs = this(1) **and** zs = this(2)
  have vmtf-ns-x′-x: ‹vmtf-ns (xs′ @ [x′, x] @ (y′ # zs′)) m ns› **and**
    vmtf-ns-x-y: ‹vmtf-ns ((xs′ @ [x′]) @ [x, y′] @ zs′) m ns›
    using vmtf-ns **unfolding** l xs zs **by** simp-all
  have [simp]: ‹y′ < length ns› ‹x < length ns› ‹x ≠ y′› ‹x′ ≠ y′› ‹x′ < length ns› ‹y′ ≠ x′›
    ‹y′ ≠ x› ‹y′ ∉ set xs› ‹y′ ∉ set zs′›
    and x-zs′: ‹x ∉ set zs′› **and** x′-zs′: ‹x′ ∉ set zs′› **and** y′-xs′: ‹y′ ∉ set xs′›
    using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] **unfolding** l xs zs
    by auto

```
    have ‹get-next (ns!x) = Some y'› ‹get-prev (ns!x) = Some x'›
      using vmtf-ns-last-mid-get-prev[OF vmtf-ns-x'-x] vmtf-ns-last-mid-get-next[OF vmtf-ns-x-y]
      by fast+
    then show ?thesis
      using notin x-zs' x'-zs' y'-xs' unfolding l xs zs
      by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def)
  qed
qed

lemma vmtf-ns-stamp-distinct:
  assumes ‹vmtf-ns l m ns›
  shows ‹distinct (map (λa. stamp (ns!a)) l)›
  using assms
proof (induction rule: vmtf-ns.induct)
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(9) and a-le-y = this(2) and
    zs-a = this(3) and ab = this(4) and a-l = this(5) and mn = this(6) and xs' = this(7) and
    nn' = this(8)
  have [simp]: ‹map (λaa. stamp
               (if b = aa
                then VMTF-Node (stamp (xs ! aa)) (Some a) (get-next (xs ! aa))
                else xs ! aa)) l =
        map (λaa. stamp (xs ! aa)) l
       › for a
    apply (rule map-cong)
    subgoal ..
    subgoal using vmtf-ns-distinct[OF vmtf-ns] by auto
    done
  show ?case
    using Cons vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns]
    by (auto simp: sorted-many-eq-append leD vmtf-ns-stamp-sorted cong: if-cong)
qed auto

lemma vmtf-ns-thighten-stamp:
  assumes vmtf-ns: ‹vmtf-ns l m xs› and n: ‹∀ a∈set l. stamp (xs ! a) ≤ n›
  shows ‹vmtf-ns l n xs›
proof –
  consider
    (empty) ‹l = []› |
    (single) x where ‹l = [x]› |
    (more-than-two) x y ys where ‹l = x # y # ys›
    by (cases l; cases ‹tl l›) auto
  then show ?thesis
  proof cases
    case empty
    then show ?thesis by (auto intro: vmtf-ns.intros)
  next
    case (single x)
    then show ?thesis using n vmtf-ns by (auto simp: vmtf-ns-single-iff)
  next
    case (more-than-two x y ys) note l = this
    then have vmtf-ns': ‹vmtf-ns ([] @ [x, y] @ ys) m xs›
      using vmtf-ns by auto
    from vmtf-ns-append-decomp[OF this] have
      ‹vmtf-ns ([x]) m (xs[x := VMTF-Node (stamp (xs ! x)) (get-prev (xs ! x)) None])› and
      vmtf-ns-y-ys: ‹vmtf-ns (y # ys) (stamp (xs ! y))
        (xs[y := VMTF-Node (stamp (xs ! y)) None (get-next (xs ! y))])› and
```

⟨*stamp* (*xs* ! *y*) < *stamp* (*xs* ! *x*)⟩
      by *auto*
    have [*simp*]: ⟨*x* ≠ *y*⟩ ⟨*x* ∉ *set ys*⟩ ⟨*x* < *length xs*⟩ ⟨*y* < *length xs*⟩
      using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l* **by** *auto*
    show *?thesis*
      unfolding *l*
      apply (*rule vmtf-ns.Cons*[*OF vmtf-ns-y-ys*, *of* - ⟨*stamp* (*xs* ! *x*)⟩])
      subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l* **by** *auto*
      subgoal using *vmtf-ns* **unfolding** *l* **by** (*cases* ⟨*xs* ! *x*⟩) (*auto elim*: *vmtf-nsE*)
      subgoal by *simp*
      subgoal by *simp*
      subgoal using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *vmtf-ns-stamp-distinct*[*OF vmtf-ns*]
       by (*auto simp*: *l sorted-many-eq-append*)
      subgoal
        using *vmtf-ns vmtf-ns-last-mid-get-prev*[*OF vmtf-ns'*]
        apply (*cases* ⟨*xs* ! *y*⟩)
        by *simp* (*auto simp*: *l eq-commute*[*of* ⟨*xs* ! *y*⟩])
      subgoal using *n* **unfolding** *l* **by** *auto*
      done
  qed
qed

lemma *vmtf-ns-rescale*:
  assumes
    ⟨*vmtf-ns l m xs*⟩ **and**
    ⟨*sorted* (*map* (λ*a*. *st* ! *a*) (*rev l*))⟩ **and** ⟨*distinct* (*map* (λ*a*. *st* ! *a*) *l*)⟩
    ⟨∀ *a* ∈ *set l*. *get-prev* (*zs* ! *a*) = *get-prev* (*xs* ! *a*)⟩ **and**
    ⟨∀ *a* ∈ *set l*. *get-next* (*zs* ! *a*) = *get-next* (*xs* ! *a*)⟩ **and**
    ⟨∀ *a* ∈ *set l*. *stamp* (*zs* ! *a*) = *st* ! *a*⟩ **and**
    ⟨*length xs* ≤ *length zs*⟩ **and**
    ⟨∀ *a*∈*set l*. *a* < *length st*⟩ **and**
    m': ⟨∀ *a* ∈ *set l*. *st* ! *a* < *m'*⟩
  shows ⟨*vmtf-ns l m' zs*⟩
  using *assms*
proof (*induction arbitrary*: *zs m' rule*: *vmtf-ns.induct*)
  case (*Nil st xs*)
  then show *?case* **by** (*auto intro*: *vmtf-ns.intros*)
next
  case (*Cons1 a xs m n*)
  then show *?case* **by** (*cases* ⟨*zs* ! *a*⟩) (*auto simp*: *vmtf-ns-single-iff intro*!: *Max-ge nth-mem*)
next
  case (*Cons b l m xs a n xs' n' zs m'*) **note** *vmtf-ns* = *this*(*1*) **and** *a-le-y* = *this*(*2*) **and**
    zs-a = *this*(*3*) **and** *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and**
    nn' = *this*(*8*) **and** *IH* = *this*(*9*) **and** *H* = *this*(*10*−)
  have [*simp*]: ⟨*b* < *length xs*⟩ ⟨*b* ≠ *a*⟩ ⟨*a* ≠ *b*⟩ ⟨*b* ∉ *set l*⟩ ⟨*b* < *length zs*⟩ ⟨*a* < *length zs*⟩
    using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] *ab H*(*6*) *a-le-y* **unfolding** *xs'*
    by *force*+

  have *simp-is-stupid*[*simp*]: ⟨*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *a* ≠ *x*⟩ ⟨*a* ∈ *set l* ⟹ *x* ∉ *set l* ⟹ *x* ≠ *a*⟩
for *a x*
    by *auto*
  define *zs'* **where** ⟨*zs'* ≡ (*zs*[*b* := *VMTF-Node* (*st* ! *b*) (*get-prev* (*xs* ! *b*)) (*get-next* (*xs* ! *b*)),
        a := *VMTF-Node* (*st* ! *a*) *None* (*Some b*)])⟩
  have *zs-upd-zs*: ⟨*zs* = *zs*
    [*b* := *VMTF-Node* (*st* ! *b*) (*get-prev* (*xs* ! *b*)) (*get-next* (*xs* ! *b*)),
     a := *VMTF-Node* (*st* ! *a*) *None* (*Some b*),

```
    b := VMTF-Node (st ! b) (Some a) (get-next (xs ! b))]
  ›
  using H(2−5) xs' zs-a ‹b < length xs›
  by (metis list.set-intros(1) list.set-intros(2) list-update-id list-update-overwrite
    nth-list-update-eq nth-list-update-neq vmtf-node.collapse vmtf-node.sel(2,3))

have vtmf-b-l: ‹vmtf-ns (b # l) m' zs'›
  unfolding zs'-def
  apply (rule IH)
  subgoal using H(1) by (simp add: sorted-many-eq-append)
  subgoal using H(2) by auto
  subgoal using H(3,4,5) xs' zs-a a-l ab by (auto split: if-splits)
  subgoal using H(4) xs' zs-a a-l ab by auto
  subgoal using H(5) xs' a-l ab by auto
  subgoal using H(6) xs' by auto
  subgoal using H(7) xs' by auto
  subgoal using H(8) by auto
  done
then have ‹vmtf-ns (b # l) (stamp (zs' ! b)) zs'›
  by (rule vmtf-ns-thighten-stamp)
    (use vmtf-ns-stamp-sorted[OF vtmf-b-l] in ‹auto simp: sorted-append›)

then show ?case
  apply (rule vmtf-ns.Cons[of - - - - - ‹st ! a›])
  unfolding zs'-def
  subgoal using a-le-y H(6) xs' by auto
  subgoal using a-le-y by auto
  subgoal using ab.
  subgoal using a-l .
  subgoal using nn' mn H(1,2) by (auto simp: sorted-many-eq-append)
  subgoal using zs-upd-zs by auto
  subgoal using H by (auto intro!: Max-ge nth-mem)
  done
qed

lemma vmtf-ns-last-prev:
  assumes vmtf: ‹vmtf-ns (xs @ [x]) m ns›
  shows ‹get-prev (ns ! x) = option-last xs›
proof (cases xs rule: rev-cases)
  case Nil
  then show ?thesis using vmtf by (cases ‹ns!x›) (auto simp: vmtf-ns-single-iff)
next
  case (snoc xs' y')
  then show ?thesis
    using vmtf-ns-last-mid-get-prev[of xs' y' x ‹[]› m ns] vmtf by auto
qed
```

Abstract Invariants Invariants

- The atoms of *xs* and *ys* are always disjoint.
- The atoms of *ys* are *always* set.
- The atoms of *xs* *can* be set but do not have to.
- The atoms of *zs* are either in *xs* and *ys*.

definition *vmtf*-$\mathcal{L}_{all}$ :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat abs-vmtf-ns-remove* ⇒ *bool*› **where**
‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ≡ λ((*xs*, *ys*), *zs*).
(∀ *L*∈*ys*. *L* ∈ *atm-of* ‘ *lits-of-l M*) ∧
xs ∩ *ys* = {} ∧
zs ⊆ *xs* ∪ *ys* ∧
xs ∪ *ys* = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)
›

abbreviation *abs-vmtf-ns-inv* :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat abs-vmtf-ns* ⇒ *bool*› **where**
‹*abs-vmtf-ns-inv* $\mathcal{A}$ *M vm* ≡ *vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*vm*, {})›

Implementation

type-synonym (**in** −) *vmtf* = ‹*nat-vmtf-node list* × *nat* × *nat* × *nat* × *nat option*›
type-synonym (**in** −) *vmtf-remove-int* = ‹*vmtf* × *nat set*›

We use the opposite direction of the VMTF paper: The latest added element *fst-As* is at the beginning.

definition *vmtf* :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int set*› **where**
‹*vmtf* $\mathcal{A}$ *M* = {((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(∃ *xs'* *ys'*.
vmtf-ns (*ys'* @ *xs'*) *m ns* ∧ *fst-As* = *hd* (*ys'* @ *xs'*) ∧ *lst-As* = *last* (*ys'* @ *xs'*)
∧ *next-search* = *option-hd xs'*
∧ *vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)
∧ *vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*
∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧ (∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))
)}›

lemma *vmtf-consD*:
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ (*L* # *M*)›
proof −
obtain *xs'* *ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ (*L* # *M*) ((*set xs'*, *set ys'*), *remove*)›
using *abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def* **by** *auto*
ultimately have ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns* ∧
fst-As = *hd* (*ys'* @ *xs'*) ∧
lst-As = *last* (*ys'* @ *xs'*) ∧
next-search = *option-hd xs'* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*L* # *M*) ((*set xs'*, *set ys'*), *remove*) ∧
vmtf-ns-notin (*ys'* @ *xs'*) *m ns* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧
(∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show *?thesis*
unfolding *vmtf-def* **by** *fast*
qed

type-synonym (**in** −) *vmtf-option-fst-As* = ‹*nat-vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*›

definition (**in** −) *vmtf-dequeue* :: ‹*nat* ⇒ *vmtf* ⇒ *vmtf-option-fst-As*› **where**
‹*vmtf-dequeue* ≡ (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
  (*let fst-As′* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*);
    next-search′ = *if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*;
    lst-As′ = *if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As in*
  (*ns-vmtf-dequeue L ns*, *m*, *fst-As′*, *lst-As′*, *next-search′*)))›

It would be better to distinguish whether L is set in M or not.

definition *vmtf-enqueue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf*› **where**
‹*vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
  (*case fst-As of*
    None ⇒ (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
      (*if defined-lit M* (*Pos L*) *then None else Some L*))
  | *Some fst-As* ⇒
    let fst-As′ = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*)) *in*
    (*ns*[*L* := *VMTF-Node* (*m*+*1*) *None* (*Some fst-As*), *fst-As* := *fst-As′*],
      m+*1*, *L*, *the lst-As*, (*if defined-lit M* (*Pos L*) *then next-search else Some L*))))›

definition (**in** −) *vmtf-en-dequeue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf*› **where**
‹*vmtf-en-dequeue* = (λ*M L vm*. *vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *abs-vmtf-ns-bump-vmtf-dequeue*:
  fixes *M*
  assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
    L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
    dequeue: ‹(*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*) =
      vmtf-dequeue L (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)› **and**
    $\mathcal{A}_{in}$*-nempty*: ‹*isasat-input-nempty* $\mathcal{A}$›
  shows ‹∃ *xs′ ys′*. *vmtf-ns* (*ys′* @ *xs′*) *m′ ns′* ∧ *fst-As′* = *option-hd* (*ys′* @ *xs′*)
   ∧ *lst-As′* = *option-last* (*ys′* @ *xs′*)
   ∧ *next-search′* = *option-hd xs′*
   ∧ *next-search′* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)
   ∧ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set xs′*), *set ys′*), *to-remove*)
   ∧ *vmtf-ns-notin* (*ys′* @ *xs′*) *m′ ns′* ∧
   L ∉ *set* (*ys′* @ *xs′*) ∧ (∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
  unfolding *vmtf-def*
proof −
  have *ns′*: ‹*ns′* = *ns-vmtf-dequeue L ns*› **and**
    fst-As′: ‹*fst-As′* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*)› **and**
    lst-As′: ‹*lst-As′* = (*if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As*)› **and**
    m′m: ‹*m′* = *m*› **and**
    next-search-L-next:
      ‹*next-search′* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)›
    using *dequeue* **unfolding** *vmtf-dequeue-def* **by** *auto*
  obtain *xs ys* **where**
    vmtf: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
    notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**
    next-search: ‹*next-search* = *option-hd xs*› **and**
    abs-inv: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs*, *set ys*), *to-remove*)› **and**
    fst-As: ‹*fst-As* = *hd* (*ys* @ *xs*)› **and**
    lst-As: ‹*lst-As* = *last* (*ys* @ *xs*)› **and**
    atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
    L-ys-xs: ‹∀ *L*∈*set* (*ys* @ *xs*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›

```
  using vmtf unfolding vmtf-def by auto
have [dest]: ⟨xs = [] ⟹ ys = [] ⟹ False⟩
  using abs-inv 𝒜_in-nempty unfolding atms-of-ℒ_all-𝒜_in vmtf-ℒ_all-def
  by auto
let ?ys = ⟨ys⟩
let ?xs = ⟨xs⟩
have dist: ⟨distinct (xs @ ys)⟩
  using vmtf-ns-distinct[OF vmtf] by auto
have xs-ys: ⟨remove1 L ys @ remove1 L xs = remove1 L (ys @ xs)⟩
  using dist by (auto simp: remove1-append remove1-idem disjoint-iff-not-equal
      intro!: remove1-idem)
have atm-L-A: ⟨L < length ns⟩
  using atm-A L by blast

have ⟨vmtf-ns (remove1 L ys @ remove1 L xs) m' ns'⟩
  using vmtf-ns-ns-vmtf-dequeue[OF vmtf notin, of L] dequeue dist atm-L-A
  unfolding vmtf-dequeue-def by (auto split: if-splits simp: xs-ys)
moreover have next-search': ⟨next-search' = option-hd (remove1 L xs)⟩
proof –
  have ⟨[hd xs, hd (tl xs)] @ tl (tl xs) = xs⟩
    if ⟨xs ≠ []⟩ ⟨tl xs ≠ []⟩
    apply (cases xs; cases ⟨tl xs⟩)
     using that by (auto simp: tl-append split: list.splits)
  then have [simp]: ⟨get-next (ns ! hd xs) = option-hd (remove1 (hd xs) xs)⟩ if ⟨xs ≠ []⟩
    using vmtf-ns-last-mid-get-next[of ⟨?ys⟩ ⟨hd ?xs⟩
        ⟨hd (tl ?xs)⟩ ⟨tl (tl ?xs)⟩ m ns] vmtf vmtf-ns-distinct[OF vmtf] that
      distinct-remove1-last-butlast[of xs]
    by (cases xs; cases ⟨tl xs⟩)
      (auto simp: tl-append vmtf-ns-last-next split: list.splits elim: vmtf-nsE)
  have ⟨xs ≠ [] ⟹ xs ≠ [L] ⟹  L ≠ hd xs ⟹ hd xs = hd (remove1 L xs)⟩
    by (induction xs) (auto simp: remove1-Nil-iff)
  then have [simp]: ⟨option-hd xs = option-hd (remove1 L xs)⟩ if ⟨L ≠ hd xs⟩
    using that vmtf-ns-distinct[OF vmtf]
    by (auto simp: option-hd-def remove1-Nil-iff)
  show ?thesis
    using dequeue dist atm-L-A next-search next-search unfolding vmtf-dequeue-def
    by (auto split: if-splits simp: xs-ys dest: last-in-set)
  qed
moreover {
  have ⟨[hd ys, hd (tl ys)] @ tl (tl ys) = ys⟩
    if ⟨ys ≠ []⟩ ⟨tl ys ≠ []⟩
     using that by (auto simp: tl-append split: list.splits)
  then have ⟨get-next (ns ! hd (ys @ xs)) = option-hd (remove1 (hd (ys @ xs)) (ys @ xs))⟩
    if ⟨ys @ xs ≠ []⟩
    using vmtf-ns-last-next[of ⟨?xs @ butlast ?ys⟩ ⟨last ?ys⟩] that
    using vmtf-ns-last-next[of ⟨butlast ?xs⟩ ⟨last ?xs⟩]  vmtf dist
      distinct-remove1-last-butlast[of ⟨?ys @ ?xs⟩]
    by (cases ys; cases ⟨tl ys⟩)
     (auto simp: tl-append vmtf-ns-last-prev remove1-append hd-append remove1-Nil-iff
        split: list.splits if-splits elim: vmtf-nsE)
  moreover have ⟨hd ys ∉ set xs⟩ if ⟨ys ≠ []⟩
    using vmtf-ns-distinct[OF vmtf] that by (cases ys) auto
  ultimately have ⟨fst-As' = option-hd (remove1 L (ys @ xs))⟩
    using dequeue dist atm-L-A fst-As vmtf-ns-distinct[OF vmtf] vmtf
    unfolding vmtf-dequeue-def
    apply (cases ys)
```

subgoal by (*cases xs*) (*auto simp*: *remove1-append option-hd-def remove1-Nil-iff split*: *if-splits*)
subgoal by (*auto simp*: *remove1-append option-hd-def remove1-Nil-iff*)
done
}
moreover have ‹*lst-As'* = *option-last* (*remove1 L* (*ys* @ *xs*))›
apply (*cases* ‹*ys* @ *xs*› *rule*: *rev-cases*)
using *lst-As vmtf-ns-distinct*[*OF vmtf*] *vmtf-ns-last-prev vmtf*
by (*auto simp*: *lst-As' remove1-append simp del*: *distinct-append*) *auto*
moreover have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set* (*remove1 L xs*)), *set* (*remove1 L ys*)), *to-remove*)›
using *abs-inv L dist*
unfolding *vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto dest*: *in-set-remove1D*)
moreover have ‹*vmtf-ns-notin* (*remove1 L ?ys* @ *remove1 L ?xs*) *m' ns'*›
unfolding *xs-ys ns'*
apply (*rule vmtf-ns-notin-dequeue*)
subgoal using *vmtf* **unfolding** *m'm* **.**
subgoal using *notin* **unfolding** *m'm* **.**
subgoal using *atm-L-A* **.**
done
moreover have ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns'*›
using *atm-A* **unfolding** *ns'* **by** *auto*
moreover have ‹*L* ∉ *set* (*remove1 L ys* @ *remove1 L xs*)›
using *dist* **by** *auto*
moreover have ‹∀ *L*∈*set* (*remove1 L* (*ys* @ *xs*)). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L-ys-xs* **by** (*auto dest*: *in-set-remove1D*)
ultimately show *?thesis*
using *next-search-L-next*
apply −
apply (*rule exI*[*of* - ‹*remove1 L xs*›])
apply (*rule exI*[*of* - ‹*remove1 L ys*›])
unfolding *xs-ys*
by *blast*
qed

lemma *vmtf-ns-get-prev-not-itself*:
‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-prev* (*ns* ! *L*) ≠ *Some L*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by *auto*
subgoal by (*auto simp*: *vmtf-ns-single-iff*)
subgoal by *auto*
done

lemma *vmtf-ns-get-next-not-itself*:
‹*vmtf-ns xs m ns* ⟹ *L* ∈ *set xs* ⟹ *L* < *length ns* ⟹ *get-next* (*ns* ! *L*) ≠ *Some L*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by *auto*
subgoal by (*auto simp*: *vmtf-ns-single-iff*)
subgoal by *auto*
done

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue*:
fixes *M*
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
to-remove: ‹*to-remove'* ⊆ *to-remove* − {*L*}› **and**

nempty: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹(*vmtf-en-dequeue M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*›
unfolding *vmtf-def*
proof *clarify*
fix *xxs yys zzs ns' m' fst-As' lst-As' next-search'*
assume *dequeue*: ‹(*ns'*, *m'*, *fst-As'*, *lst-As'*, *next-search'*) =
vmtf-en-dequeue M L (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›
obtain *xs ys* **where**
vmtf-ns: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**
next-search: ‹*next-search* = *option-hd xs*› **and**
abs-inv: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs*, *set ys*), *to-remove*)› **and**
fst-As: ‹*fst-As* = *hd* (*ys* @ *xs*)› **and**
lst-As: ‹*lst-As* = *last* (*ys* @ *xs*)› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys-xs-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys* @ *xs*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *assms* **unfolding** *vmtf-def* **by** *auto*
have *atm-L-A*: ‹*L* < *length ns*›
using *atm-A L* **by** *blast*

d stands for dequeue

obtain *nsd md fst-Asd lst-Asd next-searchd* **where**
de: ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) = (*nsd*, *md*, *fst-Asd*, *lst-Asd*, *next-searchd*) ›
by (*cases* ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›)
obtain *xs' ys'* **where**
vmtf-ns': ‹*vmtf-ns* (*ys'* @ *xs'*) *md nsd*› **and**
fst-Asd: ‹*fst-Asd* = *option-hd* (*ys'* @ *xs'*)› **and**
lst-Asd: ‹*lst-Asd* = *option-last* (*ys'* @ *xs'*)› **and**
next-searchd-hd: ‹*next-searchd* = *option-hd xs'*› **and**
next-searchd-L-next:
‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)› **and**
abs-l: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set xs'*), *set ys'*), *to-remove*)› **and**
not-in: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *md nsd*› **and**
L-xs'-ys': ‹*L* ∉ *set* (*ys'* @ *xs'*)› **and**
L-xs'-ys'-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *abs-vmtf-ns-bump-vmtf-dequeue*[*OF vmtf L de*[*symmetric*] *nempty*] **by** *blast*

have [*simp*]: ‹*length ns'* = *length ns*› ‹*length nsd* = *length ns*›
using *dequeue de* **unfolding** *vmtf-en-dequeue-def comp-def vmtf-dequeue-def*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have *nsd*: ‹*nsd* = *ns-vmtf-dequeue L ns*›
using *de* **unfolding** *vmtf-dequeue-def* **by** *auto*
have [*simp*]: ‹*fst-As* = *L*› **if** ‹*ys'* = []› **and** ‹*xs'* = []›
proof −
have *1*: ‹*set-mset* $\mathcal{A}$ = {*L*}›
using *abs-l* **unfolding** *that vmtf*-$\mathcal{L}_{all}$-*def* **by** (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
show *?thesis*
using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs*-$\mathcal{L}_{all}$ *abs-inv*
unfolding *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *1 fst-As vmtf*-$\mathcal{L}_{all}$-*def*
by (*cases* ‹*ys* @ *xs*›) *auto*
qed
have *fst-As'*: ‹*fst-As'* = *L*› **and** *m'*: ‹*m'* = *md* + *1*› **and**
lst-As': ‹*fst-Asd* ≠ *None* ⟶ *lst-As'* = *the* (*lst-Asd*)›
‹*fst-Asd* = *None* ⟶ *lst-As'* = *L*›
using *dequeue* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)

have ‹*lst-As* = *L*› **if** ‹*ys*′ = []› **and** ‹*xs*′ = []›
proof −
  have *1*: ‹*set-mset* $\mathcal{A}$ = {*L*}›
    using *abs-l* **unfolding** *that vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
  then have ‹*set* (*ys* @ *xs*) = {*L*} ›
    using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv*
    unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 fst-As vmtf-*$\mathcal{L}_{all}$*-def*
    by *auto*
  then have ‹*ys* @ *xs* = [*L*]›
    using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv vmtf-*$\mathcal{L}_{all}$*-def*
    unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 fst-As*
    by (*cases* ‹*ys* @ *xs*› *rule*: *rev-cases*) (*auto simp del*: *set-append distinct-append*
      simp: *set-append*[*symmetric*], *auto*)
  then show *?thesis*
    using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv vmtf-*$\mathcal{L}_{all}$*-def*
    unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 lst-As*
    by (*auto simp del*: *set-append distinct-append simp*: *set-append*[*symmetric*])
qed
then have [*simp*]: ‹*lst-As*′ = *L*› **if** ‹*ys*′ = []› **and** ‹*xs*′ = []›
  using *lst-As*′ *fst-Asd* **unfolding** *that* **by** *auto*
have [*simp*]: ‹*lst-As*′ = *last* (*ys*′ @ *xs*′)› **if** ‹*ys*′ ≠ [] ∨ *xs*′ ≠ []›
  using *lst-As*′ *fst-Asd that lst-Asd* **by** *auto*

have ‹*get-prev* (*nsd* ! *i*) ≠ *Some L*› (**is** *?prev*) **and**
  ‹*get-next* (*nsd* ! *i*) ≠ *Some L*› (**is** *?next*)
  if
    i-le-A: ‹*i* < *length ns*› **and**
    i-L: ‹*i* ≠ *L*› **and**
    i-ys′: ‹*i* ∉ *set ys*′› **and**
    i-xs′: ‹*i* ∉ *set xs*′›
  for *i*
proof −
  have ‹*i* ∉ *set xs*› ‹*i* ∉ *set ys*› **and** *L-xs-ys*: ‹*L* ∈ *set xs* ∨ *L* ∈ *set ys*›
    using *i-ys*′ *i-xs*′ *abs-l abs-inv i-L* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
    by *auto*
  then have
    ‹*get-next* (*ns* ! *i*) = *None*›
    ‹*get-prev* (*ns* ! *i*) = *None*›
    using *notin i-le-A* **unfolding** *nsd vmtf-ns-notin-def ns-vmtf-dequeue-def*
    by (*auto simp*: *Let-def split*: *option.splits*)
  moreover have ‹*get-prev* (*ns* ! *L*) ≠ *Some L*› **and** ‹*get-next* (*ns* ! *L*) ≠ *Some L*›
    using *vmtf-ns-get-prev-not-itself*[*OF vmtf-ns*, *of L*] *L-xs-ys atm-L-A*
      vmtf-ns-get-next-not-itself[*OF vmtf-ns*, *of L*] **by** *auto*
  ultimately show *?next* **and** *?prev*
    using *i-le-A L-xs-ys* **unfolding** *nsd ns-vmtf-dequeue-def vmtf-ns-notin-def*
    by (*auto simp*: *Let-def split*: *option.splits*)
qed
then have *vmtf-ns-notin*′: ‹*vmtf-ns-notin* (*L* # *ys*′ @ *xs*′) *m*′ *ns*′›
  using *not-in dequeue fst-Asd* **unfolding** *vmtf-en-dequeue-def comp-def de vmtf-ns-notin-def*
    ns-vmtf-dequeue-def
  by (*auto simp add*: *vmtf-enqueue-def hd-append split*: *option.splits if-splits*)

consider
  (*defined*) ‹*defined-lit M* (*Pos L*)› |
  (*undef*) ‹*undefined-lit M* (*Pos L*)›

by *blast*
then show ‹∃ *xs′ ys′*.
vmtf-ns (*ys′* @ *xs′*) *m′ ns′* ∧
fst-As′ = *hd* (*ys′* @ *xs′*) ∧
lst-As′ = *last* (*ys′* @ *xs′*) ∧
next-search′ = *option-hd xs′* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove′*) ∧
vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* ∧
(∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) ∧
(∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof *cases*
case *defined*
have *L-in-M*: ‹*L* ∈ *atm-of* ‘ *lits-of-l M*›
using *defined* **by** (*auto simp*: *defined-lit-map lits-of-def*)
have *next-search′*: ‹*fst-Asd* ≠ *None* ⟶ *next-search′* = *next-searchd*›
‹*fst-Asd* = *None* ⟶ *next-search′* = *None*›
using *dequeue defined* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have *next-searchd*:
‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*)›
using *de* **by** (*auto simp*: *vmtf-dequeue-def*)
have *abs′*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *insert L* (*set ys′*)), *to-remove′*)›
using *abs-l to-remove L-in-M L-xs′-ys′* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto 5 5 dest*: *in-diffD*)

have *vmtf-ns*: ‹*vmtf-ns* (*L* # (*ys′* @ *xs′*)) *m′ ns′*›
proof (*cases* ‹*ys′* @ *xs′*›)
case *Nil*
then have ‹*fst-Asd* = *None*›
using *fst-Asd* **by** *auto*
then show *?thesis*
using *atm-L-A dequeue Nil* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
next
case (*Cons z zs*)
let *?m* = ‹(*stamp* (*nsd*!*z*))›
let *?Ad* = ‹*nsd*[*L* := *VMTF-Node m′ None* (*Some z*)]›
have *L-z-zs*: ‹*L* ∉ *set* (*z* # *zs*)›
using *L-xs′-ys′ atm-L-A* **unfolding** *Cons*
by *simp*
have *vmtf-ns-z*: ‹*vmtf-ns* (*z* # *zs*) *md nsd*›
using *vmtf-ns′* **unfolding** *Cons* **.**

have *vmtf-ns-A*: ‹*vmtf-ns* (*z* # *zs*) *md ?Ad*›
apply (*rule vmtf-ns-eq-iffI*[*of* - - *nsd*])
subgoal using *L-z-zs atm-L-A* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-z*] **by** *auto*
subgoal using *vmtf-ns-z* **.**
done
have [*simp*]: ‹*fst-Asd* = *Some z*›
using *fst-Asd* **unfolding** *Cons* **by** *simp*
show *?thesis*
unfolding *Cons*
apply (*rule vmtf-ns.Cons*[*of* - - *md ?Ad* - *m′*])
subgoal using *vmtf-ns-A* **.**
subgoal using *atm-L-A* **by** *simp*

subgoal using *atm-L-A* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *m'* **by** *simp-all*
subgoal
using *atm-L-A dequeue L-z-zs* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
apply (*cases* ‹*ns-vmtf-dequeue z ns* ! *z*›)
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
subgoal by *linarith*
done
qed
have *L-xs'-ys'-*$\mathcal{L}_{all}$*'*: ‹∀ *L'*∈*set* ((*L* # *ys'*) @ *xs'*). *L'* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L L-xs'-ys'-*$\mathcal{L}_{all}$ **by** *auto*
have *next-search'-xs'*: ‹*next-search'* = *option-hd xs'*›
using *next-searchd-L-next next-search' next-searchd-hd lst-As' fst-Asd*
by (*auto split*: *if-splits*)
show *?thesis*
apply (*rule exI*[*of* - ‹*xs'*›])
apply (*rule exI*[*of* - ‹*L* # *ys'*›])
using *fst-As' next-search' abs' atm-A vmtf-ns-notin' vmtf-ns ys-xs-*$\mathcal{L}_{all}$ *L-xs'-ys'-*$\mathcal{L}_{all}$*'*
next-searchd next-search'-xs'
by *simp*
next
case *undef*
have *next-search'*: ‹*next-search'* = *Some L*›
using *dequeue undef* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have *next-searchd*:
‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*)›
using *de* **by** (*auto simp*: *vmtf-dequeue-def*)
have *abs'*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*insert L* (*set* (*ys'* @ *xs'*)), *set* []), *to-remove'*)›
using *abs-l to-remove L-xs'-ys'* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto 5 5 dest*: *in-diffD*)

have *vmtf-ns*: ‹*vmtf-ns* (*L* # (*ys'* @ *xs'*)) *m' ns'*›
proof (*cases* ‹*ys'* @ *xs'*›)
case *Nil*
then have ‹*fst-Asd* = *None*›
using *fst-Asd* **by** *auto*
then show *?thesis*
using *atm-L-A dequeue Nil* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
next
case (*Cons z zs*)
let *?m* = ‹(*stamp* (*nsd*!*z*))›
let *?Ad* = ‹*nsd*[*L* := *VMTF-Node m' None* (*Some z*)]›
have *L-z-zs*: ‹*L* ∉ *set* (*z* # *zs*)›
using *L-xs'-ys' atm-L-A* **unfolding** *Cons*
by *simp*
have *vmtf-ns-z*: ‹*vmtf-ns* (*z* # *zs*) *md nsd*›
using *vmtf-ns'* **unfolding** *Cons* **.**

have *vmtf-ns-A*: ‹*vmtf-ns* (*z* # *zs*) *md ?Ad*›
apply (*rule vmtf-ns-eq-iffI*[*of* - - *nsd*])
subgoal using *L-z-zs atm-L-A* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-z*] **by** *auto*

subgoal using *vmtf-ns-z* **.**
done
have [*simp*]: ‹*fst-Asd* = *Some z*›
using *fst-Asd* **unfolding** *Cons* **by** *simp*
show *?thesis*
unfolding *Cons*
apply (*rule vmtf-ns.Cons*[*of - - md ?Ad - m'*])
subgoal using *vmtf-ns-A* **.**
subgoal using *atm-L-A* **by** *simp*
subgoal using *atm-L-A* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *L-z-zs* **by** *simp*
subgoal using *m'* **by** *simp-all*
subgoal
using *atm-L-A dequeue L-z-zs* **unfolding** *Nil vmtf-en-dequeue-def comp-def de nsd*
apply (*cases* ‹*ns-vmtf-dequeue z ns* ! *z*›)
by (*auto simp*: *vmtf-ns-single-iff vmtf-enqueue-def split*: *option.splits*)
subgoal by *linarith*
done
qed
have *L-xs'-ys'-*$\mathcal{L}_{all}$*'*: ‹∀ *L'*∈*set* ((*L* # *ys'*) @ *xs'*). *L'* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L L-xs'-ys'-*$\mathcal{L}_{all}$ **by** *auto*
show *?thesis*
apply (*rule exI*[*of -* ‹(*L* # *ys'*) @ *xs'*›])
apply (*rule exI*[*of -* ‹[]›])
using *fst-As' next-search' abs' atm-A vmtf-ns-notin' vmtf-ns ys-xs-*$\mathcal{L}_{all}$ *L-xs'-ys'-*$\mathcal{L}_{all}$*'*
next-searchd
by *simp*
qed
qed

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue'*:
fixes *M*
assumes
vmtf: ‹(*vm*, *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
to-remove: ‹*to-remove'* ⊆ *to-remove* − {*L*}› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹(*vmtf-en-dequeue M L vm*, *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*›
using *abs-vmtf-ns-bump-vmtf-en-dequeue assms* **by** (*cases vm*) *blast*

definition (**in** −) *vmtf-unset* :: ‹*nat* ⇒ *vmtf-remove-int* ⇒ *vmtf-remove-int*› **where**
‹*vmtf-unset* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(*if next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)
then ((*ns*, *m*, *fst-As*, *lst-As*, *Some L*), *to-remove*)
else ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)))›

lemma *vmtf-atm-of-ys-iff*:
assumes
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
shows ‹*L* ∈ *set ys'* ⟷ *next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)›
proof −

let *?xs′* = ‹*set xs′*›
let *?ys′* = ‹*set ys′*›
have *L-xs-ys*: ‹$L \in$ *?xs′* $\cup$ *?ys′*›
using *abs-vmtf L* **unfolding** *vmtf-$\mathcal{L}_{all}$-def*
by (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
have *dist*: ‹*distinct* (*xs′* @ *ys′*)›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*

have *sorted*: ‹*sorted* (*map* (λa. *stamp* (*ns* ! *a*)) (*rev xs′* @ *rev ys′*))› **and**
distinct: ‹*distinct* (*map* (λa. *stamp* (*ns* ! *a*)) (*xs′* @ *ys′*))›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *vmtf-ns-stamp-distinct*[*OF vmtf-ns*]
by (*auto simp*: *rev-map*[*symmetric*])
have *next-search-xs*: ‹*?xs′* = {} $\longleftrightarrow$ *next-search* = *None*›
using *next-search* **by** *auto*

have ‹*stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*) $\Longrightarrow$ $L \notin$ *?xs′*›
if ‹*xs′* $\neq$ []›
using *that sorted distinct L-xs-ys* **unfolding** *next-search*
by (*cases xs′*) (*auto simp*: *sorted-append*)
moreover have ‹*stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)› (**is** ‹*?n* < *?L*›)
if *xs′*: ‹*xs′* $\neq$ []› **and** ‹$L \in$ *?ys′*›
proof −
have ‹*?n* $\leq$ *?L*›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns*] *that last-in-set*[*OF xs′*]
by (*cases xs′*)
(*auto simp*: *rev-map*[*symmetric*] *next-search sorted-append sorted2*)
moreover have ‹*?n* $\neq$ *?L*›
using *vmtf-ns-stamp-distinct*[*OF vmtf-ns*] *that last-in-set*[*OF xs′*]
by (*cases xs′*) (*auto simp*: *rev-map*[*symmetric*] *next-search*)
ultimately show *?thesis*
by *arith*
qed
ultimately show *?thesis*
using *L-xs-ys next-search-xs dist* **by** *auto*
qed

lemma *vmtf-$\mathcal{L}_{all}$-to-remove-mono*:
assumes
‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*a*, *b*), *to-remove*)› **and**
‹*to-remove′* $\subseteq$ *to-remove*›
shows ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*a*, *b*), *to-remove′*)›
using *assms* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** (*auto simp*: *mset-subset-eqD*)

lemma *abs-vmtf-ns-unset-vmtf-unset*:
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹$L \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
to-remove: ‹*to-remove′* $\subseteq$ *to-remove*›
shows ‹(*vmtf-unset L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove′*)) $\in$ *vmtf* $\mathcal{A}$ *M*› (**is** ‹*?S* $\in$ -›)
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**

atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
L-ys′-xs′-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
obtain *ns′ m′ fst-As′ next-search′ to-remove′′ lst-As′* **where**
S: ‹*?S* = ((*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*), *to-remove′′*)›
by (*cases ?S*) *auto*
have *L-ys′-iff*: ‹*L* ∈ *set ys′* ⟷ (*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
using *vmtf-atm-of-ys-iff*[*OF vmtf-ns next-search abs-vmtf L-N*] **.**
have ‹*L* ∈ *set* (*xs′* @ *ys′*)›
using *abs-vmtf L-N* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
then have *L-ys′-xs′*: ‹*L* ∈ *set ys′* ⟷ *L* ∉ *set xs′*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **by** *auto*
have ‹∃ *xs′ ys′*.
vmtf-ns (*ys′* @ *xs′*) *m′ ns′* ∧
fst-As′ = *hd* (*ys′* @ *xs′*) ∧
lst-As′ = *last* (*ys′* @ *xs′*) ∧
next-search′ = *option-hd xs′* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove′′*) ∧
vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) ∧
(∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof (*cases* ‹*L* ∈ *set xs′*›)
case *True*
then have *C*: ‹¬(*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *abs-vmtf*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove′′*)›
apply (*rule vmtf-*$\mathcal{L}_{all}$*-to-remove-mono*)
apply (*rule abs-vmtf*)
using *to-remove S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)
show *?thesis*
using *S True* **unfolding** *vmtf-unset-def L-ys′-xs′*[*symmetric*]
apply −
apply (*simp add*: *C*)
using *vmtf-ns fst-As next-search abs-vmtf notin atm-A to-remove L-ys′-xs′-*$\mathcal{L}_{all}$ *lst-As*
by *auto*
next
case *False*
then have *C*: ‹*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*)›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *L-ys*: ‹*L* ∈ *set ys′*›
by (*use False L-ys′-xs′* **in** *auto*)
define *y-ys* **where** ‹*y-ys* ≡ *takeWhile* ((≠) *L*) *ys′*›
define *x-ys* **where** ‹*x-ys* ≡ *drop* (*length y-ys*) *ys′*›
let *?ys′* = ‹*y-ys*›
let *?xs′* = ‹*x-ys* @ *xs′*›
have *x-ys-take-ys′*: ‹*y-ys* = *take* (*length y-ys*) *ys′*›
unfolding *y-ys-def*
by (*subst take-length-takeWhile-eq-takeWhile*[*of* ‹(≠) *L*› ‹*ys′*›, *symmetric*]) *standard*
have *ys′-y-x*: ‹*ys′* = *y-ys* @ *x-ys*›
by (*subst x-ys-take-ys′*) (*auto simp*: *x-ys-def*)
have *y-ys-le-ys′*: ‹*length y-ys* < *length ys′*›
using *L-ys* **by** (*metis* (*full-types*) *append-eq-conv-conj append-self-conv le-antisym*
length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys′ y-ys-def)
from *nth-length-takeWhile*[*OF this*[*unfolded y-ys-def*]] **have** [*simp*]: ‹*x-ys* ≠ []› ‹*hd x-ys* = *L*›
using *y-ys-le-ys′* **unfolding** *x-ys-def y-ys-def*
by (*auto simp*: *x-ys-def y-ys-def hd-drop-conv-nth*)

have [*simp*]: ‹*ns'* = *ns*› ‹*m'* = *m*› ‹*fst-As'* = *fst-As*› ‹*next-search'* = *Some L*› ‹*to-remove''* = *to-remove'*›
‹*lst-As'* = *lst-As*›
using *S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)

have ‹*vmtf-ns* (*?ys'* @ *?xs'*) *m ns*›
using *vmtf-ns* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*fst-As'* = *hd* (*?ys'* @ *?xs'*)›
using *fst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*lst-As'* = *last* (*?ys'* @ *?xs'*)›
using *lst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*next-search'* = *option-hd ?xs'*›
by *auto*
moreover {
have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *to-remove*)›
using *abs-vmtf vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def ys'-y-x*
by *auto*
then have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *to-remove'*)›
by (*rule vmtf-*$\mathcal{L}_{all}$*-to-remove-mono*) (*use to-remove* **in** *auto*)
}
moreover have ‹*vmtf-ns-notin* (*?ys'* @ *?xs'*) *m ns*›
using *notin* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹∀ *L*∈*set* (*?ys'* @ *?xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *L-ys'-xs'-*$\mathcal{L}_{all}$ **unfolding** *ys'-y-x* **by** *auto*
ultimately show *?thesis*
using *S False atm-A* **unfolding** *vmtf-unset-def L-ys'-xs'*[*symmetric*]
by (*fastforce simp add*: *C*)
qed
then show *?thesis*
unfolding *vmtf-def S*
by *fast*
qed

definition (**in** −) *vmtf-dequeue-pre* **where**
‹*vmtf-dequeue-pre* = (λ(*L*,*ns*). *L* < *length ns* ∧
(*get-next* (*ns*!*L*) ≠ *None* ⟶ *the* (*get-next* (*ns*!*L*)) < *length ns*) ∧
(*get-prev* (*ns*!*L*) ≠ *None* ⟶ *the* (*get-prev* (*ns*!*L*)) < *length ns*))›

lemma (**in** −) *vmtf-dequeue-pre-alt-def*:
‹*vmtf-dequeue-pre* = (λ(*L*, *ns*). *L* < *length ns* ∧
(∀ *a*. *Some a* = *get-next* (*ns*!*L*) ⟶ *a* < *length ns*) ∧
(∀ *a*. *Some a* = *get-prev* (*ns*!*L*) ⟶ *a* < *length ns*))›
apply (*intro ext*, *rename-tac x*)
subgoal for *x*
by (*cases* ‹*get-next* ((*snd x*)!(*fst x*))›; *cases* ‹*get-prev* ((*snd x*)!(*fst x*))›)
(*auto simp*: *vmtf-dequeue-pre-def intro*!: *ext*)
done

definition *vmtf-en-dequeue-pre* :: ‹*nat multiset* ⇒ ((*nat*, *nat*) *ann-lits* × *nat*) × *vmtf* ⇒ *bool*› **where**
‹*vmtf-en-dequeue-pre* $\mathcal{A}$ = (λ((*M*, *L*),(*ns*,*m*,*fst-As*, *lst-As*, *next-search*)).
L < *length ns* ∧ *vmtf-dequeue-pre* (*L*, *ns*) ∧
fst-As < *length ns* ∧ (*get-next* (*ns* ! *fst-As*) ≠ *None* ⟶ *get-prev* (*ns* ! *lst-As*) ≠ *None*) ∧
(*get-next* (*ns* ! *fst-As*) = *None* ⟶ *fst-As* = *lst-As*) ∧
m+*1* ≤ *uint64-max* ∧
Pos L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)›

lemma (**in** −) *id-reorder-list*:
⟨(*RETURN o id*, *reorder-list vm*) ∈ ⟨*nat-rel*⟩*list-rel* $\rightarrow_f$ ⟨⟨*nat-rel*⟩*list-rel*⟩*nres-rel*⟩
unfolding *reorder-list-def* **by** (*intro frefI nres-relI*) *auto*

lemma *vmtf-vmtf-en-dequeue-pre-to-remove*:
assumes *vmtf*: ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*⟩ **and**
i: ⟨*A* ∈ *to-remove*⟩ **and**
m-le: ⟨*m* + *1* ≤ *uint64-max*⟩ **and**
nempty: ⟨*isasat-input-nempty* $\mathcal{A}$⟩
shows ⟨*vmtf-en-dequeue-pre* $\mathcal{A}$ ((*M*, *A*), (*ns*, *m*, *fst-As*, *lst-As*, *next-search*))⟩
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ⟨*vmtf-ns* (*ys′* @ *xs′*) *m ns*⟩ **and**
fst-As: ⟨*fst-As* = *hd* (*ys′* @ *xs′*)⟩ **and**
lst-As: ⟨*lst-As* = *last* (*ys′* @ *xs′*)⟩ **and**
next-search: ⟨*next-search* = *option-hd xs′*⟩ **and**
abs-vmtf: ⟨*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*)⟩ **and**
notin: ⟨*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*⟩ **and**
atm-A: ⟨∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*⟩ **and**
L-ys′-xs′-$\mathcal{L}_{all}$: ⟨∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
have [*dest*]: *False* **if** ⟨*ys′* = []⟩ **and** ⟨*xs′* = []⟩
proof −
have *1*: ⟨*set-mset* $\mathcal{A}$ = {}⟩
using *abs-vmtf* **unfolding** *that vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
then show *?thesis*
using *nempty* **by** *auto*
qed

have ⟨*A* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
using *abs-vmtf i* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
then have *remove-i-le-A*: ⟨*A* < *length ns*⟩ **and**
i-L: ⟨*Pos A* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩
using *atm-A* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *atms-of-def*)
moreover have ⟨*fst-As* < *length ns*⟩
using *fst-As atm-A L-ys′-xs′-*$\mathcal{L}_{all}$ **by** (*cases ys′*; *cases xs′*) *auto*
moreover have ⟨*get-prev* (*ns* ! *lst-As*) ≠ *None*⟩ **if** ⟨*get-next* (*ns* ! *fst-As*) ≠ *None*⟩
using *that vmtf-ns-hd-next*[*of* ⟨*hd* (*ys′* @ *xs′*)⟩ ⟨*hd* (*tl* (*ys′* @ *xs′*))⟩ ⟨*tl* (*tl* (*ys′* @ *xs′*))⟩]
vmtf-ns vmtf-ns-last-prev[*of* ⟨*butlast* (*ys′* @ *xs′*)⟩ ⟨*last* (*ys′* @ *xs′*)⟩]
vmtf-ns-last-next[*of* ⟨*butlast* (*ys′* @ *xs′*)⟩ ⟨*last* (*ys′* @ *xs′*)⟩]
by (*cases* ⟨*ys′* @ *xs′*⟩; *cases* ⟨*tl* (*ys′* @ *xs′*)⟩)
(*auto simp*: *fst-As lst-As*)
moreover have ⟨*vmtf-dequeue-pre* (*A*, *ns*)⟩
proof −
have ⟨*A* < *length ns*⟩
using *i abs-vmtf atm-A* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
moreover have ⟨*y* < *length ns*⟩ **if** *get-next*: ⟨*get-next* (*ns* ! (*A*)) = *Some y*⟩ **for** *y*
proof (*cases* ⟨*A* ∈ *set* (*ys′* @ *xs′*)⟩)
case *False*
then show *?thesis*
using *notin get-next remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
next
case *True*
then obtain *zs zs′* **where** *zs*: ⟨*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*⟩
using *split-list* **by** *fastforce*
moreover have ⟨*set* (*ys′* @ *xs′*) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩

using *abs-vmtf* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
ultimately show *?thesis*
using *vmtf-ns-last-mid-get-next-option-hd*[*of zs′ A zs m ns*] *vmtf-ns atm-A get-next L-ys′-xs′-$\mathcal{L}_{all}$* **unfolding** *zs* **by** *force*
qed
moreover have ‹*y* < *length ns*› **if** *get-prev*: ‹*get-prev* (*ns* ! (*A*)) = *Some y*› **for** *y*
proof (*cases* ‹*A* ∈ *set* (*ys′* @ *xs′*)›)
case *False*
then show *?thesis*
using *notin get-prev remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
next
case *True*
then obtain *zs zs′* **where** *zs*: ‹*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*›
using *split-list* **by** *fastforce*
moreover have ‹*set* (*ys′* @ *xs′*) = *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *abs-vmtf* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
ultimately show *?thesis*
using *vmtf-ns-last-mid-get-prev-option-last*[*of zs′ A zs m ns*] *vmtf-ns atm-A get-prev L-ys′-xs′-$\mathcal{L}_{all}$* **unfolding** *zs* **by** *force*
qed
ultimately show *?thesis*
unfolding *vmtf-dequeue-pre-def* **by** *auto*
qed
moreover have ‹*get-next* (*ns* ! *fst-As*) = *None* ⟶ *fst-As* = *lst-As*›
using *vmtf-ns-hd-next*[*of* ‹*hd* (*ys′* @ *xs′*)› ‹*hd* (*tl* (*ys′* @ *xs′*))› ‹*tl* (*tl* (*ys′* @ *xs′*))›]
vmtf-ns vmtf-ns-last-prev[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
vmtf-ns-last-next[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
by (*cases* ‹*ys′* @ *xs′*›; *cases* ‹*tl* (*ys′* @ *xs′*)›)
(*auto simp*: *fst-As lst-As*)
ultimately show *?thesis*
using *m-le* **unfolding** *vmtf-en-dequeue-pre-def* **by** *auto*
qed

lemma *vmtf-vmtf-en-dequeue-pre-to-remove′*:
assumes *vmtf*: ‹(*vm*, *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
i: ‹*A* ∈ *to-remove*› **and** ‹*fst* (*snd vm*) + *1* ≤ *uint64-max*› **and**
A: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹*vmtf-en-dequeue-pre* $\mathcal{A}$ ((*M*, *A*), *vm*)›
using *vmtf-vmtf-en-dequeue-pre-to-remove assms*
by (*cases vm*) *auto*

lemma *wf-vmtf-get-next*:
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹*wf* {(*get-next* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧ *the a* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}› (**is** ‹*wf ?R*›)
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *f* **where**
f: ‹(*f* (*Suc i*), *f i*) ∈ *?R*› **for** *i*
unfolding *wf-iff-no-infinite-down-chain* **by** *blast*

obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*)› **and**

notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?f0* = ‹*the* (*f 0*)›
have *f-None*: ‹*f i* ≠ *None*› **for** *i*
using *f*[*of i*] **by** *fast*
have *f-Suc* : ‹*f* (*Suc n*) = *get-next* (*ns* ! *the* (*f n*))› **for** *n*
using *f*[*of n*] **by** *auto*
have *f0-length*: ‹*?f0* < *length ns*›
using *f*[*of 0*] *atm-A*
by *auto*
have ‹*?f0* ∈ *set* (*ys'* @ *xs'*)›
apply (*rule ccontr*)
using *notin f-Suc*[*of 0*] *f0-length* **unfolding** *vmtf-ns-notin-def*
by (*auto simp*: *f-None*)
then obtain *i0* **where**
i0: ‹(*ys'* @ *xs'*) ! *i0* = *?f0*› ‹*i0* < *length* (*ys'* @ *xs'*)›
by (*meson in-set-conv-nth*)
define *zs* **where** ‹*zs* = *ys'* @ *xs'*›
have *H*: ‹*ys'* @ *xs'* = *take m* (*ys'* @ *xs'*) @ [(*ys'* @ *xs'*) ! *m*, (*ys'* @ *xs'*) ! (*m*+*1*)] @
drop (*m*+*2*) (*ys'* @ *xs'*)›
if ‹*m*+*1* < *length* (*ys'* @ *xs'*)›
for *m*
using *that*
unfolding *zs-def*[*symmetric*]
apply −
apply (*subst id-take-nth-drop*[*of m*])
by (*auto simp*: *Cons-nth-drop-Suc simp del*: *append-take-drop-id*)

have ‹*the* (*f n*) = (*ys'* @ *xs'*) ! (*i0* + *n*) ∧ *i0* + *n* < *length* (*ys'* @ *xs'*)› **for** *n*
proof (*induction n*)
case *0*
then show *?case* **using** *i0* **by** *simp*
next
case (*Suc n'*)
have *i0-le*: ‹*i0* + *n'* + *1* < *length* (*ys'* @ *xs'*)›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then have ‹*i0* + *n'* + *1* = *length* (*ys'* @ *xs'*)›
using *Suc* **by** *auto*
then have ‹*ys'* @ *xs'* = *butlast* (*ys'* @ *xs'*) @ [*the* (*f n'*)]›
using *Suc* **by** (*metis add-diff-cancel-right' append-butlast-last-id length-0-conv length-butlast less-one not-add-less2 nth-append-length*)
then show *False*
using *vmtf-ns-last-next*[*of* ‹*butlast* (*ys'* @ *xs'*)› ‹*the* (*f n'*)› *m ns*] *vmtf-ns f-Suc*[*of n'*] **by** (*auto simp*: *f-None*)
qed
have *get-next*: ‹*get-next* (*ns* ! ((*ys'* @ *xs'*) ! (*i0* + *n'*))) = *Some* ((*ys'* @ *xs'*) ! (*i0* + *n'* + *1*))›
apply(*rule vmtf-ns-last-mid-get-next*[*of* ‹*take* (*i0* + *n'*) (*ys'* @ *xs'*)›
‹(*ys'* @ *xs'*) ! (*i0* + *n'*)›
‹(*ys'* @ *xs'*) ! ((*i0* + *n'*) + *1*)›
‹*drop* ((*i0* + *n'*) + *2*) (*ys'* @ *xs'*)›
m ns])
apply (*subst H*[*symmetric*])
subgoal using *i0-le* **.**
subgoal using *vmtf-ns* **by** *simp*

done
then show *?case*
using *f-Suc*[*of n'*] *Suc i0-le* **by** *auto*
qed
then show *False*
by *blast*
qed

lemma *vmtf-next-search-take-next*:
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
n: ‹*next-search* ≠ *None*› **and**
def-n: ‹*defined-lit M* (*Pos* (*the next-search*))›
shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *get-next* (*ns*!*the next-search*)), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
unfolding *vmtf-def*
proof *clarify*
obtain *xs' ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys'-xs'-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?xs'* = ‹*tl xs'*›
let *?ys'* = ‹*ys'* @ [*hd xs'*]›
have [*simp*]: ‹*xs'* ≠ []›
using *next-search n* **by** *auto*
have ‹*vmtf-ns* (*?ys'* @ *?xs'*) *m ns*›
using *vmtf-ns* **by** (*cases xs'*) *auto*
moreover have ‹*fst-As* = *hd* (*?ys'* @ *?xs'*)›
using *fst-As* **by** *auto*
moreover have ‹*lst-As* = *last* (*?ys'* @ *?xs'*)›
using *lst-As* **by** *auto*
moreover have ‹*get-next* (*ns* ! *the next-search*) = *option-hd ?xs'*›
using *next-search n vmtf-ns*
by (*cases xs'*) (*auto dest*: *vmtf-ns-last-mid-get-next-option-hd*)
moreover {
have [*dest*]: ‹*defined-lit M* (*Pos a*) ⟹ *a* ∈ *atm-of* ‘ *lits-of-l M*› **for** *a*
by (*auto simp*: *defined-lit-map lits-of-def*)
have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *to-remove*)›
using *abs-vmtf def-n next-search n vmtf-ns-distinct*[*OF vmtf-ns*]
unfolding *vmtf-*$\mathcal{L}_{all}$*-def*
by (*cases xs'*) *auto* **}**
moreover have ‹*vmtf-ns-notin* (*?ys'* @ *?xs'*) *m ns*›
using *notin* **by** *auto*
moreover have ‹∀ *L*∈*set* (*?ys'* @ *?xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *ys'-xs'-*$\mathcal{L}_{all}$ **by** *auto*
ultimately show ‹∃ *xs' ys'*. *vmtf-ns* (*ys'* @ *xs'*) *m ns* ∧
fst-As = *hd* (*ys'* @ *xs'*) ∧
lst-As = *last* (*ys'* @ *xs'*) ∧
get-next (*ns* ! *the next-search*) = *option-hd xs'* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*) ∧
vmtf-ns-notin (*ys'* @ *xs'*) *m ns* ∧

($\forall L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}).\ L < length\ ns) \wedge$
($\forall L \in set\ (ys' \mathbin{@} xs').\ L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}))$›
using *atm-A* **by** *blast*
qed

definition *vmtf-find-next-undef* :: ‹*nat multiset* ⇒ *vmtf-remove-int* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat option*) *nres*› **where**
‹*vmtf-find-next-undef* $\mathcal{A}$ = (λ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) *M*. *do* {
$WHILE_T{}^{\lambda next\text{-}search.\ ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ to\text{-}remove) \in vmtf\ \mathcal{A}\ M\ \wedge}$ $(next\text{-}search \neq None \longrightarrow Pos\ ($
(λ*next-search*. *next-search* ≠ *None* ∧ *defined-lit M* (*Pos* (*the next-search*)))
(λ*next-search*. *do* {
ASSERT(*next-search* ≠ *None*);
let n = *the next-search*;
ASSERT(*Pos n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
ASSERT (*n* < *length ns*);
RETURN (*get-next* (*ns*!*n*))
}
)
next-search
})›

lemma *vmtf-find-next-undef-ref*:
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹*vmtf-find-next-undef* $\mathcal{A}$ ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) *M*
≤ ⇓ *Id* (*SPEC* (λ*L*. ((*ns*, *m*, *fst-As*, *lst-As*, *L*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M* ∧
(*L* = *None* ⟶ (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M L*)) ∧
(*L* ≠ *None* ⟶ *Pos* (*the L*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *undefined-lit M* (*Pos* (*the L*)))))›
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
have *no-next-search-all-defined*:
‹((*ns′*, *m′*, *fst-As′*, *lst-As′*, *None*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *x* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *defined-lit M x*›
for *x ns′ m′ fst-As′ lst-As′ remove*
by (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
defined-lit-map lits-of-def)
have *next-search-*$\mathcal{L}_{all}$:
‹((*ns′*, *m′*, *fst-As′*, *lst-As′*, *Some y*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *y* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
for *ns′ m′ fst-As′ remove y lst-As′*
by (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
defined-lit-map lits-of-def)
have *next-search-le-A′*:
‹((*ns′*, *m′*, *fst-As′*, *lst-As′*, *Some y*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *y* < *length ns′*›
for *ns′ m′ fst-As′ remove y lst-As′*
by (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
defined-lit-map lits-of-def)
show *?thesis*
unfolding *vmtf-find-next-undef-def*

apply (*refine-vcg*
WHILEIT-rule[**where** *R*=‹{(*get-next* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧ *the a* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}›])
subgoal using *vmtf* **by** (*rule wf-vmtf-get-next*)
subgoal using *next-search vmtf* **by** *auto*
subgoal using *vmtf* **by** (*auto dest*!: *next-search-*$\mathcal{L}_{all}$ *simp*: *image-image in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
subgoal using *vmtf* **by** *auto*
subgoal using *vmtf* **by** *auto*
subgoal using *vmtf* **by** (*auto dest*: *next-search-le-A′*)
subgoal by (*auto simp*: *image-image in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
(*metis next-search-*$\mathcal{L}_{all}$ *option.distinct*(*1*) *option.sel vmtf-next-search-take-next*)
subgoal by (*auto simp*: *image-image in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
(*metis next-search-*$\mathcal{L}_{all}$ *option.distinct*(*1*) *option.sel vmtf-next-search-take-next*)
subgoal by (*auto dest*: *no-next-search-all-defined next-search-*$\mathcal{L}_{all}$)
subgoal by (*auto dest*: *next-search-le-A′*)
subgoal for *x1 ns′ x2 m′ x2a fst-As′ next-search′ x2c s*
by (*auto dest*: *no-next-search-all-defined next-search-*$\mathcal{L}_{all}$)
subgoal by (*auto dest*: *vmtf-next-search-take-next*)
subgoal by (*auto simp*: *image-image in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
done
qed

definition *vmtf-mark-to-rescore*
:: ‹*nat* ⇒ *vmtf-remove-int* ⇒ *vmtf-remove-int*›
where
‹*vmtf-mark-to-rescore L* = (λ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
((*ns*, *m*, *fst-As*, *next-search*), *insert L to-remove*))›

lemma *vmtf-mark-to-rescore*:
assumes
L: ‹*L* ∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹*vmtf-mark-to-rescore L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*›
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *to-remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *insert L to-remove*)›
using *abs-vmtf L* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
ultimately show *?thesis*
unfolding *vmtf-def vmtf-mark-to-rescore-def* **by** *fast*
qed

lemma *vmtf-unset-vmtf-tl*:
fixes *M*
defines [*simp*]: ‹*L* ≡ *atm-of* (*lit-of* (*hd M*))›
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* ≠ []›
shows ‹(*vmtf-unset L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)) ∈ *vmtf* $\mathcal{A}$ (*tl M*)›

(**is** ‹*?S* ∈ -›)
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys′-xs′-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
obtain *ns′ m′ fst-As′ next-search′ remove″ lst-As′* **where**
S: ‹*?S* = ((*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*), *remove″*)›
by (*cases ?S*) *auto*
have *L-ys′-iff*: ‹*L* ∈ *set ys′* ⟷ (*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
using *vmtf-atm-of-ys-iff*[*OF vmtf-ns next-search abs-vmtf L-N*] **.**
have *dist*: ‹*distinct* (*ys′* @ *xs′*)›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **.**
have ‹*L* ∈ *set* (*xs′* @ *ys′*)›
using *abs-vmtf L-N* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
then have *L-ys′-xs′*: ‹*L* ∈ *set ys′* ⟷ *L* ∉ *set xs′*›
using *dist* **by** *auto*
have [*simp*]: ‹*remove″* = *remove*›
using *S* **unfolding** *vmtf-unset-def* **by** (*auto split*: *if-splits*)
have ‹∃ *xs′ ys′*.
vmtf-ns (*ys′* @ *xs′*) *m′ ns′* ∧
fst-As′ = *hd* (*ys′* @ *xs′*) ∧
lst-As′ = *last* (*ys′* @ *xs′*) ∧
next-search′ = *option-hd xs′* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set xs′*, *set ys′*), *remove″*) ∧
vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) ∧
(∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof (*cases* ‹*L* ∈ *set xs′*›)
case *True*
then have *C*[*unfolded L-def*]: ‹¬(*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *abs-vmtf*: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set xs′*, *set ys′*), *remove*)›
using *S abs-vmtf dist L-ys′-xs′ True* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def vmtf-unset-def*
by (*cases M*) (*auto simp*: *C*)
show *?thesis*
using *S True* **unfolding** *vmtf-unset-def L-ys′-xs′*[*symmetric*]
apply −
apply (*simp add*: *C*)
using *vmtf-ns fst-As next-search abs-vmtf notin atm-A ys′-xs′-*$\mathcal{L}_{all}$ *lst-As*
by *auto*
next
case *False*
then have *C*[*unfolded L-def*]: ‹*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*)›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *L-ys*: ‹*L* ∈ *set ys′*›
by (*use False L-ys′-xs′* **in** *auto*)
define *y-ys* **where** ‹*y-ys* ≡ *takeWhile* ((≠) *L*) *ys′*›
define *x-ys* **where** ‹*x-ys* ≡ *drop* (*length y-ys*) *ys′*›

let *?ys'* = ‹*y-ys*›
let *?xs'* = ‹*x-ys* @ *xs'*›
have *x-ys-take-ys'*: ‹*y-ys* = *take* (*length y-ys*) *ys'*›
unfolding *y-ys-def*
by (*subst take-length-takeWhile-eq-takeWhile*[*of* ‹(≠) *L*› ‹*ys'*›, *symmetric*]) *standard*
have *ys'-y-x*: ‹*ys'* = *y-ys* @ *x-ys*›
by (*subst x-ys-take-ys'*) (*auto simp*: *x-ys-def*)
have *y-ys-le-ys'*: ‹*length y-ys* < *length ys'*›
using *L-ys* **by** (*metis* (*full-types*) *append-eq-conv-conj append-self-conv le-antisym length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys' y-ys-def*)
from *nth-length-takeWhile*[*OF this*[*unfolded y-ys-def*]] **have** [*simp*]: ‹*x-ys* ≠ []› ‹*hd x-ys* = *L*›
using *y-ys-le-ys'* **unfolding** *x-ys-def y-ys-def*
by (*auto simp*: *x-ys-def y-ys-def hd-drop-conv-nth*)
have [*simp*]: ‹*ns'* = *ns*› ‹*m'* = *m*› ‹*fst-As'* = *fst-As*› ‹*next-search'* = *Some* (*atm-of* (*lit-of* (*hd M*)))›
‹*lst-As'* = *lst-As*›
using *S* **unfolding** *vmtf-unset-def* **by** (*auto simp*: *C*)
have *L-y-ys*: ‹*L* ∉ *set y-ys*›
unfolding *y-ys-def* **by** (*metis* (*full-types*) *takeWhile-eq-all-conv takeWhile-idem*)
have ‹*vmtf-ns* (*?ys'* @ *?xs'*) *m ns*›
using *vmtf-ns* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*fst-As'* = *hd* (*?ys'* @ *?xs'*)›
using *fst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*lst-As'* = *last* (*?ys'* @ *?xs'*)›
using *lst-As* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹*next-search'* = *option-hd ?xs'*›
by *auto*
moreover {
have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set ?xs'*, *set ?ys'*), *remove*)›
using *abs-vmtf dist* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def ys'-y-x*
by *auto*
then have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) ((*set ?xs'*, *set ?ys'*), *remove*)›
using *dist L-y-ys* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def ys'-y-x ys'-y-x*
by (*cases M*) *auto*
}
moreover have ‹*vmtf-ns-notin* (*?ys'* @ *?xs'*) *m ns*›
using *notin* **unfolding** *ys'-y-x* **by** *simp*
moreover have ‹∀ *L*∈*set* (*?ys'* @ *?xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *ys'-xs'*-$\mathcal{L}_{all}$ **unfolding** *ys'-y-x* **by** *simp*
ultimately show *?thesis*
using *S False atm-A* **unfolding** *vmtf-unset-def L-ys'-xs'*[*symmetric*]
by (*fastforce simp add*: *C*)
qed
then show *?thesis*
unfolding *vmtf-def S*
by *fast*
qed

definition *vmtf-mark-to-rescore-and-unset* :: ‹*nat* ⇒ *vmtf-remove-int* ⇒ *vmtf-remove-int*› **where**
‹*vmtf-mark-to-rescore-and-unset L M* = *vmtf-mark-to-rescore L* (*vmtf-unset L M*)›

lemma *vmtf-append-remove-iff*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *insert L b*) ∈ *vmtf* $\mathcal{A}$ *M* ⟷
L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧ ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *b*) ∈ *vmtf* $\mathcal{A}$ *M*›
(**is** ‹*?A* ⟷ *?L* ∧ *?B*›)
proof
assume *vmtf*: *?A*

obtain *xs'* *ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *insert L b*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *b*)› **and** *L*: *?L*
using *abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def* **by** *auto*
ultimately have ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns* ∧
fst-As = *hd* (*ys'* @ *xs'*) ∧
next-search = *option-hd xs'* ∧
lst-As = *last* (*ys'* @ *xs'*) ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *b*) ∧
vmtf-ns-notin (*ys'* @ *xs'*) *m ns* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧
(∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show ‹*?L* ∧ *?B*›
using *L* **unfolding** *vmtf-def* **by** *fast*
next
assume *vmtf*: ‹*?L* ∧ *?B*›
obtain *xs'* *ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *b*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
moreover have ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *insert L b*)›
using *vmtf abs-vmtf* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def* **by** *auto*
ultimately have ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns* ∧
fst-As = *hd* (*ys'* @ *xs'*) ∧
next-search = *option-hd xs'* ∧
lst-As = *last* (*ys'* @ *xs'*) ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *insert L b*) ∧
vmtf-ns-notin (*ys'* @ *xs'*) *m ns* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧
(∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *fast*
then show *?A*
unfolding *vmtf-def* **by** *fast*
qed

lemma *vmtf-append-remove-iff'*:
‹(*vm*, *insert L b*) ∈ *vmtf* $\mathcal{A}$ *M* ⟷
L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧ (*vm*, *b*) ∈ *vmtf* $\mathcal{A}$ *M*›
by (*cases vm*) (*auto simp*: *vmtf-append-remove-iff*)

lemma *vmtf-mark-to-rescore-unset*:
fixes *M*
defines [*simp*]: ‹*L* ≡ *atm-of* (*lit-of* (*hd M*))›

assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* ≠ []›
shows ‹(*vmtf-mark-to-rescore-and-unset L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)) ∈ *vmtf* $\mathcal{A}$ (*tl M*)›
(**is** ‹*?S* ∈ -›)
using *vmtf-unset-vmtf-tl*[*OF assms*(*2*−)[*unfolded assms*(*1*)]] *L-N*
unfolding *vmtf-mark-to-rescore-and-unset-def vmtf-mark-to-rescore-def*
by (*cases* ‹*vmtf-unset* (*atm-of* (*lit-of* (*hd M*))) ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)›)
(*auto simp*: *vmtf-append-remove-iff*)

lemma *vmtf-insert-sort-nth-code-preD*:
assumes *vmtf*: ‹*vm* ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹∀ *x*∈*snd vm*. *x* < *length* (*fst* (*fst vm*))›
proof −
obtain *ns m fst-As lst-As next-search remove* **where**
vm: ‹*vm* = ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*)›
by (*cases vm*) *auto*

obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*), *remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def vm* **by** *fast*
show *?thesis*
using *atm-A abs-vmtf* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto simp*: *vm*)
qed

lemma *vmtf-ns-Cons*:
assumes
vmtf: ‹*vmtf-ns* (*b* # *l*) *m xs*› **and**
a-xs: ‹*a* < *length xs*› **and**
ab: ‹*a* ≠ *b*› **and**
a-l: ‹*a* ∉ *set l*› **and**
nm: ‹*n* > *m*› **and**
xs′: ‹*xs′* = *xs*[*a* := *VMTF-Node n None* (*Some b*),
b := *VMTF-Node* (*stamp* (*xs*!*b*)) (*Some a*) (*get-next* (*xs*!*b*))]› **and**
nn′: ‹*n′* ≥ *n*›
shows ‹*vmtf-ns* (*a* # *b* # *l*) *n′ xs′*›
proof −
have ‹*vmtf-ns* (*b* # *l*) *m* (*xs*[*a* := *VMTF-Node n None* (*Some b*)])›
apply (*rule vmtf-ns-eq-iffI*[*OF* - - *vmtf*])
subgoal using *ab a-l a-xs* **by** *auto*
subgoal using *a-xs vmtf-ns-le-length*[*OF vmtf*] **by** *auto*
done
then show *?thesis*
apply (*rule vmtf-ns.Cons*[*of* - - - - - *n*])
subgoal using *a-xs* **by** *simp*
subgoal using *a-xs* **by** *simp*
subgoal using *ab* **.**

subgoal using *a-l* **.**
subgoal using *nm* **.**
subgoal using *xs′ ab a-xs* **by** (*cases* ‹*xs* ! *b*›) *auto*
subgoal using *nn′* **.**
done
qed

definition (**in** −) *vmtf-cons* **where**
‹*vmtf-cons ns L cnext st* =
(*let*
ns = *ns*[*L* := *VMTF-Node* (*Suc st*) *None cnext*];
ns = (*case cnext of None* ⇒ *ns*
| *Some cnext* ⇒ *ns*[*cnext* := *VMTF-Node* (*stamp* (*ns*!*cnext*)) (*Some L*) (*get-next* (*ns*!*cnext*))]) *in*
ns)
›

lemma *vmtf-notin-vmtf-cons*:
assumes
vmtf-ns: ‹*vmtf-ns-notin xs m ns*› **and**
cnext: ‹*cnext* = *option-hd xs*› **and**
L-xs: ‹*L* ∉ *set xs*›
shows
‹*vmtf-ns-notin* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
case *Nil*
then show *?thesis*
using *assms* **by** (*auto simp*: *vmtf-ns-notin-def vmtf-cons-def elim*: *vmtf-nsE*)
next
case (*Cons L′ xs′*) **note** *xs* = *this*
show *?thesis*
using *assms* **unfolding** *xs vmtf-ns-notin-def xs vmtf-cons-def* **by** *auto*
qed

lemma *vmtf-cons*:
assumes
vmtf-ns: ‹*vmtf-ns xs m ns*› **and**
cnext: ‹*cnext* = *option-hd xs*› **and**
L-A: ‹*L* < *length ns*› **and**
L-xs: ‹*L* ∉ *set xs*›
shows
‹*vmtf-ns* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
case *Nil*
then show *?thesis*
using *assms* **by** (*auto simp*: *vmtf-ns-single-iff vmtf-cons-def elim*: *vmtf-nsE*)
next
case (*Cons L′ xs′*) **note** *xs* = *this*
show *?thesis*
unfolding *xs*
apply (*rule vmtf-ns-Cons*[*OF vmtf-ns*[*unfolded xs*], *of* - ‹*Suc m*›])
subgoal using *L-A* **.**
subgoal using *L-xs* **unfolding** *xs* **by** *simp*
subgoal using *L-xs* **unfolding** *xs* **by** *simp*
subgoal by *simp*
subgoal using *cnext L-xs*
by (*auto simp*: *vmtf-cons-def Let-def xs*)

subgoal by *linarith*
done
qed

lemma *length-vmtf-cons*[*simp*]: ‹*length* (*vmtf-cons ns L n m*) = *length ns*›
by (*auto simp*: *vmtf-cons-def Let-def split*: *option.splits*)

lemma *wf-vmtf-get-prev*:
assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) $\in$ *vmtf* $\mathcal{A}$ *M*›
shows ‹*wf* {(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* $\neq$ *None* $\wedge$ *the a* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}› (**is** ‹*wf ?R*›)
proof (*rule ccontr*)
assume ‹$\neg$ *?thesis*›
then obtain *f* **where**
f: ‹(*f* (*Suc i*), *f i*) $\in$ *?R*› **for** *i*
unfolding *wf-iff-no-infinite-down-chain* **by** *blast*

obtain *xs' ys'* **where**
vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹$\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?f0* = ‹*the* (*f 0*)›
have *f-None*: ‹*f i* $\neq$ *None*› **for** *i*
using *f*[*of i*] **by** *fast*
have *f-Suc* : ‹*f* (*Suc n*) = *get-prev* (*ns* ! *the* (*f n*))› **for** *n*
using *f*[*of n*] **by** *auto*
have *f0-length*: ‹*?f0* < *length ns*›
using *f*[*of 0*] *atm-A*
by *auto*
have *f0-in*: ‹*?f0* $\in$ *set* (*ys'* @ *xs'*)›
apply (*rule ccontr*)
using *notin f-Suc*[*of 0*] *f0-length* **unfolding** *vmtf-ns-notin-def*
by (*auto simp*: *f-None*)
then obtain *i0* **where**
i0: ‹(*ys'* @ *xs'*) ! *i0* = *?f0*› ‹*i0* < *length* (*ys'* @ *xs'*)›
by (*meson in-set-conv-nth*)
define *zs* **where** ‹*zs* = *ys'* @ *xs'*›
have *H*: ‹*ys'* @ *xs'* = *take m* (*ys'* @ *xs'*) @ [(*ys'* @ *xs'*) ! *m*, (*ys'* @ *xs'*) ! (*m*+*1*)] @
drop (*m*+*2*) (*ys'* @ *xs'*)›
if ‹*m* + *1* < *length* (*ys'* @ *xs'*)›
for *m*
using *that*
unfolding *zs-def*[*symmetric*]
apply −
apply (*subst id-take-nth-drop*[*of m*])
by (*auto simp*: *take-Suc-conv-app-nth Cons-nth-drop-Suc simp del*: *append-take-drop-id*)

have ‹*the* (*f n*) = (*ys'* @ *xs'*) ! (*i0* − *n*) $\wedge$ *i0* − *n* $\geq$ *0* $\wedge$ *n* $\leq$ *i0*› **for** *n*
proof (*induction n*)
case *0*
then show *?case* **using** *i0* **by** *simp*
next

```
    case (Suc n')
    have i0-le: ‹n' < i0›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have ‹i0 = n'›
        using Suc by auto
      then have ‹ys' @ xs' = [the (f n')] @ tl (ys' @ xs')›
        using Suc f0-in
        by (cases ‹ys' @ xs'›) auto
      then show False
        using vmtf-ns-hd-prev[of ‹the (f n')› ‹tl (ys' @ xs')› m ns] vmtf-ns
         f-Suc[of n'] by (auto simp: f-None)
    qed
    have get-prev: ‹get-prev (ns ! ((ys' @ xs') ! (i0 − (n' +1) + 1))) =
         Some ((ys' @ xs') ! ((i0 − (n' + 1))))›
      apply (rule vmtf-ns-last-mid-get-prev[of ‹take (i0 − (n' +1)) (ys' @ xs')› - -
        ‹drop ((i0 − (n' + 1)) + 2) (ys' @ xs')› m])
      apply (subst H[symmetric])
      subgoal using i0-le i0 by auto
      subgoal using vmtf-ns by simp
      done
    then show ?case
      using f-Suc[of n'] Suc i0-le by auto
  qed
  from this[of ‹Suc i0›] show False
    by auto
qed

fun update-stamp where
  ‹update-stamp xs n a = xs[a := VMTF-Node n (get-prev (xs!a)) (get-next (xs!a))]›

definition vmtf-rescale :: ‹vmtf ⇒ vmtf nres› where
‹vmtf-rescale = (λ(ns, m, fst-As, lst-As :: nat, next-search). do {
  (ns, m, -) ← WHILE_T^{λ-. True}
     (λ(ns, n, lst-As). lst-As ≠None)
     (λ(ns, n, a). do {
       ASSERT(a ≠ None);
       ASSERT(n+1 ≤ uint32-max);
       ASSERT(the a < length ns);
       RETURN (update-stamp ns n (the a), n+1, get-prev (ns ! the a))
     })
     (ns, 0, Some lst-As);
  RETURN ((ns, m, fst-As, lst-As, next-search))
  })
›


lemma vmtf-rescale-vmtf:
  assumes vmtf: ‹(vm, to-remove) ∈ vmtf 𝒜 M› and
    nempty: ‹isasat-input-nempty 𝒜› and
    bounded: ‹isasat-input-bounded 𝒜›
  shows
    ‹vmtf-rescale vm ≤ SPEC (λvm. (vm, to-remove) ∈ vmtf 𝒜 M ∧ fst (snd vm) ≤ uint32-max)›
    (is ‹?A ≤ ?R›)
proof −
  obtain ns m fst-As lst-As next-search where
```

vm: ⟨$vm = ((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search))$⟩
by (*cases vm*) *auto*

obtain xs' ys' **where**
vmtf-ns: ⟨*vmtf-ns* (ys' @ xs') *m ns*⟩ **and**
fst-As: ⟨*fst-As* = *hd* (ys' @ xs')⟩ **and**
lst-As: ⟨*lst-As* = *last* (ys' @ xs')⟩ **and**
next-search: ⟨*next-search* = *option-hd* xs'⟩ **and**
abs-vmtf: ⟨*vmtf*-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set* xs', *set* ys'), *to-remove*)⟩ **and**
notin: ⟨*vmtf-ns-notin* (ys' @ xs') *m ns*⟩ **and**
atm-A: ⟨$\forall$ *L*$\in$*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*⟩ **and**
in-lall: ⟨$\forall$ *L*$\in$*set* (ys' @ xs'). *L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
using *vmtf* **unfolding** *vmtf-def vm* **by** *fast*
have [*dest*]: ⟨$ys' = [] \Longrightarrow xs' = [] \Longrightarrow False$⟩ **and**
[*simp*]: ⟨$ys' = [] \longrightarrow xs' \neq []$⟩
using *abs-vmtf nempty* **unfolding** *vmtf*-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
have *1*: ⟨*RES* (*vmtf* $\mathcal{A}$ *M*) = *do* {
a ← *RETURN* ();
RES (*vmtf* $\mathcal{A}$ *M*)
}⟩
by *auto*
define *zs* **where** ⟨*zs* ≡ ys' @ xs'⟩

define I' **where**
⟨$I' \equiv \lambda(ns',\ n::nat,\ lst::nat\ option).$
$map\ get\text{-}prev\ ns = map\ get\text{-}prev\ ns' \wedge$
$map\ get\text{-}next\ ns = map\ get\text{-}next\ ns' \wedge$
$(\forall\, i<n.\ stamp\ (ns'\ !\ (rev\ zs\ !\ i)) = i) \wedge$
$(lst \neq None \longrightarrow n < length\ (zs) \wedge the\ lst = zs\ !\ (length\ zs - Suc\ n)) \wedge$
$(lst = None \longrightarrow n = length\ zs) \wedge$
$n \leq length\ zs$⟩
have [*simp*]: ⟨$zs \neq []$⟩
unfolding *zs-def* **by** *auto*
have $I'0$: ⟨I' (*ns*, *0*, *Some lst-As*)⟩
using *vmtf lst-As* **unfolding** I'-*def vm zs-def*[*symmetric*] **by** (*auto simp*: *last-conv-nth*)

have *lits*: ⟨*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* ‘# *mset zs*)⟩ **and**
dist: ⟨*distinct zs*⟩
using *abs-vmtf vmtf-ns-distinct*[*OF vmtf-ns*] **unfolding** *vmtf-def zs-def*
vmtf-$\mathcal{L}_{all}$-*def*
by (*auto simp*: *literals-are-in*-$\mathcal{L}_{in}$-*alt-def inj-on-def*)
have *dist*: ⟨*distinct-mset* (*Pos* ‘# *mset zs*)⟩
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ⟨*auto simp*: *inj-on-def*⟩)
have *tauto*: ⟨¬ *tautology* (*poss* (*mset zs*))⟩
by (*auto simp*: *tautology-decomp*)

have *length-zs-le*: ⟨*length zs* < *uint32-max*⟩ **using** *vmtf-ns-distinct*[*OF vmtf-ns*]
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)

have ⟨*wf* {(*a*, *b*). (*a*, *b*) ∈ {(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧ *the a* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}}⟩
by (*rule wf-subset*[*OF wf-vmtf-get-prev*[*OF vmtf*[*unfolded vm*]]]) *auto*
from *wf-snd-wf-pair*[*OF wf-snd-wf-pair*[*OF this*]]

```
have wf: ‹wf {((-, -, a), (-, -, b)). (a, b) ∈ {(get-prev (ns ! the a), a) |a. a ≠ None ∧
    the a ∈ atms-of (ℒall 𝒜)}}›
  by (rule wf-subset) auto
have zs-lall: ‹zs ! (length zs − Suc n) ∈ atms-of (ℒall 𝒜)› for n
  using abs-vmtf nth-mem[of ‹length zs − Suc n› zs] unfolding zs-def vmtf-ℒall-def
  by auto
then have zs-le-ns[simp]: ‹zs ! (length zs − Suc n) < length ns› for n
  using atm-A by auto
have loop-body: ‹(case s' of
      (ns, n, a) ⇒ do {
          ASSERT (a ≠ None);
          ASSERT (n + 1 ≤ uint32-max);
          ASSERT(the a < length ns);
          RETURN (update-stamp ns n (the a), n + 1, get-prev (ns ! the a))
        })
      ≤ SPEC
        (λs'a. True ∧
              I' s'a ∧
              (s'a, s')
              ∈ {((-, -, a), -, -, b).
                (a, b)
                ∈ {(get-prev (ns ! the a), a) |a.
                    a ≠ None ∧ the a ∈ atms-of (ℒall 𝒜)}})›
  if
    I': ‹I' s'› and
    cond: ‹case s' of (ns, n, lst-As) ⇒ lst-As ≠ None›
  for s'
proof −
  obtain ns' n' a' where s': ‹s' = (ns', n' , a')›
    by (cases s')
  have
    a[simp]: ‹a' = Some (zs ! (length zs − Suc n'))› and
    eq-prev: ‹map get-prev ns = map get-prev ns'› and
    eq-next: ‹map get-next ns = map get-next ns'› and
    eq-stamps: ‹⋀i. i<n' ⟹ stamp (ns' ! (rev zs ! i)) = i› and
    n'-le: ‹n' < length zs›
    using I' cond unfolding I'-def prod.simps s'
    by auto
  have [simp]: ‹length ns' = length ns›
    using arg-cong[OF eq-prev, of length] by auto
  have vmtf-as: ‹vmtf-ns
    (take (length zs − (n' + 1)) zs @
     zs ! (length zs − (n' + 1)) #
     drop (Suc (length zs − (n' + 1))) zs)
    m ns›
    apply (subst Cons-nth-drop-Suc)
    subgoal by auto
    apply (subst append-take-drop-id)
    using vmtf-ns unfolding zs-def[symmetric] .

  have ‹get-prev (ns' ! the a') ≠ None ⟶
      n' + 1 < length zs ∧
      the (get-prev (ns' ! the a')) = zs ! (length zs − Suc (n' + 1))›
    using n'-le vmtf-ns arg-cong[OF eq-prev, of ‹λxs. xs ! (zs ! (length zs − Suc n'))›]
      vmtf-ns-last-mid-get-prev-option-last[OF vmtf-as]
    by (auto simp: last-conv-nth)
```

```
  moreover have ‹map get-prev ns = map get-prev (update-stamp ns' n' (the a'))›
    unfolding update-stamp.simps
    apply (subst map-update)
    apply (subst list-update-id')
    subgoal by auto
    subgoal using eq-prev .
    done
  moreover have ‹map get-next ns = map get-next (update-stamp ns' n' (the a'))›
    unfolding update-stamp.simps
    apply (subst map-update)
    apply (subst list-update-id')
    subgoal by auto
    subgoal using eq-next .
    done
  moreover have ‹i<n' + 1 ⟹ stamp (update-stamp ns' n' (the a') ! (rev zs ! i)) = i› for i
    using eq-stamps[of i] vmtf-ns-distinct[OF vmtf-ns] n'-le
    unfolding zs-def[symmetric]
    by (cases ‹i < n'›)
      (auto simp: rev-nth nth-eq-iff-index-eq)
  moreover have ‹n' + 1 ≤ length zs›
   using n'-le by (auto simp: Suc-le-eq)
  moreover have ‹get-prev (ns' ! the a') = None ⟹ n' + 1 = length zs›
    using n'-le vmtf-ns arg-cong[OF eq-prev, of ‹λxs. xs ! (zs ! (length zs − Suc n'))›]
      vmtf-ns-last-mid-get-prev-option-last[OF vmtf-as]
    by auto
  ultimately have I'-f: ‹I' (update-stamp ns' n' (the a'), n' + 1, get-prev (ns' ! the a'))›
    using cond n'-le unfolding I'-def prod.simps s'
    by simp

  show ?thesis
    unfolding s' prod.case
    apply refine-vcg
    subgoal using cond by auto
    subgoal using length-zs-le n'-le by auto
    subgoal by auto
    subgoal by fast
    subgoal by (rule I'-f)
    subgoal
      using arg-cong[OF eq-prev, of ‹λxs. xs ! (zs ! (length zs − Suc n'))›] zs-lall
      by auto
    done
qed
have loop-final: ‹s ∈ {x. (case x of
            (ns, m, uua-) ⇒
              RETURN ((ns, m, fst-As, lst-As, next-search)))
            ≤ ?R}›
  if
    ‹True› and
    ‹I' s› and
    ‹¬ (case s of (ns, n, lst-As) ⇒ lst-As ≠ None)›
  for s
proof −
  obtain ns' n' a' where s: ‹s = (ns', n' , a')›
    by (cases s)
  have
    [simp]:‹a' = None› and
```

eq-prev: ‹*map get-prev ns* = *map get-prev ns'*› **and**
eq-next: ‹*map get-next ns* = *map get-next ns'*› **and**
stamp: ‹∀ *i*<*n'*. *stamp* (*ns'* ! (*rev zs* ! *i*)) = *i*› **and**
[*simp*]: ‹*n'* = *length zs*›
using *that* **unfolding** *I'-def s prod.case* **by** *auto*
have [*simp*]: ‹*length ns'* = *length ns*›
using *arg-cong*[*OF eq-prev*, *of length*] **by** *auto*
have [*simp*]: ‹*map* ((!) (*map stamp ns'*)) (*rev zs*) = [*0*..<*length zs*]›
apply (*subst list-eq-iff-nth-eq*, *intro conjI*)
subgoal by *auto*
subgoal using *stamp* **by** (*auto simp*: *rev-nth*)
done
then have *stamps-zs*[*simp*]: ‹*map* ((!) (*map stamp ns'*)) *zs* = *rev* [*0*..<*length zs*]›
unfolding *rev-map*[*symmetric*]
using *rev-swap* **by** *blast*

have ‹*sorted* (*map* ((!) (*map stamp ns'*)) (*rev zs*))›
by *simp*
moreover have ‹*distinct* (*map* ((!) (*map stamp ns'*)) *zs*)›
by *simp*
moreover have ‹∀ *a*∈*set zs*. *get-prev* (*ns'* ! *a*) = *get-prev* (*ns* ! *a*)›
using *eq-prev map-eq-nth-eq* **by** *fastforce*
moreover have ‹∀ *a*∈*set zs*. *get-next* (*ns'* ! *a*) = *get-next* (*ns* ! *a*)›
using *eq-next map-eq-nth-eq* **by** *fastforce*
moreover have ‹∀ *a*∈*set zs*. *stamp* (*ns'* ! *a*) = *map stamp ns'* ! *a*›
using *vmtf-ns vmtf-ns-le-length zs-def* **by** *auto*
moreover have ‹*length ns* ≤ *length ns'*›
by *simp*
moreover have ‹∀ *a*∈*set zs*. *a* < *length* (*map stamp ns'*)›
using *vmtf-ns vmtf-ns-le-length zs-def* **by** *auto*
moreover have ‹∀ *a*∈*set zs*. *map stamp ns'* ! *a* < *n'*›
proof
fix *a*
assume ‹*a* ∈ *set zs*›
then have ‹*map stamp ns'* ! *a* ∈ *set* (*map* ((!) (*map stamp ns'*)) *zs*)›
by (*metis in-set-conv-nth length-map nth-map*)
then show ‹*map stamp ns'* ! *a* < *n'*›
unfolding *stamps-zs* **by** *simp*
qed
ultimately have ‹*vmtf-ns zs n' ns'*›
using *vmtf-ns-rescale*[*OF vmtf-ns*, *of* ‹*map stamp ns'*› *ns'*, *unfolded zs-def*[*symmetric*]]
by *fast*
moreover have ‹*vmtf-ns-notin zs* (*length zs*) *ns'*›
using *notin map-eq-nth-eq*[*OF eq-prev*] *map-eq-nth-eq*[*OF eq-next*]
unfolding *zs-def*[*symmetric*]
by (*auto simp*: *vmtf-ns-notin-def*)
ultimately have ‹((*ns'*, *n'*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* 𝒜 *M*›
using *fst-As lst-As next-search abs-vmtf atm-A notin in-lall*
unfolding *vmtf-def in-pair-collect-simp prod.case* **apply** −
apply (*rule exI*[*of - xs'*])
apply (*rule exI*[*of - ys'*])
unfolding *zs-def*[*symmetric*]
by *auto*
then show *?thesis*
using *length-zs-le*
by (*auto simp*: *s*)

qed

have H: ‹$WHILE_T{}^{\lambda\text{-}.\ True}$ (λ(*ns*, *n*, *lst-As*). *lst-As* $\neq$ *None*)
(λ(*ns*, *n*, *a*). *do* {
- $\leftarrow$ *ASSERT* (*a* $\neq$ *None*);
- $\leftarrow$ *ASSERT* (*n* + *1* $\leq$ *uint32-max*);
ASSERT(*the a* < *length ns*);
RETURN (*update-stamp ns n* (*the a*), *n* + *1*, *get-prev* (*ns* ! *the a*))
})
(*ns*, *0*, *Some lst-As*)
$\leq$ *SPEC*
(λx. (*case x of*
(*ns*, *m*, *uua-*) $\Rightarrow$
RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)))
$\leq$ *?R*)
›
apply (*rule WHILEIT-rule-stronger-inv-RES*[**where** I' = I' **and**
R = ‹{((-, -, *a*), (-, -, *b*)). (*a*, *b*) $\in$
{(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* $\neq$ *None* $\wedge$ *the a* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}}›])
subgoal
by (*rule wf*)
subgoal by *fast*
subgoal by (*rule* $I'0$)
subgoal for s'
by (*rule loop-body*)
subgoal for *s*
by (*rule loop-final*)
done

show *?thesis*
unfolding *vmtf-rescale-def vm prod.case*
apply (*subst bind-rule-complete-RES*)
apply (*rule H*)
done
qed

definition *vmtf-flush*
:: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*) *ann-lits* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$ *vmtf-remove-int nres*›
where
‹*vmtf-flush* $\mathcal{A}_{in}$ = (λM (*vm*, *to-remove*). *RES* (*vmtf* $\mathcal{A}_{in}$ *M*))›

definition *atoms-hash-rel* :: ‹*nat multiset* $\Rightarrow$ (*bool list* $\times$ *nat set*) *set*› **where**
‹*atoms-hash-rel* $\mathcal{A}$ = {(*C*, *D*). ($\forall L \in D$. *L* < *length C*) $\wedge$ ($\forall L$ < *length C*. *C* ! *L* $\longleftrightarrow$ $L \in D$) $\wedge$
($\forall L \in\# \mathcal{A}$. *L* < *length C*) $\wedge$ $D \subseteq$ *set-mset* $\mathcal{A}$}›

definition *distinct-hash-atoms-rel*
:: ‹*nat multiset* $\Rightarrow$ (('*v list* $\times$ '*v set*) $\times$ '*v set*) *set*›
where
‹*distinct-hash-atoms-rel* $\mathcal{A}$ = {((*C*, *h*), *D*). *set C* = *D* $\wedge$ *h* = *D* $\wedge$ *distinct C*}›

definition *distinct-atoms-rel*
:: ‹*nat multiset* $\Rightarrow$ ((*nat list* $\times$ *bool list*) $\times$ *nat set*) *set*›
where
‹*distinct-atoms-rel* $\mathcal{A}$ = (*Id* $\times_r$ *atoms-hash-rel* $\mathcal{A}$) *O distinct-hash-atoms-rel* $\mathcal{A}$›

lemma *distinct-atoms-rel-alt-def*:

‹$distinct\text{-}atoms\text{-}rel\ \mathcal{A} = \{((D', C), D).\ (\forall L \in D.\ L < length\ C) \wedge (\forall L < length\ C.\ C\ !\ L \longleftrightarrow L \in D) \wedge$
$(\forall L \in\#\ \mathcal{A}.\ L < length\ C) \wedge set\ D' = D \wedge distinct\ D' \wedge set\ D' \subseteq set\text{-}mset\ \mathcal{A}\}$›
unfolding *distinct-atoms-rel-def atoms-hash-rel-def distinct-hash-atoms-rel-def prod-rel-def*
apply *rule*
subgoal
by (*auto simp*: *mset-set-set*)
subgoal
by (*auto simp*: *mset-set-set*)
done

lemma *distinct-atoms-rel-empty-hash-iff*:
‹$(([], h), \{\}) \in distinct\text{-}atoms\text{-}rel\ \mathcal{A} \longleftrightarrow (\forall L \in\#\ \mathcal{A}.\ L < length\ h) \wedge (\forall i{\in}set\ h.\ i = False)$›
unfolding *distinct-atoms-rel-alt-def all-set-conv-nth*
by *auto*

definition *atoms-hash-del-pre* **where**
‹$atoms\text{-}hash\text{-}del\text{-}pre\ i\ xs = (i < length\ xs)$›

definition *atoms-hash-del* **where**
‹$atoms\text{-}hash\text{-}del\ i\ xs = xs[i := False]$›

definition *vmtf-flush-int* :: ‹$nat\ multiset \Rightarrow (nat,nat)\ ann\text{-}lits \Rightarrow \text{-} \Rightarrow \text{-}\ nres$› **where**
‹$vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in} = (\lambda M\ (vm, (to\text{-}remove, h)).\ do\ \{$
$ASSERT(\forall x{\in}set\ to\text{-}remove.\ x < length\ (fst\ vm));$
$ASSERT(length\ to\text{-}remove \leq uint32\text{-}max);$
$to\text{-}remove' \leftarrow reorder\text{-}list\ vm\ to\text{-}remove;$
$ASSERT(length\ to\text{-}remove' \leq uint32\text{-}max);$
$vm \leftarrow (if\ length\ to\text{-}remove' + fst\ (snd\ vm) \geq uint64\text{-}max$
$then\ vmtf\text{-}rescale\ vm\ else\ RETURN\ vm);$
$ASSERT(length\ to\text{-}remove' + fst\ (snd\ vm) \leq uint64\text{-}max);$
$(\text{-}, vm, h) \leftarrow WHILE_T{}^{\lambda(i,\ vm',\ h).\ i \leq length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge \quad (i < length\ to\text{-}remove}$
$(\lambda(i, vm, h).\ i < length\ to\text{-}remove')$
$(\lambda(i, vm, h).\ do\ \{$
$ASSERT(i < length\ to\text{-}remove');$
$ASSERT(to\text{-}remove'!i \in\#\ \mathcal{A}_{in});$
$ASSERT(atoms\text{-}hash\text{-}del\text{-}pre\ (to\text{-}remove'!i)\ h);$
$RETURN\ (i{+}1, vmtf\text{-}en\text{-}dequeue\ M\ (to\text{-}remove'!i)\ vm, atoms\text{-}hash\text{-}del\ (to\text{-}remove'!i)\ h)\})$
$(0, vm, h);$
$RETURN\ (vm, (emptied\text{-}list\ to\text{-}remove', h))$
$\})$›

lemma *vmtf-change-to-remove-order*:
assumes
vmtf: ‹$((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove) \in vmtf\ \mathcal{A}_{in}\ M$› **and**
CD-rem: ‹$(((C, D), to\text{-}remove) \in distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}$› **and**
nempty: ‹$isasat\text{-}input\text{-}nempty\ \mathcal{A}_{in}$› **and**
bounded: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}$›
shows ‹$vmtf\text{-}flush\text{-}int\ \mathcal{A}_{in}\ M\ ((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), (C, D))$
$\leq \Downarrow(Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in})$
$(vmtf\text{-}flush\ \mathcal{A}_{in}\ M\ ((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove))$›
proof −
let $?vm$ = ‹$((ns, m, fst\text{-}As, lst\text{-}As, next\text{-}search), to\text{-}remove)$›

have *vmtf-flush-alt-def*: ⟨*vmtf-flush* $\mathcal{A}_{in}$ *M ?vm* = *do* {
  - ← *RETURN* ();
  - ← *RETURN* ();
  vm ← *RES*(*vmtf* $\mathcal{A}_{in}$ *M*);
  RETURN (*vm*)
}⟩
  unfolding *vmtf-flush-def* **by** (*auto simp*: *RES-RES-RETURN-RES RES-RETURN-RES vmtf*)

have *pre-sort*: ⟨∀ *x*∈*set x1a*. *x* < *length* (*fst x1*)⟩
  if
    ⟨*x2* = (*x1a*, *x2a*)⟩ **and**
    ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)⟩
  for *x1 x2 x1a x2a*
proof −
  show *?thesis*
    using *vmtf CD-rem that* **by** (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*
      distinct-atoms-rel-alt-def)
qed

have *length-le*: ⟨*length x1a* ≤ *uint32-max*⟩
  if
    ⟨*x2* = (*x1a*, *x2a*)⟩ **and**
    ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)⟩ **and**
    ⟨∀ *x*∈*set x1a*. *x* < *length* (*fst x1*)⟩
    for *x1 x2 x1a x2a*
proof −
  have *lits*: ⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*Pos* '# *mset x1a*)⟩ **and**
    dist: ⟨*distinct x1a*⟩
    using *that vmtf CD-rem* **unfolding** *vmtf-def*
      vmtf-$\mathcal{L}_{all}$*-def*
    by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def distinct-atoms-rel-alt-def inj-on-def*)
  have *dist*: ⟨*distinct-mset* (*Pos* '# *mset x1a*)⟩
    by (*subst distinct-image-mset-inj*)
      (*use dist* **in** ⟨*auto simp*: *inj-on-def*⟩)
  have *tauto*: ⟨¬ *tautology* (*poss* (*mset x1a*))⟩
    by (*auto simp*: *tautology-decomp*)

  show *?thesis*
    using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
    by (*auto simp*: *uint32-max-def*)
qed

have [*refine0*]:
  ⟨*reorder-list x1 x1a* ≤ *SPEC* (λ*c*. (*c*, ()) ∈
    {(*c*, *c'*). ((*c*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$ ∧ *to-remove* = *set c* ∧
      length C = *length c*})⟩
  (**is** ⟨- ≤ *SPEC*(λ-. - ∈ *?reorder-list*)⟩)
  if
    ⟨*x2* = (*x1a*, *x2a*)⟩ **and**
    ⟨((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *C*, *D*) = (*x1*, *x2*)⟩
  for *x1 x2 x1a x2a*
proof −
  show *?thesis*
    using *that assms* **by** (*force simp*: *reorder-list-def distinct-atoms-rel-alt-def*
      dest: *mset-eq-setD same-mset-distinct-iff mset-eq-length*)

```
qed


have [refine0]: ‹(if uint64-max ≤ length to-remove' + fst (snd x1) then vmtf-rescale x1
    else RETURN x1)
    ≤ SPEC (λc. (c, ()) ∈
      {(vm ,vm'). uint64-max ≥ length to-remove' + fst (snd vm) ∧
        (vm, set to-remove') ∈ vmtf 𝒜in M}) ›
  (is ‹- ≤ SPEC(λc. (c, ()) ∈ ?rescale)› is ‹- ≤ ?H›)
if
  ‹x2 = (x1a, x2a)› and
  ‹((ns, m, fst-As, lst-As, next-search), C, D) = (x1, x2)› and
  ‹∀ x∈set x1a. x < length (fst x1)› and
  ‹length x1a ≤ uint32-max› and
  ‹(to-remove', uu) ∈ ?reorder-list› and
  ‹length to-remove' ≤ uint32-max›
for x1 x2 x1a x2a to-remove' uu
proof –
  have ‹vmtf-rescale x1 ≤ ?H›
    apply (rule order-trans)
    apply (rule vmtf-rescale-vmtf[of - to-remove 𝒜in M])
    subgoal using vmtf that by auto
    subgoal using nempty by fast
    subgoal using bounded by fast
    subgoal using that by (auto intro!: RES-refine simp: uint64-max-def uint32-max-def)
    done
  then show ?thesis
    using that vmtf
    by (auto intro!: RETURN-RES-refine)
qed


have loop-ref: ‹WHILE_T^λ(i, vm', h).                i ≤ length to-remove' ∧ fst (snd vm') = i + fst (snd x1) ∧
      (λ(i, vm, h). i < length to-remove')
      (λ(i, vm, h). do {
          ASSERT (i < length to-remove');
          ASSERT(to-remove'!i ∈# 𝒜in);
          ASSERT(atoms-hash-del-pre (to-remove'!i) h);
          RETURN
            (i + 1, vmtf-en-dequeue M (to-remove' ! i) vm,
            atoms-hash-del (to-remove'!i) h)
        })
      (0, x1, x2a)
      ≤ ⇓ {((i, vm::vmtf, h:: -), vm'). (vm, {}) = vm' ∧ (∀ i∈set h. i = False) ∧ i = length to-remove'
∧
          ((drop i to-remove', h), set(drop i to-remove')) ∈ distinct-atoms-rel 𝒜in}
    (RES (vmtf 𝒜in M))›
  if
    x2: ‹x2 = (x1a, x2a)› and
    CD: ‹((ns, m, fst-As, lst-As, next-search), C, D) = (x1', x2)› and
    x1: ‹(x1, u') ∈ ?rescale to-remove'›
    ‹(to-remove', u) ∈ ?reorder-list›
  for x1 x2 x1a x2a to-remove' u u' x1'
proof –
  define I where ‹I ≡ λ(i, vm'::vmtf, h::bool list).
          i ≤ length to-remove' ∧ fst (snd vm') = i + fst (snd x1) ∧
```

```
            (i < length to-remove' ⟶
              vmtf-en-dequeue-pre 𝒜in ((M, to-remove' ! i), vm'))›
define I' where ‹I' ≡ λ(i, vm::vmtf, h:: bool list).
   ((drop i to-remove', h), set(drop i to-remove')) ∈ distinct-atoms-rel 𝒜in ∧
        (vm, set (drop i to-remove')) ∈ vmtf 𝒜in M›
have [simp]:
    ‹x2 = (C, D)›
    ‹x1' = (ns, m, fst-As, lst-As, next-search)›
    ‹x1a = C›
    ‹x2a = D› and
  rel: ‹((to-remove', D), to-remove) ∈ distinct-atoms-rel 𝒜in› and
  to-rem: ‹to-remove = set to-remove'›
  using that by (auto simp: )
have D: ‹set to-remove' = to-remove› and dist: ‹distinct to-remove'›
  using rel unfolding distinct-atoms-rel-alt-def by auto
have in-lall: ‹to-remove' ! x1 ∈ atms-of (ℒall 𝒜in)› if le': ‹x1 < length to-remove'› for x1
  using vmtf to-rem nth-mem[OF le'] by (auto simp: vmtf-def vmtf-ℒall-def)
have bound: ‹fst (snd x1) + 1 ≤ uint64-max› if ‹0 < length to-remove'›
    using rel vmtf to-rem that x1 by (cases to-remove') auto
have I-init: ‹I (0, x1, x2a)› (is ?A)
  for x1a x2 x1aa x2aa
proof −
  have ‹vmtf-en-dequeue-pre 𝒜in ((M, to-remove' ! 0), x1)› if ‹0 < length to-remove'›
    apply (rule vmtf-vmtf-en-dequeue-pre-to-remove'[of - ‹set to-remove'›])
    using rel vmtf to-rem that x1 bound nempty by (auto simp: )
  then show ?A
    unfolding I-def by auto
qed
have I'-init: ‹I' (0, x1, x2a)› (is ?B)
  for x1a x2 x1aa x2aa
proof −
  show ?B
    using rel to-rem CD-rem that vmtf by (auto simp: distinct-atoms-rel-def I'-def)
qed
have post-loop: ‹do {
        ASSERT (x2 < length to-remove');
        ASSERT(to-remove' ! x2 ∈# 𝒜in);
        ASSERT(atoms-hash-del-pre (to-remove' ! x2) x2a');
        RETURN
          (x2 + 1, vmtf-en-dequeue M (to-remove' ! x2) x2aa,
              atoms-hash-del (to-remove'!x2) x2a')
      } ≤ SPEC
          (λs'. I s' ∧ I' s' ∧ (s', x1a) ∈ measure (λ(i, vm, h). Suc (length to-remove') − i))›
  if
    I: ‹I x1a› and
    I': ‹I' x1a› and
    ‹case x1a of (i, vm, h) ⇒ i < length to-remove'› and
    x1aa: ‹x1aa = (x2aa, x2a')› ‹x1a = (x2, x1aa)›
  for s x2 x1a x2a x1a' x2a' x1aa x2aa
proof −
  let ?x2a' = ‹set (drop x2 to-remove')›
  have le: ‹x2 < length to-remove'› and vm: ‹(x2aa, set (drop x2 to-remove')) ∈ vmtf 𝒜in M› and
    x2a': ‹fst (snd x2aa) = x2 + fst (snd x1)›
    using that unfolding I-def I'-def by (auto simp: distinct-atoms-rel-alt-def)
  have 1: ‹(vmtf-en-dequeue M (to-remove' ! x2) x2aa, ?x2a'− {to-remove' ! x2}) ∈ vmtf 𝒜in M›
    by (rule abs-vmtf-ns-bump-vmtf-en-dequeue'[OF vm in-lall[OF le]])
```

(*use nempty* **in** *auto*)
have *2*: ‹*to-remove'* ! *Suc x2* ∈ *?x2a'*− {*to-remove'* ! *x2*}›
if ‹*Suc x2* < *length to-remove'*›
using *I I' le dist that x1aa* **unfolding** *I-def I'-def*
by (*auto simp*: *distinct-atoms-rel-alt-def in-set-drop-conv-nth I'-def*
nth-eq-iff-index-eq x2 intro: *bex-geI*[*of* - ‹*Suc x2*›])
have *3*: ‹*fst* (*snd x2aa*) = *fst* (*snd x1*) + *x2*›
using *I I' le dist that CD*[*unfolded x2*] *x2a'* **unfolding** *I-def I'-def x2 x2a' x1aa*
by (*auto simp*: *distinct-atoms-rel-def in-set-drop-conv-nth I'-def*
nth-eq-iff-index-eq x2 intro: *bex-geI*[*of* - ‹*Suc x2*›])
then have *4*: ‹*fst* (*snd* (*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*)) + *1* ≤ *uint64-max*›
if ‹*Suc x2* < *length to-remove'*›
using *x1 le that*
by (*cases x2aa*)
(*auto simp*: *vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def*
split: *option.splits*)
have *1*: ‹*vmtf-en-dequeue-pre* $\mathcal{A}_{in}$
((*M*, *to-remove'* ! *Suc x2*), *vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*)›
if ‹*Suc x2* < *length to-remove'*›
by (*rule vmtf-vmtf-en-dequeue-pre-to-remove'*)
(*rule 1*, *rule 2*, *rule that*, *rule 4*[*OF that*], *rule nempty*)
have *3*: ‹(*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*, *?x2a'* − {*to-remove'* ! *x2*}) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
by (*rule abs-vmtf-ns-bump-vmtf-en-dequeue'*[*OF vm in-lall*[*OF le*]]) (*use nempty* **in** *auto*)
have *4*: ‹((*drop* (*Suc x2*) *to-remove'*, *atoms-hash-del* (*to-remove'* ! *x2*) *x2a'*),
set (*drop* (*Suc x2*) *to-remove'*))
∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
3: ‹(*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*, *set* (*drop* (*Suc x2*) *to-remove'*))
∈ *vmtf* $\mathcal{A}_{in}$ *M*›
using *3 I' le to-rem that* **unfolding** *I'-def distinct-atoms-rel-alt-def atoms-hash-del-def*
by (*auto simp*: *Cons-nth-drop-Suc*[*symmetric*] *intro*: *mset-le-add-mset-decr-left1*)

have *A*: ‹*to-remove'* ! *x2* ∈ *?x2a'*›
using *I I' le dist that x1aa* **unfolding** *I-def I'-def*
by (*auto simp*: *distinct-atoms-rel-def in-set-drop-conv-nth I'-def*
nth-eq-iff-index-eq x2 x2a' intro: *bex-geI*[*of* - ‹*x2*›])
moreover have ‹*I* (*Suc x2*, *vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove'* ! *x2*) *x2a'*)›
using *that 1* **unfolding** *I-def*
by (*cases x2aa*)
(*auto simp*: *vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def*
split: *option.splits*)
moreover have ‹*length to-remove'* − *x2* < *Suc* (*length to-remove'*) − *x2*›
using *le* **by** *auto*
moreover have ‹*I'* (*Suc x2*, *vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove'* ! *x2*) *x2a'*)›
using *that 3 4 I'* **unfolding** *I'-def*
by *auto*
moreover have ‹*atoms-hash-del-pre* (*to-remove'* ! *x2*) *x2a'*›
unfolding *atoms-hash-del-pre-def*
using *that le A* **unfolding** *I-def I'-def* **by** (*auto simp*: *distinct-atoms-rel-alt-def*)
ultimately show *?thesis*
using *that in-lall*[*OF le*]
by (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
qed
have [*simp*]: ‹∀ *L*<*length ba*. ¬ *ba* ! *L* ⟹ *True* ∉ *set ba*› **for** *ba*
by (*simp add*: *in-set-conv-nth*)

have *post-rel*: ‹*RETURN s*
$\le \Downarrow \{((i, vm, h), vm').$
$(vm, \{\}) = vm' \wedge$
$(\forall i \in set\ h.\ i = False) \wedge$
$i = length\ to\text{-}remove' \wedge$
$((drop\ i\ to\text{-}remove', h), set\ (drop\ i\ to\text{-}remove'))$
$\in distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}\}\qquad (RES\ (vmtf\ \mathcal{A}_{in}\ M))$›
if
‹$\neg$ (*case s of* $(i, vm, h) \Rightarrow i < length\ to\text{-}remove'$)› **and**
‹*I s*› **and**
‹*I′ s*›
for *s*
proof −
obtain *i vm h* **where** *s*: ‹$s = (i, vm, h)$› **by** (*cases s*)
have [*simp*]: ‹$i = length\ (to\text{-}remove')$› **and** [*iff*]: ‹$True \notin set\ h$› **and**
[*simp*]: ‹$(([], h), \{\}) \in distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}$›
‹$(vm, \{\}) \in vmtf\ \mathcal{A}_{in}\ M$›
using *that* **unfolding** *s I-def I′-def* **by** (*auto simp*: *distinct-atoms-rel-empty-hash-iff*)
show *?thesis*
unfolding *s*
by (*rule RETURN-RES-refine*) *auto*
qed

show *?thesis*
unfolding *I-def*[*symmetric*]
apply (*refine-rcg*
WHILEIT-rule-stronger-inv-RES′[**where** *R*=‹*measure* (λ(*i*, *vm*::*vmtf*, *h*). *Suc* (*length to-remove′*)
−*i*)› **and**
I′=‹*I′*›])
subgoal by *auto*
subgoal by (*rule I-init*)
subgoal by (*rule I′-init*)
subgoal for *x1″ x2″ x1a″ x2a″* **by** (*rule post-loop*)
subgoal by (*rule post-rel*)
done
qed

show *?thesis*
unfolding *vmtf-flush-int-def vmtf-flush-alt-def*
apply (*refine-rcg*)
subgoal by (*rule pre-sort*)
subgoal by (*rule length-le*)
apply (*assumption+*)[*2*]
subgoal by *auto*
apply (*assumption+*)[*5*]
subgoal by *auto*
apply (*rule loop-ref*; *assumption*)
subgoal by (*auto simp*: *emptied-list-def*)
done
qed

lemma *vmtf-change-to-remove-order′*:
‹(*uncurry* (*vmtf-flush-int* $\mathcal{A}_{in}$), *uncurry* (*vmtf-flush* $\mathcal{A}_{in}$)) ∈
$[\lambda(M, vm).\ vm \in vmtf\ \mathcal{A}_{in}\ M \wedge isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in} \wedge isasat\text{-}input\text{-}nempty\ \mathcal{A}_{in}]_f$
$Id \times_r (Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in}) \rightarrow \langle(Id \times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}_{in})\rangle\ nres\text{-}rel$›

by (*intro frefI nres-relI*)
  (*use vmtf-change-to-remove-order* **in** *auto*)

4.7.2 Phase saving

type-synonym *phase-saver* = ‹*bool list*›

definition *phase-saving* :: ‹*nat multiset* ⇒ *phase-saver* ⇒ *bool*› **where**
‹*phase-saving* $\mathcal{A}$ φ ⟷ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length* φ)›

Save phase as given (e.g. for literals in the trail):

definition *save-phase* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
  ‹*save-phase* *L* φ = φ[*atm-of* *L* := *is-pos* *L*]›

lemma *phase-saving-save-phase*[*simp*]:
  ‹*phase-saving* $\mathcal{A}$ (*save-phase* *L* φ) ⟷ *phase-saving* $\mathcal{A}$ φ›
  by (*auto simp*: *phase-saving-def save-phase-def*)

Save opposite of the phase (e.g. for literals in the conflict clause):

definition *save-phase-inv* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
  ‹*save-phase-inv* *L* φ = φ[*atm-of* *L* := ¬*is-pos* *L*]›

end
theory *LBD*
  imports *IsaSAT-Literals*
begin

Chapter 5

LBD

LBD (literal block distance) or glue is a measure of usefulness of clauses: It is the number of different levels involved in a clause. This measure has been introduced by Glucose in 2009 (Audemart and Simon).

LBD has also another advantage, explaining why we implemented it even before working on restarts: It can speed the conflict minimisation. Indeed a literal might be redundant only if there is a literal of the same level in the conflict.

The LBD data structure is well-suited to do so: We mark every level that appears in the conflict in a hash-table like data structure.

Remark that we combine the LBD with a MTF scheme.

5.1 Types and relations

type-synonym *lbd* = ‹*bool list*›
type-synonym *lbd-ref* = ‹*bool list* × *nat* × *nat*›

Beside the actual "lookup" table, we also keep the highest level marked so far to unmark all levels faster (but we currently don't save the LBD and have to iterate over the data structure). We also handle growing of the structure by hand instead of using a proper hash-table. We do so, because there are much stronger guarantees on the key that there is in a general hash-table (especially, our numbers are all small).

definition *lbd-ref* **where**
‹$lbd\text{-}ref = \{((lbd,\ n,\ m),\ lbd').\ lbd = lbd' \wedge n < length\ lbd \wedge$
$(\forall\, k > n.\ k < length\ lbd \longrightarrow \neg lbd!k) \wedge$
$length\ lbd \leq Suc\ (Suc\ (uint32\text{-}max\ div\ 2)) \wedge n < length\ lbd \wedge$
$m = length\ (filter\ id\ lbd)\}$›

5.2 Testing if a level is marked

definition *level-in-lbd* :: ‹*nat* ⇒ *lbd* ⇒ *bool*› **where**
‹$level\text{-}in\text{-}lbd\ i = (\lambda lbd.\ i < length\ lbd \wedge lbd!i)$›

definition *level-in-lbd-ref* :: ‹*nat* ⇒ *lbd-ref* ⇒ *bool*› **where**
‹$level\text{-}in\text{-}lbd\text{-}ref = (\lambda i\ (lbd,\ \text{-}).\ i < length\text{-}uint32\text{-}nat\ lbd \wedge lbd!i)$›

lemma *level-in-lbd-ref-level-in-lbd*:
‹$(uncurry\ (RETURN\ oo\ level\text{-}in\text{-}lbd\text{-}ref),\ uncurry\ (RETURN\ oo\ level\text{-}in\text{-}lbd)) \in$
$nat\text{-}rel \times_r lbd\text{-}ref \rightarrow_f \langle bool\text{-}rel\rangle nres\text{-}rel$›

by (*intro frefI nres-relI*) (*auto simp*: *level-in-lbd-ref-def level-in-lbd-def lbd-ref-def*)

5.3 Marking more levels

definition *list-grow* **where**
⟨*list-grow xs n x = xs @ replicate (n − length xs) x*⟩

definition *lbd-write* :: ⟨*lbd ⇒ nat ⇒ lbd*⟩ **where**
⟨*lbd-write = (λlbd i.*
(if i < length lbd then (lbd[i := True])
else ((list-grow lbd (i + 1) False)[i := True])))⟩

definition *lbd-ref-write* :: ⟨*lbd-ref ⇒ nat ⇒ lbd-ref nres*⟩ **where**
⟨*lbd-ref-write = (λ(lbd, m, n) i. do {*
ASSERT(length lbd ≤ uint32-max ∧ n + 1 ≤ uint32-max);
(if i < length-uint32-nat lbd then
let n = if lbd ! i then n else n+1 in
RETURN (lbd[i := True], max i m, n)
else do {
ASSERT(i + 1 ≤ uint32-max);
RETURN ((list-grow lbd (i + 1) False)[i := True], max i m, n + 1)
})
})⟩

lemma *length-list-grow*[*simp*]:
⟨*length (list-grow xs n a) = max (length xs) n*⟩
by (*auto simp*: *list-grow-def*)

lemma *list-update-append2*: ⟨*i ≥ length xs ⟹ (xs @ ys)[i := x] = xs @ ys[i − length xs := x]*⟩
by (*induction xs arbitrary*: *i*) (*auto split*: *nat.splits*)

lemma *lbd-ref-write-lbd-write*:
⟨*(uncurry (lbd-ref-write), uncurry (RETURN oo lbd-write)) ∈*
$[\lambda(lbd, i).\ i \le Suc\ (uint32\text{-}max\ div\ 2)]_f$
$lbd\text{-}ref \times_f nat\text{-}rel \rightarrow \langle lbd\text{-}ref\rangle nres\text{-}rel$⟩
unfolding *lbd-ref-write-def lbd-write-def*
by (*intro frefI nres-relI*)
(*auto simp*: *level-in-lbd-ref-def level-in-lbd-def lbd-ref-def list-grow-def*
nth-append uint32-max-def length-filter-update-true list-update-append2
length-filter-update-false
intro!: *ASSERT-leI le-trans*[*OF length-filter-le*])

5.4 Cleaning the marked levels

definition *lbd-emtpy-inv* :: ⟨*nat ⇒ bool list × nat ⇒ bool*⟩ **where**
⟨*lbd-emtpy-inv m = (λ(xs, i). i ≤ Suc m ∧ (∀j < i. xs ! j = False) ∧*
(∀j > m. j < length xs ⟶ xs ! j = False))⟩

definition *lbd-empty-ref* **where**
⟨*lbd-empty-ref = (λ(xs, m, -). do {*
(xs, i) ←
$WHILE_T{}^{lbd\text{-}emtpy\text{-}inv\ m}$
(λ(xs, i). i ≤ m)
(λ(xs, i). do {

```
          ASSERT(i < length xs);
          ASSERT(i + 1 < uint32-max);
          RETURN (xs[i := False], i + 1)})
        (xs, 0);
      RETURN (xs, 0, 0)
  })›

definition lbd-empty where
  ‹lbd-empty xs = RETURN (replicate (length xs) False)›

lemma lbd-empty-ref:
  assumes ‹((xs, m, n), xs) ∈ lbd-ref›
  shows
    ‹lbd-empty-ref (xs, m, n) ≤ ⇓ lbd-ref (RETURN (replicate (length xs) False))›
proof –
  have m-xs: ‹m ≤ length xs› and [simp]: ‹xs ≠ []› and le-xs: ‹length xs ≤ uint32-max div 2 + 2›
    using assms by (auto simp: lbd-ref-def)
  have [iff]: ‹(∀ j. ¬ j < (b :: nat)) ⟷ b = 0› for b
    by auto
  have init: ‹lbd-emtpy-inv m (xs, 0)›
    using assms m-xs unfolding lbd-emtpy-inv-def
    by (auto simp: lbd-ref-def)
  have lbd-remove: ‹lbd-emtpy-inv m
       (a[b := False], b + 1)›
    if
      inv: ‹lbd-emtpy-inv m s› and
      ‹case s of (ys, i) ⇒ length ys = length xs› and
      cond: ‹case s of (xs, i) ⇒ i ≤ m› and
      s: ‹s = (a, b)› and
      [simp]: ‹b < length a›
    for s a b
  proof –
    have 1: ‹a[b := False] ! j = False› if ‹j<b› for j
      using inv that unfolding lbd-emtpy-inv-def s
      by auto
    have 2: ‹∀ j>m. j < length (a[b := False]) ⟶ a[b := False] ! j = False›
      using inv that unfolding lbd-emtpy-inv-def s
      by auto
    have ‹b + 1 ≤ Suc m›
      using cond unfolding s by simp
    moreover have ‹a[b := False] ! j = False› if ‹j<b + 1› for j
      using 1[of j] that cond by (cases ‹j = b›) auto
    moreover have ‹∀ j>m. j < length (a[b := False]) ⟶ a[b := False] ! j = False›
      using 2 by auto
    ultimately show ?thesis
      unfolding lbd-emtpy-inv-def by auto
  qed
  have lbd-final: ‹((a, 0, 0), replicate (length xs) False) ∈ lbd-ref›
    if
      lbd: ‹lbd-emtpy-inv m s› and
      I': ‹case s of (ys, i) ⇒ length ys = length xs› and
      cond: ‹¬ (case s of (xs, i) ⇒ i ≤ m)› and
      s: ‹s = (a, b)›
      for s a b
  proof –
    have 1: ‹a[b := False] ! j = False› if ‹j<b› for j
```

using *lbd that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have *2*: ‹*j*>*m* ⟶ *j* < *length a* ⟶ *a* ! *j* = *False*› **for** *j*
using *lbd that* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have [*simp*]: ‹*length a* = *length xs*›
using *I'* **unfolding** *s* **by** *auto*
have [*simp*]: ‹*b* = *Suc m*›
using *cond lbd* **unfolding** *lbd-emtpy-inv-def s*
by *auto*
have [*dest*]: ‹*i* < *length xs* ⟹ ¬*a* ! *i*› **for** *i*
using *1*[*of i*] *2*[*of i*] **by** (*cases* ‹*i* < *Suc m*›) *auto*

have [*simp*]: ‹*a* = *replicate* (*length xs*) *False*›
unfolding *list-eq-iff-nth-eq*
apply (*intro conjI*)
subgoal by *simp*
subgoal by *auto*
done
show *?thesis*
using *le-xs* **by** (*auto simp*: *lbd-ref-def*)
qed
show *?thesis*
unfolding *lbd-empty-ref-def conc-fun-RETURN*
apply *clarify*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*xs*, *i*). *Suc m* − *i*)› **and** *I'* = ‹λ(*ys*, *i*). *length ys* = *length xs*›])
subgoal by *auto*
subgoal by (*rule init*)
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *lbd-ref-def*)
subgoal using *m-xs le-xs* **by** (*auto simp*: *uint32-max-def*)
subgoal by (*rule lbd-remove*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule lbd-final*)
done
qed

lemma *lbd-empty-ref-lbd-empty*:
‹(*lbd-empty-ref*, *lbd-empty*) ∈ *lbd-ref* $\to_f$ ⟨*lbd-ref*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *lbd m lbd'*
using *lbd-empty-ref*[*of lbd m*]
by (*auto simp*: *lbd-empty-def lbd-ref-def*)
done

definition (**in** −)*empty-lbd* :: ‹*lbd*› **where**
‹*empty-lbd* = (*replicate 32 False*)›

definition *empty-lbd-ref* :: ‹*lbd-ref*› **where**
‹*empty-lbd-ref* = (*replicate 32 False*, *0*, *0*)›

lemma *empty-lbd-ref-empty-lbd*:
‹(λ-. (*RETURN empty-lbd-ref*), λ-. (*RETURN empty-lbd*)) ∈ *unit-rel* $\to_f$ ⟨*lbd-ref*⟩*nres-rel*›

by (*intro frefI nres-relI*) (*auto simp*: *empty-lbd-def lbd-ref-def empty-lbd-ref-def uint32-max-def nth-Cons split*: *nat.splits*)

5.5 Extracting the LBD

We do not prove correctness of our algorithm, as we don't care about the actual returned value (for correctness).

definition *get-LBD* :: ‹*lbd* ⇒ *nat nres*› **where**
‹*get-LBD lbd* = *SPEC*(λ-. *True*)›

definition *get-LBD-ref* :: ‹*lbd-ref* ⇒ *nat nres*› **where**
‹*get-LBD-ref* = (λ(*xs*, *m*, *n*). *RETURN n*)›

lemma *get-LBD-ref*:
‹((*lbd*, *m*), *lbd'*) ∈ *lbd-ref* ⟹ *get-LBD-ref* (*lbd*, *m*) ≤ ⇓ *nat-rel* (*get-LBD lbd'*)›
unfolding *get-LBD-ref-def get-LBD-def*
by (*auto split*:*prod.splits*)

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *lbd m n lbd'*
using *get-LBD-ref*[*of lbd*]
by (*auto simp*: *lbd-empty-def lbd-ref-def*)
done

end
theory *LBD-LLVM*
imports *LBD IsaSAT-Literals-LLVM*
begin

no-notation *WB-More-Refinement.fref* ([-]$_f$ - → - [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (- $\rightarrow_f$ - [*60*,*60*] *60*)

type-synonym *'a larray64* = (*'a*,*64*) *larray*
type-synonym *lbd-assn* = ‹(*1 word*) *larray64* × *32 word* × *32 word*›

abbreviation *lbd-int-assn* :: ‹*lbd-ref* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-int-assn* ≡ *larray64-assn bool1-assn* $\times_a$ *uint32-nat-assn* $\times_a$ *uint32-nat-assn*›

definition *lbd-assn* :: ‹*lbd* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-assn* ≡ *hr-comp lbd-int-assn lbd-ref*›

Testing if a level is marked **sepref-def** *level-in-lbd-code*
is [] ‹*uncurry* (*RETURN oo level-in-lbd-ref*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *lbd-int-assn*^{k} $\rightarrow_a$ *bool1-assn*›
supply [[*goals-limit*=*1*]]
unfolding *level-in-lbd-ref-def short-circuit-conv length-uint32-nat-def*
apply (*rewrite* **in** ⌑ < - *annot-unat-snat-upcast*[**where** *'l*=*64*])
apply (*rewrite* **in** - ! ⌑ *annot-unat-snat-upcast*[**where** *'l*=*64*])
by *sepref*

lemma *level-in-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry level-in-lbd-code*, *uncurry* (*RETURN* ∘∘ *level-in-lbd*)) ∈ *uint32-nat-assn*^{k} $*_a$
lbd-assn^k $\rightarrow_a$ *bool1-assn*›
supply *lbd-ref-def*[*simp*] *uint32-max-def*[*simp*]
using *level-in-lbd-code.refine*[*FCOMP level-in-lbd-ref-level-in-lbd*[*unfolded convert-fref*]]
unfolding *lbd-assn-def*[*symmetric*]
by *simp*

sepref-def *lbd-empty-code*
is [] ‹*lbd-empty-ref*›
:: ‹*lbd-int-assn*^{d} $\rightarrow_a$ *lbd-int-assn*›
unfolding *lbd-empty-ref-def*
supply [[*goals-limit=1*]]
apply (*rewrite at* ‹- + ⌑› *snat-const-fold*[**where** *'a=64*])+
apply (*rewrite at* ‹(-, ⌑)› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const TYPE*(*32*))
apply (*rewrite* **in** - ≤ ⌑ *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
‹(*lbd-empty-code*, *lbd-empty*) ∈ *lbd-assn*^{d} $\rightarrow_a$ *lbd-assn*›
using *lbd-empty-code.refine*[*FCOMP lbd-empty-ref-lbd-empty*[*unfolded convert-fref*]]
unfolding *lbd-assn-def* **.**

sepref-def *empty-lbd-code*
is [] ‹*uncurry0* (*RETURN empty-lbd-ref*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *lbd-int-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-lbd-ref-def larray-fold-custom-replicate*
apply (*rewrite at* ‹*op-larray-custom-replicate* ⌑ -› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemma *empty-lbd-ref-empty-lbd*:
‹((*uncurry0* (*RETURN empty-lbd-ref*), *uncurry0* (*RETURN empty-lbd*)) ∈ *unit-rel* $\rightarrow_f$ ⟨*lbd-ref*⟩*nres-rel*›
using *empty-lbd-ref-empty-lbd* **unfolding** *uncurry0-def convert-fref* **.**

lemma *empty-lbd-hnr*[*sepref-fr-rules*]:
‹(*Sepref-Misc.uncurry0 empty-lbd-code*, *Sepref-Misc.uncurry0* (*RETURN empty-lbd*)) ∈ *unit-assn*^{k} $\rightarrow_a$
lbd-assn›
using *empty-lbd-code.refine*[*FCOMP empty-lbd-ref-empty-lbd*]
unfolding *lbd-assn-def* **.**

sepref-def *get-LBD-code*
is [] ‹*get-LBD-ref*›
:: ‹*lbd-int-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *get-LBD-ref-def*
by *sepref*

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) ∈ *lbd-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
using *get-LBD-code.refine*[*FCOMP get-LBD-ref-get-LBD*[*unfolded convert-fref*],
unfolded lbd-assn-def[*symmetric*]] **.**

Marking more levels **lemmas** *list-grow-alt* = *list-grow-def*[*unfolded op-list-grow-init'-def*[*symmetric*]]

sepref-def *lbd-write-code*
  is [] ‹*uncurry lbd-ref-write*›
  :: ‹ [λ(*lbd*, *i*). $i \leq$ *Suc* (*uint32-max div 2*)]$_a$
    lbd-int-assn^d $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *lbd-int-assn*›
  supply [[*goals-limit=1*]]
  unfolding *lbd-ref-write-def length-uint32-nat-def list-grow-alt max-def*
    op-list-grow-init'-alt
  apply (*rewrite at* ‹- + ¤› *unat-const-fold*[**where** *'a=32*])
  apply (*rewrite at* ‹- + ¤› *unat-const-fold*[**where** *'a=32*])
  apply (*rewrite* **in** *If* (¤ < -) *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*rewrite* **in** *If* (- ! ¤) *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*rewrite* **in** -[¤ := -] *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*rewrite* **in** *op-list-grow-init* - ¤ - *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*rewrite at* (-[¤ := -], -, - + -) *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*annot-unat-const TYPE*(*32*))
  by *sepref*

lemma *lbd-write-hnr-*[*sepref-fr-rules*]:
  ‹(*uncurry lbd-write-code*, *uncurry* (*RETURN* ∘∘ *lbd-write*))
    $\in$ [λ(*lbd*, *i*). $i \leq$ *Suc* (*uint32-max div 2*)]$_a$
     lbd-assn^d $*_a$ *uint32-nat-assn*^{k} $\rightarrow$ *lbd-assn*›
  using *lbd-write-code.refine*[*FCOMP lbd-ref-write-lbd-write*[*unfolded convert-fref*]]
  unfolding *lbd-assn-def* **.**

experiment begin

export-llvm
  level-in-lbd-code
  lbd-empty-code
  empty-lbd-code
  get-LBD-code
  lbd-write-code

end

end
theory *Version*
  imports *Main*
begin

This code was taken from IsaFoR and adapted to git.

local-setup ‹
  let
    val version =
      trim-line (*#1* (*Isabelle-System.bash-output* (*cd* $*ISAFOL/* && *git rev−parse −−short HEAD* ||
echo unknown)))
  in
    Local-Theory.define
      ((***binding***‹*version*›, *NoSyn*),
        ((***binding***‹*version-def*›, []), *HOLogic.mk-literal version*)) *#> #2*
  end
›

declare *version-def* [*code*]

end
theory *IsaSAT-Watch-List*
  imports *IsaSAT-Literals*
begin

Chapter 6

Refinement of the Watched Function

There is not much to say about watch lists since they are arrays of resizeable arrays, which are defined in a separate theory.

However, when replacing the elements in our watch lists from (*nat* × *uint32*) to (*nat* × *uint32* × *bool*) to enable special handling of binary clauses, we got a huge and unexpected slowdown, due to a much higher number of cache misses (roughly 3.5 times as many on a eq.atree.braun.8.unsat.cnf which also took 66s instead of 50s). While toying with the generated ML code, we found out that our version of the tuples with booleans were using 40 bytes instead of 24 previously. Just merging the *uint32* and the *bool* to a single *uint64* was sufficient to get the performance back.

Remark that however, the evaluation of terms like (*2*::*uint64*) ^ *32* was not done automatically and even worse, was redone each time, leading to a complete performance blow-up (75s on my macbook for eq.atree.braun.7.unsat.cnf instead of 7s).

None of the problems appears in the LLVM code.

6.1 Definition

definition *map-fun-rel* :: ‹(*nat* × *'key*) *set* ⇒ (*'b* × *'a*) *set* ⇒ (*'b list* × (*'key* ⇒ *'a*)) *set*› **where**
map-fun-rel-def-internal:
‹$map\text{-}fun\text{-}rel\ D\ R = \{(m, f).\ \forall (i, j){\in}D.\ i < length\ m \wedge (m\ !\ i, f\ j) \in R\}$›

lemma *map-fun-rel-def*:
‹$\langle R\rangle map\text{-}fun\text{-}rel\ D = \{(m, f).\ \forall (i, j){\in}D.\ i < length\ m \wedge (m\ !\ i, f\ j) \in R\}$›
unfolding *relAPP-def map-fun-rel-def-internal* **by** *auto*

definition *mop-append-ll* :: *'a list list* ⇒ *nat literal* ⇒ *'a* ⇒ *'a list list nres* **where**
‹*mop-append-ll xs i x* = *do* {
ASSERT(*nat-of-lit i* < *length xs*);
RETURN (*append-ll xs* (*nat-of-lit i*) *x*)
}›

6.2 Operations

lemma *length-ll-length-ll-f*:
‹(*uncurry* (*RETURN oo length-ll*), *uncurry* (*RETURN oo length-ll-f*)) ∈
$[\lambda(W, L).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}_{in}]_f$ ((⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}_{in}$)) $\times_r$ *nat-lit-rel*) →
⟨*nat-rel*⟩ *nres-rel*›
unfolding *length-ll-def length-ll-f-def*

by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def nat-lit-rel-def*)

lemma *mop-append-ll*:
‹(*uncurry2 mop-append-ll*, *uncurry2* (*RETURN ooo* ($\lambda W\ i\ x.\ W(i := W\ i\ @\ [x])$))) $\in$
$[\lambda((W,\ i),\ x).\ i \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f\ \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \times_f Id \times_f Id \to \langle\langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A})\rangle nres\text{-}rel$›
unfolding *uncurry-def mop-append-ll-def*
by (*intro frefI nres-relI*)
(*auto intro*!: *ASSERT-leI simp*: *map-fun-rel-def append-ll-def*)

definition *delete-index-and-swap-update* :: ‹($'a \Rightarrow 'b\ list$) $\Rightarrow 'a \Rightarrow nat \Rightarrow 'a \Rightarrow 'b\ list$› **where**
‹*delete-index-and-swap-update W K w* = *W*(*K* := *delete-index-and-swap* (*W K*) *w*)›

The precondition is not necessary.

lemma *delete-index-and-swap-ll-delete-index-and-swap-update*:
‹(*uncurry2* (*RETURN ooo delete-index-and-swap-ll*), *uncurry2* (*RETURN ooo delete-index-and-swap-update*))
$\in[\lambda((W,\ L),\ i).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f$ ($\langle Id\rangle$*map-fun-rel* ($D_0\ \mathcal{A}$) $\times_r$ *nat-lit-rel*) $\times_r$ *nat-rel* $\to$
$\langle\langle Id\rangle$*map-fun-rel* ($D_0\ \mathcal{A}$)$\rangle$*nres-rel*›
by (*auto simp*: *delete-index-and-swap-ll-def uncurry-def fref-def nres-rel-def delete-index-and-swap-update-def map-fun-rel-def p2rel-def nat-lit-rel-def br-def nth-list-update' nat-lit-rel-def*
simp del: *literal-of-nat.simps*)

definition *append-update* :: ‹($'a \Rightarrow 'b\ list$) $\Rightarrow 'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b\ list$› **where**
‹*append-update W L a* = *W*(*L*:= *W* (*L*) @ [*a*])›

type-synonym *nat-clauses-l* = ‹*nat list list*›

Refinement of the Watched Function

definition *watched-by-nth* :: ‹*nat twl-st-wl* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat watcher*› **where**
‹*watched-by-nth* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) *L i*. *W L* ! *i*)›

definition *watched-app*
:: ‹(*nat literal* $\Rightarrow$ (*nat watcher*) *list*) $\Rightarrow$ *nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat watcher*› **where**
‹*watched-app M L i* $\equiv$ *M L* ! *i*›

lemma *watched-by-nth-watched-app*:
‹*watched-by S K* ! *w* = *watched-app* ((*snd o snd o snd o snd o snd o snd o snd o snd*) *S*) *K w*›
by (*cases S*) (*auto simp*: *watched-app-def*)

lemma *nth-ll-watched-app*:
‹(*uncurry2* (*RETURN ooo nth-rll*), *uncurry2* (*RETURN ooo watched-app*)) $\in$
$[\lambda((W,\ L),\ i).\ L \in\#\ (\mathcal{L}_{all}\ \mathcal{A})]_f$ (($\langle Id\rangle$*map-fun-rel* ($D_0\ \mathcal{A}$)) $\times_r$ *nat-lit-rel*) $\times_r$ *nat-rel* $\to$
$\langle$*nat-rel* $\times_r$ *Id*$\rangle$ *nres-rel*›
unfolding *watched-app-def nth-rll-def*
by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def nat-lit-rel-def*)

end
theory *IsaSAT-Watch-List-LLVM*
imports *IsaSAT-Watch-List IsaSAT-Literals-LLVM*

begin

type-synonym *watched-wl-uint32*
  = ‹(*64*,(*64* *word* × *32* *word* × *1* *word*),*64*)*array-array-list*›

abbreviation *watcher-fast-assn* ≡ *sint64-nat-assn* $\times_a$ *unat-lit-assn* $\times_a$ *bool1-assn*

end
theory *IsaSAT-Lookup-Conflict*
  imports
    IsaSAT-Literals
    Watched-Literals.CDCL-Conflict-Minimisation
    LBD
    IsaSAT-Clauses
    IsaSAT-Watch-List
    IsaSAT-Trail
begin

Chapter 7

Clauses Encoded as Positions

We use represent the conflict in two data structures close to the one used by the most SAT solvers: We keep an array that represent the clause (for efficient iteration on the clause) and a "hash-table" to efficiently test if a literal belongs to the clause.

The first data structure is simply an array to represent the clause. This theory is only about the second data structure. We refine it from the clause (seen as a multiset) in two steps:

1. First, we represent the clause as a "hash-table", where the i-th position indicates *Some True* (respectively *Some False*, *None*) if *Pos* i is present in the clause (respectively *Neg* i, not at all). This allows to represent every not-tautological clause whose literals fits in the underlying array.

2. Then we refine it to an array of booleans indicating if the atom is present or not. This information is redundant because we already know that a literal can only appear negated compared to the trail.

The first step makes it easier to reason about the clause (since we have the full clause), while the second step should generate (slightly) more efficient code.

Most solvers also merge the underlying array with the array used to cache information for the conflict minimisation (see theory *Watched-Literals.CDCL-Conflict-Minimisation*, where we only test if atoms appear in the clause, not literals).

As far as we know, versat stops at the first refinement (stating that there is no significant overhead, which is probably true, but the second refinement is not much additional work anyhow and we don't have to rely on the ability of the compiler to not represent the option type on booleans as a pointer, which it might be able to or not).

This is the first level of the refinement. We tried a few different definitions (including a direct one, i.e., mapping a position to the inclusion in the set) but the inductive version turned out to the easiest one to use.

inductive *mset-as-position* :: ‹*bool option list* ⇒ *nat literal multiset* ⇒ *bool*› **where**
empty:
  ‹*mset-as-position* (*replicate n None*) {#}› |
add:
  ‹*mset-as-position xs'* (*add-mset L P*)›
  if ‹*mset-as-position xs P*› **and** ‹*atm-of L* < *length xs*› **and** ‹$L \notin\#$ *P*› **and** ‹$-L \notin\#$ *P*› **and**
    ‹*xs'* = *xs*[*atm-of L* := *Some* (*is-pos L*)]›

lemma *mset-as-position-distinct-mset*:

‹mset-as-position xs P ⟹ distinct-mset P›
  by (*induction rule*: *mset-as-position.induct*) *auto*

lemma *mset-as-position-atm-le-length*:
  ‹mset-as-position xs P ⟹ L ∈# P ⟹ atm-of L < length xs›
  by (*induction rule*: *mset-as-position.induct*) (*auto simp*: *nth-list-update′ atm-of-eq-atm-of*)

lemma *mset-as-position-nth*:
  ‹mset-as-position xs P ⟹ L ∈# P ⟹ xs ! (atm-of L) = Some (is-pos L)›
  by (*induction rule*: *mset-as-position.induct*)
    (*auto simp*: *nth-list-update′ atm-of-eq-atm-of dest*: *mset-as-position-atm-le-length*)

lemma *mset-as-position-in-iff-nth*:
  ‹mset-as-position xs P ⟹ atm-of L < length xs ⟹ L ∈# P ⟷ xs ! (atm-of L) = Some (is-pos L)›
  by (*induction rule*: *mset-as-position.induct*)
    (*auto simp*: *nth-list-update′ atm-of-eq-atm-of is-pos-neg-not-is-pos*
      dest: *mset-as-position-atm-le-length*)

lemma *mset-as-position-tautology*: ‹mset-as-position xs C ⟹ ¬tautology C›
  by (*induction rule*: *mset-as-position.induct*) (*auto simp*: *tautology-add-mset*)

lemma *mset-as-position-right-unique*:
  assumes
    map: ‹mset-as-position xs D› **and**
    map′: ‹mset-as-position xs D′›
  shows ‹D = D′›
proof (*rule distinct-set-mset-eq*)
  show ‹distinct-mset D›
    using *mset-as-position-distinct-mset*[*OF map*] **.**
  show ‹distinct-mset D′›
    using *mset-as-position-distinct-mset*[*OF map′*] **.**
  show ‹set-mset D = set-mset D′›
    using *mset-as-position-in-iff-nth*[*OF map*] *mset-as-position-in-iff-nth*[*OF map′*]
      mset-as-position-atm-le-length[*OF map*] *mset-as-position-atm-le-length*[*OF map′*]
    by *blast*
qed

lemma *mset-as-position-mset-union*:
  fixes *P xs*
  defines ‹xs′ ≡ fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs›
  assumes
    mset: ‹mset-as-position xs P′› **and**
    atm-P-xs: ‹∀ L ∈ set P. atm-of L < length xs› **and**
    uL-P: ‹∀ L ∈ set P. −L ∉# P′› **and**
    dist: ‹distinct P› **and**
    tauto: ‹¬tautology (mset P)›
  shows ‹mset-as-position xs′ (mset P ∪# P′)›
  using *atm-P-xs uL-P dist tauto* **unfolding** *xs′-def*
proof (*induction P*)
  case *Nil*
  show *?case* **using** *mset* **by** *auto*
next
  case (*Cons L P*) **note** *IH* = *this*(*1*) **and** *atm-P-xs* = *this*(*2*) **and** *uL-P* = *this*(*3*) **and** *dist* = *this*(*4*)
    and *tauto* = *this*(*5*)
  then have *mset*:
    ‹mset-as-position (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) (mset P ∪# P′)›

by (*auto simp*: *tautology-add-mset*)
show *?case*
proof (*cases* ‹$L \in\#$ P'›)
case *True*
then show *?thesis*
using *mset dist*
by (*metis* (*no-types*, *lifting*) *add-mset-union assms*(*2*) *distinct.simps*(*2*) *fold-simps*(*2*) *insert-DiffM list-update-id mset.simps*(*2*) *mset-as-position-nth set-mset-mset sup-union-right1*)
next
case *False*
have [*simp*]: ‹*length* (*fold* (λL *xs*. *xs*[*atm-of* L := *Some* (*is-pos* L)]) P *xs*) = *length xs*›
by (*induction* P *arbitrary*: *xs*) *auto*
moreover have ‹$- L \notin$ *set* P›
using *tauto* **by** (*metis list.set-intros*(*1*) *list.set-intros*(*2*) *set-mset-mset tautology-minus*)
moreover have
‹*fold* (λL *xs*. *xs*[*atm-of* L := *Some* (*is-pos* L)]) P (*xs*[*atm-of* L := *Some* (*is-pos* L)]) =
(*fold* (λL *xs*. *xs*[*atm-of* L := *Some* (*is-pos* L)]) P *xs*) [*atm-of* L := *Some* (*is-pos* L)]›
using *uL-P dist tauto*
apply (*induction* P *arbitrary*: *xs*)
subgoal by *auto*
subgoal for L' P
by (*cases* ‹*atm-of* L = *atm-of* L'›)
(*auto simp*: *tautology-add-mset list-update-swap atm-of-eq-atm-of*)
done
ultimately show *?thesis*
using *mset atm-P-xs dist uL-P False* **by** (*auto intro*!: *mset-as-position.add*)
qed
qed

lemma *mset-as-position-empty-iff*: ‹*mset-as-position xs* {#} $\longleftrightarrow$ ($\exists$ *n*. *xs* = *replicate n None*)›
apply (*rule iffI*)
subgoal
by (*cases rule*: *mset-as-position.cases*, *assumption*) *auto*
subgoal
by (*auto intro*: *mset-as-position.intros*)
done

type-synonym (**in** −) *lookup-clause-rel* = ‹*nat* × *bool option list*›

definition *lookup-clause-rel* :: ‹*nat multiset* ⇒ (*lookup-clause-rel* × *nat literal multiset*) *set*› **where**
‹*lookup-clause-rel* $\mathcal{A}$ = {((*n*, *xs*), *C*). *n* = *size C* ∧ *mset-as-position xs C* ∧
(∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*)}›

lemma *lookup-clause-rel-empty-iff*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *n* = *0* ⟷ *C* = {#}›
by (*auto simp*: *lookup-clause-rel-def*)

lemma *conflict-atm-le-length*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
L < *length xs*›
by (*auto simp*: *lookup-clause-rel-def*)

lemma *conflict-le-length*:
assumes
c-rel: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›

shows ⟨*atm-of* L < *length* *xs*⟩
proof −
have
size: ⟨n = *size* C⟩ **and**
mset-pos: ⟨*mset-as-position* *xs* C⟩ **and**
atms-le: ⟨$\forall$ L∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). L < *length* *xs*⟩
using *c-rel* **unfolding** *lookup-clause-rel-def* **by** *blast+*
have ⟨*atm-of* L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩
using L-$\mathcal{L}_{all}$ **by** (*simp add*: *atms-of-def*)
then show *?thesis*
using *atms-le* **by** *blast*
qed

lemma *lookup-clause-rel-atm-in-iff*:
⟨((n, *xs*), C) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ L ∈# C ⟷ *xs*!(*atm-of* L) = *Some* (*is-pos* L)⟩
by (*rule* *mset-as-position-in-iff-nth*)
(*auto simp*: *lookup-clause-rel-def* *atms-of-def*)

lemma
assumes
c: ⟨((n,*xs*), C) ∈ *lookup-clause-rel* $\mathcal{A}$⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows
lookup-clause-rel-not-tautolgy: ⟨¬*tautology* C⟩ **and**
lookup-clause-rel-distinct-mset: ⟨*distinct-mset* C⟩ **and**
lookup-clause-rel-size: ⟨*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}$ C ⟹ *size* C ≤ *1* + *uint32-max div 2*⟩
proof −
have *mset*: ⟨*mset-as-position* *xs* C⟩ **and** ⟨n = *size* C⟩ **and** ⟨$\forall$ L∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). L < *length* *xs*⟩
using *c* **unfolding** *lookup-clause-rel-def* **by** *fast+*
show ⟨¬*tautology* C⟩
using *mset*
apply (*induction rule*: *mset-as-position.induct*)
subgoal by (*auto simp*: *literals-are-in*-$\mathcal{L}_{in}$-*def*)
subgoal by (*auto simp*: *tautology-add-mset*)
done
show ⟨*distinct-mset* C⟩
using *mset* *mset-as-position-distinct-mset* **by** *blast*
then show ⟨*literals-are-in*-$\mathcal{L}_{in}$ $\mathcal{A}$ C ⟹ *size* C ≤ *1* + *uint32-max div 2*⟩
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ⟨C⟩] ⟨¬*tautology* C⟩ *bounded* **by** *auto*
qed

definition *option-bool-rel* :: ⟨(*bool* × *'a option*) *set*⟩ **where**
⟨*option-bool-rel* = {(b, x). b ⟷ ¬(*is-None* x)}⟩

definition *NOTIN* :: ⟨*bool option*⟩ **where**
[*simp*]: ⟨*NOTIN* = *None*⟩

definition *ISIN* :: ⟨*bool* ⇒ *bool option*⟩ **where**
[*simp*]: ⟨*ISIN* b = *Some* b⟩

definition *is-NOTIN* :: ⟨*bool option* ⇒ *bool*⟩ **where**
[*simp*]: ⟨*is-NOTIN* x ⟷ x = *NOTIN*⟩

lemma *is-NOTIN-alt-def*:
  ‹*is-NOTIN x* $\longleftrightarrow$ *is-None x*›
  by (*auto split*: *option.splits*)

definition *option-lookup-clause-rel* **where**
‹*option-lookup-clause-rel* $\mathcal{A}$ = {((*b*,(*n*,*xs*)), *C*). *b* = (*C* = *None*) ∧
  (*C* = *None* $\longrightarrow$ ((*n*,*xs*), {#}) ∈ *lookup-clause-rel* $\mathcal{A}$) ∧
  (*C* ≠ *None* $\longrightarrow$ ((*n*,*xs*), *the C*) ∈ *lookup-clause-rel* $\mathcal{A}$)}
  ›

lemma *option-lookup-clause-rel-lookup-clause-rel-iff*:
  ‹((*False*, (*n*, *xs*)), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ $\longleftrightarrow$
  ((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
  unfolding *option-lookup-clause-rel-def* **by** *auto*

type-synonym (**in** −) *conflict-option-rel* = ‹*bool* × *nat* × *bool option list*›

definition (**in** −) *lookup-clause-assn-is-None* :: ‹- ⇒ *bool*› **where**
  ‹*lookup-clause-assn-is-None* = (λ(*b*, -, -). *b*)›

lemma *lookup-clause-assn-is-None-is-None*:
  ‹(*RETURN o lookup-clause-assn-is-None*, *RETURN o is-None*) ∈
   option-lookup-clause-rel $\mathcal{A}$ $\rightarrow_f$ ⟨*bool-rel*⟩*nres-rel*›
  by (*intro nres-relI frefI*)
   (*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-None-def split*: *option.splits*)

definition (**in** −) *lookup-clause-assn-is-empty* :: ‹- ⇒ *bool*› **where**
  ‹*lookup-clause-assn-is-empty* = (λ(-, *n*, -). *n* = *0*)›

lemma *lookup-clause-assn-is-empty-is-empty*:
  ‹(*RETURN o lookup-clause-assn-is-empty*, *RETURN o* (λ*D*. *Multiset.is-empty*(*the D*))) ∈
  [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
  by (*intro nres-relI frefI*)
   (*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-empty-def lookup-clause-rel-def*
     Multiset.is-empty-def split: *option.splits*)

definition *size-lookup-conflict* :: ‹- ⇒ *nat*› **where**
  ‹*size-lookup-conflict* = (λ(-, *n*, -). *n*)›

definition *size-conflict-wl-heur* :: ‹- ⇒ *nat*› **where**
  ‹*size-conflict-wl-heur* = (λ(*M*, *N*, *U*, *D*, -, -, -, -). *size-lookup-conflict D*)›

lemma (**in** −) *mset-as-position-length-not-None*:
   ‹*mset-as-position x2 C* ⟹ *size C* = *length* (*filter* ((≠) *None*) *x2*)›
proof (*induction rule*: *mset-as-position.induct*)
  case (*empty n*)
  then show *?case* **by** *auto*
next
  case (*add xs P L xs'*) **note** *m-as-p* = *this*(*1*) **and** *atm-L* = *this*(*2*)
  have *xs-L*: ‹*xs* ! (*atm-of L*) = *None*›
  proof −
    obtain *bb* :: ‹*bool option* ⇒ *bool*› **where**
      f1: ‹∀ *z*. *z* = *None* ∨ *z* = *Some* (*bb z*)›
      by (*metis option.exhaust*)

    have *f2*: ‹*xs* ! *atm-of L* ≠ *Some* (*is-pos L*)›
      using *add.hyps*(*1*) *add.hyps*(*2*) *add.hyps*(*3*) *mset-as-position-in-iff-nth* **by** *blast*
    have *f3*: ‹∀ *z b*. ((*Some b* = *z* ∨ *z* = *None*) ∨ *bb z*) ∨ *b*›
      using *f1* **by** *blast*
    have *f4*: ‹∀ *zs*. (*zs* ! *atm-of L* ≠ *Some* (*is-pos* (− *L*)) ∨ ¬ *atm-of L* < *length zs*)
          ∨ ¬ *mset-as-position zs P*›
      by (*metis add.hyps*(*4*) *atm-of-uminus mset-as-position-in-iff-nth*)
    have ‹∀ *z b*. ((*Some b* = *z* ∨ *z* = *None*) ∨ ¬ *bb z*) ∨ ¬ *b*›
      using *f1* **by** *blast*
    then show *?thesis*
      using *f4 f3 f2* **by** (*metis add.hyps*(*1*) *add.hyps*(*2*) *is-pos-neg-not-is-pos*)
  qed
  obtain *xs1 xs2* **where**
    xs-xs12: ‹*xs* = *xs1* @ *None* # *xs2*› **and**
    xs1: ‹*length xs1* = *atm-of L*›
    using *atm-L upd-conv-take-nth-drop*[*of* ‹*atm-of L*› *xs* ‹*None*›] **apply** −
    apply (*subst*(*asm*)(*2*) *xs-L*[*symmetric*])
    by (*force simp del*: *append-take-drop-id*)+
  then show *?case*
    using *add* **by** (*auto simp*: *list-update-append*)
qed

definition (**in** −) *is-in-lookup-conflict* **where**
  ‹*is-in-lookup-conflict* = (λ(*n*, *xs*) *L*. ¬*is-None* (*xs* ! *atm-of L*))›

lemma *mset-as-position-remove*:
  ‹*mset-as-position xs D* ⟹ *L* < *length xs* ⟹
   mset-as-position (*xs*[*L* := *None*]) (*remove1-mset* (*Pos L*) (*remove1-mset* (*Neg L*) *D*))›
proof (*induction rule*: *mset-as-position.induct*)
  case (*empty n*)
  then have [*simp*]: ‹(*replicate n None*)[*L* := *None*] = *replicate n None*›
    using *list-update-id*[*of* ‹*replicate n None*› *L*] **by** *auto*
  show *?case* **by** (*auto intro*: *mset-as-position.intros*)
next
  case (*add xs P K xs'*)
  show *?case*
  proof (*cases* ‹*L* = *atm-of K*›)
    case *True*
    then show *?thesis*
      using *add* **by** (*cases K*) *auto*
  next
    case *False*
    have *map*: ‹*mset-as-position* (*xs*[*L* := *None*]) (*remove1-mset* (*Pos L*) (*remove1-mset* (*Neg L*) *P*))›
      using *add* **by** *auto*
    have ‹*K* ∉# *P* − {#*Pos L*, *Neg L*#}› ‹−*K* ∉# *P* − {#*Pos L*, *Neg L*#}›
      by (*auto simp*: *add.hyps dest*!: *in-diffD*)
    then show *?thesis*
      using *mset-as-position.add*[*OF map*, *of* ‹*K*› ‹*xs*[*L* := *None*, *atm-of K* := *Some* (*is-pos K*)]›]
        add False list-update-swap[*of* ‹*atm-of K*› *L xs*] **apply** *simp*
      apply (*subst diff-add-mset-swap*)
      by *auto*
  qed
qed

lemma *mset-as-position-remove2*:

‹*mset-as-position xs D* ⟹ *atm-of L* < *length xs* ⟹
mset-as-position (*xs*[*atm-of L* := *None*]) (*D* − {#*L*, −*L*#})›
using *mset-as-position-remove*[*of xs D* ‹*atm-of* (−*L*)›]
by (*smt add-mset-commute add-mset-diff-bothsides atm-of-uminus insert-DiffM literal.exhaust-sel minus-notin-trivial2 remove-1-mset-id-iff-notin uminus-not-id'*)

definition (**in** −) *delete-from-lookup-conflict*
:: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel nres*› **where**
‹*delete-from-lookup-conflict* = (λ*L* (*n*, *xs*). *do* {
ASSERT(*n*≥*1*);
ASSERT(*atm-of L* < *length xs*);
RETURN (*n* − *1*, *xs*[*atm-of L* := *None*])
})›

lemma *delete-from-lookup-conflict-op-mset-delete*:
‹(*uncurry delete-from-lookup-conflict*, *uncurry* (*RETURN oo remove1-mset*)) ∈
[λ(*L*, *D*). −*L* ∉# *D* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *L* ∈# *D*]$_f$ *Id* ×$_f$ *lookup-clause-rel* $\mathcal{A}$ →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *mset-as-position-remove*[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*atm-of* (*fst y*)›]
apply (*cases x*; *cases y*; *cases* ‹*fst y*›)
by (*auto simp*: *delete-from-lookup-conflict-def lookup-clause-rel-def*
dest!: *multi-member-split*
intro!: *ASSERT-refine-left*)
done

definition *delete-from-lookup-conflict-pre* **where**
‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ = (λ(*a*, *b*). − *a* ∉# *b* ∧ *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *a* ∈# *b*)›

definition *set-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*set-conflict-m M N i* - - - - - =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*N*∝*i*)) ∧ *n* = *card-max-lvl M* (*mset* (*N*∝*i*)) ∧
out-learned M C outl)›

definition *merge-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m M N i D* - - - =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *merge-conflict-m-g*
:: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m-g init M Ni D* =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*drop init* (*Ni*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *add-to-lookup-conflict* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*add-to-lookup-conflict* = (λ*L* (*n*, *xs*). (*if xs* ! *atm-of L* = *NOTIN then n* + *1 else n*,
xs[*atm-of L* := *ISIN* (*is-pos L*)]))›

definition *lookup-conflict-merge′-step*
:: ‹*nat multiset* ⇒ *nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *nat* ⇒ *lookup-clause-rel* ⇒ *nat clause-l* ⇒
nat clause ⇒ *out-learned* ⇒ *bool*›
where
‹*lookup-conflict-merge′-step* $\mathcal{A}$ *init M i clvls zs D C outl* = (
let D′ = *mset* (*take* (*i* − *init*) (*drop init D*));
E = *remdups-mset* (*D′* + *C*) *in*
((*False*, *zs*), *Some E*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
out-learned M (*Some E*) *outl* ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ *E* ∧ *clvls* = *card-max-lvl M E*)›

lemma *option-lookup-clause-rel-update-None*:
assumes ‹((*False*, (*n*, *xs*)), *Some D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and** *L-xs* : ‹*L* < *length xs*›
shows ‹((*False*, (*if xs*!*L* = *None then n else n* − *1*, *xs*[*L* := *None*])),
Some (*D* − {# *Pos L*, *Neg L* #})) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
proof −
have [*simp*]: ‹*L* ∉# *A* ⟹ *A* − *add-mset L′* (*add-mset L B*) = *A* − *add-mset L′ B*›
for *A B* :: ‹′*a multiset*› **and** *L L′*
by (*metis add-mset-commute minus-notin-trivial*)
have ‹*n* = *size D*› **and** *map*: ‹*mset-as-position xs D*›
using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)
have *xs-None-iff*: ‹*xs* ! *L* = *None* ⟷ *Pos L* ∉# *D* ∧ *Neg L* ∉# *D*›
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ‹*Pos L*›]
mset-as-position-in-iff-nth[*of xs D* ‹*Neg L*›]
by (*cases* ‹*xs* ! *L*›) *auto*

have *1*: ‹*xs* ! *L* = *None* ⟹ *D* − {#*Pos L*, *Neg L*#} = *D*›
using *assms* **by** (*auto simp*: *xs-None-iff minus-notin-trivial*)
have *2*: ‹*xs* ! *L* = *None* ⟹ *xs*[*L* := *None*] = *xs*›
using *map list-update-id*[*of xs L*] **by** (*auto simp*: *1*)
have *3*: ‹*xs* ! *L* = *Some y* ⟷ (*y* ∧ *Pos L* ∈# *D* ∧ *Neg L* ∉# *D*) ∨ (¬*y* ∧ *Pos L* ∉# *D* ∧ *Neg L* ∈#
D)›
for *y*
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ‹*Pos L*›]
mset-as-position-in-iff-nth[*of xs D* ‹*Neg L*›]
by (*cases* ‹*xs* ! *L*›) *auto*

show *?thesis*
using *assms mset-as-position-remove*[*of xs D L*]
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def 1 2 3 size-remove1-mset-If*
minus-notin-trivial mset-as-position-remove)
qed

lemma *add-to-lookup-conflict-lookup-clause-rel*:
assumes
confl: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
uL-C: ‹−*L* ∉# *C*› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹(*add-to-lookup-conflict L* (*n*, *xs*), {#*L*#} ∪# *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
proof −

have
  n: ‹*n* = *size C*› **and**
  mset: ‹*mset-as-position xs C*› **and**
  atm: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*›
  using *confl* **unfolding** *lookup-clause-rel-def* **by** *blast*+
have ‹*distinct-mset C*›
  using *mset* **by** (*blast dest*: *mset-as-position-distinct-mset*)
have *atm-L-xs*: ‹*atm-of L* < *length xs*›
  using *atm L*-$\mathcal{L}_{all}$ **by** (*auto simp*: *atms-of-def*)
then show *?thesis*
proof (*cases* ‹*L* ∈# *C*›)
  case *True*
  with *mset* **have** *xs*: ‹*xs* ! *atm-of L* = *Some* (*is-pos L*)› ‹*xs* ! *atm-of L* ≠ *None*›
    by (*auto dest*: *mset-as-position-nth*)
  moreover have ‹{#*L*#} ∪# *C* = *C*›
    using *True* **by** (*simp add*: *subset-mset.sup.absorb2*)
  ultimately show *?thesis*
    using *n mset atm True*
    by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def xs*[*symmetric*])
next
  case *False*
  with *mset* **have** ‹*xs* ! *atm-of L* = *None*›
    using *mset-as-position-in-iff-nth*[*of xs C L*]
     mset-as-position-in-iff-nth[*of xs C* ‹−*L*›] *atm-L-xs uL-C*
    by (*cases L*; *cases* ‹*xs* ! *atm-of L*›) *auto*
  then show *?thesis*
    using *n mset atm False atm-L-xs uL-C*
    by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def*
        intro!: *mset-as-position.intros*)
qed
qed

definition *outlearned-add*
  :: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
  ‹*outlearned-add* = (λ*M L zs outl*.
    (*if get-level M L* < *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
        else outl))›

definition *clvls-add*
  :: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
  ‹*clvls-add* = (λ*M L zs clvls*.
    (*if get-level M L* = *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
        else clvls))›

definition *lookup-conflict-merge*
  :: ‹*nat* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
      out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
  ‹*lookup-conflict-merge init M D* = (λ(*b*, *xs*) *clvls lbd outl*. *do* {
   (-, *clvls*, *zs*, *lbd*, *outl*) ← $WHILE_T$ λ(*i*::*nat*, *clvls* :: *nat*, *zs*, *lbd*, *outl*).   *length* (*snd zs*) = *length* (*snd xs*) ∧   *Su*
     (λ(*i* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *i* < *length-uint32-nat D*)
     (λ(*i* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *do* {
        ASSERT(*i* < *length-uint32-nat D*);
        ASSERT(*Suc i* ≤ *uint32-max*);
        let lbd = *lbd-write lbd* (*get-level M* (*D*!*i*));

ASSERT(¬*is-in-lookup-conflict zs* (*D*!*i*) ⟶ *length outl* < *uint32-max*);
let outl = *outlearned-add M* (*D*!*i*) *zs outl*;
let clvls = *clvls-add M* (*D*!*i*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*D*!*i*) *zs*;
RETURN(*Suc i*, *clvls*, *zs*, *lbd*, *outl*)
})
(*init*, *clvls*, *xs*, *lbd*, *outl*);
RETURN ((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

definition *resolve-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*resolve-lookup-conflict-aa M N i xs clvls lbd outl* =
lookup-conflict-merge 1 M (*N* ∝ *i*) *xs clvls lbd outl*›

definition *set-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒(*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*set-lookup-conflict-aa M C i xs clvls lbd outl* =
lookup-conflict-merge 0 M (*C*∝*i*) *xs clvls lbd outl*›

definition *isa-outlearned-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
‹*isa-outlearned-add* = (λ*M L zs outl*.
(*if get-level-pol M L* < *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
else outl))›

lemma *isa-outlearned-add-outlearned-add*:
‹(*M*′, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-outlearned-add M′ *L zs outl* = *outlearned-add M L zs outl*›
by (*auto simp*: *isa-outlearned-add-def outlearned-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

definition *isa-clvls-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*isa-clvls-add* = (λ*M L zs clvls*.
(*if get-level-pol M L* = *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
else clvls))›

lemma *isa-clvls-add-clvls-add*:
‹(*M*′, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-clvls-add M′ *L zs outl* = *clvls-add M L zs outl*›
by (*auto simp*: *isa-clvls-add-def clvls-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

definition *isa-lookup-conflict-merge*
:: ‹*nat* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*isa-lookup-conflict-merge init M N i* = (λ(*b*, *xs*) *clvls lbd outl*. *do* {
ASSERT(*arena-is-valid-clause-idx N i*);
(-, *clvls*, *zs*, *lbd*, *outl*) ← $WHILE_T{}^{\lambda(i::nat,\ clvls\ ::\ nat,\ zs,\ lbd,\ outl).}$ *length* (*snd zs*) = *length* (*snd xs*) ∧ Su

$(\lambda(j :: nat, clvls, zs, lbd, outl). j < i + arena\text{-}length\ N\ i)$
$(\lambda(j :: nat, clvls, zs, lbd, outl). do\ \{$
$ASSERT(j < length\ N);$
$ASSERT(arena\text{-}lit\text{-}pre\ N\ j);$
$ASSERT(get\text{-}level\text{-}pol\text{-}pre\ (M, arena\text{-}lit\ N\ j));$
$ASSERT(get\text{-}level\text{-}pol\ M\ (arena\text{-}lit\ N\ j) \le Suc\ (uint32\text{-}max\ div\ 2));$
$let\ lbd = lbd\text{-}write\ lbd\ (get\text{-}level\text{-}pol\ M\ (arena\text{-}lit\ N\ j));$
$ASSERT(atm\text{-}of\ (arena\text{-}lit\ N\ j) < length\ (snd\ zs));$
$ASSERT(\neg is\text{-}in\text{-}lookup\text{-}conflict\ zs\ (arena\text{-}lit\ N\ j) \longrightarrow length\ outl < uint32\text{-}max);$
$let\ outl = isa\text{-}outlearned\text{-}add\ M\ (arena\text{-}lit\ N\ j)\ zs\ outl;$
$let\ clvls = isa\text{-}clvls\text{-}add\ M\ (arena\text{-}lit\ N\ j)\ zs\ clvls;$
$let\ zs = add\text{-}to\text{-}lookup\text{-}conflict\ (arena\text{-}lit\ N\ j)\ zs;$
$RETURN(Suc\ j, clvls, zs, lbd, outl)$
$\})$
$(i{+}init, clvls, xs, lbd, outl);$
$RETURN\ ((False, zs), clvls, lbd, outl)$
$\})$›

lemma *isa-lookup-conflict-merge-lookup-conflict-merge-ext*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#\ dom\text{-}m\ N$› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset ‘# ran-mf N*)› **and**
bxs: ‹$((b, xs), C) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A}$› **and**
M′M: ‹$(M', M) \in trail\text{-}pol\ \mathcal{A}$› **and**
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isa-lookup-conflict-merge init* M' *arena i* (*b*, *xs*) *clvls lbd outl* $\le \Downarrow Id$
(*lookup-conflict-merge init M* ($N \propto i$) (*b*, *xs*) *clvls lbd outl*)›
proof −
have [*refine0*]: ‹$((i + init, clvls, xs, lbd, outl), init, clvls, xs, lbd, outl) \in$
$\{(k, l). k = l + i\} \times_r nat\text{-}rel \times_r Id \times_r Id \times_r Id$›
by *auto*
have ‹*no-dup M*›
using *assms* **by** (*auto simp*: *trail-pol-def*)
have ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
using *M′M* **by** (*auto simp*: *trail-pol-def literals-are-in-*$\mathcal{L}_{in}$*-trail-def*)
from *literals-are-in-*$\mathcal{L}_{in}$*-trail-get-level-uint32-max*[*OF bound this* ‹*no-dup M*›]
have *lev-le*: ‹*get-level M L* $\le$ *Suc* (*uint32-max div 2*)› **for** *L* **.**

show *?thesis*
unfolding *isa-lookup-conflict-merge-def lookup-conflict-merge-def prod.case*
apply *refine-vcg*
subgoal using *assms* **unfolding** *arena-is-valid-clause-idx-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *valid i* **by** (*auto simp*: *arena-lifting*)
subgoal using *valid i* **by** (*auto simp*: *arena-lifting ac-simps*)
subgoal using *valid i*
by (*auto simp*: *arena-lifting arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of - i*])
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g*
using *i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits valid M′M*
by (*auto simp*: *arena-lifting ac-simps image-image intro*!: *get-level-pol-pre*)
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g′*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*

by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol[*OF M'M*, *symmetric*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*] *lev-le*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g*
using *i literals-are-in-$\mathcal{L}_{in}$-mm-in-$\mathcal{L}_{all}$*[*of $\mathcal{A}$ N i x1*] *lits valid M'M*
using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff arena-lifting ac-simps)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g'*
using *valid i literals-are-in-$\mathcal{L}_{in}$-mm-in-$\mathcal{L}_{all}$*[*of $\mathcal{A}$ N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol[*OF M'M*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*])
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g'*
using *valid i literals-are-in-$\mathcal{L}_{in}$-mm-in-$\mathcal{L}_{all}$*[*of $\mathcal{A}$ N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol[*OF M'M*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*])
subgoal using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff get-level-get-level-pol[*OF M'M*])
done
qed

lemma (**in** −) *arena-is-valid-clause-idx-le-uint64-max*:
⟨*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *uint64-max* ⟹
bd + *arena-length be bd* ≤ *uint64-max*⟩
⟨*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *uint64-max* ⟹
bd ≤ *uint64-max*⟩
using *arena-lifting*(*10*)[*of be - - bd*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

definition *isa-set-lookup-conflict-aa* **where**
⟨*isa-set-lookup-conflict-aa* = *isa-lookup-conflict-merge 0*⟩

definition *isa-set-lookup-conflict-aa-pre* **where**
⟨*isa-set-lookup-conflict-aa-pre* =
(λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), *out*). *i* < *length N*)⟩

lemma *lookup-conflict-merge'-spec*:
assumes
o: ⟨((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
dist: ⟨*distinct D*⟩ **and**
lits: ⟨*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*mset D*)⟩ **and**
tauto: ⟨¬*tautology* (*mset D*)⟩ **and**
lits-C: ⟨*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ *C*⟩ **and**
⟨*clvls* = *card-max-lvl M C*⟩ **and**
CD: ⟨⋀*L*. *L* ∈ *set* (*drop init D*) ⟹ −*L* ∉# *C*⟩ **and**
⟨*Suc init* ≤ *uint32-max*⟩ **and**
⟨*out-learned M* (*Some C*) *outl*⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows
⟨*lookup-conflict-merge init M D* (*b*, *n*, *xs*) *clvls lbd outl* ≤
⇓(*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*merge-conflict-m-g init M D* (*Some C*))⟩
(**is** ⟨- ≤ ⇓ *?Ref ?Spec*⟩)

proof −
  define *lbd-upd* **where**
    ‹*lbd-upd lbd i* ≡ *lbd-write lbd* (*get-level M* (*D*!*i*))› **for** *lbd i*
  let *?D* = ‹*drop init D*›
  have *le-D-le-upper*[*simp*]: ‹$a < length\ D \Longrightarrow Suc\ (Suc\ a) \leq uint32\text{-}max$› **for** *a*
    using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*mset D*›] *assms* **by** (*auto simp*: *uint32-max-def*)
  have *Suc-N-uint32-max*: ‹$Suc\ n \leq uint32\text{-}max$› **and**
    size-C-uint32-max: ‹$size\ C \leq 1 + uint32\text{-}max\ div\ 2$› **and**
    clvls: ‹*clvls* = *card-max-lvl M C*› **and**
    tauto-C: ‹¬ *tautology C*› **and**
    dist-C: ‹*distinct-mset C*› **and**
    atms-le-xs: ‹$\forall L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A}).\ L < length\ xs$› **and**
    map: ‹*mset-as-position xs C*›
    using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ *C*] *mset-as-position-distinct-mset*[*of xs C*]
      lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
    unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
    by (*auto simp*: *uint32-max-def*)
  then have *clvls-uint32-max*: ‹$clvls \leq 1 + uint32\text{-}max\ div\ 2$›
    using *size-filter-mset-lesseq*[*of* ‹λ*L*. *get-level M L* = *count-decided M*› *C*]
    unfolding *uint32-max-def card-max-lvl-def* **by** *linarith*
  have [*intro*]: ‹$((b, a, ba), Some\ C) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \Longrightarrow literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ C \Longrightarrow$
      $Suc\ (Suc\ a) \leq uint32\text{-}max$› **for** *b a ba C*
    using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
      lookup-clause-rel-def uint32-max-def)
  have [*simp*]: ‹*remdups-mset C* = *C*›
    using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def*
      lookup-clause-rel-def distinct-mset-remdups-mset-id)
  have ‹¬*tautology C*›
    using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def*
      lookup-clause-rel-def)
  have ‹*distinct-mset C*›
    using *mset-as-position-distinct-mset*[*of - C*] *o*
    unfolding *option-lookup-clause-rel-def lookup-clause-rel-def* **by** *auto*
  let *?C′* = ‹λ*a*. (*mset* (*take* (*a* − *init*) (*drop init D*)) + *C*)›
  have *tauto-C′*: ‹¬ *tautology* (*?C′ a*)› **if** ‹$a \geq init$› **for** *a*
    using *that tauto tauto-C dist dist-C CD* **unfolding** *tautology-decomp′*
    by (*force dest*: *in-set-takeD in-diffD dest*: *in-set-dropD*
      simp: *distinct-mset-in-diff in-image-uminus-uminus*)

  define *I* **where**
    ‹*I xs* = (λ(*i*, *clvls*, *zs* :: *lookup-clause-rel*, *lbd* :: *lbd*, *outl* :: *out-learned*).
        length (*snd zs*) =
        length (*snd xs*) ∧
        $Suc\ i \leq uint32\text{-}max$ ∧
        $Suc\ (fst\ zs) \leq uint32\text{-}max$ ∧
        $Suc\ clvls \leq uint32\text{-}max$)›
   for *xs* :: *lookup-clause-rel*
  define *I′* **where** ‹*I′* = (λ(*i*, *clvls*, *zs*, *lbd* :: *lbd*, *outl*).
      lookup-conflict-merge′-step $\mathcal{A}$ *init M i clvls zs D C outl* ∧ $i \geq init$)›

  have *dist-D*: ‹*distinct-mset* (*mset D*)›
    using *dist* **by** *auto*
  have *dist-CD*: ‹*distinct-mset* (*C* − *mset D* − *uminus* ‘# *mset D*)›
    using ‹*distinct-mset C*› **by** *auto*
  have [*simp*]: ‹*remdups-mset* (*mset* (*drop init D*) + *C*) = *mset* (*drop init D*) ∪# *C*›
    apply (*rule distinct-mset-rempdups-union-mset*[*symmetric*])

using *dist-C dist* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*)› **for** *a*
using *lits-C lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def all-lits-of-m-def*
dest!: *in-set-takeD in-set-dropD*)
then have *size-outl-le*: ‹*size* (*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*) ≤ *Suc uint32-max div 2*› **if** ‹*a* ≥ *init*› **for** *a*
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*mset* (*take* (*a* − *init*) (*drop init D*)) ∪# *C*›]
tauto-C′[*OF that*] ‹*distinct-mset C*› *dist-D*
by (*auto simp*: *uint32-max-def*)

have
if-True-I: ‹*I x2* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*, *lbd-upd lbd′ a*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I*) **and**
if-true-I′: ‹*I′* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*, *lbd-upd lbd′ a*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I′*)
if
I: ‹*I x2 s*› **and**
I′: ‹*I′ s*› **and**
cond: ‹*case s of* (*i*, *clvls*, *zs*, *lbd*, *outl*) ⇒ *i* < *length D*› **and**
s: ‹*s* = (*a*, *ba*)› ‹*ba* = (*aa*, *baa2*)› ‹*baa2* = (*baa*, *lbdL′*)› ‹(*b*, *n*, *xs*) = (*x1*, *x2*)›
‹*lbdL′* = (*lbd′*, *outl*)› **and**
a-le-D: ‹*a* < *length D*› **and**
a-uint32-max: ‹*Suc a* ≤ *uint32-max*›
for *x1 x2 s a ba aa baa baa2 lbd′ lbdL′ outl*
proof −
have [*simp*]:
‹*s* = (*a*, *aa*, *baa*, *lbd′*, *outl*)›
‹*ba* = (*aa*, *baa*, *lbd′*, *outl*)›
‹*x2* = (*n*, *xs*)›
using *s* **by** *auto*
obtain *ab b* **where** *baa*[*simp*]: ‹*baa* = (*ab*, *b*)› **by** (*cases baa*)

have *aa*: ‹*aa* = *card-max-lvl M* (*remdups-mset* (*?C′ a*))› **and**
ocr: ‹((*False*, *ab*, *b*), *Some* (*remdups-mset* (*?C′ a*))) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*remdups-mset* (*?C′ a*))› **and**
outl: ‹*out-learned M* (*Some* (*remdups-mset* (*?C′ a*))) *outl*›
using *I′*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def*
by *auto*
have
ab: ‹*ab* = *size* (*remdups-mset* (*?C′ a*))› **and**
map: ‹*mset-as-position b* (*remdups-mset* (*?C′ a*))› **and**
‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length b*› **and**
cr: ‹((*ab*, *b*), *remdups-mset* (*?C′ a*)) ∈ *lookup-clause-rel* $\mathcal{A}$›
using *ocr* **unfolding** *option-lookup-clause-rel-def lookup-clause-rel-def*
by *auto*
have *a-init*: ‹*a* ≥ *init*›
using *I′* **unfolding** *I′-def* **by** *auto*
have ‹*size* (*card-max-lvl M* (*remdups-mset* (*?C′ a*))) ≤ *size* (*remdups-mset* (*?C′ a*))›
unfolding *card-max-lvl-def*
by *auto*
then have [*simp*]: ‹*Suc* (*Suc aa*) ≤ *uint32-max*› ‹*Suc aa* ≤ *uint32-max*›
using *size-C-uint32-max lits a-init*
simple-clss-size-upper-div2[*of* $\mathcal{A}$ ‹*remdups-mset* (*?C′ a*)›, *OF bounded*]

unfolding *uint32-max-def aa*[*symmetric*]
by (*auto simp*: *tauto-C′*)
have [*simp*]: ‹*length b* = *length xs*›
using *I* **unfolding** *I-def* **by** *simp-all*

have *ab-upper*: ‹*Suc* (*Suc ab*) ≤ *uint32-max*›
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*remdups-mset* (*?C′ a*)›]
lookup-clause-rel-not-tautolgy[*OF cr bounded*] *a-le-D lits mset-as-position-distinct-mset*[*OF map*]
unfolding *ab literals-are-in-*$\mathcal{L}_{in}$*-remdups uint32-max-def* **by** *auto*
show *?I*
using *le-D-le-upper a-le-D ab-upper a-init*
unfolding *I-def add-to-lookup-conflict-def baa clvls-add-def* **by** *auto*

have *take-Suc-a*[*simp*]: ‹*take* (*Suc a* − *init*) *?D* = *take* (*a* − *init*) *?D* @ [*D* ! *a*]›
by (*smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop*
length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
have [*simp*]: ‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
using *dist tauto a-le-D* **apply** (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc a* − *init*›],
subst append-take-drop-id[*symmetric*, *of* - ‹*Suc a* − *init*›])
apply (*subst* (*asm*) *distinct-append*, *subst nth-append*)
by (*auto simp*: *in-set-distinct-take-drop-iff*)
have [*simp*]: ‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
proof
assume ‹− *D* ! *a* ∈ *set* (*take* (*a* − *init*) (*drop init D*))›
then have (*if is-pos* (*D* ! *a*) *then Neg else Pos*) (*atm-of* (*D* ! *a*)) ∈ *set D*
by (*metis* (*no-types*) *in-set-dropD in-set-takeD uminus-literal-def*)
then show *False*
using *a-le-D tauto* **by** *force*
qed

have *D-a-notin*: ‹*D* ! *a* ∉# (*mset* (*take* (*a* − *init*) *?D*) + *uminus* ‘# *mset* (*take* (*a* − *init*) *?D*))›
by (*auto simp*: *uminus-lit-swap*[*symmetric*])
have *uD-a-notin*: ‹−*D* ! *a* ∉# (*mset* (*take* (*a* − *init*) *?D*) + *uminus* ‘# *mset* (*take* (*a* − *init*) *?D*))›
by (*auto simp*: *uminus-lit-swap*[*symmetric*])

show *?I′*
proof (*cases* ‹(*get-level M* (*D* ! *a*) = *count-decided M* ∧ ¬ *is-in-lookup-conflict baa* (*D* ! *a*))›)
case *if-cond*: *True*
have [*simp*]: ‹*D* ! *a* ∉# *C*› ‹−*D* ! *a* ∉# *C*› ‹*b* ! *atm-of* (*D* ! *a*) = *None*›
using *if-cond mset-as-position-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-nth[*OF map*, *of* ‹−*D* ! *a*›] *D-a-notin uD-a-notin*
by (*auto simp*: *is-in-lookup-conflict-def split*: *option.splits bool.splits*
dest: *in-diffD*)
have [*simp*]: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)

have *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)), *Some* (*remdups-mset* (*?C′* (*Suc a*))))
∈ *option-lookup-clause-rel* $\mathcal{A}$›
using *ocr D-a-notin uD-a-notin*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def*
by (*auto dest*: *in-diffD simp*: *minus-notin-trivial*
intro!: *mset-as-position.intros*)
have ‹*out-learned M* (*Some* (*remdups-mset* (*?C′* (*Suc a*)))) (*outlearned-add M* (*D* ! *a*) (*ab*, *b*) *outl*)›
using *D-a-notin uD-a-notin ocr lits if-cond a-init outl*
unfolding *outlearned-add-def out-learned-def*

by *auto*
then show *?I′*
using *D-a-notin uD-a-notin ocr lits if-cond a-init*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def clvls-add-def*
by (*auto simp*: *minus-notin-trivial literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
card-max-lvl-add-mset aa)
next
case *if-cond*: *False*
have *atm-D-a-le-xs*: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
have [*simp*]: ‹*D* ! *a* ∉# *C* − *add-mset* (− *D* ! *a*)
(*add-mset* (*D* ! *a*)
(*mset* (*take a D*) + *uminus* ‘# *mset* (*take a D*)))›
using *dist-C in-diffD*[*of* ‹*D* ! *a*› *C* ‹*add-mset* (− *D* ! *a*)
(*mset* (*take a D*) + *uminus* ‘# *mset* (*take a D*))›,
THEN multi-member-split]
by (*meson distinct-mem-diff-mset member-add-mset*)
have *a-init*: ‹*a* ≥ *init*›
using *I′* **unfolding** *I′-def* **by** *auto*
have *take-Suc-a*[*simp*]: ‹*take* (*Suc a* − *init*) *?D* = *take* (*a* − *init*) *?D* @ [*D* ! *a*]›
by (*smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop*
length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
have [*iff*]: ‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
using *dist tauto a-le-D*
apply (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc a* − *init*›],
subst append-take-drop-id[*symmetric*, *of* - ‹*Suc a* − *init*›])
apply (*subst* (*asm*) *distinct-append*, *subst nth-append*)
by (*auto simp*: *in-set-distinct-take-drop-iff*)
have [*simp*]: ‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
proof
assume − *D* ! *a* ∈ *set* (*take* (*a* − *init*) (*drop init D*))
then have (*if is-pos* (*D* ! *a*) *then Neg else Pos*) (*atm-of* (*D* ! *a*)) ∈ *set D*
by (*metis* (*no-types*) *in-set-dropD in-set-takeD uminus-literal-def*)
then show *False*
using *a-le-D tauto* **by** *force*
qed
have ‹*D* ! *a* ∈ *set* (*drop init D*)›
using *a-init a-le-D* **by** (*meson in-set-drop-conv-nth*)
from *CD*[*OF this*] **have** [*simp*]: ‹−*D* ! *a* ∉# *C*› **.**
consider
(*None*) ‹*b* ! *atm-of* (*D* ! *a*) = *None*› |
(*Some-in*) *i* **where** ‹*b* ! *atm-of* (*D* ! *a*) = *Some i*› **and**
‹(*if i then Pos* (*atm-of* (*D* ! *a*)) *else Neg* (*atm-of* (*D* ! *a*))) ∈# *C*›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
by (*cases* ‹*b* ! *atm-of* (*D* ! *a*)›) (*auto simp*: *is-pos-neg-not-is-pos*)
then have *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)),
Some (*remdups-mset* (*?C′* (*Suc a*)))) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
proof *cases*
case [*simp*]: *None*
have [*simp*]: ‹*D* ! *a* ∉# *C*›
using *if-cond mset-as-position-nth*[*OF map*, *of* ‹*D* ! *a*›]
if-cond mset-as-position-nth[*OF map*, *of* ‹−*D* ! *a*›]
by (*auto simp*: *is-in-lookup-conflict-def split*: *option.splits bool.splits*
dest: *in-diffD*)

have [*simp*]: ‹*atm-of* (*D* ! *a*) < *length xs*› ‹*D* ! *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF* ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› *a-le-D*] *atms-le-xs*
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)

show *ocr*: ‹((*False*, *add-to-lookup-conflict* (*D* ! *a*) (*ab*, *b*)),
Some (*remdups-mset* (*?C′* (*Suc a*)))) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
using *ocr*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def*
by (*auto dest*: *in-diffD simp*: *minus-notin-trivial*
intro!: *mset-as-position.intros*)
next
case *Some-in*
then have ‹*remdups-mset* (*?C′ a*) = *remdups-mset* (*?C′* (*Suc a*))›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›] *a-init*
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
by (*auto simp*: *is-neg-neg-not-is-neg*)
moreover
have *1*: ‹*Some i* = *Some* (*is-pos* (*D* ! *a*))›
using *if-cond mset-as-position-in-iff-nth*[*OF map*, *of* ‹*D* ! *a*›] *a-init Some-in*
if-cond mset-as-position-in-iff-nth[*OF map*, *of* ‹−*D* ! *a*›] *atm-D-a-le-xs*(*1*)
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)› ‹−*D* ! *a* ∉# *C*›
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) *?D*)›
by (*cases* ‹*D* ! *a*›) (*auto simp*: *is-neg-neg-not-is-neg*)
moreover have ‹*b*[*atm-of* (*D* ! *a*) := *Some i*] = *b*›
unfolding *1*[*symmetric*] *Some-in*(*1*)[*symmetric*] **by** *simp*
ultimately show *?thesis*
using *dist-C atms-le-xs Some-in*(*1*) *map*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def ab*
by (*auto simp*: *distinct-mset-in-diff minus-notin-trivial*
intro: *mset-as-position.intros*
simp del: *remdups-mset-singleton-sum*)
qed
have *notin-lo-in-C*: ‹¬*is-in-lookup-conflict* (*ab*, *b*) (*D* ! *a*) ⟹ *D* ! *a* ∉# *C*›
using *mset-as-position-in-iff-nth*[*OF map*, *of* ‹*Pos* (*atm-of* (*D*!*a*))›]
mset-as-position-in-iff-nth[*OF map*, *of* ‹*Neg* (*atm-of* (*D*!*a*))›] *atm-D-a-le-xs*(*1*)
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹−*D* ! *a* ∉# *C*› *a-init*
by (*cases* ‹*b* ! (*atm-of* (*D* ! *a*))›; *cases* ‹*D* ! *a*›)
(*auto simp*: *is-in-lookup-conflict-def dist-C distinct-mset-in-diff*
split: *option.splits bool.splits*
dest: *in-diffD*)
have *in-lo-in-C*: ‹*is-in-lookup-conflict* (*ab*, *b*) (*D* ! *a*) ⟹ *D* ! *a* ∈# *C*›
using *mset-as-position-in-iff-nth*[*OF map*, *of* ‹*Pos* (*atm-of* (*D*!*a*))›]
mset-as-position-in-iff-nth[*OF map*, *of* ‹*Neg* (*atm-of* (*D*!*a*))›] *atm-D-a-le-xs*(*1*)
‹− *D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹*D* ! *a* ∉ *set* (*take* (*a* − *init*) (*drop init D*))›
‹−*D* ! *a* ∉# *C*› *a-init*
by (*cases* ‹*b* ! (*atm-of* (*D* ! *a*))›; *cases* ‹*D* ! *a*›)
(*auto simp*: *is-in-lookup-conflict-def dist-C distinct-mset-in-diff*
split: *option.splits bool.splits*
dest: *in-diffD*)
moreover have ‹*out-learned M* (*Some* (*remdups-mset* (*?C′* (*Suc a*))))
(*outlearned-add M* (*D* ! *a*) (*ab*, *b*) *outl*)›
using *D-a-notin uD-a-notin ocr lits if-cond a-init outl in-lo-in-C notin-lo-in-C*
unfolding *outlearned-add-def out-learned-def*

by *auto*
ultimately show *?I′*
using *ocr lits if-cond atm-D-a-le-xs a-init*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def clvls-add-def*
by (*auto simp*: *minus-notin-trivial literals-are-in-$\mathcal{L}_{in}$-add-mset*
card-max-lvl-add-mset aa)
qed
qed
have *uL-C-if-L-C*: ‹$-L \notin\# C$› **if** ‹$L \in\# C$› **for** *L*
using *tauto-C that* **unfolding** *tautology-decomp′* **by** *blast*

have *outl-le*: ‹*length bc* < *uint32-max*›
if
‹*I x2 s*› **and**
‹*I′ s*› **and**
‹*s* = (*a*, *ba*)› **and**
‹*ba* = (*aa*, *baa*)› **and**
‹*baa* = (*ab*, *bb*)› **and**
‹*bb* = (*ac*, *bc*)› **for** *x1 x2 s a ba aa baa ab bb ac bc*
proof −
have ‹*mset* (*tl bc*) ⊆# (*remdups-mset* (*mset* (*take* (*a* −*init*) (*drop init D*)) + *C*))› **and** ‹*init* ≤ *a*›
using *that* **by** (*auto simp*: *I-def I′-def lookup-conflict-merge′-step-def Let-def out-learned-def*)
from *size-mset-mono*[*OF this*(*1*)] *this*(*2*) **show** *?thesis* **using** *size-outl-le*[*of a*] *dist-C dist-D*
by (*auto simp*: *uint32-max-def distinct-mset-rempdups-union-mset*)
qed
show *confl*: ‹*lookup-conflict-merge init M D* (*b*, *n*, *xs*) *clvls lbd outl*
≤ ⇓ *?Ref* (*merge-conflict-m-g init M D* (*Some C*))›
supply [[*goals-limit*=*1*]]
unfolding *resolve-lookup-conflict-aa-def lookup-conflict-merge-def*
distinct-mset-rempdups-union-mset[*OF dist-D dist-CD*] *I-def*[*symmetric*] *conc-fun-SPEC*
lbd-upd-def[*symmetric*] *Let-def length-uint32-nat-def merge-conflict-m-g-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*j*, -). *length D* − *j*)› **and**
I′ = *I′*])
subgoal by *auto*
subgoal
using *clvls-uint32-max Suc-N-uint32-max* ‹*Suc init* ≤ *uint32-max*›
unfolding *uint32-max-def I-def* **by** *auto*
subgoal using *assms*
unfolding *lookup-conflict-merge′-step-def Let-def option-lookup-clause-rel-def I′-def*
by (*auto simp add*: *uint32-max-def lookup-conflict-merge′-step-def option-lookup-clause-rel-def*)
subgoal by *auto*
subgoal unfolding *I-def* **by** *fast*
subgoal for *x1 x2 s a ba aa baa ab bb ac bc* **by** (*rule outl-le*)
subgoal by (*rule if-True-I*)
subgoal by (*rule if-true-I′*)
subgoal for *b′ n′ s j zs*
using *dist lits tauto*
by (*auto simp*: *option-lookup-clause-rel-def take-Suc-conv-app-nth*
literals-are-in-$\mathcal{L}_{in}$-in-$\mathcal{L}_{all}$)
subgoal using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-conflict-merge′-step-def*
Let-def I-def I′-def)
done
qed

lemma *literals-are-in-$\mathcal{L}_{in}$-mm-literals-are-in-$\mathcal{L}_{in}$*:
assumes *lits*: ‹*literals-are-in-$\mathcal{L}_{in}$-mm $\mathcal{A}$* (*mset* ‘# *ran-mf N*)› **and**

i: ‹*i* ∈# *dom-m N*›
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))›
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-def*
proof (*standard*)
fix *L*
assume ‹*L* ∈# *all-lits-of-m* (*mset* (*N* ∝ *i*))›
then have ‹*atm-of L* ∈ *atms-of-mm* (*mset* '# *ran-mf N*)›
using *i* **unfolding** *ran-m-def in-all-lits-of-m-ain-atms-of-iff*
by (*auto dest*!: *multi-member-split*)
then show ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *lits atm-of-notin-atms-of-iff in-all-lits-of-mm-ain-atms-of-iff*
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-mm-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
by *blast*
qed

lemma *isa-set-lookup-conflict*:
‹(*uncurry6 isa-set-lookup-conflict-aa*, *uncurry6 set-conflict-m*) ∈
[λ((((((*M*, *N*), *i*), *xs*), *clvls*), *lbd*), *outl*). *i* ∈# *dom-m N* ∧ *xs* = *None* ∧ *distinct* (*N* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*) ∧
¬*tautology* (*mset* (*N* ∝ *i*)) ∧ *clvls* = *0* ∧
out-learned M None outl ∧
isasat-input-bounded $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ $\times_f$
nat-rel $\times_f$ *Id*
$\times_f$ *Id* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*›
proof −
have *H*: ‹*set-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls lbd outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*set-conflict-m M N i None clvls lbd outl*)›
if
i: ‹*i* ∈# *dom-m N*› **and**
ocr: ‹((*b*, *n*, *xs*), *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
‹*clvls* = *0*› **and**
out: ‹*out-learned M None outl*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for *b n xs N i M clvls lbd outl*
proof −
have *lookup-conflict-merge-normalise*:
‹*lookup-conflict-merge 0 M C* (*b*, *zs*) = *lookup-conflict-merge 0 M C* (*False*, *zs*)›
for *M C zs*
unfolding *lookup-conflict-merge-def* **by** *auto*
have [*simp*]: ‹*out-learned M* (*Some* {#}) *outl*›
using *out* **by** (*cases outl*) (*auto simp*: *out-learned-def*)
have *T*: ‹((*False*, *n*, *xs*), *Some* {#}) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
using *ocr* **unfolding** *option-lookup-clause-rel-def* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))›
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$[*OF lits i*] **.**
then show *?thesis* **unfolding** *set-lookup-conflict-aa-def set-conflict-m-def*
using *lookup-conflict-merge'-spec*[*of False n xs* ‹{#}› $\mathcal{A}$ ‹*N*∝*i*› *0* - *0 outl lbd*] *that dist T*
by (*auto simp*: *lookup-conflict-merge-normalise uint32-max-def merge-conflict-m-g-def*)
qed

have H: ‹*isa-set-lookup-conflict-aa* M' *arena* i (b, n, *xs*) *clvls lbd outl*
$\leq \Downarrow$ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*set-conflict-m* M N i *None clvls lbd outl*)›
if
i: ‹$i \in\#$ *dom-m* N› **and**
ocr: ‹((b, n, *xs*), *None*) $\in$ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* ($N \propto i$)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* N)› **and**
tauto: ‹$\neg$*tautology* (*mset* ($N \propto i$))› **and**
‹*clvls* = *0*› **and**
out: ‹*out-learned* M *None outl*› **and**
valid: ‹*valid-arena arena* N *vdom*› **and**
$M'M$: ‹(M', M) $\in$ *trail-pol* $\mathcal{A}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for b n *xs* N i M *clvls lbd outl arena vdom* M'
unfolding *isa-set-lookup-conflict-aa-def*
apply (*rule order.trans*)
apply (*rule isa-lookup-conflict-merge-lookup-conflict-merge-ext*[*OF valid i lits ocr* $M'M$ *bounded*])
unfolding *lookup-conflict-merge-def*[*symmetric*] *set-lookup-conflict-aa-def*[*symmetric*]
by (*auto intro*: H[*OF that*(*1*−*7*,*10*)])
show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
by (*intro nres-relI WB-More-Refinement.frefI*) (*auto intro*!: H)
qed

definition *merge-conflict-m-pre* **where**
‹*merge-conflict-m-pre* $\mathcal{A}$ =
(λ((((((M, N), i), *xs*), *clvls*), *lbd*), *out*). $i \in\#$ *dom-m* N $\wedge$ *xs* $\neq$ *None* $\wedge$ *distinct* ($N \propto i$) $\wedge$
$\neg$*tautology* (*mset* ($N \propto i$)) $\wedge$
($\forall L \in$ *set* (*tl* ($N \propto i$)). $-$ $L \notin\#$ *the xs*) $\wedge$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) $\wedge$ *clvls* = *card-max-lvl* M (*the xs*) $\wedge$
out-learned M *xs out* $\wedge$ *no-dup* M $\wedge$
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* N) $\wedge$
isasat-input-bounded $\mathcal{A}$)›

definition *isa-resolve-merge-conflict-gt2* **where**
‹*isa-resolve-merge-conflict-gt2* = *isa-lookup-conflict-merge 1*›

lemma *isa-resolve-merge-conflict-gt2*:
‹(*uncurry6 isa-resolve-merge-conflict-gt2*, *uncurry6 merge-conflict-m*) $\in$
[*merge-conflict-m-pre* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, N). *valid-arena arena* N *vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$
$\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$ *Id* $\rightarrow$
$\langle$*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* $\rangle$*nres-rel*›
proof −
have $H1$: ‹*resolve-lookup-conflict-aa* M N i (b, n, *xs*) *clvls lbd outl*
$\leq \Downarrow$ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id*)
(*merge-conflict-m* M N i C *clvls lbd outl*)›
if
i: ‹$i \in\#$ *dom-m* N› **and**
ocr: ‹((b, n, *xs*), C) $\in$ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* ($N \propto i$)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* N)› **and**
lits': ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the* C)› **and**
tauto: ‹$\neg$*tautology* (*mset* ($N \propto i$))› **and**
out: ‹*out-learned* M C *outl*› **and**

```
    not-neg: ‹⋀L. L ∈ set (tl (N ∝ i)) ⟹ − L ∉# the C› and
    ‹clvls = card-max-lvl M (the C)› and
    C-None: ‹C ≠ None› and
   bounded: ‹isasat-input-bounded 𝒜›
   for b n xs N i M clvls lbd outl C
proof −
  have lookup-conflict-merge-normalise:
      ‹lookup-conflict-merge 1 M C (b, zs) = lookup-conflict-merge 1 M C (False, zs)›
    for M C zs
    unfolding lookup-conflict-merge-def by auto
  have ‹literals-are-in-ℒin 𝒜 (mset (N ∝ i))›
    using literals-are-in-ℒin-mm-literals-are-in-ℒin[OF lits i] .
  then show ?thesis unfolding resolve-lookup-conflict-aa-def merge-conflict-m-def
    using lookup-conflict-merge'-spec[of b n xs ‹the C› 𝒜 ‹N∝i› clvls M 1 outl lbd] that dist
      not-neg ocr C-None lits'
    by (auto simp: lookup-conflict-merge-normalise uint32-max-def merge-conflict-m-g-def
      drop-Suc)
qed

have H2: ‹isa-resolve-merge-conflict-gt2 M' arena i (b, n, xs) clvls lbd outl
  ≤ ⇓ (Id ×r Id)
    (resolve-lookup-conflict-aa M N i (b, n, xs) clvls lbd outl)›
  if
    i: ‹i ∈# dom-m N› and
    ocr: ‹((b, n, xs), C) ∈ option-lookup-clause-rel 𝒜› and
    dist: ‹distinct (N ∝ i)› and
    lits: ‹literals-are-in-ℒin-mm 𝒜 (mset ‘# ran-mf N)› and
    lits': ‹literals-are-in-ℒin 𝒜 (the C)› and
    tauto: ‹¬tautology (mset (N ∝ i))› and
    out: ‹out-learned M C outl› and
    not-neg: ‹⋀L. L ∈ set (tl (N ∝ i)) ⟹ − L ∉# the C› and
    ‹clvls = card-max-lvl M (the C)› and
    C-None: ‹C ≠ None› and
    valid: ‹valid-arena arena N vdom› and

     i: ‹i ∈# dom-m N› and
    dist: ‹distinct (N ∝ i)› and
    lits: ‹literals-are-in-ℒin-mm 𝒜 (mset ‘# ran-mf N)› and
    tauto: ‹¬tautology (mset (N ∝ i))› and
    ‹clvls = card-max-lvl M (the C)› and
    out: ‹out-learned M C outl› and
    bounded: ‹isasat-input-bounded 𝒜› and
    M'M: ‹(M', M) ∈ trail-pol 𝒜›
  for b n xs N i M clvls lbd outl arena vdom C M'
  unfolding isa-resolve-merge-conflict-gt2-def
  apply (rule order.trans)
  apply (rule isa-lookup-conflict-merge-lookup-conflict-merge-ext[OF valid i lits ocr M'M])
  unfolding resolve-lookup-conflict-aa-def[symmetric] set-lookup-conflict-aa-def[symmetric]
  using bounded by (auto intro: H1[OF that(1−6)])
show ?thesis
  unfolding lookup-conflict-merge-def uncurry-def
  apply (intro nres-relI frefI)
  apply clarify
  subgoal
    unfolding merge-conflict-m-pre-def
    apply (rule order-trans)
```

apply (*rule H2*; *auto*; *auto*; *fail*)
by (*auto intro*!: *H1 simp*: *merge-conflict-m-pre-def*)
done
qed

definition (**in** −) *is-in-conflict* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *bool*› **where**
[*simp*]: ‹*is-in-conflict L C* ⟷ *L* ∈# *the C*›

definition (**in** −) *is-in-lookup-option-conflict*
:: ‹*nat literal* ⇒ (*bool* × *nat* × *bool option list*) ⇒ *bool*›
where
‹*is-in-lookup-option-conflict* = (λ*L* (-, -, *xs*). *xs* ! *atm-of L* = *Some* (*is-pos L*))›

lemma *is-in-lookup-option-conflict-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-lookup-option-conflict*),
uncurry (*RETURN oo is-in-conflict*)) ∈
[λ(*L*, *C*). *C* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_r$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*Id*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
subgoal for *Lxs LC*
using *lookup-clause-rel-atm-in-iff*[*of* - ‹*snd* (*snd* (*snd Lxs*))›]
apply (*cases Lxs*)
by (*auto simp*: *is-in-lookup-option-conflict-def option-lookup-clause-rel-def*)
done

definition *conflict-from-lookup* **where**
‹*conflict-from-lookup* = (λ(*n*, *xs*). *SPEC*(λ*D*. *mset-as-position xs D* ∧ *n* = *size D*))›

lemma *Ex-mset-as-position*:
‹*Ex* (*mset-as-position xs*)›
proof (*induction* ‹*size* {#*x* ∈# *mset xs*. *x* ≠ *None*#}› *arbitrary*: *xs*)
case *0*
then have *xs*: ‹*xs* = *replicate* (*length xs*) *None*›
by (*auto simp*: *filter-mset-empty-conv dest*: *replicate-length-same*)
show *?case*
by (*subst xs*) (*auto simp*: *mset-as-position.empty intro*!: *exI*[*of* - ‹{#}›])
next
case (*Suc x*) **note** *IH* = *this*(*1*) **and** *xs* = *this*(*2*)
obtain *i* **where**
[*simp*]: ‹*i* < *length xs*› **and**
xs-i: ‹*xs* ! *i* ≠ *None*›
using *xs*[*symmetric*]
by (*auto dest*!: *size-eq-Suc-imp-elem simp*: *in-set-conv-nth*)
let *?xs* = ‹*xs* [*i* := *None*]›
have ‹*x* = *size* {#*x* ∈# *mset ?xs*. *x* ≠ *None*#}›
using *xs*[*symmetric*] *xs-i* **by** (*auto simp*: *mset-update size-remove1-mset-If*)
from *IH*[*OF this*] **obtain** *D* **where**
map: ‹*mset-as-position ?xs D*›
by *blast*
have [*simp*]: ‹*Pos i* ∉# *D*› ‹*Neg i* ∉# *D*›
using *xs-i mset-as-position-nth*[*OF map*, *of* ‹*Pos i*›]
mset-as-position-nth[*OF map*, *of* ‹*Neg i*›]
by *auto*
have [*simp*]: ‹*xs* ! *i* = *a* ⟹ *xs*[*i* := *a*] = *xs*› **for** *a*
by *auto*

have ‹*mset-as-position xs* (*add-mset* (*if the* (*xs* ! *i*) *then Pos i else Neg i*) *D*)›
  using *mset-as-position.add*[*OF map*, *of* ‹*if the* (*xs* ! *i*) *then Pos i else Neg i*› *xs*]
    xs-i[*symmetric*]
  by (*cases* ‹*xs* ! *i*›) *auto*
then show *?case* **by** *blast*
qed

lemma *id-conflict-from-lookup*:
‹(*RETURN o id*, *conflict-from-lookup*) ∈ [λ(*n*, *xs*). ∃ *D*. ((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$]$_f$ *Id* →
  ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
  (*auto simp*: *lookup-clause-rel-def conflict-from-lookup-def RETURN-RES-refine-iff*)

lemma *lookup-clause-rel-exists-le-uint32-max*:
assumes *ocr*: ‹((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and** ‹*n* > *0*› **and**
  le-i: ‹∀ *k*<*i*. *xs* ! *k* = *None*› **and** *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
  bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
  ‹∃ *j*. *j* ≥ *i* ∧ *j* < *length xs* ∧ *j* < *uint32-max* ∧ *xs* ! *j* ≠ *None*›
proof −
have
  n-D: ‹*n* = *size D*› **and**
  map: ‹*mset-as-position xs D*› **and**
  le-xs: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*›
  using *ocr* **unfolding** *lookup-clause-rel-def* **by** *auto*
have *map-empty*: ‹*mset-as-position xs* {#} ⟷ (*xs* = [] ∨ *set xs* = {*None*})›
  by (*subst mset-as-position.simps*) (*auto simp add*: *list-eq-replicate-iff*)
have *ex-not-none*: ‹∃ *j*. *j* ≥ *i* ∧ *j* < *length xs* ∧ *xs* ! *j* ≠ *None*›
proof (*rule ccontr*)
  assume ‹¬ *?thesis*›
  then have ‹*xs* = [] ∨ *set xs* = {*None*}›
    using *le-i* **by** (*fastforce simp*: *in-set-conv-nth*)
  then have ‹*mset-as-position xs* {#}›
    using *map-empty* **by** *auto*
  then show *False*
    using *mset-as-position-right-unique*[*OF map*] ‹*n* > *0*› *n-D* **by** (*cases D*) *auto*
qed
then obtain *j* **where**
   j: ‹*j* ≥ *i*›‹*j* < *length xs*›‹*xs* ! *j* ≠ *None*›
  by *blast*
let *?L* = ‹*if the* (*xs* ! *j*) *then Pos j else Neg j*›
have ‹*?L* ∈# *D*›
  using *j mset-as-position-in-iff-nth*[*OF map*, *of ?L*] **by** *auto*
then have ‹*nat-of-lit ?L* ≤ *uint32-max*›
  using *lits bounded*
  by (*auto 5 5 dest*!: *multi-member-split*[*of - D*]
      simp: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset split*: *if-splits*)
then have ‹*j* < *uint32-max*›
  by (*auto simp*: *uint32-max-def split*: *if-splits*)
then show *?thesis*
  using *j* **by** *blast*
qed

During the conflict analysis, the literal of highest level is at the beginning. During the rest of the time the conflict is *None*.

definition *highest-lit* **where**
⟨$highest\text{-}lit\ M\ C\ L \longleftrightarrow$
$(L = None \longrightarrow C = \{\#\}) \wedge$
$(L \neq None \longrightarrow get\text{-}level\ M\ (fst\ (the\ L)) = snd\ (the\ L) \wedge$
$snd\ (the\ L) = get\text{-}maximum\text{-}level\ M\ C \wedge$
$fst\ (the\ L) \in\#\ C$
$)$⟩

Conflict Minimisation **definition** *iterate-over-conflict-inv* **where**
⟨$iterate\text{-}over\text{-}conflict\text{-}inv\ M\ D_0{}' = (\lambda(D,\ D').\ D \subseteq\#\ D_0{}' \wedge D' \subseteq\#\ D)$⟩

definition *is-literal-redundant-spec* **where**
⟨$is\text{-}literal\text{-}redundant\text{-}spec\ K\ NU\ UNE\ D\ L = SPEC(\lambda b.\ b \longrightarrow$
$NU + UNE \models pm\ remove1\text{-}mset\ L\ (add\text{-}mset\ K\ D))$⟩

definition *iterate-over-conflict*
:: ⟨$'v\ literal \Rightarrow ('v,\ 'mark)\ ann\text{-}lits \Rightarrow 'v\ clauses \Rightarrow 'v\ clauses \Rightarrow\ 'v\ clause \Rightarrow$
$'v\ clause\ nres$⟩
where
⟨$iterate\text{-}over\text{-}conflict\ K\ M\ NU\ UNE\ D_0{}' = do\ \{$
$(D,\ \text{-}) \leftarrow$
$WHILE_T{}^{iterate\text{-}over\text{-}conflict\text{-}inv\ M\ D_0{}'}$
$(\lambda(D,\ D').\ D' \neq \{\#\})$
$(\lambda(D,\ D').\ do\{$
$x \leftarrow SPEC\ (\lambda x.\ x \in\#\ D');$
$red \leftarrow is\text{-}literal\text{-}redundant\text{-}spec\ K\ NU\ UNE\ D\ x;$
$if\ \neg red$
$then\ RETURN\ (D,\ remove1\text{-}mset\ x\ D')$
$else\ RETURN\ (remove1\text{-}mset\ x\ D,\ remove1\text{-}mset\ x\ D')$
$\})$
$(D_0{}',\ D_0{}');$
$RETURN\ D$
$\}$⟩

definition *minimize-and-extract-highest-lookup-conflict-inv* **where**
⟨$minimize\text{-}and\text{-}extract\text{-}highest\text{-}lookup\text{-}conflict\text{-}inv = (\lambda(D,\ i,\ s,\ outl).$
$length\ outl \leq uint32\text{-}max \wedge mset\ (tl\ outl) = D \wedge outl \neq [] \wedge i \geq 1)$⟩

type-synonym $'v\ conflict\text{-}highest\text{-}conflict = $ ⟨$('v\ literal \times nat)\ option$⟩

definition (**in** $-$) *atm-in-conflict* **where**
⟨$atm\text{-}in\text{-}conflict\ L\ D \longleftrightarrow L \in atms\text{-}of\ D$⟩

definition *atm-in-conflict-lookup* :: ⟨$nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool$⟩ **where**
⟨$atm\text{-}in\text{-}conflict\text{-}lookup = (\lambda L\ (\text{-},\ xs).\ xs\ !\ L \neq None)$⟩

definition *atm-in-conflict-lookup-pre* :: ⟨$nat \Rightarrow lookup\text{-}clause\text{-}rel \Rightarrow bool$⟩ **where**
⟨$atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre\ L\ xs \longleftrightarrow L < length\ (snd\ xs)$⟩

lemma *atm-in-conflict-lookup-atm-in-conflict*:
⟨$(uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict\text{-}lookup),\ uncurry\ (RETURN\ oo\ atm\text{-}in\text{-}conflict)) \in$
$[\lambda(L,\ xs).\ L \in atms\text{-}of\ (\mathcal{L}_{all}\ \mathcal{A})]_f\ Id \times_f lookup\text{-}clause\text{-}rel\ \mathcal{A} \to \langle bool\text{-}rel\rangle nres\text{-}rel$⟩
apply (*intro frefI nres-relI*)
subgoal for $x\ y$

using *mset-as-position-in-iff-nth*[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*Pos* (*fst x*)›]
mset-as-position-in-iff-nth[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*Neg* (*fst x*)›]
by (*cases x*; *cases y*)
(*auto simp*: *atm-in-conflict-lookup-def atm-in-conflict-def*
lookup-clause-rel-def atm-iff-pos-or-neg-lit
pos-lit-in-atms-of neg-lit-in-atms-of)
done

lemma *atm-in-conflict-lookup-pre*:
fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
assumes
‹*x1n* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*x2f*, *x2a*) $\in$ *lookup-clause-rel* $\mathcal{A}$›
shows ‹*atm-in-conflict-lookup-pre* (*atm-of x1n*) *x2f*›
proof −
show *?thesis*
using *assms*
by (*auto simp*: *lookup-clause-rel-def atm-in-conflict-lookup-pre-def atms-of-def*)
qed

definition *is-literal-redundant-lookup-spec* **where**
‹*is-literal-redundant-lookup-spec* $\mathcal{A}$ *M NU NUE D′ L s* =
SPEC(λ(*s′*, *b*). *b* ⟶ (∀ *D*. (*D′*, *D*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟶
(*mset* ‘# *mset* (*tl NU*)) + *NUE* ⊨*pm remove1-mset L D*))›

type-synonym (**in** −) *conflict-min-cach-l* = ‹*minimize-status list* × *nat list*›

definition (**in** −) *conflict-min-cach-set-removable-l*
:: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *conflict-min-cach-l nres*›
where
‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if cach* ! *L* = *SEEN-UNKNOWN then sup* @ [*L*] *else sup*)
})›

definition (**in** −) *conflict-min-cach* :: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *minimize-status*› **where**
[*simp*]: ‹*conflict-min-cach cach L* = *cach L*›

definition *lit-redundant-reason-stack2*
:: ‹′*v literal* ⇒ ′*v clauses-l* ⇒ *nat* ⇒ (*nat* × *nat* × *bool*)› **where**
‹*lit-redundant-reason-stack2 L NU C′* =
(*if length* (*NU* ∝ *C′*) > *2 then* (*C′*, *1*, *False*)
else if NU ∝ *C′* ! *0* = *L then* (*C′*, *1*, *False*)
else (*C′*, *0*, *True*))›

definition *ana-lookup-rel*
:: ‹*nat clauses-l* ⇒ ((*nat* × *nat* × *bool*) × (*nat* × *nat* × *nat* × *nat*)) *set*›
where
‹*ana-lookup-rel NU* = {((*C*, *i*, *b*), (*C′*, *k′*, *i′*, *len′*)).
C = *C′* ∧ *k′* = (*if b then 1 else 0*) ∧ *i* = *i′* ∧
len′ = (*if b then 1 else length* (*NU* ∝ *C*))}›

lemma *ana-lookup-rel-alt-def*:

‹((C, i, b), (C′, k′, i′, len′)) ∈ *ana-lookup-rel NU* ⟷
C = C′ ∧ k′ = (*if b then 1 else 0*) ∧ i = i′ ∧
len′ = (*if b then 1 else length* (NU ∝ C))›
unfolding *ana-lookup-rel-def*
by *auto*

abbreviation *ana-lookups-rel* **where**
‹*ana-lookups-rel NU* ≡ ⟨*ana-lookup-rel NU*⟩*list-rel*›

definition *ana-lookup-conv* :: ‹*nat clauses-l* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv NU* = (λ(C, i, b). (C, (*if b then 1 else 0*), i, (*if b then 1 else length* (NU ∝ C))))›

definition *get-literal-and-remove-of-analyse-wl2*
:: ‹′*v clause-l* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ ′*v literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*get-literal-and-remove-of-analyse-wl2 C analyse* =
(*let* (i, j, b) = *last analyse in*
(C ! j, *analyse*[*length analyse* − 1 := (i, j + 1, b)]))›

definition *lit-redundant-rec-wl-inv2* **where**
‹*lit-redundant-rec-wl-inv2 M NU D* =
(λ(*cach*, *analyse*, b). ∃ *analyse*′. (*analyse*, *analyse*′) ∈ *ana-lookups-rel NU* ∧
lit-redundant-rec-wl-inv M NU D (*cach*, *analyse*′, b))›

definition *mark-failed-lits-stack-inv2* **where**
‹*mark-failed-lits-stack-inv2 NU analyse* = (λ*cach*.
∃ *analyse*′. (*analyse*, *analyse*′) ∈ *ana-lookups-rel NU* ∧
mark-failed-lits-stack-inv NU analyse′ *cach*)›

definition *lit-redundant-rec-wl-lookup*
:: ‹*nat multiset* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒
- ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›
where
‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* =
$WHILE_T{}^{lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D}$
(λ(*cach*, *analyse*, b). *analyse* ≠ [])
(λ(*cach*, *analyse*, b). *do* {
ASSERT(*analyse* ≠ []);
ASSERT(*length analyse* ≤ *length M*);
let (C,k, i, *len*) = *ana-lookup-conv NU* (*last analyse*);
ASSERT(C ∈# *dom-m NU*);
ASSERT(*length* (NU ∝ C) > k); — >= 2 would work too
ASSERT (NU ∝ C ! k ∈ *lits-of-l M*);
ASSERT(NU ∝ C ! k ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (NU ∝ C)));
ASSERT(*length* (NU ∝ C) ≤ *Suc* (*uint32-max div 2*));
ASSERT(*len* ≤ *length* (NU ∝ C)); — makes the refinement easier
let C = NU ∝ C;
if i ≥ *len*
then
RETURN(*cach* (*atm-of* (C ! k) := *SEEN-REMOVABLE*), *butlast analyse*, *True*)
else do {
let (L, *analyse*) = *get-literal-and-remove-of-analyse-wl2 C analyse*;
ASSERT(L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
let b = ¬*level-in-lbd* (*get-level M L*) *lbd*;
if (*get-level M L* = 0 ∨
conflict-min-cach cach (*atm-of L*) = *SEEN-REMOVABLE* ∨

atm-in-conflict (*atm-of* *L*) *D*)
then *RETURN* (*cach*, *analyse*, *False*)
else *if* *b* $\vee$ *conflict-min-cach* *cach* (*atm-of* *L*) = *SEEN-FAILED*
then *do* {
ASSERT(*mark-failed-lits-stack-inv2* *NU* *analyse* *cach*);
cach $\leftarrow$ *mark-failed-lits-wl* *NU* *analyse* *cach*;
RETURN (*cach*, [], *False*)
}
else *do* {
ASSERT($-$ *L* $\in$ *lits-of-l* *M*);
C $\leftarrow$ *get-propagation-reason* *M* ($-L$);
case *C* *of*
Some *C* $\Rightarrow$ *do* {
ASSERT(*C* $\in\#$ *dom-m* *NU*);
ASSERT(*length* (*NU* $\propto$ *C*) $\geq$ *2*);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* $\propto$ *C*)));
ASSERT(*length* (*NU* $\propto$ *C*) $\leq$ *Suc* (*uint32-max* *div* *2*));
RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack2* ($-L$) *NU* *C*], *False*)
}
| *None* $\Rightarrow$ *do* {
ASSERT(*mark-failed-lits-stack-inv2* *NU* *analyse* *cach*);
cach $\leftarrow$ *mark-failed-lits-wl* *NU* *analyse* *cach*;
RETURN (*cach*, [], *False*)
}
}
}
})
(*cach*, *analysis*, *False*)›

lemma *lit-redundant-rec-wl-ref-butlast*:
‹*lit-redundant-rec-wl-ref* *NU* *x* $\Longrightarrow$ *lit-redundant-rec-wl-ref* *NU* (*butlast* *x*)›
by (*cases* *x* *rule*: *rev-cases*)
(*auto* *simp*: *lit-redundant-rec-wl-ref-def* *dest*: *in-set-butlastD*)

lemma *lit-redundant-rec-wl-lookup-mark-failed-lits-stack-inv*:
assumes
‹(*x*, *x'*) $\in$ *Id*› **and**
‹*case* *x* *of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*lit-redundant-rec-wl-inv* *M* *NU* *D* *x'*› **and**
‹$\neg$ *snd* (*snd* (*snd* (*last* *x1a*))) $\leq$ *fst* (*snd* (*snd* (*last* *x1a*)))› **and**
‹*get-literal-and-remove-of-analyse-wl* (*NU* $\propto$ *fst* (*last* *x1c*)) *x1c* = (*x1e*, *x2e*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x'* = (*x1*, *x2*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x* = (*x1b*, *x2b*)›
shows ‹*mark-failed-lits-stack-inv* *NU* *x2e* *x1b*›
proof $-$
show *?thesis*
using *assms*
unfolding *mark-failed-lits-stack-inv-def* *lit-redundant-rec-wl-inv-def*
lit-redundant-rec-wl-ref-def *get-literal-and-remove-of-analyse-wl-def*
by (*cases* ‹*x1a*› *rule*: *rev-cases*)
(*auto* *simp*: *elim!*: *in-set-upd-cases*)
qed

context

fixes M D $\mathcal{A}$ NU *analysis* *analysis'*
assumes
M-D: ⟨$M \models as\ CNot\ D$⟩ **and**
n-d: ⟨*no-dup M*⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*⟩ **and**
ana: ⟨(*analysis*, *analysis'*) ∈ *ana-lookups-rel NU*⟩ **and**
lits-NU: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) '# *ran-m NU*)⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
begin
lemma *ccmin-rel*:
assumes ⟨*lit-redundant-rec-wl-inv M NU D* (*cach*, *analysis'*, *False*)⟩
shows ⟨((*cach*, *analysis*, *False*), *cach*, *analysis'*, *False*)
∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}⟩
proof −
show *?thesis* **using** *ana assms* **by** *auto*
qed

context
fixes *x* :: ⟨(*nat* ⇒ *minimize-status*) × (*nat* × *nat* × *bool*) *list* × *bool*⟩ **and**
x' :: ⟨(*nat* ⇒ *minimize-status*) × (*nat* × *nat* × *nat* × *nat*) *list* × *bool*⟩
assumes *x-x'*: ⟨(*x*, *x'*) ∈ {((*cach*, *ana*, *b*), (*cach'*, *ana'*, *b'*)).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧ *b* = *b'* ∧ *cach* = *cach'* ∧
lit-redundant-rec-wl-inv M NU D (*cach*, *ana'*, *b*)}⟩
begin

lemma *ccmin-lit-redundant-rec-wl-inv2*:
assumes ⟨*lit-redundant-rec-wl-inv M NU D x'*⟩
shows ⟨*lit-redundant-rec-wl-inv2 M NU D x*⟩
using *x-x'* **unfolding** *lit-redundant-rec-wl-inv2-def*
by *auto*

context
assumes
⟨*lit-redundant-rec-wl-inv2 M NU D x*⟩ **and**
⟨*lit-redundant-rec-wl-inv M NU D x'*⟩
begin

lemma *ccmin-cond*:
fixes *x1* :: ⟨*nat* ⇒ *minimize-status*⟩ **and**
x2 :: ⟨(*nat* × *nat* × *bool*) *list* × *bool*⟩ **and**
x1a :: ⟨(*nat* × *nat* × *bool*) *list*⟩ **and**
x2a :: ⟨*bool*⟩ **and** *x1b* :: ⟨*nat* ⇒ *minimize-status*⟩ **and**
x2b :: ⟨(*nat* × *nat* × *nat* × *nat*) *list* × *bool*⟩ **and**
x1c :: ⟨(*nat* × *nat* × *nat* × *nat*) *list*⟩ **and** *x2c* :: ⟨*bool*⟩
assumes
⟨*x2* = (*x1a*, *x2a*)⟩
⟨*x* = (*x1*, *x2*)⟩
⟨*x2b* = (*x1c*, *x2c*)⟩
⟨*x'* = (*x1b*, *x2b*)⟩
shows ⟨(*x1a* ≠ []) = (*x1c* ≠ [])⟩
using *assms x-x'*
by *auto*

end

context
  assumes
    ‹*case x of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
    ‹*case x′ of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
    inv2: ‹*lit-redundant-rec-wl-inv2 M NU D x*› **and**
    ‹*lit-redundant-rec-wl-inv M NU D x′*›
begin

context
  fixes *x1* :: ‹*nat* ⇒ *minimize-status*› **and**
  x2 :: ‹(*nat* × *nat* × *nat* × *nat*) *list* × *bool*› **and**
  x1a :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and** *x2a* :: ‹*bool*› **and**
  x1b :: ‹*nat* ⇒ *minimize-status*› **and**
  x2b :: ‹(*nat* × *nat* × *bool*) *list* × *bool*› **and**
  x1c :: ‹(*nat* × *nat* × *bool*) *list*› **and**
  x2c :: ‹*bool*›
  assumes *st*:
    ‹*x2* = (*x1a*, *x2a*)›
    ‹*x′* = (*x1*, *x2*)›
    ‹*x2b* = (*x1c*, *x2c*)›
    ‹*x* = (*x1b*, *x2b*)› **and**
    x1a: ‹*x1a* ≠ []›
begin

private lemma *st*:
    ‹*x2* = (*x1a*, *x2a*)›
    ‹*x′* = (*x1*, *x1a*, *x2a*)›
    ‹*x2b* = (*x1c*, *x2a*)›
    ‹*x* = (*x1*, *x1c*, *x2a*)›
    ‹*x1b* = *x1*›
    ‹*x2c* = *x2a*› **and**
  x1c: ‹*x1c* ≠ []›
  using *st x-x′ x1a* **by** *auto*

lemma *ccmin-nempty*:
  shows ‹*x1c* ≠ []›
  using *x-x′ x1a*
  by (*auto simp*: *st*)

context
  notes -[*simp*] = *st*
  fixes *x1d* :: ‹*nat*› **and** *x2d* :: ‹*nat* × *nat* × *nat*› **and**
    x1e :: ‹*nat*› **and** *x2e* :: ‹*nat* × *nat*› **and**
    x1f :: ‹*nat*› **and**
    x2f :: ‹*nat*› **and** *x1g* :: ‹*nat*› **and**
    x2g :: ‹*nat* × *nat* × *nat*› **and**
    x1h :: ‹*nat*› **and**
    x2h :: ‹*nat* × *nat*› **and**
    x1i :: ‹*nat*› **and**
    x2i :: ‹*nat*›
  assumes
    ana-lookup-conv: ‹*ana-lookup-conv NU* (*last x1c*) = (*x1g*, *x2g*)› **and**
    last: ‹*last x1a* = (*x1d*, *x2d*)› **and**

dom: ‹*x1d* ∈# *dom-m NU*› **and**
le: ‹*x1e* < *length* (*NU* ∝ *x1d*)› **and**
in-lits: ‹*NU* ∝ *x1d* ! *x1e* ∈ *lits-of-l M*› **and**
st2:
‹*x2g* = (*x1h*, *x2h*)›
‹*x2e* = (*x1f*, *x2f*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x2h* = (*x1i*, *x2i*)›
begin

private lemma *x1g-x1d*:
‹*x1g* = *x1d*›
‹*x1h* = *x1e*›
‹*x1i* = *x1f*›
using *st2 last ana-lookup-conv x-x′ x1a last*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *ana-lookup-conv-def ana-lookup-rel-def*
list-rel-append-single-iff; *fail*)+

private definition *j* **where**
‹*j* = *fst* (*snd* (*last x1c*))›

private definition *b* **where**
‹*b* = *snd* (*snd* (*last x1c*))›

private lemma *last-x1c*[*simp*]:
‹*last x1c* = (*x1d*, *x1f*, *b*)›
using *inv2 x1a last x-x′* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def)

private lemma
ana: ‹(*x1d*, (*if b then 1 else 0*), *x1f*, (*if b then 1 else length* (*NU* ∝ *x1d*))) = (*x1d*, *x1e*, *x1f*, *x2i*)› **and**
st3:
‹*x1e* = (*if b then 1 else 0*)›
‹*x1f* = *j*›
‹*x2f* = (*if b then 1 else length* (*NU* ∝ *x1d*))›
‹*x2d* = (*if b then 1 else 0*, *j*, *if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*j* ≤ (*if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*x1d* ∈# *dom-m NU*› **and**
‹*0* < *x1d*› **and**
‹(*if b then 1 else length* (*NU* ∝ *x1d*)) ≤ *length* (*NU* ∝ *x1d*)› **and**
‹(*if b then 1 else 0*) < *length* (*NU* ∝ *x1d*)› **and**
dist: ‹*distinct* (*NU* ∝ *x1d*)› **and**
tauto: ‹¬ *tautology* (*mset* (*NU* ∝ *x1d*))›
subgoal
using *inv2 x1a last x-x′ x1c ana-lookup-conv*
unfolding *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def ana-lookup-conv-def
simp del: *x1c*)
subgoal

using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal

using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x′ x1c* **unfolding** *x1g-x1d st j-def b-def*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
done

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
using *dom* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
using *le* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
using *in-lits* **unfolding** *x1g-x1d* **.**

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›
using *lits-NU multi-member-split*[*OF x1g-dom*]
by (*auto simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)

lemma *ccmin-le-uint32-max*:
‹*length* (*NU* ∝ *x1g*) ≤ *Suc* (*uint32-max div 2*)›
using *simple-clss-size-upper-div2*[*OF bounded ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*]
dist tauto **unfolding** *x1g-x1d*
by *auto*

lemma *ccmin-in-all-lits*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1h*]
le **unfolding** *x1g-x1d* **by** *auto*

lemma *ccmin-less-length*:
shows ‹*x2i* ≤ *length* (*NU* ∝ *x1g*)›
using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *ccmin-same-cond*:
shows ‹(*x2i* ≤ *x1i*) = (*x2f* ≤ *x1f*)›

using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *list-rel-butlast*:
assumes *rel*: ‹(*xs*, *ys*) ∈ ⟨*R*⟩*list-rel*›
shows ‹(*butlast xs*, *butlast ys*) ∈ ⟨*R*⟩*list-rel*›
proof −
have ‹*length xs* = *length ys*›
using *assms list-rel-imp-same-length* **by** *blast*
then show *?thesis*
using *rel*
by (*induction xs ys rule*: *list-induct2*) (*auto split*: *nat.splits*)
qed

lemma *ccmin-set-removable*:
assumes
‹*x2i* ≤ *x1i*› **and**
‹*x2f* ≤ *x1f*› **and** ‹*lit-redundant-rec-wl-inv2 M NU D x*›
shows ‹((*x1b*(*atm-of* (*NU* ∝ *x1g* ! *x1h*) := *SEEN-REMOVABLE*), *butlast x1c*, *True*),
x1(*atm-of* (*NU* ∝ *x1d* ! *x1e*) := *SEEN-REMOVABLE*), *butlast x1a*, *True*)
∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
using *x-x'* **by** (*auto simp*: *x1g-x1d lit-redundant-rec-wl-ref-butlast lit-redundant-rec-wl-inv-def*
dest: *list-rel-butlast*)

context
assumes
le: ‹¬ *x2i* ≤ *x1i*› ‹¬ *x2f* ≤ *x1f*›
begin

context
notes -[*simp*]= *x1g-x1d st2 last*
fixes *x1j* :: ‹*nat literal*› **and** *x2j* :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and**
x1k :: ‹*nat literal*› **and** *x2k* :: ‹(*nat* × *nat* × *bool*) *list*›
assumes
rem: ‹*get-literal-and-remove-of-analyse-wl* (*NU* ∝ *x1d*) *x1a* = (*x1j*, *x2j*)› **and**
rem2:‹*get-literal-and-remove-of-analyse-wl2* (*NU* ∝ *x1g*) *x1c* = (*x1k*, *x2k*)› **and**
‹*fst* (*snd* (*snd* (*last x2j*))) ≠ *0*› **and**
ux1j-M: ‹− *x1j* ∈ *lits-of-l M*›
begin

private lemma *confl-min-last*: ‹(*last x1c*, *last x1a*) ∈ *ana-lookup-rel NU*›
using *x1a x1c x-x' rem rem2 last ana-lookup-conv* **unfolding** *x1g-x1d st2 b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

private lemma *rel*: ‹(*x1c*[*length x1c* − *Suc 0* := (*x1d*, *Suc x1f*, *b*)], *x1a*
[*length x1a* − *Suc 0* := (*x1d*, *x1e*, *Suc x1f*, *x2f*)])
∈ *ana-lookups-rel NU*›
using *x1a x1c x-x' rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

private lemma *x1k-x1j*: ‹*x1k* = *x1j*› ‹*x1j* = *NU* ∝ *x1d* ! *x1f*› **and**
x2k-x2j: ‹(*x2k*, *x2j*) ∈ *ana-lookups-rel NU*›
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
done

lemma *ccmin-x1k-all*:
shows ‹*x1k* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
unfolding *x1k-x1j*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1f*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*OF lits* ‹− *x1j* ∈ *lits-of-l M*›]
le st3 **unfolding** *x1g-x1d* **by** (*auto split*: *if-splits simp*: *x1k-x1j uminus-*$\mathcal{A}_{in}$*-iff*)

context
notes -[*simp*]= *x1k-x1j*
fixes *b* :: ‹*bool*› **and** *lbd*
assumes *b*: ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd*, *b*) ∈ *bool-rel*›
begin

private lemma *in-conflict-atm-in*:
‹− *x1e′* ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e′*) *D* ⟷ *x1e′* ∈# *D*› **for** *x1e′*
using *M-D n-d*
by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*
atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)

lemma *ccmin-already-seen*:
shows ‹(*get-level M x1k* = *0* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*) =
(*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
using *in-lits ana ux1j-M*
by (*auto simp add*: *in-conflict-atm-in*)

private lemma *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j*, *False*)›
using *x-x′ last ana-lookup-conv rem rem2 x1a x1c le*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)

(*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

lemma *ccmin-already-seen-rel*:
assumes
‹*get-level M x1k* = *0* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*› **and**
‹*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*›
shows ‹((*x1b*, *x2k*, *False*), *x1*, *x2j*, *False*)
∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
using *x2k-x2j ccmin-lit-redundant-rec-wl-inv* **by** *auto*

context
assumes
‹¬ (*get-level M x1k* = *0* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1k*) *D*)› **and**
‹¬ (*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
begin
lemma *ccmin-already-failed*:
shows ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*) =
(*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
using *b* **by** *auto*

context
assumes
‹¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*› **and**
‹*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-lbd*:
shows ‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
using *x1a x1c x2k-x2j rem rem2 x-x' le last*
unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def*
lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
unfolding *mark-failed-lits-stack-inv2-def*
apply −
apply (*rule exI*[*of* - *x2j*])
apply (*cases* ‹*x1a*› *rule*: *rev-cases*; *cases* ‹*x1c*› *rule*: *rev-cases*)
by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-wl-lbd*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
≤ ⇓ *Id*
(*mark-failed-lits-wl NU x2j x1*)›
by (*auto simp*: *mark-failed-lits-wl-def*)

lemma *ccmin-rel-lbd*:

fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
assumes ‹(*cach*, *cacha*) ∈ *Id*›
shows ‹((*cach*, [], *False*), *cacha*, [], *False*) ∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
using *x-x' assms* **by** (*auto simp*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*)

end

context
assumes
‹¬ (¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*)› **and**
‹¬ (*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
begin

lemma *ccmin-lit-in-trail*:
‹− *x1k* ∈ *lits-of-l M*›
using ‹− *x1j* ∈ *lits-of-l M*› *x1k-x1j*(*1*) **by** *blast*

lemma *ccmin-lit-eq*:
‹− *x1k* = − *x1j*›
by *auto*

context
fixes *xa* :: ‹*nat option*› **and** *x'a* :: ‹*nat option*›
assumes *xa-x'a*: ‹(*xa*, *x'a*) ∈ ⟨*nat-rel*⟩*option-rel*›
begin

lemma *ccmin-lit-eq2*:
‹(*xa*, *x'a*) ∈ *Id*›
using *xa-x'a* **by** *auto*

context
assumes
[*simp*]: ‹*xa* = *None*› ‹*x'a* = *None*›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-dec*:
‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
using *x1a x1c x2k-x2j rem rem2 x-x' le last*
unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def*
lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
unfolding *mark-failed-lits-stack-inv2-def*
apply −
apply (*rule exI*[*of - x2j*])
apply (*cases* ‹*x1a*› *rule*: *rev-cases*; *cases* ‹*x1c*› *rule*: *rev-cases*)
by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-stack-wl-dec*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
≤ ⇓ *Id*
(*mark-failed-lits-wl NU x2j x1*)›
by (*auto simp*: *mark-failed-lits-wl-def*)

lemma *ccmin-rel-dec*:
  fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
  assumes ‹(*cach*, *cacha*) ∈ *Id*›
  shows ‹((*cach*, [], *False*), *cacha*, [], *False*)
      ∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
    (*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
    b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
  using *assms* **by** (*auto simp*: *lit-redundant-rec-wl-ref-def lit-redundant-rec-wl-inv-def*)

end

context
  fixes *xb* :: ‹*nat*› **and** *x'b* :: ‹*nat*›
  assumes *H*:
    ‹*xa* = *Some xb*›
    ‹*x'a* = *Some x'b*›
    ‹(*xb*, *x'b*) ∈ *nat-rel*›
    ‹*x'b* ∈# *dom-m NU*›
    ‹*2* ≤ *length* (*NU* ∝ *x'b*)›
    ‹*x'b* > *0*›
    ‹*distinct* (*NU* ∝ *x'b*) ∧ ¬ *tautology* (*mset* (*NU* ∝ *x'b*))›
begin

lemma *ccmin-stack-pre*:
  shows ‹*xb* ∈# *dom-m NU*› ‹*2* ≤ *length* (*NU* ∝ *xb*)›
  using *H* **by** *auto*

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*:
  shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *xb*))›
  using *lits-NU multi-member-split*[*of xb* ‹*dom-m NU*›] *H*
  by (*auto simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)

lemma *ccmin-le-uint32-max-xb*:
  ‹*length* (*NU* ∝ *xb*) ≤ *Suc* (*uint32-max div 2*)›
  using *simple-clss-size-upper-div2*[*OF bounded ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*]
    H **unfolding** *x1g-x1d*
  by *auto*

private lemma *ccmin-lit-redundant-rec-wl-inv3*: ‹*lit-redundant-rec-wl-inv M NU D*
    (*x1*, *x2j* @ [*lit-redundant-reason-stack* (− *NU* ∝ *x1d* ! *x1f*) *NU x'b*], *False*)›
  using *ccmin-stack-pre H x-x' last ana-lookup-conv rem rem2 x1a x1c le*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)
    (*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
    lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
    list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

lemma *ccmin-stack-rel*:
  shows ‹((*x1b*, *x2k* @ [*lit-redundant-reason-stack2* (− *x1k*) *NU xb*], *False*), *x1*,
        x2j @ [*lit-redundant-reason-stack* (− *x1j*) *NU x'b*], *False*)
      ∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
    (*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
    b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›

using *x2k-x2j H ccmin-lit-redundant-rec-wl-inv3*
by (*auto simp*: *list-rel-append-single-iff ana-lookup-rel-alt-def*
lit-redundant-reason-stack2-def lit-redundant-reason-stack-def)

end
end
end
end
end
end
end
end
end
end
end
end

lemma *lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*:
assumes
M-D: ‹$M \models as\ CNot\ D$› **and**
n-d: ‹*no-dup M*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
‹(*analysis*, *analysis'*) $\in$ *ana-lookups-rel NU*› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* $\circ$ *fst*) '# *ran-m NU*)› **and**
‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* $\leq$
$\Downarrow$ (*Id* $\times_r$ (*ana-lookups-rel NU*) $\times_r$ *bool-rel*) (*lit-redundant-rec-wl M NU D cach analysis' lbd*)›
proof −
have *M*: ‹$\forall$ *a* $\in$ *lits-of-l M*. *a* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l lits* **by** *blast*
have [*simp*]: ‹$-$ *x1e* $\in$ *lits-of-l M* $\Longrightarrow$ *atm-in-conflict* (*atm-of x1e*) *D* $\longleftrightarrow$ *x1e* $\in\#$ *D*› **for** *x1e*
using *M-D n-d*
by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*
atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)
have [*simp*, *intro*]: ‹$-$ *x1e* $\in$ *lits-of-l M* $\Longrightarrow$ *atm-of x1e* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹*x1e* $\in$ *lits-of-l M* $\Longrightarrow$ *x1e* $\in\#$ ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹$-$ *x1e* $\in$ *lits-of-l M* $\Longrightarrow$ *x1e* $\in\#$ ($\mathcal{L}_{all}$ $\mathcal{A}$)› **for** *x1e*
using *lits atm-of-notin-atms-of-iff literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l* **apply** *blast*
using *M uminus-*$\mathcal{A}_{in}$*-iff* **by** *auto*
have [*refine-vcg*]: ‹(*a*, *b*) $\in$ *Id* $\Longrightarrow$ (*a*, *b*) $\in$ $\langle$*Id*$\rangle$*option-rel*› **for** *a b* **by** *auto*
have [*refine-vcg*]: ‹*get-propagation-reason M x*
$\leq$ $\Downarrow$ ($\langle$*nat-rel*$\rangle$*option-rel*) (*get-propagation-reason M y*)› **if** ‹*x* = *y*› **for** *x y*
by (*use that* **in** *auto*)
have [*refine-vcg*]:‹*RETURN* ($\neg$ *level-in-lbd* (*get-level M L*) *lbd*) $\leq$ $\Downarrow$ *Id* (*RES UNIV*)› **for** *L*
by *auto*
have [*refine-vcg*]: ‹*mark-failed-lits-wl NU a b*
$\leq$ $\Downarrow$ *Id*
(*mark-failed-lits-wl NU a' b'*)› **if** ‹*a* = *a'*› **and** ‹*b* = *b'*› **for** *a a' b b'*
unfolding *that* **by** *auto*

have *H*: ‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* $\leq$
$\Downarrow$ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) $\in$ *ana-lookups-rel NU* $\wedge$
b = *b'* $\wedge$ *cach* = *cach'* $\wedge$ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}
(*lit-redundant-rec-wl M NU D cach analysis' lbd*)›

using *assms* **apply** −
unfolding *lit-redundant-rec-wl-lookup-def lit-redundant-rec-wl-def WHILET-def*
apply (*refine-vcg*)
subgoal by (*rule ccmin-rel*)
subgoal by (*rule ccmin-lit-redundant-rec-wl-inv2*)
subgoal by (*rule ccmin-cond*)
subgoal by (*rule ccmin-nempty*)
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*rule ccmin-in-dom*)
subgoal by (*rule ccmin-in-dom-le-length*)
subgoal by (*rule ccmin-in-trail*)
subgoal by (*rule ccmin-in-all-lits*)
subgoal by (*rule ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*)
subgoal by (*rule ccmin-le-uint32-max*)
subgoal by (*rule ccmin-less-length*)
subgoal by (*rule ccmin-same-cond*)
subgoal by (*rule ccmin-set-removable*)
subgoal by (*rule ccmin-x1k-all*)
subgoal by (*rule ccmin-already-seen*)
subgoal by (*rule ccmin-already-seen-rel*)
subgoal by (*rule ccmin-already-failed*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-lbd*)
apply (*rule ccmin-mark-failed-lits-wl-lbd*; *assumption*)
subgoal by (*rule ccmin-rel-lbd*)
subgoal by (*rule ccmin-lit-in-trail*)
subgoal by (*rule ccmin-lit-eq*)
subgoal by (*rule ccmin-lit-eq2*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-dec*)
apply (*rule ccmin-mark-failed-lits-stack-wl-dec*; *assumption*)
subgoal by (*rule ccmin-rel-dec*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*)
subgoal by (*rule ccmin-le-uint32-max-xb*)
subgoal by (*rule ccmin-stack-rel*)
done
show *?thesis*
by (*rule H*[*THEN order-trans*], *rule conc-fun-R-mono*)
auto
qed

definition *literal-redundant-wl-lookup* **where**
‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* = *do* {
ASSERT($L \in\# \mathcal{L}_{all}\ \mathcal{A}$);
if get-level M L = *0* ∨ *cach* (*atm-of L*) = *SEEN-REMOVABLE*
then RETURN (*cach*, [], *True*)
else if cach (*atm-of L*) = *SEEN-FAILED*
then RETURN (*cach*, [], *False*)
else do {
ASSERT(−*L* ∈ *lits-of-l M*);
C ← *get-propagation-reason M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*C* ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ *C*) ≥ *2*);

$ASSERT(literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (NU \propto C)));$
$ASSERT(distinct\ (NU \propto C) \wedge \neg tautology\ (mset\ (NU \propto C)));$
$ASSERT(length\ (NU \propto C) \leq Suc\ (uint32\text{-}max\ div\ 2));$
lit-redundant-rec-wl-lookup $\mathcal{A}$ *M NU D cach* [*lit-redundant-reason-stack2* (−*L*) *NU C*] *lbd*
}
| *None* ⇒ *do* {
RETURN (*cach*, [], *False*)
}
}
}›

lemma *literal-redundant-wl-lookup-literal-redundant-wl*:
assumes ‹*M* ⊨*as CNot D*› ‹*no-dup M*› ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**
‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* ≤
⇓ (*Id* $\times_f$ (*ana-lookups-rel NU* $\times_f$ *bool-rel*)) (*literal-redundant-wl M NU D cach L lbd*)›
proof −
have *M*: ‹∀ *a* ∈ *lits-of-l M*. *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l assms* **by** *blast*
have [*simp*, *intro*!]: ‹− *x1e* ∈ *lits-of-l M* ⟹ *atm-of x1e* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹− *x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)› **for** *x1e*
using *assms atm-of-notin-atms-of-iff literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l* **apply** *blast*
using *M uminus-*$\mathcal{A}_{in}$*-iff* **by** *auto*
have [*refine*]: ‹(*x*, *x*′) ∈ *Id* ⟹ (*x*, *x*′) ∈ ⟨*Id*⟩*option-rel*› **for** *x x*′
by *auto*
have [*refine-vcg*]: ‹*get-propagation-reason M x*
≤ ⇓ ({(*C*, *C*′). (*C*, *C*′) ∈ ⟨*nat-rel*⟩*option-rel*})
(*get-propagation-reason M y*)› **if** ‹*x* = *y*› **and** ‹*y* ∈ *lits-of-l M*› **for** *x y*
by (*use that* **in** ‹*auto simp*: *get-propagation-reason-def intro*: *RES-refine*›)
show *?thesis*
unfolding *literal-redundant-wl-lookup-def literal-redundant-wl-def*
apply (*refine-vcg lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
using *assms* **by** (*auto dest*!: *multi-member-split simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*mset* (*NU* ∝ -)›] **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *lit-redundant-reason-stack2-def lit-redundant-reason-stack-def*
ana-lookup-rel-def)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
done
qed

definition (**in** −) *lookup-conflict-nth* **where**
[*simp*]: ‹*lookup-conflict-nth* = (λ(-, *xs*) *i*. *xs* ! *i*)›

definition (**in** −) *lookup-conflict-size* **where**
[*simp*]: ‹*lookup-conflict-size* = (λ(*n*, *xs*). *n*)›

definition (**in** −) *lookup-conflict-upd-None* **where**
[*simp*]: ‹*lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. (*n*−*1*, *xs* [*i* :=*None*]))›

definition *minimize-and-extract-highest-lookup-conflict*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒ (*nat* ⇒ *minimize-status*) ⇒ *lbd* ⇒
out-learned ⇒ (*nat clause* × (*nat* ⇒ *minimize-status*) × *out-learned*) *nres*›
where
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ = (λ*M NU nxs s lbd outl*. *do* {
(*D*, -, *s*, *outl*) ←
$WHILE_T{}^{minimize\text{-}and\text{-}extract\text{-}highest\text{-}lookup\text{-}conflict\text{-}inv}$
(λ(*nxs*, *i*, *s*, *outl*). *i* < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(*x* < *length outl*);
let L = *outl* ! *x*;
ASSERT(*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
(*s'*, -, *red*) ← *literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x*+*1*, *s'*, *outl*)
else do {
ASSERT (*delete-from-lookup-conflict-pre* $\mathcal{A}$ (*L*, *nxs*));
RETURN (*remove1-mset L nxs*, *x*, *s'*, *delete-index-and-swap outl x*)
}
})
(*nxs*, *1*, *s*, *outl*);
RETURN (*D*, *s*, *outl*)
})›

lemma *entails-uminus-filter-to-poslev-can-remove*:
assumes *NU-uL-E*: ‹*NU* ⊨*p add-mset* (− *L*) (*filter-to-poslev M' L E*)› **and**
NU-E: ‹*NU* ⊨*p E*› **and** *L-E*: ‹*L* ∈# *E*›
shows ‹*NU* ⊨*p remove1-mset L E*›
proof −
have ‹*filter-to-poslev M' L E* ⊆# *remove1-mset L E*›
by (*induction E*)
(*auto simp add*: *filter-to-poslev-add-mset remove1-mset-add-mset-If subset-mset-trans-add-mset*
intro: *diff-subset-eq-self subset-mset.dual-order.trans*)
then have ‹*NU* ⊨*p add-mset* (− *L*) (*remove1-mset L E*)›
using *NU-uL-E*
by (*meson conflict-minimize-intermediate-step mset-subset-eqD*)
moreover have ‹*NU* ⊨*p add-mset L* (*remove1-mset L E*)›
using *NU-E L-E* **by** *auto*
ultimately show *?thesis*
using *true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or*[*of NU L* ‹*remove1-mset L E*›
‹*remove1-mset L E*›]
by (*auto simp*: *true-clss-cls-add-self*)
qed

lemma *minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*:
fixes D :: ‹*nat clause*› **and** S' :: ‹*nat twl-st-l*› **and** NU :: ‹*nat clauses-l*› **and** S :: ‹*nat twl-st-wl*›
and S'' :: ‹*nat twl-st*›
defines
‹$S''' \equiv$ *state*$_W$*-of* S''›
defines
‹$M \equiv$ *get-trail-wl* S› **and**
NU: ‹$NU \equiv$ *get-clauses-wl* S› **and**
NU'*-def*: ‹$NU' \equiv$ *mset* ‘# *ran-mf* NU› **and**
NUE: ‹$NUE \equiv$ *get-unit-learned-clss-wl* S + *get-unit-init-clss-wl* S› **and**
NUS: ‹$NUS \equiv$ *get-subsumed-learned-clauses-wl* S + *get-subsumed-init-clauses-wl* S› **and**
M': ‹$M' \equiv$ *trail* S'''›
assumes
S-S': ‹$(S, S') \in$ *state-wl-l None*› **and**
S'-S'': ‹$(S', S'') \in$ *twl-st-l None*› **and**
D'-D: ‹*mset* (*tl outl*) = D› **and**
M-D: ‹M $\models$*as CNot* D› **and**
dist-D: ‹*distinct-mset* D› **and**
tauto: ‹¬*tautology* D› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M› **and**
struct-invs: ‹*twl-struct-invs* S''› **and**
add-inv: ‹*twl-list-invs* S'› **and**
cach-init: ‹*conflict-min-analysis-inv* M' s' (NU' + *NUE* + *NUS*) D› **and**
NU-P-D: ‹NU' + *NUE* + *NUS* $\models$*pm add-mset* K D› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ D› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* NU)› **and**
K: ‹K = *outl* ! *0*› **and**
outl-nempty: ‹*outl* ≠ []› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ M NU D s' *lbd outl* ≤
⇓ ({((E, s, *outl*), E'). $E = E'$ ∧ *mset* (*tl outl*) = E ∧ *outl* ! *0* = K ∧
E' ⊆# D ∧ *outl* ≠ []})
(*iterate-over-conflict* K M NU' (*NUE* + *NUS*) D)›
(**is** ‹- ≤ ⇓ *?R* -›)
proof −
let *?UE* = ‹*get-unit-learned-clss-wl* S›
let *?NE* = ‹*get-unit-init-clss-wl* S›
let *?US* = ‹*get-subsumed-learned-clauses-wl* S›
let *?NS* = ‹*get-subsumed-init-clauses-wl* S›
define N U **where**
‹$N \equiv$ *mset* ‘# *init-clss-lf* NU› **and**
‹$U \equiv$ *mset* ‘# *learned-clss-lf* NU›
obtain E **where**
S''': ‹$S''' = (M', N$ + *?NE* + *?NS*, U + *?UE* + *?US*, E)›
using M' S-S' S'-S'' **unfolding** S'''*-def* *N-def* *U-def* *NU*
by (*cases* S) (*auto simp*: *state-wl-l-def twl-st-l-def*
mset-take-mset-drop-mset')
then have *NU-N-U*: ‹*mset* ‘# *ran-mf* $NU = N + U$›
using *NU* S-S' S'-S'' **unfolding** S'''*-def* *N-def* *U-def*
apply (*subst all-clss-l-ran-m*[*symmetric*])
apply (*subst image-mset-union*[*symmetric*])
apply (*subst image-mset-union*[*symmetric*])
by (*auto simp*: *mset-take-mset-drop-mset'*)
let *?NU* = ‹N + *?NE* + *?NS* + U + *?UE* + *?US*›
have NU'*-N-U*: ‹$NU' = N + U$›

  unfolding *NU′-def N-def U-def mset-append*[*symmetric*] *image-mset-union*[*symmetric*]
  by *auto*
have *NU′-NUE*: ‹*NU′* + *NUE* = *N* + *get-unit-init-clss-wl S* + *U* + *get-unit-learned-clss-wl S*›
  unfolding *NUE NU′-N-U* **by** (*auto simp*: *ac-simps*)
have *struct-inv-S‴*: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*M′*, *N* + (*?NE* + *?NS*),
        U + (*?UE* + *?US*), *E*)›
  using *struct-invs* **unfolding** *twl-struct-invs-def S‴-def*[*symmetric*] *S‴ add.assoc*
  by *fast*
then have *n-d*: ‹*no-dup M′*›
  unfolding $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-M-level-inv-def*
    trail.simps **by** *fast*
then have *n-d*: ‹*no-dup M*›
  using *S-S′ S′-S″* **unfolding** *M-def M′ S‴-def* **by** (*auto simp*: *twl-st-wl twl-st-l twl-st*)

define *R* **where**
  ‹*R* = {((*D′*:: *nat clause*, *i*, *cach* :: *nat* ⇒ *minimize-status*, *outl′* :: *out-learned*),
         (*F* :: *nat clause*, *E* :: *nat clause*)).
         i ≤ *length outl′* ∧
         F ⊆# *D* ∧
         E ⊆# *F* ∧
         mset (*drop i outl′*) = *E* ∧
         mset (*tl outl′*) = *F* ∧
         conflict-min-analysis-inv M′ cach (*?NU*) *F* ∧
         ?NU ⊨*pm add-mset K F* ∧
         mset (*tl outl′*) = *D′* ∧
         i > *0* ∧ *outl′* ≠ [] ∧
         outl′ ! *0* = *K*
     }›
have [*simp*]: ‹*add-mset K* (*mset* (*tl outl*)) = *mset outl*›
  using *D′-D K*
  by (*cases outl*) (*auto simp*: *drop-Suc outl-nempty*)
have ‹*Suc 0* < *length outl* ⟹
  highest-lit M (*mset* (*take* (*Suc 0*) (*tl outl*)))
   (*Some* (*outl* ! *Suc 0*, *get-level M* (*outl* ! *Suc 0*)))›
  using *outl-nempty*
  by (*cases outl*; *cases* ‹*tl outl*›) (*auto simp*: *highest-lit-def get-maximum-level-add-mset*)
 then have *init-args-ref*: ‹((*D*, *1*, *s′*, *outl*), *D*, *D*) ∈ *R*›
  using *D′-D cach-init NU-P-D dist-D tauto K*
  unfolding *R-def NUE NU′-def NU-N-U NUS*
  by (*auto simp*: *ac-simps drop-Suc outl-nempty ac-simps*)

 have *init-lo-inv*: ‹*minimize-and-extract-highest-lookup-conflict-inv s′*›
  if
   ‹(*s′*, *s*) ∈ *R*› **and**
   ‹*iterate-over-conflict-inv M D s*›
  for *s′ s*
 proof −
   have [*dest!*]: ‹*mset b* ⊆# *D* ⟹ *length b* ≤ *size D*› **for** *b*
     using *size-mset-mono* **by** *fastforce*
  show *?thesis*
    using *that simple-clss-size-upper-div2*[*OF bounded lits-D dist-D tauto*]
    unfolding *minimize-and-extract-highest-lookup-conflict-inv-def*
    by (*auto simp*: *R-def uint32-max-def*)
qed
have *cond*: ‹(*m* < *length outl′*) = (*D′* ≠ {#})›
  if

st′-st: ‹(*st′*, *st*) ∈ *R*› **and**
‹*minimize-and-extract-highest-lookup-conflict-inv st′*› **and**
‹*iterate-over-conflict-inv M D st*› **and**
st:
‹*x2b* = (*j*, *outl′*)›
‹*x2a* = (*m*, *x2b*)›
‹*st′* = (*nxs*, *x2a*)›
‹*st* = (*E*, *D′*)›
for *st′ st nxs x2a m x2b j x2c D′ E st2 st3 outl′*
proof −
show *?thesis*
using *st′-st* **unfolding** *st R-def*
by *auto*
qed

have *redundant*: ‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs cach*
(*outl′* ! *x1d*) *lbd*
≤ ⇓ {((*s′*, *a′*, *b′*), *b*). *b* = *b′* ∧
(*b* ⟶ *?NU* ⊨*pm remove1-mset L* (*add-mset K E*) ∧
conflict-min-analysis-inv M′ s′ ?NU (*remove1-mset L E*)) ∧
(¬*b* ⟶ *?NU* ⊨*pm add-mset K E* ∧ *conflict-min-analysis-inv M′ s′ ?NU E*)}
(*is-literal-redundant-spec K NU′* (*NUE*+*NUS*) *E L*)›
(**is** ‹- ≤ ⇓ *?red* -›)
if
R: ‹(*x*, *x′*) ∈ *R*› **and**
‹*case x′ of* (*D*, *D′*) ⇒ *D′* ≠ {#}› **and**
‹*minimize-and-extract-highest-lookup-conflict-inv x*› **and**
‹*iterate-over-conflict-inv M D x′*› **and**
st:
‹*x′* = (*E*, *x1a*)›
‹*x2d* = (*cach*, *outl′*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x* = (*nxs*, *x2c*)› **and**
L: ‹(*outl′*!*x1d*, *L*) ∈ *Id*›
‹*x1d* < *length outl′*›
for *x x′ E x2 x1a x2a nxs x2c x1d x2d x1e x2e cach highest L outl′ st3*
proof −
let *?L* = ‹(*outl′* ! *x1d*)›
have
‹*x1d* < *length outl′*› **and**
‹*x1d* ≤ *length outl′*› **and**
‹*mset* (*tl outl′*) ⊆# *D*› **and**
‹*E* = *mset* (*tl outl′*)› **and**
cach: ‹*conflict-min-analysis-inv M′ cach ?NU E*› **and**
NU-P-E: ‹*?NU* ⊨*pm add-mset K* (*mset* (*tl outl′*))› **and**
‹*nxs* = *mset* (*tl outl′*)› **and**
‹*0* < *x1d*› **and**
[*simp*]: ‹*L* = *outl′*!*x1d*› **and**
‹*E* ⊆# *D*›
‹*E* = *mset* (*tl outl′*)› **and**
‹*E* = *nxs*›
using *R L* **unfolding** *R-def st*
by *auto*

have *M-x1*: ‹*M* ⊨*as CNot E*›
by (*metis CNot-plus M-D* ‹*E* ⊆# *D*› *subset-mset.le-iff-add true-annots-union*)

then have *M′-x1*: ‹*M′* ⊨*as CNot E*›
  using *S-S′ S′-S″* **unfolding** *M′ M-def S‴-def* **by** (*auto simp*: *twl-st twl-st-wl twl-st-l*)
have ‹*outl′* ! *x1d* ∈# *E*›
  using ‹*E* = *mset* (*tl outl′*)› ‹*x1d* < *length outl′*› ‹*0* < *x1d*›
  by (*auto simp*: *nth-in-set-tl*)

have *1*:
  ‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs cach ?L lbd* ≤ ⇓ (*Id* $\times_f$ (*ana-lookups-rel NU* $\times_f$ *bool-rel*))
(*literal-redundant-wl M NU nxs cach ?L lbd*)›
  by (*rule literal-redundant-wl-lookup-literal-redundant-wl*)
   (*use lits-NU n-d lits M-x1 struct-invs bounded add-inv* ‹*outl′* ! *x1d* ∈# *E*› ‹*E* = *nxs*› **in** *auto*)

have *2*:
  ‹*literal-redundant-wl M NU nxs cach ?L lbd* ≤ ⇓
   (*Id* $\times_r$ {(*analyse*, *analyse′*). *analyse′* = *convert-analysis-list NU analyse* ∧
     lit-redundant-rec-wl-ref NU analyse} $\times_r$ *bool-rel*)
   (*literal-redundant M′ NU′ nxs cach ?L*)›
  by (*rule literal-redundant-wl-literal-redundant*[*of S S′ S″*,
       unfolded M-def[*symmetric*] *NU*[*symmetric*] *M′*[*symmetric*] *S‴-def*[*symmetric*]
       NU′-def[*symmetric*], *THEN order-trans*])
    (*use bounded S-S′ S′-S″ M-x1 struct-invs add-inv* ‹*outl′* ! *x1d* ∈# *E*› ‹*E* = *nxs*› **in**
     ‹*auto simp*: *NU*›)

have *NU-alt-def*: ‹*?NU* = *N* + (*?NE* + *?NS*) + *U* + (*?UE* + *?US*)›
    by (*auto simp*: *ac-simps*)
have *3*:
   ‹*literal-redundant M′* (*N* + *U*) *nxs cach ?L* ≤
     literal-redundant-spec M′ (*N* + *U* + (*?NE* + *?NS*) + (*?UE* + *?US*)) *nxs ?L*›
  unfolding ‹*E* = *nxs*›[*symmetric*]
  apply (*rule literal-redundant-spec*)
    apply (*rule struct-inv-S‴*)
  apply (*rule cach*[*unfolded NU-alt-def*])
   apply (*rule* ‹*outl′* ! *x1d* ∈# *E*›)
  apply (*rule M′-x1*)
  done

then have *3*:
  ‹*literal-redundant M′* (*NU′*) *nxs cach ?L* ≤ *literal-redundant-spec M′ ?NU nxs ?L*›
  by (*auto simp*: *ac-simps NU′-N-U*)

have *ent*: ‹*?NU* ⊨*pm add-mset* (− *L*) (*filter-to-poslev M′ L* (*add-mset K E*))›
  if ‹*?NU* ⊨*pm add-mset* (− *L*) (*filter-to-poslev M′ L E*)›
  using *that* **by** (*auto simp*: *filter-to-poslev-add-mset add-mset-commute*)
show *?thesis*
  apply (*rule order.trans*)
   apply (*rule 1*)
  apply (*rule order.trans*)
  apply (*rule ref-two-step′*)
   apply (*rule 2*)
   apply (*subst conc-fun-chain*)
  apply (*rule order.trans*)
   apply (*rule ref-two-step′*[*OF 3*])
  unfolding *literal-redundant-spec-def is-literal-redundant-spec-def*
     conc-fun-SPEC NU′-NUE[*symmetric*]
  apply (*rule SPEC-rule*)
  apply *clarify*

using *NU-P-E ent* ⟨$E = nxs$⟩ ⟨$E = mset\ (tl\ outl')$⟩[*symmetric*] ⟨$outl'\ !\ x1d \in\# E$⟩ *NU'-NUE*
apply (*auto intro*!: *entails-uminus-filter-to-poslev-can-remove*[*of - - M'*] *NUE NUS ac-simps*
filter-to-poslev-conflict-min-analysis-inv ac-simps simp del: *diff-union-swap2*)
apply (*smt NU'-NUE NUS add.assoc add.commute set-mset-union*)
apply (*smt NU'-NUE NUS add.assoc add.commute set-mset-union*)
done
qed

have
outl'-F: ⟨$outl'\ !\ i \in\# F$⟩ (**is** *?out*) **and**
outl'-$\mathcal{L}_{all}$: ⟨$outl'\ !\ i \in\# \mathcal{L}_{all}\ \mathcal{A}$⟩ (**is** *?out-L*)
if
R: ⟨$(S, T) \in R$⟩ **and**
⟨*case S of* $(nxs, i, s, outl) \Rightarrow i < length\ outl$⟩ **and**
⟨*case T of* $(D, D') \Rightarrow D' \neq \{\#\}$⟩ **and**
⟨*minimize-and-extract-highest-lookup-conflict-inv S*⟩ **and**
⟨*iterate-over-conflict-inv M D T*⟩ **and**
st:
⟨$T = (F', F)$⟩
⟨$S2 = (cach, outl')$⟩
⟨$S1 = (i, S2)$⟩
⟨$S = (D', S1)$⟩
⟨$i < length\ outl'$⟩
for *S T F' T1 F highest' D' S1 i S2 cach S3 highest outl'*
proof −
have *?out* **and** ⟨$F \subseteq\# D$⟩
using *R* ⟨$i < length\ outl'$⟩ **unfolding** *R-def st*
by (*auto simp*: *set-drop-conv*)
show *?out*
using ⟨*?out*⟩ **.**
then have ⟨$outl'\ !\ i \in\# D$⟩
using ⟨$F \subseteq\# D$⟩ **by** *auto*
then show *?out-L*
using *lits-D* **by** (*auto dest*!: *multi-member-split simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
qed

have
not-red: ⟨$\neg\ red \Longrightarrow ((D', i + 1, cachr, outl'), F',$
remove1-mset L F) $\in R$⟩ (**is** ⟨$- \Longrightarrow$ *?not-red*⟩) **and**
red: ⟨$\neg\ \neg\ red \Longrightarrow$
((*remove1-mset* ($outl'\ !\ i$) D', i, *cachr*, *delete-index-and-swap outl' i*),
remove1-mset L F', *remove1-mset L F*) $\in R$⟩ (**is** ⟨$- \Longrightarrow$ *?red*⟩) **and**
del: ⟨*delete-from-lookup-conflict-pre* $\mathcal{A}$ ($outl'\ !\ i$, D')⟩ (**is** *?del*)
if
R: ⟨$(S, T) \in R$⟩ **and**
⟨*case S of* $(nxs, i, s, outl) \Rightarrow i < length\ outl$⟩ **and**
⟨*case T of* $(D, D') \Rightarrow D' \neq \{\#\}$⟩ **and**
⟨*minimize-and-extract-highest-lookup-conflict-inv S*⟩ **and**
⟨*iterate-over-conflict-inv M D T*⟩ **and**
st:
⟨$T = (F', F)$⟩
⟨$S2 = (cach, outl')$⟩
⟨$S1 = (i, S2)$⟩
⟨$S = (D', S1)$⟩
⟨$cachred1 = (stack, red)$⟩
⟨$cachred = (cachr, cachred1)$⟩ **and**

⟨$i < length\ outl'$⟩ **and**
L: ⟨$(outl' \mathbin{!} i, L) \in Id$⟩ **and**
⟨$outl' \mathbin{!} i \in\# \mathcal{L}_{all}\ \mathcal{A}$⟩ **and**
$cach$: ⟨$(cachred, red') \in (?red\ F'\ L)$⟩
for $S\ T\ F'\ T1\ F\ D'\ S1\ i\ S2\ cach\ S3\ highest\ outl'\ L\ cachred\ red'\ cachr$
$cachred1\ stack\ red$
proof −
have ⟨$L = outl' \mathbin{!} i$⟩ **and**
⟨$i \leq length\ outl'$⟩ **and**
⟨$mset\ (tl\ outl') \subseteq\# D$⟩ **and**
⟨$mset\ (drop\ i\ outl') \subseteq\# mset\ (tl\ outl')$⟩ **and**
F: ⟨$F = mset\ (drop\ i\ outl')$⟩ **and**
F': ⟨$F' = mset\ (tl\ outl')$⟩ **and**
⟨$conflict\text{-}min\text{-}analysis\text{-}inv\ M'\ cach\ ?NU\ (mset\ (tl\ outl'))$⟩ **and**
⟨$?NU \models pm\ add\text{-}mset\ K\ (mset\ (tl\ outl'))$⟩ **and**
⟨$D' = mset\ (tl\ outl')$⟩ **and**
⟨$0 < i$⟩ **and**
[$simp$]: ⟨$D' = F'$⟩ **and**
$F'\text{-}D$: ⟨$F' \subseteq\# D$⟩ **and**
$F'\text{-}F$: ⟨$F \subseteq\# F'$⟩ **and**
⟨$outl' \neq []$⟩ ⟨$outl' \mathbin{!} 0 = K$⟩
using $R\ L$ **unfolding** $R\text{-}def\ st$
by $clarify+$

have [$simp$]: ⟨$L = outl' \mathbin{!} i$⟩
using L **by** $fast$
have $L\text{-}F$: ⟨$mset\ (drop\ (Suc\ i)\ outl') = remove1\text{-}mset\ L\ F$⟩
unfolding F
apply ($subst\ (2)\ Cons\text{-}nth\text{-}drop\text{-}Suc[symmetric]$)
using ⟨$i < length\ outl'$⟩ $F'\text{-}D$
by ($auto$)
have ⟨$remove1\text{-}mset\ (outl' \mathbin{!} i)\ F \subseteq\# F'$⟩
using ⟨$F \subseteq\# F'$⟩
by $auto$
have ⟨$red' = red$⟩ **and**
red: ⟨$red \longrightarrow ?NU \models pm\ remove1\text{-}mset\ L\ (add\text{-}mset\ K\ F') \wedge$
$conflict\text{-}min\text{-}analysis\text{-}inv\ M'\ cachr\ ?NU\ (remove1\text{-}mset\ L\ F')$⟩ **and**
$not\text{-}red$: ⟨$\neg\ red \longrightarrow ?NU \models pm\ add\text{-}mset\ K\ F' \wedge conflict\text{-}min\text{-}analysis\text{-}inv\ M'\ cachr\ ?NU\ F'$⟩
using $cach$
unfolding st
by $auto$
have [$simp$]: ⟨$mset\ (drop\ (Suc\ i)\ (swap\ outl'\ (Suc\ 0)\ i)) = mset\ (drop\ (Suc\ i)\ outl')$⟩
by ($subst\ drop\text{-}swap\text{-}irrelevant$) ($use$ ⟨$0 < i$⟩ **in** $auto$)
have [$simp$]: ⟨$mset\ (tl\ (swap\ outl'\ (Suc\ 0)\ i)) = mset\ (tl\ outl')$⟩
apply ($cases\ outl'$; $cases\ i$)
using ⟨$i > 0$⟩ ⟨$outl' \neq []$⟩ ⟨$i < length\ outl'$⟩
apply ($auto\ simp$: $WB\text{-}More\text{-}Refinement\text{-}List.swap\text{-}def$)
unfolding $WB\text{-}More\text{-}Refinement\text{-}List.swap\text{-}def[symmetric]$
by ($auto\ simp$:)
have [$simp$]: ⟨$mset\ (take\ (Suc\ i)\ (tl\ (swap\ outl'\ (Suc\ 0)\ i))) = mset\ (take\ (Suc\ i)\ (tl\ outl'))$⟩
using ⟨$i > 0$⟩ ⟨$outl' \neq []$⟩ ⟨$i < length\ outl'$⟩
by ($auto\ simp$: $take\text{-}tl\ take\text{-}swap\text{-}relevant\ tl\text{-}swap\text{-}relevant$)
have [$simp$]: ⟨$mset\ (take\ i\ (tl\ (swap\ outl'\ (Suc\ 0)\ i))) = mset\ (take\ i\ (tl\ outl'))$⟩
using ⟨$i > 0$⟩ ⟨$outl' \neq []$⟩ ⟨$i < length\ outl'$⟩
by ($auto\ simp$: $take\text{-}tl\ take\text{-}swap\text{-}relevant\ tl\text{-}swap\text{-}relevant$)

```
    have [simp]: ‹¬ Suc 0 < a ⟷ a = 0 ∨ a = 1› for a :: nat
      by auto
     show ?not-red if ‹¬red›
      using ‹i < length outl'› F'-D L-F ‹remove1-mset (outl' ! i) F ⊆# F'› not-red that
         ‹i > 0› ‹outl' ! 0 = K›
      by (auto simp: R-def F[symmetric] F'[symmetric]  drop-swap-irrelevant)

    have [simp]: ‹length (delete-index-and-swap outl' i) = length outl' − 1›
      by auto
    have last: ‹¬ length outl' ≤ Suc i ⟹last outl' ∈ set (drop (Suc i) outl')›
      by (metis List.last-in-set drop-eq-Nil last-drop not-le-imp-less)
    then have H: ‹mset (drop i (delete-index-and-swap outl' i)) = mset (drop (Suc i) outl')›
      using ‹i < length outl'›
      by (cases ‹drop (Suc i) outl' = []›)
        (auto simp: butlast-list-update mset-butlast-remove1-mset)
    have H': ‹mset (tl (delete-index-and-swap outl' i)) = remove1-mset (outl' ! i) (mset (tl outl'))›
      apply (rule mset-tl-delete-index-and-swap)
      using ‹i < length outl'› ‹i > 0› by fast+
    have [simp]: ‹Suc 0 < i ⟹ delete-index-and-swap outl' i ! Suc 0 = outl' ! Suc 0›
      using ‹i < length outl'› ‹i > 0›
      by (auto simp: nth-butlast)
    have ‹remove1-mset (outl' ! i) F ⊆# remove1-mset (outl' ! i) F'›
      using ‹F ⊆# F'›
      using mset-le-subtract by blast
    have [simp]: ‹delete-index-and-swap outl' i ≠ []›
      using ‹outl' ≠ []› ‹i > 0› ‹i < length outl'›
      by (cases outl') (auto simp: butlast-update'[symmetric] split: nat.splits)
    have [simp]: ‹delete-index-and-swap outl' i ! 0 = outl' ! 0›
      using  ‹outl' ! 0 = K› ‹i < length outl'› ‹i > 0›
      by (auto simp: butlast-update'[symmetric] nth-butlast)
    have ‹(outl' ! i) ∈# F'›
      using ‹i < length outl'› ‹i > 0› unfolding F' by (auto simp: nth-in-set-tl)
    then show ?red if ‹¬¬red›
      using ‹i < length outl'› F'-D L-F ‹remove1-mset (outl' ! i) F ⊆# remove1-mset (outl' ! i) F'›
        red that ‹i > 0› ‹outl' ! 0 = K› unfolding R-def
      by (auto simp: R-def F[symmetric] F'[symmetric] H H' drop-swap-irrelevant
          simp del: delete-index-and-swap.simps)

    have ‹outl' ! i ∈# ℒ_all 𝒜› ‹outl' ! i ∈# D›
      using ‹(outl' ! i) ∈# F'› F'-D lits-D
      by (force simp: literals-are-in-ℒ_in-add-mset
          dest!: multi-member-split[of ‹outl' ! i› D])+
    then show ?del
      using ‹(outl' ! i) ∈# F'› lits-D F'-D tauto
      by (auto simp: delete-from-lookup-conflict-pre-def
          literals-are-in-ℒ_in-add-mset)
  qed
  show ?thesis
    unfolding minimize-and-extract-highest-lookup-conflict-def iterate-over-conflict-def
    apply (refine-vcg WHILEIT-refine[where R = R])
    subgoal by (rule init-args-ref)
    subgoal for s' s by (rule init-lo-inv)
    subgoal by (rule cond)
    subgoal by auto
    subgoal by (rule outl'-F)
    subgoal by (rule outl'-ℒ_all)
```

apply (*rule redundant*; *assumption*)
subgoal by *auto*
subgoal by (*rule not-red*)
subgoal by (*rule del*)
subgoal
by (*rule red*)
subgoal for $x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ x2b\ x1c\ x2c$
unfolding *R-def* **by** (*cases x1b*) *auto*
done
qed

definition *cach-refinement-list*
:: ‹*nat multiset* ⇒ (*minimize-status list* × (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement-list* $\mathcal{A}_{in}$ = ⟨*Id*⟩*map-fun-rel* {(*a*, *a'*). *a* = *a'* ∧ *a* ∈# $\mathcal{A}_{in}$}›

definition *cach-refinement-nonull*
:: ‹*nat multiset* ⇒ ((*minimize-status list* × *nat list*) × *minimize-status list*) *set*›
where
‹*cach-refinement-nonull* $\mathcal{A}$ = {((*cach*, *support*), *cach'*). *cach* = *cach'* ∧
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}$}›

definition *cach-refinement*
:: ‹*nat multiset* ⇒ ((*minimize-status list* × *nat list*) × (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement* $\mathcal{A}_{in}$ = *cach-refinement-nonull* $\mathcal{A}_{in}$ *O* *cach-refinement-list* $\mathcal{A}_{in}$›

lemma *cach-refinement-alt-def*:
‹*cach-refinement* $\mathcal{A}_{in}$ = {((*cach*, *support*), *cach'*).
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
(∀ *L* ∈# $\mathcal{A}_{in}$. *L* < *length cach* ∧ *cach* ! *L* = *cach' L*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}_{in}$}›
unfolding *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*
apply (*rule*; *rule*)
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply *blast*
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply (*rule-tac b=x1a* **in** *relcomp.relcompI*)
apply *blast*
apply *blast*
done

lemma *in-cach-refinement-alt-def*:
‹((*cach*, *support*), *cach'*) ∈ *cach-refinement* $\mathcal{A}_{in}$ ⟷
(*cach*, *cach'*) ∈ *cach-refinement-list* $\mathcal{A}_{in}$ ∧
(∀ *L*<*length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}_{in}$›
by (*auto simp*: *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*)

definition (**in** −) *conflict-min-cach-l* :: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *minimize-status*› **where**
‹*conflict-min-cach-l* = (λ(*cach*, *sup*) *L*.

(*cach* ! *L*)
)›

definition *conflict-min-cach-l-pre* **where**
‹*conflict-min-cach-l-pre* = (λ((*cach*, *sup*), *L*). *L* < *length cach*)›

lemma *conflict-min-cach-l-pre*:
fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
assumes
‹*x1n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*x1l*, *x1j*) ∈ *cach-refinement* $\mathcal{A}$›
shows ‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1n*)›
proof –
show *?thesis*
using *assms* **by** (*auto simp*: *cach-refinement-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *conflict-min-cach-l-pre-def*)
qed

lemma *nth-conflict-min-cach*:
‹(*uncurry* (*RETURN oo conflict-min-cach-l*), *uncurry* (*RETURN oo conflict-min-cach*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *map-fun-rel-def*
in-cach-refinement-alt-def cach-refinement-list-def conflict-min-cach-l-def)

definition (**in** –) *conflict-min-cach-set-failed*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where
[*simp*]: ‹*conflict-min-cach-set-failed cach L* = *cach*(*L* := *SEEN-FAILED*)›

definition (**in** –) *conflict-min-cach-set-failed-l*
:: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *conflict-min-cach-l nres*›
where
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if cach* ! *L* = *SEEN-UNKNOWN then sup* @ [*L*] *else sup*)
})›

lemma *bounded-included-le*:
assumes *bounded*: ‹*isasat-input-bounded* $\mathcal{A}$› **and** ‹*distinct n*› **and** ‹*set n* ⊆ *set-mset* $\mathcal{A}$›
shows ‹*length n* ≤ *Suc* (*uint32-max div 2*)›
proof –
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* ‘# *mset n*)› **and**
dist: ‹*distinct n*›
using *assms*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def inj-on-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset n*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset n*))›
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)
qed

lemma *conflict-min-cach-set-failed*:
‹(*uncurry conflict-min-cach-set-failed-l*, *uncurry* (*RETURN oo conflict-min-cach-set-failed*)) ∈
[λ(*cach*, *L*). $L \in\# \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *cach-refinement* $\mathcal{A}_{in} \times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
apply (*intro frefI nres-relI*)
apply (*auto simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*
conflict-min-cach-set-failed-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)
by (*fastforce dest*: *subset-add-mset-notin-subset-mset*)+

definition (**in** −) *conflict-min-cach-set-removable*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where
[*simp*]: ‹*conflict-min-cach-set-removable cach L* = *cach*(*L*:= *SEEN-REMOVABLE*)›

lemma *conflict-min-cach-set-removable*:
‹(*uncurry conflict-min-cach-set-removable-l*,
uncurry (*RETURN oo conflict-min-cach-set-removable*)) ∈
[λ(*cach*, *L*). $L \in\# \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$ *cach-refinement* $\mathcal{A}_{in} \times_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
by (*intro frefI nres-relI*)
(*auto 5 5 simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*
conflict-min-cach-set-removable-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)

definition *isa-mark-failed-lits-stack* **where**
‹*isa-mark-failed-lits-stack NU analyse cach* = *do* {
let l = *length analyse*;
ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
(-, *cach*) ← $WHILE_T{}^{\lambda(\text{-},\ cach).\ True}$
(λ(*i*, *cach*). *i* < *l*)
(λ(*i*, *cach*). *do* {
ASSERT(*i* < *length analyse*);
let (*cls-idx*, *idx*, -) = (*analyse* ! *i*);
ASSERT(*cls-idx* + *idx* ≥ *1*);
ASSERT(*cls-idx* + *idx* − *1* < *length NU*);
ASSERT(*arena-lit-pre NU* (*cls-idx* + *idx* − *1*));
cach ← *conflict-min-cach-set-failed-l cach* (*atm-of* (*arena-lit NU* (*cls-idx* + *idx* − *1*)));
RETURN (*i*+*1*, *cach*)
})
(*0*, *cach*);
RETURN cach
}›

context
begin

lemma *mark-failed-lits-stack-inv-helper1*: ‹*mark-failed-lits-stack-inv a ba a2'* $\Longrightarrow$
a1' $<$ *length ba* $\Longrightarrow$
(*a1'a*, *a2'a*) = *ba* ! *a1'* $\Longrightarrow$
a1'a $\in\#$ *dom-m a*›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2'* $\Longrightarrow$
a1' $<$ *length ba* $\Longrightarrow$
(*a1'a*, *xx*, *a2'a*, *yy*) = *ba* ! *a1'* $\Longrightarrow$
a2'a $-$ *Suc 0* $<$ *length* (*a* $\propto$ *a1'a*)›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

lemma *isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*:
assumes ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
shows ‹(*uncurry2 isa-mark-failed-lits-stack*, *uncurry2* (*mark-failed-lits-stack* $\mathcal{A}_{in}$)) $\in$
[λ((*N*, *ana*), *cach*). *length ana* $\leq$ *1* + *uint32-max div 2*]$_f$
{(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *ana-lookups-rel NU* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$ $\to$
⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
proof –
have *subset-mset-add-new*: ‹*a* $\notin\#$ *A* $\Longrightarrow$ *a* $\in\#$ *B* $\Longrightarrow$ *add-mset a A* $\subseteq\#$ *B* $\longleftrightarrow$ *A* $\subseteq\#$ *B*› **for** *a A B*
by (*metis insert-DiffM insert-subset-eq-iff subset-add-mset-notin-subset*)
have [*refine0*]: ‹((*0*, *x2c*), *0*, *x2a*) $\in$ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$›
if ‹(*x2c*, *x2a*) $\in$ *cach-refinement* $\mathcal{A}_{in}$› **for** *x2c x2a*
using *that* **by** *auto*
have *le-length-arena*: ‹*x1g* + *x2g* $-$ *1* $<$ *length x1c*› (**is** *?le*) **and**
is-lit: ‹*arena-lit-pre x1c* (*x1g* + *x2g* $-$ *1*)› (**is** *?lit*) **and**
isA: ‹*atm-of* (*arena-lit x1c* (*x1g* + *x2g* $-$ *1*)) $\in\#$ $\mathcal{A}_{in}$› (**is** *?A*) **and**
final: ‹*conflict-min-cach-set-failed-l x2e*
(*atm-of* (*arena-lit x1c* (*x1g* + *x2g* $-$ *1*)))
$\leq$ *SPEC*
(λ*cach*.
RETURN (*x1e* + *1*, *cach*)
$\leq$ *SPEC*
(λ*c*. (*c*, *x1d* + *1*, *x2d*
(*atm-of* (*x1a* $\propto$ *x1f* ! (*x2f* $-$ *1*)) := *SEEN-FAILED*))
$\in$ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$))› (**is** *?final*) **and**
ge1: ‹*x1g* + *x2g* $\geq$ *1*›
if
‹*case y of* (*x*, *xa*) $\Rightarrow$ (*case x of* (*N*, *ana*) $\Rightarrow$ λ*cach*. *length ana* $\leq$ *1* + *uint32-max div 2*) *xa*› **and**
xy: ‹(*x*, *y*) $\in$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *ana-lookups-rel NU*
$\times_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**
st:
‹*x1* = (*x1a*, *x2*)›
‹*y* = (*x1*, *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = (*x1b*, *x2c*)›
‹*x'* = (*x1d*, *x2d*)›
‹*xa* = (*x1e*, *x2e*)›
‹*x2f2* = (*x2f*, *x2f3*)›
‹*x2f0* = (*x2f1*, *x2f2*)›
‹*x2* ! *x1d* = (*x1f*, *x2f0*)›
‹*x2g0* = (*x2g*, *x2g2*)›
‹*x2b* ! *x1e* = (*x1g*, *x2g0*)› **and**
xax': ‹(*xa*, *x'*) $\in$ *nat-rel* $\times_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**

cond: ‹*case xa of* (*i*, *cach*) ⇒ *i* < *length x2b*› **and**
cond': ‹*case x' of* (*i*, *cach*) ⇒ *i* < *length x2*› **and**
inv: ‹*case x' of* (-, *x*) ⇒ *mark-failed-lits-stack-inv x1a x2 x*› **and**
le: ‹*x1d* < *length x2*› ‹*x1e* < *length x2b*› **and**
atm: ‹*atm-of* (*x1a* ∝ *x1f* ! (*x2f* − *1*)) ∈# $\mathcal{A}_{in}$›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c xa x' x1d x2d x1e x2e x1f x2f x1g x2g*
x2f0 x2f1 x2f2 x2f3 x2g0 x2g1 x2g2 x2g3
proof −
obtain *i cach* **where** *x'*: ‹*x'* = (*i*, *cach*)› **by** (*cases x'*)
have [*simp*]:
‹*x1* = (*x1a*, *x2*)›
‹*y* = ((*x1a*, *x2*), *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = ((*x1c*, *x2b*), *x2c*)›
‹*x'* = (*x1d*, *x2d*)›
‹*xa* = (*x1d*, *x2e*)›
‹*x1f* = *x1g*›
‹*x1e* = *x1d*›
‹*x2f0* = (*x2f1*, *x2f*, *x2f3*)›
‹*x2g* = *x2f*›
‹*x2g0* = (*x2g*, *x2g2*)› **and**
st': ‹*x2* ! *x1d* = (*x1g*, *x2f0*)› **and**
cach:‹(*x2e*, *x2d*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **and**
‹(*x2c*, *x2a*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **and**
x2f0-x2g0: ‹((*x1g*, *x2g*, *x2g2*), (*x1f*, *x2f1*, *x2f*, *x2f3*)) ∈ *ana-lookup-rel NU*›
using *xy st xax' param-nth*[*of x1e x2 x1d x2b* ‹*ana-lookup-rel NU*›] *le*
by (*auto intro*: *simp*: *ana-lookup-rel-alt-def*)

have *arena*: ‹*valid-arena x1c x1a vdom*›
using *xy* **unfolding** *st* **by** *auto*
have ‹*x2* ! *x1e* ∈ *set x2*›
using *le*
by *auto*
then have ‹*x2* ! *x1d* ∈ *set x2*› **and**
x2f: ‹*x2f* ≤ *length* (*x1a* ∝ *x1f*)› **and**
x1f: ‹*x1g* ∈# *dom-m x1a*› **and**
x2g: ‹*x2g* > *0*› **and**
x2g-u1-le: ‹*x2g* − *1* < *length* (*x1a* ∝ *x1f*)›
using *inv le x2f0-x2g0 nth-mem*[*of x1d x2*]
unfolding *mark-failed-lits-stack-inv-def x' prod.case st st'*
by (*auto simp del*: *nth-mem simp*: *st' ana-lookup-rel-alt-def split*: *if-splits*
dest!: *bspec*[*of* ‹*set x2*› - ‹(-, -, -, -)›])

have ‹*is-Lit* (*x1c* ! (*x1g* + (*x2g* − *1*)))›
by (*rule arena-lifting*[*OF arena x1f*]) (*use x2f x2g x2g-u1-le* **in** *auto*)
then show *?le* **and** *?A*
using *arena-lifting*[*OF arena x1f*] *le x2f x1f x2g atm x2g-u1-le*
by (*auto simp*: *arena-lit-def*)
show *?lit*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of* - *x1f*])
(*use arena x1f x2f x2g x2g-u1-le* **in** ‹*auto intro*!: *exI*[*of* - *x1a*] *exI*[*of* - *vdom*]›)
show ‹*x1g* + *x2g* ≥ *1*›
using *x2g* **by** *auto*
have [*simp*]: ‹*arena-lit x1c* (*x1g* + *x2g* − *Suc 0*) = *x1a* ∝ *x1g* ! (*x2g* − *Suc 0*)›
using *that x1f x2f x2g x2g-u1-le* **by** (*auto simp*: *arena-lifting*[*OF arena*])

have ‹*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *Suc 0*)) < *length* (*fst x2e*)›
using ‹*?A*› *cach* **by** (*auto simp*: *cach-refinement-alt-def dest*: *multi-member-split*)

then show *?final*
using ‹*?le*› ‹*?A*› *cach x1f x2g-u1-le x2g assms*
apply −
apply (*rule conflict-min-cach-set-failed*[*of* $\mathcal{A}_{in}$, *THEN fref-to-Down-curry*, *THEN order-trans*])
by (*cases x2e*)
(*auto simp*: *cach-refinement-alt-def RETURN-def conc-fun-RES*
arena-lifting[*OF arena*] *subset-mset-add-new*)
qed

show *?thesis*
unfolding *isa-mark-failed-lits-stack-def mark-failed-lits-stack-def uncurry-def*
apply (*rewrite at* ‹*let - = length - in -*› *Let-def*)
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c xa x′ x1d x2d x1e x2e*
by (*auto simp*: *list-rel-imp-same-length*)
subgoal by *auto*
subgoal by (*rule ge1*)
subgoal by (*rule le-length-arena*)
subgoal
by (*rule is-lit*)
subgoal
by (*rule final*)
subgoal by *auto*
done
qed

definition *isa-get-literal-and-remove-of-analyse-wl*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *nat literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl C analyse* =
(*let* (*i*, *j*, *b*) = (*last analyse*) *in*
(*arena-lit C* (*i* + *j*), *analyse*[*length analyse* − *1* := (*i*, *j* + *1*, *b*)]))›

definition *isa-get-literal-and-remove-of-analyse-wl-pre*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *bool*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ⟷
(*let* (*i*, *j*, *b*) = *last analyse in*
analyse ≠ [] ∧ *arena-lit-pre arena* (*i*+*j*) ∧ *j* < *uint32-max*)›

lemma *arena-lit-pre-le*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* ≤ *uint64-max*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *uint64-max* ⟹
arena-lit-pre a i ⟹ *i* < *uint64-max*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

definition *lit-redundant-reason-stack-wl-lookup-pre* :: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *bool*› **where**
‹*lit-redundant-reason-stack-wl-lookup-pre L NU C* ⟷
  arena-lit-pre NU C ∧
  arena-is-valid-clause-idx NU C›

definition *lit-redundant-reason-stack-wl-lookup*
  :: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *nat* × *nat* × *bool*›
where
‹*lit-redundant-reason-stack-wl-lookup L NU C* =
  (*if arena-length NU C* > *2 then* (*C*, *1*, *False*)
  else if arena-lit NU C = *L*
  then (*C*, *1*, *False*)
  else (*C*, *0*, *True*))›

definition *ana-lookup-conv-lookup* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv-lookup NU* = (λ(*C*, *i*, *b*).
  (*C*, (*if b then 1 else 0*), *i*, (*if b then 1 else arena-length NU C*)))›

definition *ana-lookup-conv-lookup-pre* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ *bool*› **where**
‹*ana-lookup-conv-lookup-pre NU* = (λ(*C*, *i*, *b*). *arena-is-valid-clause-idx NU C*)›

definition *isa-lit-redundant-rec-wl-lookup*
  :: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒
    - ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›
where
  ‹*isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd* =
      $WHILE_T{}^{\lambda\text{-}.\ True}$
        (λ(*cach*, *analyse*, *b*). *analyse* ≠ [])
        (λ(*cach*, *analyse*, *b*). *do* {
            ASSERT(*analyse* ≠ []);
            ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
            ASSERT(*arena-is-valid-clause-idx NU* (*fst* (*last analyse*)));
    ASSERT(*ana-lookup-conv-lookup-pre NU* ((*last analyse*)));
    let (*C*, *k*, *i*, *len*) = *ana-lookup-conv-lookup NU* ((*last analyse*));
            ASSERT(*C* < *length NU*);
            ASSERT(*arena-is-valid-clause-idx NU C*);
            ASSERT(*arena-lit-pre NU* (*C* + *k*));
            if i ≥ *len*
            then do {
      cach ← *conflict-min-cach-set-removable-l cach* (*atm-of* (*arena-lit NU* (*C* + *k*)));
              RETURN(*cach*, *butlast analyse*, *True*)
    }
            else do {
      ASSERT (*isa-get-literal-and-remove-of-analyse-wl-pre NU analyse*);
      let (*L*, *analyse*) = *isa-get-literal-and-remove-of-analyse-wl NU analyse*;
              ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
      ASSERT(*get-level-pol-pre* (*M*, *L*));
      let b = ¬*level-in-lbd* (*get-level-pol M L*) *lbd*;
      ASSERT(*atm-in-conflict-lookup-pre* (*atm-of L*) *D*);
      ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
      if (*get-level-pol M L* = *0* ∨
   conflict-min-cach-l cach (*atm-of L*) = *SEEN-REMOVABLE* ∨
   atm-in-conflict-lookup (*atm-of L*) *D*)
      then RETURN (*cach*, *analyse*, *False*)
      else if b ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-FAILED*
      then do {

cach ← *isa-mark-failed-lits-stack NU analyse cach*;
RETURN (*cach*, *take 0 analyse*, *False*)
}
else do {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*lit-redundant-reason-stack-wl-lookup-pre* (−*L*) *NU C*);
RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack-wl-lookup* (−*L*) *NU C*], *False*)
}
| *None* ⇒ *do* {
cach ← *isa-mark-failed-lits-stack NU analyse cach*;
RETURN (*cach*, *take 0 analyse*, *False*)
}
}
}
})
(*cach*, *analysis*, *False*)⟩

lemma *isa-lit-redundant-rec-wl-lookup-alt-def*:
⟨*isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd* =
$WHILE_T{}^{\lambda\text{-}.\ True}$
(λ(*cach*, *analyse*, *b*). *analyse* ≠ [])
(λ(*cach*, *analyse*, *b*). *do* {
ASSERT(*analyse* ≠ []);
ASSERT(*length analyse* ≤ *1* + *uint32-max div 2*);
let (*C*, *i*, *b*) = *last analyse*;
ASSERT(*arena-is-valid-clause-idx NU* (*fst* (*last analyse*)));
ASSERT(*ana-lookup-conv-lookup-pre NU* (*last analyse*));
let (*C*, *k*, *i*, *len*) = *ana-lookup-conv-lookup NU* ((*C*, *i*, *b*));
ASSERT(*C* < *length NU*);
let - = *map xarena-lit*
((*Misc.slice*
C
(*C* + *arena-length NU C*))
NU);
ASSERT(*arena-is-valid-clause-idx NU C*);
ASSERT(*arena-lit-pre NU* (*C* + *k*));
if i ≥ *len*
then do {
cach ← *conflict-min-cach-set-removable-l cach* (*atm-of* (*arena-lit NU* (*C* + *k*)));
RETURN(*cach*, *butlast analyse*, *True*)
}
else do {
ASSERT (*isa-get-literal-and-remove-of-analyse-wl-pre NU analyse*);
let (*L*, *analyse*) = *isa-get-literal-and-remove-of-analyse-wl NU analyse*;
ASSERT(*length analyse* ≤ *1*+ *uint32-max div 2*);
ASSERT(*get-level-pol-pre* (*M*, *L*));
let b = ¬*level-in-lbd* (*get-level-pol M L*) *lbd*;
ASSERT(*atm-in-conflict-lookup-pre* (*atm-of L*) *D*);
ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
if (*get-level-pol M L* = *0* ∨
conflict-min-cach-l cach (*atm-of L*) = *SEEN-REMOVABLE* ∨
atm-in-conflict-lookup (*atm-of L*) *D*)
then RETURN (*cach*, *analyse*, *False*)
else if b ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-FAILED*

```
            then do {
              cach ← isa-mark-failed-lits-stack NU analyse cach;
              RETURN (cach, [], False)
            }
            else do {
              C ← get-propagation-reason-pol M (−L);
              case C of
                Some C ⇒ do {
    ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
    RETURN (cach, analyse @ [lit-redundant-reason-stack-wl-lookup (−L) NU C], False)
  }
              | None ⇒ do {
                  cach ← isa-mark-failed-lits-stack NU analyse cach;
                  RETURN (cach, [], False)
              }
          }
        }
      })
      (cach, analysis, False)⟩
  unfolding isa-lit-redundant-rec-wl-lookup-def Let-def take-0
  by (auto simp: Let-def)
```

```
lemma lit-redundant-rec-wl-lookup-alt-def:
  ⟨lit-redundant-rec-wl-lookup 𝒜 M NU D cach analysis lbd =
      WHILE_T^{lit-redundant-rec-wl-inv2 M NU D}
        (λ(cach, analyse, b). analyse ≠ [])
        (λ(cach, analyse, b). do {
            ASSERT(analyse ≠ []);
            ASSERT(length analyse ≤ length M);
    let (C, k, i, len) = ana-lookup-conv NU (last analyse);
            ASSERT(C ∈# dom-m NU);
            ASSERT(length (NU ∝ C) > k); — >= 2 would work too
            ASSERT (NU ∝ C ! k ∈ lits-of-l M);
            ASSERT(NU ∝ C ! k ∈# 𝓛_all 𝒜);
    ASSERT(literals-are-in-𝓛_in 𝒜 (mset (NU ∝ C)));
    ASSERT(length (NU ∝ C) ≤ Suc (uint32-max div 2));
    ASSERT(len ≤ length (NU ∝ C)); — makes the refinement easier
    let (C,k, i, len) = (C,k,i,len);
            let C = NU ∝ C;
            if i ≥ len
            then
              RETURN(cach (atm-of (C ! k) := SEEN-REMOVABLE), butlast analyse, True)
            else do {
              let (L, analyse) = get-literal-and-remove-of-analyse-wl2 C analyse;
              ASSERT(L ∈# 𝓛_all 𝒜);
              let b = ¬level-in-lbd (get-level M L) lbd;
              if (get-level M L = 0 ∨
                  conflict-min-cach cach (atm-of L) = SEEN-REMOVABLE ∨
                  atm-in-conflict (atm-of L) D)
              then RETURN (cach, analyse, False)
              else if b ∨ conflict-min-cach cach (atm-of L) = SEEN-FAILED
              then do {
                ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
                cach ← mark-failed-lits-wl NU analyse cach;
                RETURN (cach, [], False)
              }
```

else do {
ASSERT(− L ∈ *lits-of-l M*);
C ← *get-propagation-reason M* (−L);
case C of
Some C ⇒ *do* {
ASSERT(C ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ C) ≥ *2*);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ C)));
ASSERT(*length* (*NU* ∝ C) ≤ *Suc* (*uint32-max div 2*));
RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack2* (−L) *NU C*], *False*)
}
| *None* ⇒ *do* {
ASSERT(*mark-failed-lits-stack-inv2 NU analyse cach*);
cach ← *mark-failed-lits-wl NU analyse cach*;
RETURN (*cach*, [], *False*)
}
}
}
})
(*cach*, *analysis*, *False*)⟩
unfolding *lit-redundant-rec-wl-lookup-def Let-def* **by** *auto*

lemma *valid-arena-nempty*:
⟨*valid-arena arena N vdom* ⟹ *i* ∈# *dom-m N* ⟹ *N* ∝ *i* ≠ []⟩
using *arena-lifting*(*19*)[*of arena N vdom i*]
arena-lifting(*4*)[*of arena N vdom i*]
by *auto*

lemma *isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup*:
assumes ⟨*isasat-input-bounded* $\mathcal{A}$⟩
shows ⟨(*uncurry5 isa-lit-redundant-rec-wl-lookup*, *uncurry5* (*lit-redundant-rec-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*⟩*nres-rel*⟩
proof −
have *isa-mark-failed-lits-stack*: ⟨*isa-mark-failed-lits-stack x2e x2z x1l*
≤ ⇓ (*cach-refinement* $\mathcal{A}$)
(*mark-failed-lits-wl x2 x2y x1j*)⟩
if
⟨*case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*uu-*, *N*) ⇒
λ- - - -.
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)) xa)
xa)
xa)
xa⟩ **and**
⟨(*x*, *y*)

$\in$ *trail-pol* $\mathcal{A}$ $\times_f$ $\{$(*arena*, *N*). *valid-arena arena N vdom*$\}$ $\times_f$
lookup-clause-rel $\mathcal{A}$ $\times_f$ *cach-refinement* $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id*› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)› **and**
‹(*xa*, *x'*) $\in$ *cach-refinement* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *bool-rel*)› **and**
‹*case xa of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*lit-redundant-rec-wl-inv2 x1d x2 x2a x'*› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x'* = (*x1j*, *x2j*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*xa* = (*x1l*, *x2l*)› **and**
‹*x1k* $\neq$ []› **and**
‹*x1m* $\neq$ []› **and**
‹*x2o* = (*x1p*, *x2p*)› **and**
‹*x2n* = (*x1o*, *x2o*)› **and**
‹*ana-lookup-conv x2* (*last x1k*) = (*x1n*, *x2n*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*last x1m* = (*x1q*, *x2q*)› **and**
‹*x1n* $\in$# *dom-m x2*› **and**
‹*x1o* < *length* (*x2* $\propto$ *x1n*)› **and**
‹*x2* $\propto$ *x1n* ! *x1o* $\in$ *lits-of-l x1d*› **and**
‹*x2* $\propto$ *x1n* ! *x1o* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*x2* $\propto$ *x1n*))› **and**
‹*length* (*x2* $\propto$ *x1n*) $\leq$ *Suc* (*uint32-max div 2*)› **and**
‹*x2p* $\leq$ *length* (*x2* $\propto$ *x1n*)› **and**
‹*arena-is-valid-clause-idx x2e* (*fst* (*last x1m*))› **and**
‹*x2t* = (*x1u*, *x2u*)› **and**
‹*x2s* = (*x1t*, *x2t*)› **and**
‹(*x1n*, *x1o*, *x1p*, *x2p*) = (*x1s*, *x2s*)› **and**
‹*x2w* = (*x1x*, *x2x*)› **and**
‹*x2v* = (*x1w*, *x2w*)› **and**
‹*ana-lookup-conv-lookup x2e* (*x1q*, *x1r*, *x2r*) = (*x1v*, *x2v*)› **and**
‹*x1v* < *length x2e*› **and**
‹*arena-is-valid-clause-idx x2e x1v*› **and**
‹*arena-lit-pre x2e* (*x1v* + *x1w*)› **and**
‹$\neg$ *x2x* $\leq$ *x1x*› **and**
‹$\neg$ *x2u* $\leq$ *x1u*› **and**
‹*isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m*› **and**
‹*get-literal-and-remove-of-analyse-wl2* (*x2* $\propto$ *x1s*) *x1k* = (*x1y*, *x2y*)› **and**
‹*isa-get-literal-and-remove-of-analyse-wl x2e x1m* = (*x1z*, *x2z*)› **and**
‹*x1y* $\in$# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*get-level-pol-pre* (*x1i*, *x1z*)› **and**
‹*atm-in-conflict-lookup-pre* (*atm-of x1z*) *x2f*› **and**
‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1z*)› **and**
‹$\neg$ (*get-level-pol x1i x1z* = *0* $\vee$
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-REMOVABLE* $\vee$
atm-in-conflict-lookup (*atm-of x1z*) *x2f*)› **and**
‹$\neg$ (*get-level x1d x1y* = *0* $\vee$

conflict-min-cach x1j (*atm-of x1y*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1y*) *x2a*⟩ **and**
⟨¬ *level-in-lbd* (*get-level-pol x1i x1z*) *x2i* ∨
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-FAILED*⟩ **and**
⟨¬ *level-in-lbd* (*get-level x1d x1y*) *x2d* ∨
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-FAILED*⟩ **and**
inv2: ⟨*mark-failed-lits-stack-inv2 x2 x2y x1j*⟩ **and**
⟨*length x1m* ≤ *1*+ *uint32-max div 2*⟩
for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z x2z*
proof −
have [*simp*]: ⟨*x2z* = *x2y*⟩
using *that*
by (*auto simp*: *isa-get-literal-and-remove-of-analyse-wl-def get-literal-and-remove-of-analyse-wl2-def*)

obtain *x2y0* **where**
x2z: ⟨(*x2y*, *x2y0*) ∈ *ana-lookups-rel x2*⟩ **and**
inv: ⟨*mark-failed-lits-stack-inv x2 x2y0 x1j*⟩
using *inv2* **unfolding** *mark-failed-lits-stack-inv2-def*
by *blast*
have *1*: ⟨*mark-failed-lits-wl x2 x2y x1j* = *mark-failed-lits-wl x2 x2y0 x1j*⟩
unfolding *mark-failed-lits-wl-def* **by** *auto*
show *?thesis*
unfolding *1*
apply (*rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *x2 x2y0 x1j x2e x2z x1l vdom x2*, *THEN order-trans*])
subgoal using *assms* **by** *fast*
subgoal using *that x2z* **by** (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have *isa-mark-failed-lits-stack2*: ⟨*isa-mark-failed-lits-stack x2e x2z x1l* ≤ ⇓ (*cach-refinement* $\mathcal{A}$) (*mark-failed-lits-wl x2 x2y x1j*)⟩
if
⟨*case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*uu-*, *N*) ⇒

λ- - - -.
literals-are-in-$\mathcal{L}_{in}$-mm $\mathcal{A}$ ((*mset* $\circ$ *fst*) ‘# *ran-m N*)) *xa*)
xa)
xa)
xa› **and**
‹(*x*, *y*)
$\in$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$ *cach-refinement* $\mathcal{A}$ $\times_f$ *Id* $\times_f$
Id› **and**
‹*ana-lookup-conv-lookup x2e* (*x1q*, *x1r*, *x2r*) = (*x1v*, *x2v*)› **and**
‹*x1v* < *length x2e*› **and**
‹*arena-is-valid-clause-idx x2e x1v*› **and**
‹*arena-lit-pre x2e* (*x1v* + *x1w*)› **and**
‹$\neg$ *x2x* $\leq$ *x1x*› **and**
‹$\neg$ *x2u* $\leq$ *x1u*› **and**
‹*isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m*› **and**
‹*get-literal-and-remove-of-analyse-wl2* (*x2* $\propto$ *x1s*) *x1k* = (*x1y*, *x2y*)› **and**
‹*isa-get-literal-and-remove-of-analyse-wl x2e x1m* = (*x1z*, *x2z*)› **and**
‹*x1y* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*get-level-pol-pre* (*x1i*, *x1z*)› **and**
‹*atm-in-conflict-lookup-pre* (*atm-of x1z*) *x2f*› **and**
‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1z*)› **and**
‹$\neg$ (*get-level-pol x1i x1z* = *0* $\vee$
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-REMOVABLE* $\vee$
atm-in-conflict-lookup (*atm-of x1z*) *x2f*)› **and**
‹$\neg$ (*get-level x1d x1y* = *0* $\vee$
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-REMOVABLE* $\vee$
atm-in-conflict (*atm-of x1y*) *x2a*)› **and**
‹$\neg$ ($\neg$ *level-in-lbd* (*get-level-pol x1i x1z*) *x2i* $\vee$
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-FAILED*)› **and**
‹$\neg$ ($\neg$ *level-in-lbd* (*get-level x1d x1y*) *x2d* $\vee$
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-FAILED*)› **and**
‹$-$ *x1y* $\in$ *lits-of-l x1d*› **and**
‹(*xb*, *x'a*) $\in$ ⟨*nat-rel*⟩*option-rel*› **and**
‹*xb* = *None*› **and**
‹*x'a* = *None*› **and**
inv2: ‹*mark-failed-lits-stack-inv2 x2 x2y x1j*› **and**
‹(*xa*, *x'*) $\in$ *cach-refinement* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *bool-rel*)› **and** ‹*case xa of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) $\Rightarrow$ *analyse* $\neq$ []› **and**
‹*lit-redundant-rec-wl-inv2 x1d x2 x2a x'*› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x'* = (*x1j*, *x2j*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*xa* = (*x1l*, *x2l*)› **and**
‹*x1k* $\neq$ []› **and**
‹*x1m* $\neq$ []› **and**
‹*x2o* = (*x1p*, *x2p*)› **and**
‹*x2n* = (*x1o*, *x2o*)› **and**
‹*ana-lookup-conv x2* (*last x1k*) = (*x1n*, *x2n*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*last x1m* = (*x1q*, *x2q*)› **and**
‹*x1n* $\in\#$ *dom-m x2*› **and**
‹*x1o* < *length* (*x2* $\propto$ *x1n*)› **and**
‹*x2* $\propto$ *x1n* ! *x1o* $\in$ *lits-of-l x1d*› **and**
‹*x2* $\propto$ *x1n* ! *x1o* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*mset* (*x2* $\propto$ *x1n*))› **and**

⟨*length* (*x2* ∝ *x1n*) ≤ *Suc* (*uint32-max div 2*)⟩ **and**
⟨*x2p* ≤ *length* (*x2* ∝ *x1n*)⟩ **and**
⟨*arena-is-valid-clause-idx x2e* (*fst* (*last x1m*))⟩ **and**
⟨*x2t* = (*x1u*, *x2u*)⟩ **and**
⟨*x2s* = (*x1t*, *x2t*)⟩ **and**
⟨(*x1n*, *x1o*, *x1p*, *x2p*) = (*x1s*, *x2s*)⟩ **and**
⟨*x2w* = (*x1x*, *x2x*)⟩ **and**
⟨*x2v* = (*x1w*, *x2w*)⟩ **and**
⟨*x1c* = (*x1d*, *x2*)⟩ **and**
⟨*x1b* = (*x1c*, *x2a*)⟩ **and**
⟨*x1a* = (*x1b*, *x2b*)⟩ **and**
⟨*x1* = (*x1a*, *x2c*)⟩ **and**
⟨*y* = (*x1*, *x2d*)⟩ **and**
⟨*x1h* = (*x1i*, *x2e*)⟩ **and**
⟨*x1g* = (*x1h*, *x2f*)⟩ **and**
⟨*x1f* = (*x1g*, *x2g*)⟩ **and**
⟨*x1e* = (*x1f*, *x2h*)⟩ **and**
⟨*x* = (*x1e*, *x2i*)⟩ **and**
⟨*length x1m* ≤ *1* + *uint32-max div 2*⟩
for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z x2z xb x′a*
proof −
have [*simp*]: ⟨*x2z* = *x2y*⟩
using *that*
by (*auto simp*: *isa-get-literal-and-remove-of-analyse-wl-def*
get-literal-and-remove-of-analyse-wl2-def)

obtain *x2y0* **where**
x2z: ⟨(*x2y*, *x2y0*) ∈ *ana-lookups-rel x2*⟩ **and**
inv: ⟨*mark-failed-lits-stack-inv x2 x2y0 x1j*⟩
using *inv2* **unfolding** *mark-failed-lits-stack-inv2-def*
by *blast*
have *1*: ⟨*mark-failed-lits-wl x2 x2y x1j* = *mark-failed-lits-wl x2 x2y0 x1j*⟩
unfolding *mark-failed-lits-wl-def* **by** *auto*
show *?thesis*
unfolding *1*
apply (*rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j x2e x2z x1l vdom x2*, *THEN order-trans*])
subgoal using *assms* **by** *fast*
subgoal using *that x2z* **by** (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have [*refine0*]: ⟨*get-propagation-reason-pol M′ L′*
≤ ⇓ (⟨*Id*⟩*option-rel*)

(*get-propagation-reason M L*)›
if ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and** ‹(*L′*, *L*) ∈ *Id*› **and** ‹*L* ∈ *lits-of-l M*›
for *M M′ L L′*
using *get-propagation-reason-pol*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*, *of M L M′ L′*] *that* **by** *auto*
note [*simp*]=*get-literal-and-remove-of-analyse-wl-def isa-get-literal-and-remove-of-analyse-wl-def arena-lifting* **and** [*split*] = *prod.splits*

show *?thesis*
supply [[*goals-limit=1*]] *ana-lookup-conv-def*[*simp*] *ana-lookup-conv-lookup-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 add*]
unfolding *isa-lit-redundant-rec-wl-lookup-alt-def lit-redundant-rec-wl-lookup-alt-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m*
by (*auto simp*: *arena-lifting*)
subgoal by (*auto simp*: *trail-pol-alt-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv2-def*)
subgoal by (*auto simp*: *ana-lookup-conv-lookup-pre-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m*
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x*
apply (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def*)
by (*rule-tac x* = ‹*x1s*› **in** *exI*; *auto simp*: *valid-arena-nempty*)+
subgoal by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def split*: *if-splits*)
subgoal using *assms*
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def bind-rule-complete-RES conc-fun-RETURN in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
intro!: *conflict-min-cach-set-removable*[*of* $\mathcal{A}$,*THEN fref-to-Down-curry*, *THEN order-trans*]
dest: *List.last-in-set*)

subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x*
by (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def*
uint32-max-def
arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def)
(*rule-tac x* = *x1s* **in** *exI*; *auto simp*: *uint32-max-def*; *fail*)+
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*auto intro*!: *get-level-pol-pre simp*: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal by (*auto intro*!: *atm-in-conflict-lookup-pre simp*: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*

x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*auto intro*!: *conflict-min-cach-l-pre*
simp: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal
by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*]
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*] *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
get-level-get-level-pol atms-of-def
get-literal-and-remove-of-analyse-wl2-def
split: *prod.splits*)
(*subst* (*asm*) *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *atms-of-def*; *fail*)+
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
subgoal by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*]
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*] *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
get-level-get-level-pol atms-of-def
simp: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
apply (*rule isa-mark-failed-lits-stack*; *assumption*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
apply *assumption*
apply (*rule isa-mark-failed-lits-stack2*; *assumption*)
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z
x2z xb x′a xc x′b
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def
simp: *valid-arena-nempty get-literal-and-remove-of-analyse-wl2-def*
lit-redundant-reason-stack-wl-lookup-def
lit-redundant-reason-stack2-def
intro!: *exI*[*of - x′b*] *bex-leI*[*of - x′b*])
subgoal premises *p* **for** *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u xb x′a xc x′b
using *p*
by (*auto simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def get-literal-and-remove-of-analyse-wl2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why ⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *header-size* (*?N* ∝ *?i*) ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Size* (*?arena* ! (*?i* − *SIZE-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *length* (*?N* ∝ *?i*) = *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?N* ∝ *?i* ! *?j* = *arena-lit ?arena* (*?i* + *?j*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *is-Lit* (*?arena* ! (*?i* + *?j*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?i* + *?j* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?N* ∝ *?i* ! *0* = *arena-lit ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Lit* (*?arena* ! *?i*)

$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *?i* + *length* (*?N* $\propto$ *?i*) $\leq$ *length ?arena*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *is-Pos* (*?arena* ! (*?i* − *POS-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *is-long-clause* (*?N* $\propto$ *?i*)$]\!]$ $\Longrightarrow$ *arena-pos ?arena ?i* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-LBD* (*?arena* ! (*?i* − *LBD-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Act* (*?arena* ! (*?i* − *ACTIVITY-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *is-Status* (*?arena* ! (*?i* − *STATUS-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *SIZE-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *LBD-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *ACTIVITY-SHIFT* $\leq$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *2* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Suc 0* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *0* $\leq$ *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Suc 0* < *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *0* < *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ (*arena-status ?arena ?i* = *LEARNED*) = ($\neg$ *irred ?N ?i*)
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *arena-status ?arena ?i* $\neq$ *DELETED*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!]$ $\Longrightarrow$ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* $\propto$ *?i*) requires to be instantiated.
  done
qed

lemma *iterate-over-conflict-spec*:
  fixes *D* :: ‹*'v clause*›
  assumes ‹*NU* + *NUE* $\models$*pm add-mset K D*› **and** *dist*: ‹*distinct-mset D*›
  shows
    ‹*iterate-over-conflict K M NU NUE D* $\leq$ $\Downarrow$ *Id* (*SPEC*(λ*D'*. *D'* $\subseteq\#$ *D* $\wedge$
      NU + *NUE* $\models$*pm add-mset K D'*))›
proof −
  define *I'* **where**
    ‹*I'* = (λ(*E*:: *'v clause*, *f* :: *'v clause*).
          E $\subseteq\#$ *D* $\wedge$ *NU* + *NUE* $\models$*pm add-mset K E* $\wedge$ *distinct-mset E* $\wedge$ *distinct-mset f*)›

  have *init-I'*: ‹*I'* (*D*, *D*)›
    using ‹*NU* + *NUE* $\models$*pm add-mset K D*› *dist* **unfolding** *I'-def highest-lit-def* **by** *auto*

  have *red*: ‹*is-literal-redundant-spec K NU NUE a x*
      $\leq$ *SPEC* (λ*red*. (*if* $\neg$ *red then RETURN* (*a*, *remove1-mset x aa*)
            else RETURN (*remove1-mset x a*, *remove1-mset x aa*))
          $\leq$ *SPEC* (λ*s'*. *iterate-over-conflict-inv M D s'* $\wedge$ *I' s'* $\wedge$
            (*s'*, *s*) $\in$ *measure* (λ(*D*, *D'*). *size D'*)))›
    if
      ‹*iterate-over-conflict-inv M D s*› **and**
      ‹*I' s*› **and**
      ‹*case s of* (*D*, *D'*) $\Rightarrow$ *D'* $\neq$ {#}› **and**
      ‹*s* = (*a*, *aa*)› **and**
      ‹*x* $\in\#$ *aa*›
    for *s a b aa x*
  proof −
    have ‹*x* $\in\#$ *a*› ‹*distinct-mset aa*›

using *that*
by (*auto simp*: *I′-def highest-lit-def*
eq-commute[*of* ‹*get-level - -*›] *iterate-over-conflict-inv-def*
get-maximum-level-add-mset add-mset-eq-add-mset
dest!: *split*: *option.splits if-splits*)
then show *?thesis*
using *that*
by (*auto simp*: *is-literal-redundant-spec-def iterate-over-conflict-inv-def*
I′-def size-mset-remove1-mset-le-iff remove1-mset-add-mset-If
intro: *mset-le-subtract*)
qed

show *?thesis*
unfolding *iterate-over-conflict-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where**
R = ‹*measure* (λ(D :: ′*v clause*, D':: ′*v clause*).
size D')› **and**
$I' = I'$])
subgoal by *auto*
subgoal by (*auto simp*: *iterate-over-conflict-inv-def highest-lit-def*)
subgoal by (*rule init-I′*)
subgoal by (*rule red*)
subgoal unfolding *I′-def iterate-over-conflict-inv-def* **by** *auto*
subgoal unfolding *I′-def iterate-over-conflict-inv-def* **by** *auto*
done
qed

end

lemma
fixes D :: ‹*nat clause*› **and** s **and** s' **and** NU :: ‹*nat clauses-l*› **and**
S :: ‹*nat twl-st-wl*› **and** S' :: ‹*nat twl-st-l*› **and** S'' :: ‹*nat twl-st*›
defines
‹$S''' \equiv$ *state*$_W$*-of* S''›
defines
‹$M \equiv$ *get-trail-wl* S› **and**
NU: ‹$NU \equiv$ *get-clauses-wl* S› **and**
NU′-def: ‹$NU' \equiv$ *mset* ‘# *ran-mf* NU› **and**
NUE: ‹$NUE \equiv$ *get-unit-learned-clss-wl* S + *get-unit-init-clss-wl* S› **and**
NUE: ‹$NUS \equiv$ *get-subsumed-learned-clauses-wl* S + *get-subsumed-init-clauses-wl* S› **and**
M': ‹$M' \equiv$ *trail* S'''›
assumes
S-S′: ‹(S, S') $\in$ *state-wl-l None*› **and**
S′-S″: ‹(S', S'') $\in$ *twl-st-l None*› **and**
D′-D: ‹*mset* (*tl outl*) = D› **and**
M-D: ‹M $\models$*as CNot* D› **and**
dist-D: ‹*distinct-mset* D› **and**
tauto: ‹$\neg$*tautology* D› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M› **and**
struct-invs: ‹*twl-struct-invs* S''› **and**
add-inv: ‹*twl-list-invs* S'› **and**
cach-init: ‹*conflict-min-analysis-inv* M' s' (NU' + NUE + NUS) D› **and**
NU-P-D: ‹NU' + NUE + NUS $\models$*pm add-mset* K D› **and**
lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ D› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* NU)› **and**
K: ‹K = *outl* ! *0*› **and**

outl-nempty: ‹*outl* ≠ []› **and**
‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ *M NU D s′ lbd outl* ≤
⇓ ({((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧ *outl*!*0* = *K* ∧
E′ ⊆# *D*})
(*SPEC* (λ*D′*. *D′* ⊆# *D* ∧ *NU′* + *NUE* + *NUS* ⊨*pm add-mset K D′*))›
proof −
show *?thesis*
apply (*rule order.trans*)
apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*OF*
assms(*8*−*23*)[*unfolded assms*(*1*−*9*)],
unfolded assms(*1*−*9*)[*symmetric*]])
apply (*rule order.trans*)
apply (*rule ref-two-step′*[*OF iterate-over-conflict-spec*[*OF NU-P-D*[*unfolded add.assoc*] *dist-D*]])
by (*auto simp*: *conc-fun-RES ac-simps*)
qed

lemma (**in** −) *lookup-conflict-upd-None-RETURN-def*:
‹*RETURN oo lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. *RETURN* (*n*− *1*, *xs* [*i* := *NOTIN*]))›
by (*auto intro*!: *ext*)

definition *isa-literal-redundant-wl-lookup* ::
trail-pol ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l*
⇒ *nat literal* ⇒ *lbd* ⇒ (*conflict-min-cach-l* × (*nat* × *nat* × *bool*) *list* × *bool*) *nres*
where
‹*isa-literal-redundant-wl-lookup M NU D cach L lbd* = *do* {
ASSERT(*get-level-pol-pre* (*M*, *L*));
ASSERT(*conflict-min-cach-l-pre* (*cach*, *atm-of L*));
if get-level-pol M L = *0* ∨ *conflict-min-cach-l cach* (*atm-of L*) = *SEEN-REMOVABLE*
then RETURN (*cach*, [], *True*)
else if conflict-min-cach-l cach (*atm-of L*) = *SEEN-FAILED*
then RETURN (*cach*, [], *False*)
else do {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*lit-redundant-reason-stack-wl-lookup-pre* (−*L*) *NU C*);
isa-lit-redundant-rec-wl-lookup M NU D cach
[*lit-redundant-reason-stack-wl-lookup* (−*L*) *NU C*] *lbd*}
| *None* ⇒ *do* {
RETURN (*cach*, [], *False*)
}
}
}›

lemma *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*[*intro*]: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of L* ∈# $\mathcal{A}$›
using *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ **by** *blast*

lemma *isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-literal-redundant-wl-lookup*, *uncurry5* (*literal-redundant-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$ *cach-refinement*
$\mathcal{A}$
$\times_f$ *Id* $\times_f$ *Id* →

$\langle$*cach-refinement* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *bool-rel*$\rangle$*nres-rel*›
proof −
have [*intro*!]: ‹$(x2g, x') \in$ *cach-refinement* $\mathcal{A} \Longrightarrow$
$(x2g, x') \in$ *cach-refinement* (*fold-mset* (+) $\mathcal{A}$ {#})› **for** *x2g x'*
by *auto*
have [*refine0*]: ‹*get-propagation-reason-pol M* (− *L*)
$\le \Downarrow$ ($\langle$*Id*$\rangle$*option-rel*)
(*get-propagation-reason M'* (− *L'*))›
if ‹$(M, M') \in$ *trail-pol* $\mathcal{A}$› **and** ‹$(L, L') \in$ *Id*› **and** ‹$-L \in$ *lits-of-l M'*›
for *M M' L L'*
using *that get-propagation-reason-pol*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*, *of M'* ‹−*L'*› *M* ‹−*L*›] **by** *auto*

show *?thesis*
unfolding *isa-literal-redundant-wl-lookup-def literal-redundant-wl-lookup-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup[*of* $\mathcal{A}$ *vdom*, *THEN fref-to-Down-curry5*])
subgoal
by (*rule get-level-pol-pre*) *auto*
subgoal by (*rule conflict-min-cach-l-pre*) *auto*
subgoal
by (*auto simp*: *get-level-get-level-pol* *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*])
(*subst* (*asm*) *nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*]; *auto*)+
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i
by (*subst nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp del: *conflict-min-cach-def*)
(*auto simp*: *get-level-get-level-pol in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$*D*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x' xb x'a
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def
simp: *valid-arena-nempty*
intro!: *exI*[*of - xb*])
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x' xb x'a
by (*simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why $[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!] \Longrightarrow$ *header-size* (*?N* $\propto$ *?i*) $\le$ *?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!] \Longrightarrow$ *?i* $<$ *length ?arena*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!] \Longrightarrow$ *is-Size* (*?arena* ! (*?i* − *SIZE-SHIFT*))
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*$]\!] \Longrightarrow$ *length* (*?N* $\propto$ *?i*) = *arena-length ?arena ?i*
$[\![$*valid-arena ?arena ?N ?vdom*; *?i* $\in\#$ *dom-m ?N*; *?j* $<$ *length* (*?N* $\propto$ *?i*)$]\!] \Longrightarrow$ *?N* $\propto$ *?i* ! *?j* = *arena-lit*

?arena (*?i* + *?j*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *is-Lit* (*?arena* ! (*?i* + *?j*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?i* + *?j* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?N* ∝ *?i* ! *0* = *arena-lit ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Lit* (*?arena* ! *?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* + *length* (*?N* ∝ *?i*) ≤ *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *is-Pos* (*?arena* ! (*?i* − *POS-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *arena-pos ?arena ?i* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-LBD* (*?arena* ! (*?i* − *LBD-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Act* (*?arena* ! (*?i* − *ACTIVITY-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Status* (*?arena* ! (*?i* − *STATUS-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *SIZE-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *LBD-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *ACTIVITY-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *2* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *LEARNED*) = (¬ *irred ?N ?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *arena-status ?arena ?i* ≠ *DELETED*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* ∝ *?i*) requires to be instantiated.
  done
qed

definition (**in** −) *lookup-conflict-remove1* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*lookup-conflict-remove1* =
  (λ*L* (*n*,*xs*). (*n*−*1*, *xs* [*atm-of L* := *NOTIN*]))›

lemma *lookup-conflict-remove1*:
‹(*uncurry* (*RETURN oo lookup-conflict-remove1*), *uncurry* (*RETURN oo remove1-mset*))
∈ [λ(*L*,*C*). *L* ∈# *C* ∧ −*L* ∉# *C* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
  Id $\times_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*case-tac y*; *case-tac x*)
subgoal for *x y a b aa ab c*
  using *mset-as-position-remove*[*of c b* ‹*atm-of aa*›]
  by (*cases* ‹*aa*›)
    (*auto simp*: *lookup-clause-rel-def lookup-conflict-remove1-def lookup-clause-rel-atm-in-iff minus-notin-trivial2 size-remove1-mset-If in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff minus-notin-trivial mset-as-position-in-iff-nth*)
done

definition (**in** −) *lookup-conflict-remove1-pre* :: ‹*nat literal* × *nat* × *bool option list* ⇒ *bool*› **where**
‹*lookup-conflict-remove1-pre* = (λ(*L*,(*n*,*xs*)). *n* > *0* ∧ *atm-of L* < *length xs*)›

definition *isa-minimize-and-extract-highest-lookup-conflict*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l* ⇒ *lbd* ⇒
  out-learned ⇒ (*lookup-clause-rel* × *conflict-min-cach-l* × *out-learned*) *nres*›

where
‹*isa-minimize-and-extract-highest-lookup-conflict* = (λ*M NU nxs s lbd outl. do* {
(D, -, s, *outl*) $\leftarrow$
$WHILE_T{}^{\lambda(nxs,\ i,\ s,\ outl).\ length\ outl\ \leq\ uint32\text{-}max}$
(λ(*nxs*, *i*, *s*, *outl*). i < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(x < *length outl*);
let L = *outl* ! x;
(s', -, *red*) $\leftarrow$ *isa-literal-redundant-wl-lookup M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x+1*, s', *outl*)
else do {
ASSERT(*lookup-conflict-remove1-pre* (*L*, *nxs*));
RETURN (*lookup-conflict-remove1 L nxs*, *x*, s', *delete-index-and-swap outl x*)
}
})
(*nxs*, *1*, *s*, *outl*);
RETURN (*D*, *s*, *outl*)
})›

lemma *isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*,
uncurry5 (*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$)) $\in$
[λ(((((-, *N*), *D*), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* $\circ$ *fst*) ‘# *ran-m N*) $\wedge$
¬*tautology D*]$_f$
trail-pol $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* $\rightarrow$
⟨*lookup-clause-rel* $\mathcal{A}$ $\times_r$ *cach-refinement* $\mathcal{A}$ $\times_r$ *Id*⟩*nres-rel*›
proof −
have *init*: ‹((*x2f*, *1*, *x2g*, *x2i*), *x2a*::*nat literal multiset*, *1*, *x2b*, *x2d*)
$\in$ *lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *cach-refinement* $\mathcal{A}$ $\times_r$ *Id* ›
if
‹(*x*, *y*)
$\in$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *lookup-clause-rel* $\mathcal{A}$ $\times_f$
cach-refinement $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id*› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)›
for *x y x1 x1a x1b x1c x1d x2 x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i **and**
x2a
proof −
show *?thesis*
using *that* **by** *auto*
qed

show *?thesis*

unfolding *isa-minimize-and-extract-highest-lookup-conflict-def uncurry-def minimize-and-extract-highest-lookup-conflict-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup*[*of* $\mathcal{A}$ *vdom*, *THEN fref-to-Down-curry5*])
apply (*rule init*; *assumption*)
subgoal by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*auto simp*: *lookup-conflict-remove1-pre-def lookup-clause-rel-def atms-of-def minimize-and-extract-highest-lookup-conflict-inv-def*)
subgoal
by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def intro*!: *lookup-conflict-remove1* [*THEN fref-to-Down-unRET-uncurry*] *simp*: *nth-in-set-tl delete-from-lookup-conflict-pre-def dest*!: *in-set-takeD*)
subgoal by *auto*
done
qed

definition *set-empty-conflict-to-none* **where**
‹*set-empty-conflict-to-none D* = *None*›

definition *set-lookup-empty-conflict-to-none* **where**
‹*set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). (*True*, *n*, *xs*))›

lemma *set-empty-conflict-to-none-hnr*:
‹(*RETURN o set-lookup-empty-conflict-to-none*, *RETURN o set-empty-conflict-to-none*) $\in$ [λD. $D = \{\#\}$]$_f$ *lookup-clause-rel* $\mathcal{A} \to \langle$*option-lookup-clause-rel* $\mathcal{A}\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def set-empty-conflict-to-none-def set-lookup-empty-conflict-to-none-def*)

definition *lookup-merge-eq2*
:: ‹*nat literal* $\Rightarrow$ (*nat*,*nat*) *ann-lits* $\Rightarrow$ *nat clause-l* $\Rightarrow$ *conflict-option-rel* $\Rightarrow$ *nat* $\Rightarrow$ *lbd* $\Rightarrow$ *out-learned* $\Rightarrow$ (*conflict-option-rel* $\times$ *nat* $\times$ *lbd* $\times$ *out-learned*) *nres*› **where**
‹*lookup-merge-eq2 L M N* = (λ(-, *zs*) *clvls lbd outl*. *do* {
ASSERT(*length N* = *2*);
let L' = (*if N* ! *0* = *L then N* ! *1 else N* ! *0*);
ASSERT(*get-level M* $L' \le$ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level M* L');
ASSERT(*atm-of* L' < *length* (*snd zs*));
ASSERT(*length outl* < *uint32-max*);
let outl = *outlearned-add M* L' *zs outl*;
ASSERT(*clvls* < *uint32-max*);
ASSERT(*fst zs* < *uint32-max*);
let clvls = *clvls-add M* L' *zs clvls*;
let zs = *add-to-lookup-conflict* L' *zs*;

RETURN((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

definition *merge-conflict-m-eq2*
:: ‹*nat literal* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *lbd* × *out-learned*) *nres*›
where
‹*merge-conflict-m-eq2 L M Ni D* =
SPEC (λ(*C*, *n*, *lbd*, *outl*). *C* = *Some* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
n = *card-max-lvl M* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
out-learned M C outl)›

lemma *lookup-merge-eq2-spec*:
assumes
o: ‹((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› **and**
lits-tr: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
n-d: ‹*no-dup M*› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C*› **and**
no-tauto: ‹⋀*K*. *K* ∈ *set* (*remove1 L D*) ⟹ − *K* ∉# *C*›
‹*clvls* = *card-max-lvl M C*› **and**
out: ‹*out-learned M* (*Some C*) *outl*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
le2: ‹*length D* = *2*› **and**
L-D: ‹*L* ∈ *set D*›
shows
‹*lookup-merge-eq2 L M D* (*b*, *n*, *xs*) *clvls lbd outl* ≤
⇓(*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *Id* $\times_r$ *Id*)
(*merge-conflict-m-eq2 L M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
proof −
define *lbd-upd* **where**
‹*lbd-upd lbd i* ≡ *lbd-write lbd* (*get-level M* (*D*!*i*))› **for** *lbd i*
let *?D* = ‹*remove1 L D*›
have *le-D-le-upper*[*simp*]: ‹*a* < *length D* ⟹ *Suc* (*Suc a*) ≤ *uint32-max*› **for** *a*
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*mset D*›] *assms* **by** (*auto simp*: *uint32-max-def*)
have *Suc-N-uint32-max*: ‹*Suc n* ≤ *uint32-max*› **and**
size-C-uint32-max: ‹*size C* ≤ *1* + *uint32-max div 2*› **and**
clvls: ‹*clvls* = *card-max-lvl M C*› **and**
tauto-C: ‹¬ *tautology C*› **and**
dist-C: ‹*distinct-mset C*› **and**
atms-le-xs: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*› **and**
map: ‹*mset-as-position xs C*›
using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ *C*] *mset-as-position-distinct-mset*[*of xs C*]
lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*auto simp*: *uint32-max-def*)
then have *clvls-uint32-max*: ‹*clvls* ≤ *1* + *uint32-max div 2*›
using *size-filter-mset-lesseq*[*of* ‹λ*L*. *get-level M L* = *count-decided M*› *C*]
unfolding *uint32-max-def card-max-lvl-def* **by** *linarith*
have [*intro*]: ‹((*b*, *a*, *ba*), *Some C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹
Suc (*Suc a*) ≤ *uint32-max*› **for** *b a ba C*
using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def uint32-max-def)

have [*simp*]: ‹*remdups-mset* $C = C$›
  using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def*
    lookup-clause-rel-def distinct-mset-remdups-mset-id)
have ‹$\neg$*tautology C*›
  using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def*
    lookup-clause-rel-def)
have ‹*distinct-mset C*›
  using *mset-as-position-distinct-mset*[*of - C*] *o*
  unfolding *option-lookup-clause-rel-def lookup-clause-rel-def* **by** *auto*
have ‹*mset* (*tl outl*) $\subseteq\#$ *C*›
   using *out* **by** (*auto simp*: *out-learned-def*)
from *size-mset-mono*[*OF this*] **have** *outl-le*: ‹*length outl* < *uint32-max*›
 using *simple-clss-size-upper-div2*[*OF bounded lits-C*] *dist-C tauto-C* **by** (*auto simp*: *uint32-max-def*)
define L' **where** ‹$L' \equiv$ *if D ! 0 = L then D ! 1 else D ! 0*›
have L'*-all*: ‹$L' \in\# \mathcal{L}_{all}\ \mathcal{A}$›
  using *lits le2* **by** (*cases D*; *cases* ‹*tl D*›)
    (*auto simp*: L'*-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
then have L': ‹*atm-of* $L' \in$ *atms-of* ($\mathcal{L}_{all}\ \mathcal{A}$)›
  by (*auto simp*: *atms-of-def*)
have *DLL*: ‹*mset D* = $\{\#L, L'\#\}$› ‹*set D* = $\{L, L'\}$› ‹$L \neq L'$› ‹*remove1 L D* = [L']›
  using *le2 L-D dist* **by** (*cases D*; *cases* ‹*tl D*›; *auto simp*: L'*-def*; *fail*)+
have ‹$- L' \in\# C \Longrightarrow$ *False*› **and** [*simp*]: ‹$- L' \notin\# C$›
  using *dist no-tauto* **by** (*auto simp*: *DLL*)
then have o': ‹((*False*, *add-to-lookup-conflict* L' (*n*, *xs*)), *Some* ($\{\#L'\#\} \cup\#$ *C*))
  $\in$ *option-lookup-clause-rel* $\mathcal{A}$›
  using *o* L'*-all* **unfolding** *option-lookup-clause-rel-def*
  by (*auto intro*!: *add-to-lookup-conflict-lookup-clause-rel*)
have [*iff*]: ‹*is-in-lookup-conflict* (*n*, *xs*) $L' \longleftrightarrow L' \in\# C$›
  using *o mset-as-position-in-iff-nth*[*of xs C* L'] L' *no-tauto*
  apply (*auto simp*: *is-in-lookup-conflict-def option-lookup-clause-rel-def*
     lookup-clause-rel-def DLL is-pos-neg-not-is-pos
    split: *option.splits*)
  by (*smt* ‹$- L' \notin\# C$› *atm-of-uminus is-pos-neg-not-is-pos mset-as-position-in-iff-nth option.inject*)
have *clvls-add*: ‹*clvls-add M* L' (*n*, *xs*) *clvls* = *card-max-lvl M* ($\{\#L'\#\} \cup\#$ *C*)›
  by (*cases* ‹$L' \in\# C$›)
    (*auto simp*: *clvls-add-def card-max-lvl-add-mset clvls add-mset-union*
    dest!: *multi-member-split*)
have out': ‹*out-learned M* (*Some* ($\{\#L'\#\} \cup\#$ *C*)) (*outlearned-add M* L' (*n*, *xs*) *outl*)›
  using *out*
  by (*cases* ‹$L' \in\# C$›)
    (*auto simp*: *out-learned-def outlearned-add-def add-mset-union*
    dest!: *multi-member-split*)

show *?thesis*
  unfolding *lookup-merge-eq2-def prod.simps* L'*-def*[*symmetric*]
  apply *refine-vcg*
  subgoal by (*rule le2*)
  subgoal using *literals-are-in-*$\mathcal{L}_{in}$*-trail-get-level-uint32-max*[*OF bounded lits-tr n-d*] **by** *blast*
  subgoal using *atms-le-xs* L' **by** *simp*
  subgoal using *outl-le* **.**
  subgoal using *clvls-uint32-max* **by** (*auto simp*: *uint32-max-def*)
  subgoal using *Suc-N-uint32-max* **by** *auto*
  subgoal
    using o' *clvls-add* out'
    by (*auto simp*: *merge-conflict-m-eq2-def DLL*
      intro!: *RETURN-RES-refine*)

done
qed

definition *isasat-lookup-merge-eq2*
:: ‹*nat literal* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒ *lbd* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *lbd* × *out-learned*) *nres*› **where**
‹*isasat-lookup-merge-eq2 L M N C* = (λ(-, *zs*) *clvls lbd outl*. *do* {
ASSERT(*arena-lit-pre N C*);
ASSERT(*arena-lit-pre N* (*C*+*1*));
let L′ = (*if arena-lit N C* = *L then arena-lit N* (*C* + *1*) *else arena-lit N C*);
ASSERT(*get-level-pol-pre* (*M*, *L′*));
ASSERT(*get-level-pol M L′* ≤ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level-pol M L′*);
ASSERT(*atm-of L′* < *length* (*snd zs*));
ASSERT(*length outl* < *uint32-max*);
let outl = *isa-outlearned-add M L′ zs outl*;
ASSERT(*clvls* < *uint32-max*);
ASSERT(*fst zs* < *uint32-max*);
let clvls = *isa-clvls-add M L′ zs clvls*;
let zs = *add-to-lookup-conflict L′ zs*;
RETURN((*False*, *zs*), *clvls*, *lbd*, *outl*)
})›

lemma *isasat-lookup-merge-eq2-lookup-merge-eq2*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
bxs: ‹((*b*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
M′M: ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isasat-lookup-merge-eq2 L M′ arena i* (*b*, *xs*) *clvls lbd outl* ≤ ⇓ *Id*
(*lookup-merge-eq2 L M* (*N* ∝ *i*) (*b*, *xs*) *clvls lbd outl*)›
proof −
define *L′* **where** ‹*L′* ≡ (*if arena-lit arena i* = *L then arena-lit arena* (*i* + *1*)
else arena-lit arena i)›
define *L″* **where** ‹*L″* ≡ (*if N* ∝ *i* ! *0* = *L then N* ∝ *i* ! *1 else N* ∝ *i* ! *0*)›

have [*simp*]: ‹*L″* = *L′*›
if ‹*length* (*N* ∝ *i*) = *2*›
using *that i valid* **by** (*auto simp*: *L″-def L′-def arena-lifting*)
have *L′-all*: ‹*L′* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
if ‹*length* (*N* ∝ *i*) = *2*›
by (*use lits i valid that*
literals-are-in-$\mathcal{L}_{in}$*-mm-add-msetD*[*of* $\mathcal{A}$
‹*mset* (*N* ∝ *i*)› - ‹*arena-lit arena* (*Suc i*)›]
literals-are-in-$\mathcal{L}_{in}$*-mm-add-msetD*[*of* $\mathcal{A}$
‹*mset* (*N* ∝ *i*)› - ‹*arena-lit arena i*›]
nth-mem[*of 0* ‹*N* ∝ *i*›] *nth-mem*[*of 1* ‹*N* ∝ *i*›]
in ‹*auto simp*: *arena-lifting ran-m-def L′-def*
simp del: *nth-mem*
dest:
dest!: *multi-member-split*›)

show *?thesis*
unfolding *isasat-lookup-merge-eq2-def lookup-merge-eq2-def prod.simps*
L′-def[*symmetric*] *L″-def*[*symmetric*]

apply *refine-vcg*
subgoal
using *valid i*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*auto intro*!: *exI*[*of - i*] *exI*[*of - N*])
subgoal
using *valid i*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*auto intro*!: *exI*[*of - i*] *exI*[*of - N*])
subgoal
by (*rule get-level-pol-pre*[*OF - M'M*])
(*use L'-all*
in ‹*auto simp*: *arena-lifting ran-m-def*
simp del: *nth-mem*
dest:
dest!: *multi-member-split*›)
subgoal
by (*subst get-level-get-level-pol*[*OF M'M*, *symmetric*])
(*use L'-all* **in** *auto*)
subgoal by *auto*
subgoal
using *M'M L'-all*
by (*auto simp*: *isa-clvls-add-clvls-add get-level-get-level-pol*
isa-outlearned-add-outlearned-add)
done
qed

definition *merge-conflict-m-eq2-pre* **where**
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ =
(λ(((((((L, M), N), i), xs), clvls), lbd), out). $i \in\#$ *dom-m N* $\wedge$ *xs* $\neq$ *None* $\wedge$ *distinct* ($N \propto i$) $\wedge$
$\neg$*tautology* (*mset* ($N \propto i$)) $\wedge$
($\forall K \in$ *set* (*remove1 L* ($N \propto i$)). $- K \notin\#$ *the xs*) $\wedge$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) $\wedge$ *clvls* = *card-max-lvl M* (*the xs*) $\wedge$
out-learned M xs out $\wedge$ *no-dup M* $\wedge$
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) $\wedge$
isasat-input-bounded $\mathcal{A}$ $\wedge$
length ($N \propto i$) = *2* $\wedge$
$L \in$ *set* ($N \propto i$))›

definition *merge-conflict-m-g-eq2* :: ‹-› **where**
‹*merge-conflict-m-g-eq2 L M N i D - - -* = *merge-conflict-m-eq2 L M* ($N \propto i$) *D*›

lemma *isasat-lookup-merge-eq2*:
‹(*uncurry7 isasat-lookup-merge-eq2*, *uncurry7 merge-conflict-m-g-eq2*) $\in$
[*merge-conflict-m-eq2-pre* $\mathcal{A}$]$_f$
Id $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel*
$\mathcal{A}$
$\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$ *Id* $\rightarrow$
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id* $\times_r$ *Id* ⟩*nres-rel*›
proof –
have *H1*: ‹*isasat-lookup-merge-eq2 a* (*aa*, *ab*, *ac*, *ad*, *ae*, *b*) *ba bb* (*af*, *ag*, *bc*) *bd be*
bf
$\leq \Downarrow$ *Id* (*lookup-merge-eq2 a bg* ($bh \propto bb$) (*af*, *ag*, *bc*) *bd be bf*)›
if

⟨*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)⟩ **and**
⟨((((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*), *bd*), *be*), *bf*),
((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)
∈ *Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$
option-lookup-clause-rel $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$
Id $\times_f$
Id⟩
for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
proof −
have
bi: ⟨*bi* ∈# *dom-m bh*⟩ **and**
⟨(*bf*, *bm*) ∈ *Id*⟩ **and**
⟨*bj* ≠ *None*⟩ **and**
⟨(*be*, *bl*) ∈ *Id*⟩ **and**
⟨*distinct* (*bh* ∝ *bi*)⟩ **and**
⟨(*bd*, *bk*) ∈ *nat-rel*⟩ **and**
⟨¬ *tautology* (*mset* (*bh* ∝ *bi*))⟩ **and**
o: ⟨((*af*, *ag*, *bc*), *bj*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**
⟨∀ *K*∈*set* (*remove1 ah* (*bh* ∝ *bi*)). − *K* ∉# *the bj*⟩ **and**
st: ⟨*bb* = *bi*⟩ **and**
⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)⟩ **and**
valid: ⟨*valid-arena ba bh vdom*⟩ **and**
⟨*bk* = *card-max-lvl bg* (*the bj*)⟩ **and**
⟨(*a*, *ah*) ∈ *Id*⟩ **and**
tr: ⟨((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) ∈ *trail-pol* $\mathcal{A}$⟩ **and**
⟨*out-learned bg bj bm*⟩ **and**
⟨*no-dup bg*⟩ **and**
lits: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf bh*)⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩ **and**
ah: ⟨*ah* ∈ *set* (*bh* ∝ *bi*)⟩
using *that* **unfolding** *merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff*
by *blast*+

show *?thesis*
by (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[*OF valid bi*[*unfolded st*[*symmetric*]]
lits o tr bounded])
qed
have *H2*: ⟨*lookup-merge-eq2 a bg* (*bh* ∝ *bb*) (*af*, *ag*, *bc*) *bd be bf*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_f$ (*nat-rel* $\times_f$ (*Id* $\times_f$ *Id*)))
(*merge-conflict-m-g-eq2 ah bg bh bi bj bk bl bm*)⟩
if
⟨*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)⟩ **and**
⟨((((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*), *bd*), *be*), *bf*),
((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*), *bl*), *bm*)
∈ *Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$
option-lookup-clause-rel $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$
Id $\times_f$
Id⟩
for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
proof −
have
bi: ⟨*bi* ∈# *dom-m bh*⟩ **and**
bj: ⟨*bj* ≠ *None*⟩ **and**
dist: ⟨*distinct* (*bh* ∝ *bi*)⟩ **and**
tauto: ⟨¬ *tautology* (*mset* (*bh* ∝ *bi*))⟩ **and**
o: ⟨((*af*, *ag*, *bc*), *bj*) ∈ *option-lookup-clause-rel* $\mathcal{A}$⟩ **and**

K: ‹∀ K∈*set* (*remove1 ah* (*bh* ∝ *bi*)). − K ∉# *the bj*› **and**
st: ‹*bb* = *bi*›
‹*bd* = *bk*›
‹*bf* = *bm*›
‹*be* = *bl*›
‹*a* = *ah*› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)› **and**
valid: ‹*valid-arena ba bh vdom*› **and**
bk: ‹*bk* = *card-max-lvl bg* (*the bj*)› **and**
tr: ‹((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) ∈ *trail-pol* $\mathcal{A}$› **and**
out: ‹*out-learned bg bj bm*› **and**
‹*no-dup bg*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf bh*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
le2: ‹*length* (*bh* ∝ *bi*) = *2*› **and**
ah: ‹*ah* ∈ *set* (*bh* ∝ *bi*)›
using *that* **unfolding** *merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff*
by *blast+*
obtain *bj′* **where** *bj′*: ‹*bj* = *Some bj′*›
using *bj* **by** (*cases bj*) *auto*
have *n-d*: ‹*no-dup bg*› **and** *lits-tr*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *bg*›
using *tr* **unfolding** *trail-pol-alt-def*
by *auto*
have *lits-bi*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*bh* ∝ *bi*))›
using *bi lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset ran-m-def*
dest!: *multi-member-split*)

show *?thesis*
unfolding *st merge-conflict-m-g-eq2-def*
apply (*rule lookup-merge-eq2-spec*[*THEN order-trans*, *OF o*[*unfolded bj′*]
dist lits-bi lits-tr n-d tauto lits-confl[*unfolded bj′ option.sel*]
- *bk*[*unfolded bj′ option.sel*] - *bounded le2 ah*])
subgoal using K **unfolding** *bj′* **by** *auto*
subgoal using *out* **unfolding** *bj′* **.**
subgoal unfolding *bj′* **by** *auto*
done
qed

show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
apply (*intro nres-relI frefI*)
apply *clarify*
subgoal for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bk bl bm*
apply (*rule H1*[*THEN order-trans*]; *assumption?*)
apply (*subst Down-id-eq*)
apply (*rule H2*)
apply *assumption+*
done
done
qed

end
theory *IsaSAT-Setup*
imports
Watched-Literals-VMTF
Watched-Literals.Watched-Literals-Watch-List-Initialisation

IsaSAT-Lookup-Conflict
IsaSAT-Clauses IsaSAT-Arena IsaSAT-Watch-List LBD
begin

Chapter 8

Complete state

We here define the last step of our refinement: the step with all the heuristics and fully deterministic code.

After the result of benchmarking, we concluded that the use of *nat* leads to worse performance than using *sint64*. As, however, the later is not complete, we do so with a switch: as long as it fits, we use the faster (called 'bounded') version. After that we switch to the 'unbounded' version (which is still bounded by memory anyhow) if we generate Standard ML code.

We have successfully killed all natural numbers when generating LLVM. However, the LLVM binding does not have a binding to GMP integers.

8.1 Statistics

We do some statistics on the run.

NB: the statistics are not proven correct (especially they might overflow), there are just there to look for regressions, do some comparisons (e.g., to conclude that we are propagating slower than the other solvers), or to test different option combination.

type-synonym *stats* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *incr-propagation* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-propagation* = (λ(*propa*, *confl*, *dec*). (*propa* + *1*, *confl*, *dec*))›

definition *incr-conflict* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-conflict* = (λ(*propa*, *confl*, *dec*). (*propa*, *confl* + *1*, *dec*))›

definition *incr-decision* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-decision* = (λ(*propa*, *confl*, *dec*, *res*). (*propa*, *confl*, *dec* + *1*, *res*))›

definition *incr-restart* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-restart* = (λ(*propa*, *confl*, *dec*, *res*, *lres*). (*propa*, *confl*, *dec*, *res* + *1*, *lres*))›

definition *incr-lrestart* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-lrestart* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*). (*propa*, *confl*, *dec*, *res*, *lres* + *1*, *uset*))›

definition *incr-uset* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-uset* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, (*uset*, *gcs*)). (*propa*, *confl*, *dec*, *res*, *lres*, *uset* + *1*, *gcs*))›

definition *incr-GC* :: ‹*stats* ⇒ *stats*› **where**
‹*incr-GC* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbds*). (*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs* + *1*, *lbds*))›

definition *add-lbd* :: ‹*64 word* ⇒ *stats* ⇒ *stats*› **where**
‹*add-lbd lbd* = (λ(*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbds*). (*propa*, *confl*, *dec*, *res*, *lres*, *uset*, *gcs*, *lbd* + *lbds*))›

8.2 Moving averages

We use (at least hopefully) the variant of EMA-14 implemented in Cadical, but with fixed-point calculation (*1* is *1* >> *32*).

Remark that the coefficient β already should not take care of the fixed-point conversion of the glue. Otherwise, *value* is wrongly updated.

type-synonym *ema* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *ema-bitshifting* **where**
‹*ema-bitshifting* = (*1* << *32*)›

definition (**in** −) *ema-update* :: ‹*nat* ⇒ *ema* ⇒ *ema*› **where**
‹*ema-update* = (λ*lbd* (*value*, α, β, *wait*, *period*).
  let lbd = (*of-nat lbd*) ∗ *ema-bitshifting in*
  let value = *if lbd* > *value then value* + (β ∗ (*lbd* − *value*) >> *32*) *else value* − (β ∗ (*value* − *lbd*) >> *32*) *in*
  if β ≤ α ∨ *wait* > *0 then* (*value*, α, β, *wait* − *1*, *period*)
  else
    let wait = *2* ∗ *period* + *1 in*
    let period = *wait in*
    let β = β >> *1 in*
    let β = *if* β ≤ α *then* α *else* β *in*
    (*value*, α, β, *wait*, *period*))›

definition (**in** −) *ema-update-ref* :: ‹*32 word* ⇒ *ema* ⇒ *ema*› **where**
‹*ema-update-ref* = (λ*lbd* (*value*, α, β, *wait*, *period*).
  let lbd = *ucast lbd* ∗ *ema-bitshifting in*
  let value = *if lbd* > *value then value* + (β ∗ (*lbd* − *value*) >> *32*) *else value* − (β ∗ (*value* − *lbd*) >> *32*) *in*
  if β ≤ α ∨ *wait* > *0 then* (*value*, α, β, *wait* − *1*, *period*)
  else
    let wait = *2* ∗ *period* + *1 in*
    let period = *wait in*
    let β = β >> *1 in*
    let β = *if* β ≤ α *then* α *else* β *in*
    (*value*, α, β, *wait*, *period*))›

definition (**in** −) *ema-init* :: ‹*64 word* ⇒ *ema*› **where**
‹*ema-init* α = (*0*, α, *ema-bitshifting*, *0*, *0*)›

fun *ema-reinit* **where**
‹*ema-reinit* (*value*, α, β, *wait*, *period*) = (*value*, α, *1* << *32*, *0*, *0*)›

fun *ema-get-value* :: ‹*ema* ⇒ *64 word*› **where**
‹*ema-get-value* (*v*, -) = *v*›

We use the default values for Cadical: $(3::'a)$ / $(10::'a)^2$ and $(1::'a)$ / $(10::'a)^5$ in our fixed-point version.

abbreviation *ema-fast-init* :: *ema* **where**
⟨*ema-fast-init* ≡ *ema-init* (*128849010*)⟩

abbreviation *ema-slow-init* :: *ema* **where**
⟨*ema-slow-init* ≡ *ema-init* *429450*⟩

8.3 Information related to restarts

definition *NORMAL-PHASE* :: ⟨*64 word*⟩ **where**
⟨*NORMAL-PHASE* = *0*⟩

definition *QUIET-PHASE* :: ⟨*64 word*⟩ **where**
⟨*QUIET-PHASE* = *1*⟩

definition *DEFAULT-INIT-PHASE* :: ⟨*64 word*⟩ **where**
⟨*DEFAULT-INIT-PHASE* = *10000*⟩

type-synonym *restart-info* = ⟨*64 word* × *64 word* × *64 word* × *64 word* × *64 word*⟩

definition *incr-conflict-count-since-last-restart* :: ⟨*restart-info* ⇒ *restart-info*⟩ **where**
⟨*incr-conflict-count-since-last-restart* = (λ(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*).
(*ccount* + *1*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*))⟩

definition *restart-info-update-lvl-avg* :: ⟨*32 word* ⇒ *restart-info* ⇒ *restart-info*⟩ **where**
⟨*restart-info-update-lvl-avg* = (λ*lvl* (*ccount*, *ema-lvl*). (*ccount*, *ema-lvl*))⟩

definition *restart-info-init* :: ⟨*restart-info*⟩ **where**
⟨*restart-info-init* = (*0*, *0*, *NORMAL-PHASE*, *DEFAULT-INIT-PHASE*, *1000*)⟩

definition *restart-info-restart-done* :: ⟨*restart-info* ⇒ *restart-info*⟩ **where**
⟨*restart-info-restart-done* = (λ(*ccount*, *lvl-avg*). (*0*, *lvl-avg*))⟩

8.4 Phase saving

type-synonym *phase-save-heur* = ⟨*phase-saver* × *nat* × *phase-saver* × *nat* × *phase-saver* × *64 word* × *64 word* × *64 word*⟩

definition *phase-save-heur-rel* :: ⟨*nat multiset* ⇒ *phase-save-heur* ⇒ *bool*⟩ **where**
⟨*phase-save-heur-rel* $\mathcal{A}$ = (λ(φ, *target-assigned*, *target*, *best-assigned*, *best*,
end-of-phase, *curr-phase*). *phase-saving* $\mathcal{A}$ φ ∧
phase-saving $\mathcal{A}$ *target* ∧ *phase-saving* $\mathcal{A}$ *best* ∧ *length* φ = *length best* ∧
length target = *length best*)⟩

definition *end-of-rephasing-phase* :: ⟨*phase-save-heur* ⇒ *64 word*⟩ **where**
⟨*end-of-rephasing-phase* = (λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*,
length-phase). *end-of-phase*)⟩

definition *phase-current-rephasing-phase* :: ⟨*phase-save-heur* ⇒ *64 word*⟩ **where**
⟨*phase-current-rephasing-phase* =
(λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*). *curr-phase*)⟩

8.5 Heuristics

type-synonym *restart-heuristics* = ‹*ema* × *ema* × *restart-info* × *64 word* × *phase-save-heur*›

fun *fast-ema-of* :: ‹*restart-heuristics* ⇒ *ema*› **where**
‹*fast-ema-of* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ) = *fast-ema*›

fun *slow-ema-of* :: ‹*restart-heuristics* ⇒ *ema*› **where**
‹*slow-ema-of* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ) = *slow-ema*›

fun *restart-info-of* :: ‹*restart-heuristics* ⇒ *restart-info*› **where**
‹*restart-info-of* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ) = *restart-info*›

fun *current-restart-phase* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*current-restart-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ)
=
restart-phase›

fun *incr-restart-phase* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*incr-restart-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ) =
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*, φ)›

fun *incr-wasted* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*incr-wasted waste* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) =
(*fast-ema*, *slow-ema*, *res-info*, *wasted* + *waste*, φ)›

fun *set-zero-wasted* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*set-zero-wasted* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) =
(*fast-ema*, *slow-ema*, *res-info*, *0*, φ)›

fun *wasted-of* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*wasted-of* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ) = *wasted*›

definition *heuristic-rel* :: ‹*nat multiset* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-rel* $\mathcal{A}$ = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *phase-save-heur-rel* $\mathcal{A}$ φ)›

definition *save-phase-heur* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*save-phase-heur L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, *target*, *best*)).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, ($\varphi[L := b]$, *target*, *best*)))›

definition *save-phase-heur-pre* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*save-phase-heur-pre L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). *L* < *length* φ)›

definition *mop-save-phase-heur* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*mop-save-phase-heur L b heur* = *do* {
ASSERT(*save-phase-heur-pre L b heur*);
RETURN (*save-phase-heur L b heur*)
}›

definition *get-saved-phase-heur-pre* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur-pre L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). *L* < *length* φ)›

definition *get-saved-phase-heur* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -)). $\varphi!L$)›

definition *current-rephasing-phase* :: ‹*restart-heuristics* ⇒ *64 word*› **where**

‹*current-rephasing-phase* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *phase-current-rephasing-phase* φ)›

definition *mop-get-saved-phase-heur* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool nres*› **where**
‹*mop-get-saved-phase-heur L heur* = *do* {
  ASSERT(*get-saved-phase-heur-pre L heur*);
  RETURN (*get-saved-phase-heur L heur*)
}›

definition *end-of-rephasing-phase-heur* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase-heur* =
  (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*). *end-of-rephasing-phase phasing*)›

lemma *heuristic-relI*[*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*incr-wasted wast heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*set-zero-wasted heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*save-phase-heur L b heur*)›
by (*clarsimp-all simp*: *heuristic-rel-def save-phase-heur-def phase-save-heur-rel-def phase-saving-def*)

lemma *save-phase-heur-preI*:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *a* ∈# $\mathcal{A}$ ⟹ *save-phase-heur-pre a b heur*›
by (*auto simp*: *heuristic-rel-def phase-saving-def save-phase-heur-pre-def*
  phase-save-heur-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

8.6 VMTF

type-synonym (**in** −) *isa-vmtf-remove-int* = ‹*vmtf* × (*nat list* × *bool list*)›

8.7 Options

type-synonym *opts* = ‹*bool* × *bool* × *bool*›

definition *opts-restart* **where**
‹*opts-restart* = (λ(*a*, *b*, *c*). *a*)›

definition *opts-reduce* **where**
‹*opts-reduce* = (λ(*a*, *b*, *c*). *b*)›

definition *opts-unbounded-mode* **where**
‹*opts-unbounded-mode* = (λ(*a*, *b*, *c*). *c*)›

type-synonym *out-learned* = ‹*nat clause-l*›

type-synonym *vdom* = ‹*nat list*›

8.7.1 Conflict

definition *size-conflict-wl* :: ‹*nat twl-st-wl* ⇒ *nat*› **where**
‹*size-conflict-wl S* = *size* (*the* (*get-conflict-wl S*))›

definition *size-conflict* :: ‹*nat clause option* ⇒ *nat*› **where**
‹*size-conflict D* = *size* (*the D*)›

definition *size-conflict-int* :: ‹*conflict-option-rel* ⇒ *nat*› **where**
‹*size-conflict-int* = (λ(-, *n*, -). *n*)›

8.8 Full state

heur stands for heuristic.

Definition **type-synonym** *twl-st-wl-heur* =
‹*trail-pol* × *arena* ×
conflict-option-rel × *nat* × (*nat watcher*) *list list* × *isa-vmtf-remove-int* ×
nat × *conflict-min-cach-l* × *lbd* × *out-learned* × *stats* × *restart-heuristics* ×
vdom × *vdom* × *nat* × *opts* × *arena*›

Accessors **fun** *get-clauses-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *arena*› **where**
‹*get-clauses-wl-heur* (*M*, *N*, *D*, -) = *N*›

fun *get-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *trail-pol*› **where**
‹*get-trail-wl-heur* (*M*, *N*, *D*, -) = *M*›

fun *get-conflict-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *conflict-option-rel*› **where**
‹*get-conflict-wl-heur* (-, -, *D*, -) = *D*›

fun *watched-by-int* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat watched*› **where**
‹*watched-by-int* (*M*, *N*, *D*, *Q*, *W*, -) *L* = *W* ! *nat-of-lit L*›

fun *get-watched-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*nat watcher*) *list list*› **where**
‹*get-watched-wl-heur* (-, -, -, -, *W*, -) = *W*›

fun *literals-to-update-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*literals-to-update-wl-heur* (*M*, *N*, *D*, *Q*, *W*, -, -) = *Q*›

fun *set-literals-to-update-wl-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*set-literals-to-update-wl-heur i* (*M*, *N*, *D*, -, *W'*) = (*M*, *N*, *D*, *i*, *W'*)›

definition *watched-by-app-heur-pre* **where**
‹*watched-by-app-heur-pre* = (λ((*S*, *L*), *K*). *nat-of-lit L* < *length* (*get-watched-wl-heur S*) ∧
K < *length* (*watched-by-int S L*))›

definition (**in** −) *watched-by-app-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app-heur S L K* = *watched-by-int S L* ! *K*›

definition (**in** −) *mop-watched-by-app-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher nres*›
where
‹*mop-watched-by-app-heur S L K* = *do* {
ASSERT(*K* < *length* (*watched-by-int S L*));
ASSERT(*nat-of-lit L* < *length* (*get-watched-wl-heur S*));
RETURN (*watched-by-int S L* ! *K*)}›

lemma *watched-by-app-heur-alt-def*:
‹*watched-by-app-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L K*. *W* ! *nat-of-lit L* ! *K*)›
by (*auto simp*: *watched-by-app-heur-def intro*!: *ext*)

definition *watched-by-app* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app* S L K = *watched-by* S L ! K›

fun *get-vmtf-heur* :: ‹*twl-st-wl-heur* ⇒ *isa-vmtf-remove-int*› **where**
‹*get-vmtf-heur* (-, -, -, -, -, *vm*, -) = *vm*›

fun *get-count-max-lvls-heur* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*get-count-max-lvls-heur* (-, -, -, -, -, -, *clvls*, -) = *clvls*›

fun *get-conflict-cach*:: ‹*twl-st-wl-heur* ⇒ *conflict-min-cach-l*› **where**
‹*get-conflict-cach* (-, -, -, -, -, -, -, *cach*, -) = *cach*›

fun *get-lbd* :: ‹*twl-st-wl-heur* ⇒ *lbd*› **where**
‹*get-lbd* (-, -, -, -, -, -, -, -, *lbd*, -) = *lbd*›

fun *get-outlearned-heur* :: ‹*twl-st-wl-heur* ⇒ *out-learned*› **where**
‹*get-outlearned-heur* (-, -, -, -, -, -, -, -, -, *out*, -) = *out*›

fun *get-fast-ema-heur* :: ‹*twl-st-wl-heur* ⇒ *ema*› **where**
‹*get-fast-ema-heur* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *fast-ema-of heur*›

fun *get-slow-ema-heur* :: ‹*twl-st-wl-heur* ⇒ *ema*› **where**
‹*get-slow-ema-heur* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *slow-ema-of heur*›

fun *get-conflict-count-heur* :: ‹*twl-st-wl-heur* ⇒ *restart-info*› **where**
‹*get-conflict-count-heur* (-, -, -, -, -, -, -, -, -, -, -, *heur*, -) = *restart-info-of heur*›

fun *get-vdom* :: ‹*twl-st-wl-heur* ⇒ *nat list*› **where**
‹*get-vdom* (-, -, -, -, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*›

fun *get-avdom* :: ‹*twl-st-wl-heur* ⇒ *nat list*› **where**
‹*get-avdom* (-, -, -, -, -, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*›

fun *get-learned-count* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*get-learned-count* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, *lcount*, -) = *lcount*›

fun *get-ops* :: ‹*twl-st-wl-heur* ⇒ *opts*› **where**
‹*get-ops* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *opts*, -) = *opts*›

fun *get-old-arena* :: ‹*twl-st-wl-heur* ⇒ *arena*› **where**
‹*get-old-arena* (-, -, -, -, -, -, -, -, -, -, -, -, -, -, -, -, *old-arena*) = *old-arena*›

8.9 Virtual domain

The virtual domain is composed of the addressable (and accessible) elements, i.e., the domain and all the deleted clauses that are still present in the watch lists.

definition *vdom-m* :: ‹*nat multiset* ⇒ (*nat literal* ⇒ (*nat* × -) *list*) ⇒ (*nat*, '*b*) *fmap* ⇒ *nat set*› **where**
‹*vdom-m* $\mathcal{A}$ W N = $\bigcup$(((‘) *fst*) ‘ *set* ‘ W ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) ∪ *set-mset* (*dom-m* N)›

lemma *vdom-m-simps*[*simp*]:
‹*bh* ∈# *dom-m* N ⟹ *vdom-m* $\mathcal{A}$ W (N(*bh* ↪ C)) = *vdom-m* $\mathcal{A}$ W N›
‹*bh* ∉# *dom-m* N ⟹ *vdom-m* $\mathcal{A}$ W (N(*bh* ↪ C)) = *insert bh* (*vdom-m* $\mathcal{A}$ W N)›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps2*[*simp*]:
‹$i \in\# \ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (W(L := W\ L\ @\ [(i,\ C)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
‹$bi \in\# \ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L:= bp\ L\ @\ [(bi,\ av')]))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps3*[*simp*]:
‹$fst\ biav' \in\# \ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ (bp(L:= bp\ L\ @\ [biav']))\ ax = vdom\text{-}m\ \mathcal{A}\ bp\ ax$›
by (*cases biav'*; *auto simp*: *dest*: *multi-member-split*[*of L*] *split*: *if-splits*)

What is the difference with the next lemma?

lemma [*simp*]:
‹$bf \in\# \ dom\text{-}m\ ax \Longrightarrow vdom\text{-}m\ \mathcal{A}\ bj\ (ax(bf \hookrightarrow C')) = vdom\text{-}m\ \mathcal{A}\ bj\ (ax)$›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps4*[*simp*]:
‹$i \in\# \ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1\ := W\ L1\ @\ [(i,\ C1)],\ L2\ := W\ L2\ @\ [(i,\ C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

This is $?i \in\# \ dom\text{-}m\ ?N \Longrightarrow vdom\text{-}m\ ?\mathcal{A}\ (?W(?L1.0 := ?W\ ?L1.0\ @\ [(?i,\ ?C1.0)],\ ?L2.0 := ?W\ ?L2.0\ @\ [(?i,\ ?C2.0)]))\ ?N = vdom\text{-}m\ ?\mathcal{A}\ ?W\ ?N$ if the assumption of distinctness is not present in the context.

lemma *vdom-m-simps4'*[*simp*]:
‹$i \in\# \ dom\text{-}m\ N \Longrightarrow$
$vdom\text{-}m\ \mathcal{A}\ (W\ (L1\ := W\ L1\ @\ [(i,\ C1),\ (i,\ C2)]))\ N = vdom\text{-}m\ \mathcal{A}\ W\ N$›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

We add a spurious dependency to the parameter of the locale:

definition *empty-watched* :: ‹$nat\ multiset \Rightarrow nat\ literal \Rightarrow (nat \times nat\ literal \times bool)\ list$› **where**
‹$empty\text{-}watched\ \mathcal{A} = (\lambda\text{-}.\ [])$›

lemma *vdom-m-empty-watched*[*simp*]:
‹$vdom\text{-}m\ \mathcal{A}\ (empty\text{-}watched\ \mathcal{A}')\ N = set\text{-}mset\ (dom\text{-}m\ N)$›
by (*auto simp*: *vdom-m-def empty-watched-def*)

The following rule makes the previous one not applicable. Therefore, we do not mark this lemma as simp.

lemma *vdom-m-simps5*:
‹$i \notin\# \ dom\text{-}m\ N \Longrightarrow vdom\text{-}m\ \mathcal{A}\ W\ (fmupd\ i\ C\ N) = insert\ i\ (vdom\text{-}m\ \mathcal{A}\ W\ N)$›
by (*force simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

lemma *in-watch-list-in-vdom*:
assumes ‹$L \in\# \ \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$w < length\ (watched\text{-}by\ S\ L)$›
shows ‹$fst\ (watched\text{-}by\ S\ L\ !\ w) \in vdom\text{-}m\ \mathcal{A}\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S)$›
using *assms*
unfolding *vdom-m-def*
by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-watch-list-in-vdom'*:
assumes ‹$L \in\# \ \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$A \in set\ (watched\text{-}by\ S\ L)$›
shows ‹$fst\ A \in vdom\text{-}m\ \mathcal{A}\ (get\text{-}watched\text{-}wl\ S)\ (get\text{-}clauses\text{-}wl\ S)$›
using *assms*
unfolding *vdom-m-def*

by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-dom-in-vdom*[*simp*]:
‹$x \in\# \ dom\text{-}m\ N \Longrightarrow x \in vdom\text{-}m\ \mathcal{A}\ W\ N$›
unfolding *vdom-m-def*
by (*auto dest*: *multi-member-split*)

lemma *in-vdom-m-upd*:
‹$x1f \in vdom\text{-}m\ \mathcal{A}\ (g(x1e := (g\ x1e)[x2 := (x1f,\ x2f)]))\ b$›
if ‹$x2 < length\ (g\ x1e)$› **and** ‹$x1e \in\# \mathcal{L}_{all}\ \mathcal{A}$›
using *that*
unfolding *vdom-m-def*
by (*auto dest*!: *multi-member-split intro*!: *set-update-memI img-fst*)

lemma *in-vdom-m-fmdropD*:
‹$x \in vdom\text{-}m\ \mathcal{A}\ ga\ (fmdrop\ C\ baa) \Longrightarrow x \in (vdom\text{-}m\ \mathcal{A}\ ga\ baa)$›
unfolding *vdom-m-def*
by (*auto dest*: *in-diffD*)

definition *cach-refinement-empty* **where**
‹$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \longleftrightarrow$
$(cach,\ \lambda\text{-}.\ SEEN\text{-}UNKNOWN) \in cach\text{-}refinement\ \mathcal{A}$›

VMTF **definition** *isa-vmtf* **where**
‹$isa\text{-}vmtf\ \mathcal{A}\ M =$
$((Id \times_r nat\text{-}rel \times_r nat\text{-}rel \times_r nat\text{-}rel \times_r \langle nat\text{-}rel\rangle option\text{-}rel) \times_f distinct\text{-}atoms\text{-}rel\ \mathcal{A})^{-1}$
$``\ vmtf\ \mathcal{A}\ M$›

lemma *isa-vmtfI*:
‹$(vm,\ to\text{-}remove') \in vmtf\ \mathcal{A}\ M \Longrightarrow (to\text{-}remove,\ to\text{-}remove') \in distinct\text{-}atoms\text{-}rel\ \mathcal{A} \Longrightarrow$
$(vm,\ to\text{-}remove) \in isa\text{-}vmtf\ \mathcal{A}\ M$›
by (*auto simp*: *isa-vmtf-def Image-iff intro*!: *bexI*[*of* - ‹(*vm*, *to-remove*′)›])

lemma *isa-vmtf-consD*:
‹$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ remove) \in isa\text{-}vmtf\ \mathcal{A}\ M \Longrightarrow$
$((ns,\ m,\ fst\text{-}As,\ lst\text{-}As,\ next\text{-}search),\ remove) \in isa\text{-}vmtf\ \mathcal{A}\ (L\ \#\ M)$›
by (*auto simp*: *isa-vmtf-def dest*: *vmtf-consD*)

lemma *isa-vmtf-consD2*:
‹$f \in isa\text{-}vmtf\ \mathcal{A}\ M \Longrightarrow$
$f \in isa\text{-}vmtf\ \mathcal{A}\ (L\ \#\ M)$›
by (*auto simp*: *isa-vmtf-def dest*: *vmtf-consD*)

vdom is an upper bound on all the address of the clauses that are used in the state. *avdom* includes the active clauses.

definition *twl-st-heur* :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹$twl\text{-}st\text{-}heur =$
$\{((M',\ N',\ D',\ j,\ W',\ vm,\ clvls,\ cach,\ lbd,\ outl,\ stats,\ heur,$
$vdom,\ avdom,\ lcount,\ opts,\ old\text{-}arena),$
$(M,\ N,\ D,\ NE,\ UE,\ NS,\ US,\ Q,\ W)).$
$(M',\ M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE + NS + US)) \land$
$valid\text{-}arena\ N'\ N\ (set\ vdom) \land$
$(D',\ D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US)) \land$
$(D = None \longrightarrow j \le length\ M) \land$

$Q = uminus$ ‘# $lit\text{-}of$ ‘# $mset$ $(drop\ j\ (rev\ M))$ ∧
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}atms\ N\ (NE + UE + NS + US)))$ ∧
$vm \in isa\text{-}vmtf\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ M$ ∧
$no\text{-}dup\ M$ ∧
$clvls \in counts\text{-}maximum\text{-}level\ M\ D$ ∧
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ cach$ ∧
$out\text{-}learned\ M\ D\ outl$ ∧
$lcount = size\ (learned\text{-}clss\text{-}lf\ N)$ ∧
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \ W\ N \subseteq set\ vdom$ ∧
$mset\ avdom \subseteq\#\ mset\ vdom$ ∧
$distinct\ vdom$ ∧
$isasat\text{-}input\text{-}bounded\ (all\text{-}atms\ N\ (NE + UE + NS + US))$ ∧
$isasat\text{-}input\text{-}nempty\ (all\text{-}atms\ N\ (NE + UE + NS + US))$ ∧
$old\text{-}arena = []$ ∧
$heuristic\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ heur$
}›

lemma *twl-st-heur-state-simp*:
assumes ‹$(S, S') \in twl\text{-}st\text{-}heur$›
shows
‹$(get\text{-}trail\text{-}wl\text{-}heur\ S,\ get\text{-}trail\text{-}wl\ S') \in trail\text{-}pol\ (all\text{-}atms\text{-}st\ S')$› **and**
twl-st-heur-state-simp-watched: ‹$C \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S') \Longrightarrow$
$watched\text{-}by\text{-}int\ S\ C = watched\text{-}by\ S'\ C$› **and**
‹$literals\text{-}to\text{-}update\text{-}wl\ S' =$
$uminus$ ‘# $lit\text{-}of$ ‘# $mset\ (drop\ (literals\text{-}to\text{-}update\text{-}wl\text{-}heur\ S)\ (rev\ (get\text{-}trail\text{-}wl\ S')))$› **and**
twl-st-heur-state-simp-watched2: ‹$C \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S') \Longrightarrow$
$nat\text{-}of\text{-}lit\ C < length(get\text{-}watched\text{-}wl\text{-}heur\ S)$›
using *assms* **unfolding** *twl-st-heur-def* **by** (*auto simp*: *map-fun-rel-def ac-simps*)

abbreviation *twl-st-heur'''*
:: ‹$nat \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$›
where
‹$twl\text{-}st\text{-}heur'''\ r \equiv \{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\ \wedge$
$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) = r\}$›

definition *twl-st-heur'* :: ‹$nat\ multiset \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$› **where**
‹$twl\text{-}st\text{-}heur'\ N = \{(S, S').\ (S, S') \in twl\text{-}st\text{-}heur \wedge dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S') = N\}$›

definition *twl-st-heur-conflict-ana*
:: ‹$(twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$›
where
‹$twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana =$
$\{((M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur, vdom,$
$avdom, lcount, opts, old\text{-}arena),$
$(M, N, D, NE, UE, NS, US, Q, W)).$
$(M', M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE + NS + US))$ ∧
$valid\text{-}arena\ N'\ N\ (set\ vdom)$ ∧
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US))$ ∧
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}atms\ N\ (NE + UE + NS + US)))$ ∧
$vm \in isa\text{-}vmtf\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ M$ ∧
$no\text{-}dup\ M$ ∧
$clvls \in counts\text{-}maximum\text{-}level\ M\ D$ ∧
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ cach$ ∧
$out\text{-}learned\ M\ D\ outl$ ∧
$lcount = size\ (learned\text{-}clss\text{-}lf\ N)$ ∧
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ W\ N \subseteq set\ vdom$ ∧

$mset\ avdom \subseteq\# \ mset\ vdom\ \wedge$
$distinct\ vdom\ \wedge$
$isasat\text{-}input\text{-}bounded\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$isasat\text{-}input\text{-}nempty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$old\text{-}arena = []\ \wedge$
$heuristic\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ heur$
$\}$›

lemma *twl-st-heur-twl-st-heur-conflict-ana*:
‹$(S, T) \in twl\text{-}st\text{-}heur \Longrightarrow (S, T) \in twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana$›
by (*auto simp*: *twl-st-heur-def twl-st-heur-conflict-ana-def ac-simps*)

lemma *twl-st-heur-ana-state-simp*:
assumes ‹$(S, S') \in twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana$›
shows
‹$(get\text{-}trail\text{-}wl\text{-}heur\ S,\ get\text{-}trail\text{-}wl\ S') \in trail\text{-}pol\ (all\text{-}atms\text{-}st\ S')$› **and**
‹$C \in\# \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S') \Longrightarrow watched\text{-}by\text{-}int\ S\ C = watched\text{-}by\ S'\ C$›
using *assms* **unfolding** *twl-st-heur-conflict-ana-def* **by** (*auto simp*: *map-fun-rel-def ac-simps*)

This relations decouples the conflict that has been minimised and appears abstractly from the refined state, where the conflict has been removed from the data structure to a separate array.

definition *twl-st-heur-bt* :: ‹$(twl\text{-}st\text{-}wl\text{-}heur \times nat\ twl\text{-}st\text{-}wl)\ set$› **where**
‹$twl\text{-}st\text{-}heur\text{-}bt =$
$\{((M', N', D', Q', W', vm, clvls, cach, lbd, outl, stats, heur, vdom, avdom, lcount, opts,$
$old\text{-}arena),$
$(M, N, D, NE, UE, NS, US, Q, W)).$
$(M', M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$valid\text{-}arena\ N'\ N\ (set\ vdom)\ \wedge$
$(D', None) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}atms\ N\ (NE + UE + NS + US)))\ \wedge$
$vm \in isa\text{-}vmtf\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ M\ \wedge$
$no\text{-}dup\ M\ \wedge$
$clvls \in counts\text{-}maximum\text{-}level\ M\ None\ \wedge$
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ cach\ \wedge$
$out\text{-}learned\ M\ None\ outl\ \wedge$
$lcount = size\ (learned\text{-}clss\text{-}l\ N)\ \wedge$
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ W\ N \subseteq set\ vdom\ \wedge$
$mset\ avdom \subseteq\# \ mset\ vdom\ \wedge$
$distinct\ vdom\ \wedge$
$isasat\text{-}input\text{-}bounded\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$isasat\text{-}input\text{-}nempty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \wedge$
$old\text{-}arena = []\ \wedge$
$heuristic\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ heur$
$\}$›

The difference between *isasat-unbounded-assn* and *isasat-bounded-assn* corresponds to the following condition:

definition *isasat-fast* :: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow bool$› **where**
‹$isasat\text{-}fast\ S \longleftrightarrow (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max - (uint32\text{-}max\ div\ 2 + 6))$›

lemma *isasat-fast-length-leD*: ‹$isasat\text{-}fast\ S \Longrightarrow length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max$›
by (*cases S*) (*auto simp*: *isasat-fast-def*)

8.10 Lift Operations to State

definition *polarity-st* :: ⟨′v twl-st-wl ⇒ ′v literal ⇒ bool option⟩ **where**
⟨*polarity-st S* = *polarity* (*get-trail-wl S*)⟩

definition *get-conflict-wl-is-None-heur* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*get-conflict-wl-is-None-heur* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *b*)⟩

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None*:
⟨(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur $\rightarrow_f$ ⟨*Id*⟩*nres-rel*⟩
unfolding *get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def*
apply (*intro WB-More-Refinement.frefI nres-relI*) **apply** *refine-rcg*
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def*
option-lookup-clause-rel-def
split: *option.splits*)

lemma *get-conflict-wl-is-None-heur-alt-def*:
⟨*RETURN o get-conflict-wl-is-None-heur* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)⟩
unfolding *get-conflict-wl-is-None-heur-def*
by *auto*

definition *count-decided-st* :: ⟨*nat twl-st-wl* ⇒ *nat*⟩ **where**
⟨*count-decided-st* = (λ(*M*, -). *count-decided M*)⟩

definition *isa-count-decided-st* :: ⟨*twl-st-wl-heur* ⇒ *nat*⟩ **where**
⟨*isa-count-decided-st* = (λ(*M*, -). *count-decided-pol M*)⟩

lemma *count-decided-st-count-decided-st*:
⟨(*RETURN o isa-count-decided-st*, *RETURN o count-decided-st*) ∈ *twl-st-heur* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*⟩
by (*intro WB-More-Refinement.frefI nres-relI*)
(*auto simp*: *count-decided-st-def twl-st-heur-def isa-count-decided-st-def*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

lemma *count-decided-st-alt-def*: ⟨*count-decided-st S* = *count-decided* (*get-trail-wl S*)⟩
unfolding *count-decided-st-def*
by (*cases S*) *auto*

definition (**in** −) *is-in-conflict-st* :: ⟨*nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*⟩ **where**
⟨*is-in-conflict-st L S* ⟷ *is-in-conflict L* (*get-conflict-wl S*)⟩

definition *atm-is-in-conflict-st-heur* :: ⟨*nat literal* ⇒ *twl-st-wl-heur* ⇒ *bool nres*⟩ **where**
⟨*atm-is-in-conflict-st-heur L* = (λ(*M*, *N*, (-, *D*), -). *do* {
ASSERT (*atm-in-conflict-lookup-pre* (*atm-of L*) *D*); *RETURN* (¬*atm-in-conflict-lookup* (*atm-of L*)
D) })⟩

lemma *atm-is-in-conflict-st-heur-alt-def*:
⟨*atm-is-in-conflict-st-heur* = (λ*L* (*M*, *N*, (-, (-, *D*)), -). *do* {*ASSERT* ((*atm-of L*) < *length D*); *RE-TURN* (*D* ! (*atm-of L*) = *None*)})⟩
unfolding *atm-is-in-conflict-st-heur-def* **by** (*auto intro*!: *ext simp*: *atm-in-conflict-lookup-def atm-in-conflict-lookup-pre-*

lemma *atm-of-in-atms-of-iff*: ⟨*atm-of x* ∈ *atms-of D* ⟷ *x* ∈# *D* ∨ −*x* ∈# *D*⟩
by (*cases x*) (*auto simp*: *atms-of-def dest*!: *multi-member-split*)

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*atm-is-in-conflict-st-heur*), *uncurry* (*mop-lit-notin-conflict-wl*)) ∈
[λ(*L*, *S*). *True*]$_f$
Id $\times_r$ *twl-st-heur* → ⟨*Id*⟩ *nres-rel*›
proof −
have *1*: ‹*aaa* ∈# $\mathcal{L}_{all}$ *A* ⟹ *atm-of aaa* ∈ *atms-of* ($\mathcal{L}_{all}$ *A*)› **for** *aaa A*
by (*auto simp*: *atms-of-def*)
show *?thesis*
unfolding *atm-is-in-conflict-st-heur-def twl-st-heur-def option-lookup-clause-rel-def uncurry-def comp-def mop-lit-notin-conflict-wl-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
apply *clarsimp*
subgoal
apply (*rule atm-in-conflict-lookup-pre*)
unfolding $\mathcal{L}_{all}$*-all-atms-all-lits*[*symmetric*]
apply *assumption*+
apply (*auto simp*: *ac-simps*)
done
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x1e x2d x2e*
apply (*subst atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*, *of* ‹*all-atms-st x2*› ‹*atm-of x1*› ‹*the* (*get-conflict-wl* (*snd y*))›])
apply (*simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def*)[]
apply (*auto simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def option-lookup-clause-rel-def ac-simps*)[]
apply (*simp add*: *atm-in-conflict-def atm-of-in-atms-of-iff*)
done
done
qed

abbreviation *nat-lit-lit-rel* **where**
‹*nat-lit-lit-rel* ≡ *Id* :: (*nat literal* × -) *set*›

8.11 More theorems

lemma *valid-arena-DECISION-REASON*:
‹*valid-arena arena NU vdom* ⟹ *DECISION-REASON* ∉# *dom-m NU*›
using *arena-lifting*[*of arena NU vdom DECISION-REASON*]
by (*auto simp*: *DECISION-REASON-def SHIFTS-def*)

definition *count-decided-st-heur* :: ‹- ⇒ -› **where**
‹*count-decided-st-heur* = (λ((-,-,-,-,*n*, -), -). *n*)›

lemma *twl-st-heur-count-decided-st-alt-def*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
unfolding *count-decided-st-def twl-st-heur-def trail-pol-def*
by (*cases S*) (*auto simp*: *count-decided-st-heur-def*)

lemma *twl-st-heur-isa-length-trail-get-trail-wl*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *isa-length-trail* (*get-trail-wl-heur S*) = *length* (*get-trail-wl T*)›
unfolding *isa-length-trail-def twl-st-heur-def trail-pol-def*

by (*cases S*) (*auto dest*: *ann-lits-split-reasons-map-lit-of*)

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *trail-pol* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *trail-pol* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *distinct-atoms-rel* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *distinct-atoms-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf*-$\mathcal{L}_{all}$-*def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
atoms-hash-rel-def
by (*auto simp*:)

lemma *phase-save-heur-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *phase-save-heur-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *phase-save-heur-rel* $\mathcal{B}$ *heur*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-save-heur-rel-def phase-saving-def*)

lemma *heuristic-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{B}$ *heur*›
using *phase-save-heur-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$ ‹(λ(-, -, -, -, *a*). *a*) *heur*›]
by (*auto simp*: *heuristic-rel-def*)

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ $L \in$ *vmtf* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf*-$\mathcal{L}_{all}$-*def*
by *auto*

lemma *isa-vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *isa-vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ $L \in$ *isa-vmtf* $\mathcal{B}$ *M*›
using *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
apply (*subst* (*asm*) *isa-vmtf-def*)
apply (*cases L*)
by (*auto intro*!: *isa-vmtfI*)

lemma *option-lookup-clause-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *option-lookup-clause-rel* $\mathcal{A}$ $\Longrightarrow$ $L \in$ *option-lookup-clause-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
apply (*cases L*)
by (*auto intro*!: *isa-vmtfI*)

lemma D_0-*cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ D_0 $\mathcal{A}$ = D_0 $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by *auto*

lemma *phase-saving-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *phase-saving* $\mathcal{A}$ = *phase-saving* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of*-$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-saving-def*)

lemma *distinct-subseteq-iff2*:
  assumes *dist*: *distinct-mset M*
  shows *set-mset M* $\subseteq$ *set-mset N* $\longleftrightarrow$ $M \subseteq\# N$
proof
  assume *set-mset M* $\subseteq$ *set-mset N*
  then show $M \subseteq\# N$
    using *dist* **by** (*metis distinct-mset-set-mset-ident mset-set-subset-iff*)
next
  assume $M \subseteq\# N$
  then show *set-mset M* $\subseteq$ *set-mset N*
    by (*metis set-mset-mono*)
qed

lemma *cach-refinement-empty-cong*:
  ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *cach-refinement-empty* $\mathcal{A}$ = *cach-refinement-empty* $\mathcal{B}$›
  using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
  by (*force simp*: *cach-refinement-empty-def cach-refinement-alt-def*
    distinct-subseteq-iff2[*symmetric*] *intro*!: *ext*)

lemma *vdom-m-cong*:
  ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *vdom-m* $\mathcal{A}$ *x y* = *vdom-m* $\mathcal{B}$ *x y*›
  using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
  by (*auto simp*: *vdom-m-def intro*!: *ext*)

lemma *isasat-input-bounded-cong*:
  ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ = *isasat-input-bounded* $\mathcal{B}$›
  using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
  by (*auto simp*: *intro*!: *ext*)

lemma *isasat-input-nempty-cong*:
  ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *isasat-input-nempty* $\mathcal{A}$ = *isasat-input-nempty* $\mathcal{B}$›
  using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
  by (*auto simp*: *intro*!: *ext*)

8.12 Shared Code Equations

definition *clause-not-marked-to-delete* **where**
  ‹*clause-not-marked-to-delete S C* $\longleftrightarrow$ $C \in\#$ *dom-m* (*get-clauses-wl S*)›

definition *clause-not-marked-to-delete-pre* **where**
  ‹*clause-not-marked-to-delete-pre* =
    (λ(*S*, *C*). $C \in$ *vdom-m* (*all-atms-st S*) (*get-watched-wl S*) (*get-clauses-wl S*))›

definition *clause-not-marked-to-delete-heur-pre* **where**
  ‹*clause-not-marked-to-delete-heur-pre* =
     (λ(*S*, *C*). *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*)›

definition *clause-not-marked-to-delete-heur* :: ‹- $\Rightarrow$ *nat* $\Rightarrow$ *bool*›
where
  ‹*clause-not-marked-to-delete-heur S C* $\longleftrightarrow$
    arena-status (*get-clauses-wl-heur S*) $C \neq$ *DELETED*›

lemma *clause-not-marked-to-delete-rel*:

‹(*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*),
uncurry (*RETURN oo clause-not-marked-to-delete*)) ∈
[*clause-not-marked-to-delete-pre*]$_f$
twl-st-heur $\times_f$ *nat-rel* → ⟨*bool-rel*⟩*nres-rel*›
by (*intro WB-More-Refinement.frefI nres-relI*)
(*use arena-dom-status-iff in-dom-in-vdom* **in**
‹*auto 5 5 simp*: *clause-not-marked-to-delete-def twl-st-heur-def*
clause-not-marked-to-delete-heur-def arena-dom-status-iff
clause-not-marked-to-delete-pre-def ac-simps›)

definition (**in** −) *access-lit-in-clauses-heur-pre* **where**
‹*access-lit-in-clauses-heur-pre* =
(λ((*S*, *i*), *j*).
arena-lit-pre (*get-clauses-wl-heur S*) (*i*+*j*))›

definition (**in** −) *access-lit-in-clauses-heur* **where**
‹*access-lit-in-clauses-heur S i j* = *arena-lit* (*get-clauses-wl-heur S*) (*i* + *j*)›

lemma *access-lit-in-clauses-heur-alt-def*:
‹*access-lit-in-clauses-heur* = (λ(*M*, *N*, -) *i j*. *arena-lit N* (*i* + *j*))›
by (*auto simp*: *access-lit-in-clauses-heur-def intro*!: *ext*)

definition (**in** −) *mop-access-lit-in-clauses-heur* **where**
‹*mop-access-lit-in-clauses-heur S i j* = *mop-arena-lit2* (*get-clauses-wl-heur S*) *i j*›

lemma *mop-access-lit-in-clauses-heur-alt-def*:
‹*mop-access-lit-in-clauses-heur* = (λ(*M*, *N*, -) *i j*. *mop-arena-lit2 N i j*)›
by (*auto simp*: *mop-access-lit-in-clauses-heur-def intro*!: *ext*)

lemma *access-lit-in-clauses-heur-fast-pre*:
‹*arena-lit-pre* (*get-clauses-wl-heur a*) (*ba* + *b*) ⟹
isasat-fast a ⟹ *ba* + *b* ≤ *sint64-max*›
by (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
dest!: *arena-lifting*(*10*)
dest!: *isasat-fast-length-leD*)[]

lemma *eq-insertD*: ‹*A* = *insert a B* ⟹ *a* ∈ *A* ∧ *B* ⊆ *A*›
by *auto*

lemma $\mathcal{L}_{all}$*-add-mset*:
‹*set-mset* ($\mathcal{L}_{all}$ (*add-mset L C*)) = *insert* (*Pos L*) (*insert* (*Neg L*) (*set-mset* ($\mathcal{L}_{all}$ *C*)))›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *correct-watching-dom-watched*:
assumes ‹*correct-watching S*› **and** ‹⋀*C*. *C* ∈# *ran-mf* (*get-clauses-wl S*) ⟹ *C* ≠ []›
shows ‹*set-mset* (*dom-m* (*get-clauses-wl S*)) ⊆
⋃(((‘) *fst*) ‘ *set* ‘ (*get-watched-wl S*) ‘ *set-mset* ($\mathcal{L}_{all}$ (*all-atms-st S*)))›
(**is** ‹*?A* ⊆ *?B*›)
proof
fix *C*
assume ‹*C* ∈ *?A*›

then obtain D **where**
  D: ‹$D \in\#$ *ran-mf* (*get-clauses-wl* S)› **and**
  D': ‹D = *get-clauses-wl* $S \propto C$› **and**
  C: ‹$C \in\#$ *dom-m* (*get-clauses-wl* S)›
  by *auto*
have ‹*atm-of* (*hd* D) $\in\#$ *atm-of* '# *all-lits-st* S›
  using D D' *assms*(*2*)[*of* D]
  by (*cases* S; *cases* D)
    (*auto simp*: *all-lits-def*
        all-lits-of-mm-add-mset all-lits-of-m-add-mset
      dest!: *multi-member-split*)
then show ‹$C \in$?*B*›
  using *assms*(*1*) *assms*(*2*)[*of* D] D D'
    multi-member-split[*OF* C]
  by (*cases* S; *cases* ‹*get-clauses-wl* $S \propto C$›;
      cases ‹*hd* (*get-clauses-wl* $S \propto C$)›)
    (*auto simp*: *correct-watching.simps clause-to-update-def*
        all-lits-of-mm-add-mset all-lits-of-m-add-mset
  $\mathcal{L}_{all}$-*add-mset*
  eq-commute[*of* ‹- # -›] *atm-of-eq-atm-of*
      simp flip: *all-atms-def*
dest!: *multi-member-split eq-insertD*
dest!: *arg-cong*[*of* ‹*filter-mset* - -› ‹*add-mset* - -› *set-mset*])
qed

8.13 Rewatch

definition *rewatch-heur* **where**
‹*rewatch-heur vdom arena W = do* {
  let - = vdom;
  nfoldli [*0*..<*length vdom*] (λ-. *True*)
  (λi W. *do* {
    ASSERT(i < *length vdom*);
    let C = *vdom* ! i;
    ASSERT(*arena-is-valid-clause-vdom arena* C);
    if arena-status arena $C \neq$ *DELETED*
    then do {
      L1 ← *mop-arena-lit2 arena* C *0*;
      L2 ← *mop-arena-lit2 arena* C *1*;
      n ← *mop-arena-length arena* C;
      let b = (n = *2*);
      ASSERT(*length* (W ! (*nat-of-lit L1*)) < *length arena*);
      W ← *mop-append-ll* W *L1* (C, *L2*, b);
      ASSERT(*length* (W ! (*nat-of-lit L2*)) < *length arena*);
      W ← *mop-append-ll* W *L2* (C, *L1*, b);
      RETURN W
    }
    else RETURN W
  })
  W
}›

lemma *rewatch-heur-rewatch*:
  assumes
    valid: ‹*valid-arena arena* N *vdom*› **and** ‹*set xs* ⊆ *vdom*› **and** ‹*distinct xs*› **and** ‹*set-mset* (*dom-m* N)

$\subseteq$ *set xs*› **and**
‹$(W, W') \in \langle Id\rangle$*map-fun-rel* $(D_0\ \mathcal{A})$› **and** *lall*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
‹*vdom-m* $\mathcal{A}$ W' $N \subseteq$ *set-mset* (*dom-m N*)›
shows
‹*rewatch-heur xs arena W* $\leq \Downarrow$ $(\{(W, W').\ (W, W') \in\langle Id\rangle$*map-fun-rel* $(D_0\ \mathcal{A}) \wedge$ *vdom-m* $\mathcal{A}$ W' N $\subseteq$ *set-mset* (*dom-m N*)$\}$) (*rewatch N W'*)›
proof −
have [*refine0*]: ‹(*xs*, *xsa*) $\in$ *Id* $\Longrightarrow$
([*0..<length xs*], [*0..<length xsa*]) $\in \langle\{(x, x').\ x = x' \wedge x <$ *length xsa* $\wedge$ *xs*!*x* $\in$ *vdom*$\}\rangle$*list-rel*›
for *xsa*
using *assms* **unfolding** *list-rel-def*
by (*auto simp*: *list-all2-same*)
show *?thesis*
unfolding *rewatch-heur-def rewatch-def*
apply (*subst* (*2*) *nfoldli-nfoldli-list-nth*)
apply (*refine-vcg mop-arena-lit*[*OF valid*] *mop-append-ll*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*, *unfolded comp-def*]
mop-arena-length[*of vdom*, *THEN fref-to-Down-curry*, *unfolded comp-def*])
subgoal
using *assms* **by** *fast*
subgoal
using *assms* **by** *fast*
subgoal
using *assms* **by** *fast*
subgoal by *fast*
subgoal by *auto*
subgoal
using *assms*
unfolding *arena-is-valid-clause-vdom-def*
by *blast*
subgoal
using *assms*
by (*auto simp*: *arena-dom-status-iff*)
subgoal for *xsa xi x si s*
using *assms*
by *auto*
subgoal by *simp*
subgoal by *linarith*
subgoal for *xsa xi x si s*
using *assms*
unfolding *arena-lit-pre-def*
by (*auto*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal for *xsa xi x si s*
using *assms*
unfolding *arena-is-valid-clause-idx-and-access-def*
arena-is-valid-clause-idx-def
by (*auto simp*: *arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *N*] *exI*[*of* - *vdom*])
subgoal for *xsa xi x si s*
using *valid-arena-size-dom-m-le-arena*[*OF assms*(*1*)] *assms*
literals-are-in-$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of* ‹*xs* ! *xi*› *0*]
by (*auto simp*: *map-fun-rel-def arena-lifting*)
subgoal for *xsa xi x si s*

using *valid-arena-size-dom-m-le-arena*[*OF assms*(*1*)] *assms*
literals-are-in-$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of* ‹*xs* ! *xi*› *0*]
by (*auto simp*: *map-fun-rel-def arena-lifting*)
subgoal using *assms* **by** (*simp add*: *arena-lifting*)
subgoal for *xsa xi x si s*
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of* ‹*xs* ! *xi*› *1*]
assms valid-arena-size-dom-m-le-arena[*OF assms*(*1*)]
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
subgoal for *xsa xi x si s*
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*OF lall*, *of* ‹*xs* ! *xi*› *1*]
assms
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
subgoal for *xsa xi x si s*
using *assms*
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
subgoal for *xsa xi x si s*
using *assms*
by (*auto simp*: *arena-lifting append-ll-def map-fun-rel-def*)
done
qed

lemma *rewatch-heur-alt-def*:
‹*rewatch-heur vdom arena W* = *do* {
let - = *vdom*;
nfoldli [*0*..<*length vdom*] (λ-. *True*)
(λ*i W*. *do* {
ASSERT(*i* < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C ≠ *DELETED*
then do {
L1 ← *mop-arena-lit2 arena C 0*;
L2 ← *mop-arena-lit2 arena C 1*;
n ← *mop-arena-length arena C*;
let b = (*n* = *2*);
ASSERT(*length* (*W* ! (*nat-of-lit L1*)) < *length arena*);
W ← *mop-append-ll W L1* (*C*, *L2*, *b*);
ASSERT(*length* (*W* ! (*nat-of-lit L2*)) < *length arena*);
W ← *mop-append-ll W L2* (*C*, *L1*, *b*);
RETURN W
}
else RETURN W
})
W
}›
unfolding *Let-def rewatch-heur-def*
by *auto*

lemma *arena-lit-pre-le-sint64-max*:
‹*length ba* ≤ *sint64-max* ⟹
arena-lit-pre ba a ⟹ *a* ≤ *sint64-max*›
using *arena-lifting*(*10*)[*of ba* - -]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def)

definition *rewatch-heur-st*

:: ‹twl-st-wl-heur ⇒ twl-st-wl-heur nres›
where
‹rewatch-heur-st = (λ(M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
    stats, heur, vdom, avdom, ccount, lcount). do {
  ASSERT(length vdom ≤ length N0);
  W ← rewatch-heur vdom N0 W;
  RETURN (M, N0, D, Q, W, vm, clvls, cach, lbd, outl,
    stats, heur, vdom, avdom, ccount, lcount)
  })›

definition *rewatch-heur-st-fast* **where**
‹rewatch-heur-st-fast = rewatch-heur-st›

definition *rewatch-heur-st-fast-pre* **where**
‹rewatch-heur-st-fast-pre S =
  ((∀ x ∈ set (get-vdom S). x ≤ sint64-max) ∧ length (get-clauses-wl-heur S) ≤ sint64-max)›

definition *rewatch-st* :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres› **where**
‹rewatch-st S = do{
  (M, N, D, NE, UE, NS, US, Q, W) ← RETURN S;
  W ← rewatch N W;
  RETURN ((M, N, D, NE, UE, NS, US, Q, W))
}›

fun *remove-watched-wl* :: ‹'v twl-st-wl ⇒ -› **where**
‹remove-watched-wl (M, N, D, NE, UE, NS, US, Q, -) = (M, N, D, NE, UE, NS, US, Q)›

lemma *rewatch-st-correctness*:
  assumes ‹get-watched-wl S = (λ-. [])› **and**
  ‹⋀x. x ∈# dom-m (get-clauses-wl S) ⟹
    distinct ((get-clauses-wl S) ∝ x) ∧ 2 ≤ length ((get-clauses-wl S) ∝ x)›
  shows ‹rewatch-st S ≤ SPEC (λT. remove-watched-wl S = remove-watched-wl T ∧
    correct-watching-init T)›
  apply (*rule SPEC-rule-conjI*)
  subgoal
    using *rewatch-correctness*[*OF assms*]
    unfolding *rewatch-st-def*
    apply (*cases S*, *case-tac* ‹*rewatch b i*›)
    by (*auto simp*: *RES-RETURN-RES*)
  subgoal
    using *rewatch-correctness*[*OF assms*]
    unfolding *rewatch-st-def*
    apply (*cases S*, *case-tac* ‹*rewatch b i*›)
    by (*force simp*: *RES-RETURN-RES*)+
  done

8.14 Fast to slow conversion

Setup to convert a list from *64 word* to *nat*.

definition *convert-wlists-to-nat-conv* :: ‹'a list list ⇒ 'a list list› **where**
‹convert-wlists-to-nat-conv = id›

abbreviation *twl-st-heur''*

:: ⟨*nat multiset* ⇒ *nat* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*⟩
where
⟨*twl-st-heur*'' $\mathcal{D}$ *r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*' $\mathcal{D}$ ∧
length (*get-clauses-wl-heur S*) = *r*}⟩

abbreviation *twl-st-heur-up*''
:: ⟨*nat multiset* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ (*twl-st-wl-heur* × *nat twl-st-wl*) *set*⟩
where
⟨*twl-st-heur-up*'' $\mathcal{D}$ *r s L* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*'' $\mathcal{D}$ *r* ∧
length (*watched-by T L*) = *s* ∧ *s* ≤ *r*}⟩

lemma *length-watched-le*:
assumes
prop-inv: ⟨*correct-watching x1*⟩ **and**
xb-x'a: ⟨(*x1a*, *x1*) ∈ *twl-st-heur*'' $\mathcal{D}1$ *r*⟩ **and**
x2: ⟨*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)⟩
shows ⟨*length* (*watched-by x1 x2*) ≤ *r* − *4*⟩
proof −
have *dist*: ⟨*distinct-watched* (*watched-by x1 x2*)⟩
using *prop-inv x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps ac-simps*)
then have *dist*: ⟨*distinct-watched* (*watched-by x1 x2*)⟩
using *xb-x'a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ⟨*distinct* (*get-vdom x1a*)⟩
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
have *x2*: ⟨*x2* ∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*)
(*get-unit-clauses-wl x1* + *get-subsumed-clauses-wl x1*))⟩
using *x2 xb-x'a* **unfolding** *all-atms-def*
by *auto*

have
valid: ⟨*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))⟩
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur'-def twl-st-heur-def*)

have ⟨*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)⟩
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def ac-simps*)

then have *subset*: ⟨*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)⟩
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur'-def twl-st-heur-def*
dest!: *multi-member-split*)
have *watched-incl*: ⟨*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)⟩
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
⟨*simp-all flip*: *distinct-mset-mset-distinct*⟩)
have *vdom-incl*: ⟨*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}⟩
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *4*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF* - *vdom-incl*] **in** *auto*)
then show *?thesis*
using *size-mset-mono*[*OF watched-incl*] *xb-x′a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

lemma *length-watched-le2*:
assumes
prop-inv: ‹*correct-watching-except i j L x1*› **and**
xb-x′a: ‹(*x1a*, *x1*) ∈ *twl-st-heur″ 𝒟1 r*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)› **and** *diff*: ‹*L* ≠ *x2*›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *4*›
proof −
from *prop-inv diff* **have** *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching-except.simps ac-simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x′a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur′-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms* (*get-clauses-wl x1*) (*get-unit-clauses-wl x1* + *get-subsumed-clauses-wl x1*))›
using *x2 xb-x′a*
by (*auto simp flip*: *all-atms-def all-lits-alt-def2 simp*: *ac-simps*)

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x′a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur′-def ac-simps simp flip*: *all-atms-def*)
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur′-def twl-st-heur-def ac-simps simp flip*: *all-atms-def all-lits-alt-def2*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *4*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF* - *vdom-incl*] **in** *auto*)
then show *?thesis*

using *size-mset-mono*[*OF watched-incl*] *xb-x′a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

lemma *atm-of-all-lits-of-m*: ‹*atm-of* ‘# (*all-lits-of-m C*) = *atm-of* ‘# *C* + *atm-of* ‘# *C*›
‹*atm-of* ‘ *set-mset* (*all-lits-of-m C*) = *atm-of* ‘*set-mset C* ›
by (*induction C*; *auto simp*: *all-lits-of-m-add-mset*)+

lemma *mop-watched-by-app-heur-mop-watched-by-at*:
‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
twl-st-heur $\times_f$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
unfolding *mop-watched-by-app-heur-def mop-watched-by-at-def uncurry-def all-lits-def*[*symmetric*] *all-lits-alt-def*[*symm*
by (*intro frefI nres-relI*, *refine-rcg*,
auto simp: *twl-st-heur-def* $\mathcal{L}_{all}$-*all-atms-all-lits map-fun-rel-def*
simp flip: *all-lits-alt-def2*)
(*auto simp*: *add.assoc*)

lemma *mop-watched-by-app-heur-mop-watched-by-at′′*:
‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
twl-st-heur-up′′ $\mathcal{D}$ *r s K* $\times_f$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
by (*rule fref-mono*[*THEN set-mp*, *OF* - - - *mop-watched-by-app-heur-mop-watched-by-at*])
(*auto simp*: $\mathcal{L}_{all}$-*all-atms-all-lits twl-st-heur′-def map-fun-rel-def*)

definition *mop-polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-pol* = (λ*M L*. *do* {
ASSERT(*polarity-pol-pre M L*);
RETURN (*polarity-pol M L*)
})›

definition *polarity-st-pre* :: ‹*nat twl-st-wl* × *nat literal* ⇒ *bool*› **where**
‹*polarity-st-pre* ≡ λ(*S*, *L*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›

definition *mop-polarity-st-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-st-heur S L* = *do* {
mop-polarity-pol (*get-trail-wl-heur S*) *L*
}›

lemma *mop-polarity-st-heur-alt-def*: ‹*mop-polarity-st-heur* = (λ(*M*, -) *L*. *do* {
mop-polarity-pol M L
})›
by (*auto simp*: *mop-polarity-st-heur-def intro*!: *ext*)

lemma *mop-polarity-st-heur-mop-polarity-wl*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur* $\times_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
unfolding *mop-polarity-wl-def mop-polarity-st-heur-def uncurry-def mop-polarity-pol-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg polarity-pol-polarity*[*of* ‹*all-atms* - -›, *THEN fref-to-Down-unRET-uncurry*])
apply (*auto simp*: *twl-st-heur-def* $\mathcal{L}_{all}$-*all-atms-all-lits ac-simps*
intro!: *polarity-pol-pre simp flip*: *all-atms-def*)
done

lemma *mop-polarity-st-heur-mop-polarity-wl′′*:

⟨(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
$[\lambda\text{-}.\ \mathit{True}]_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\times_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*⟩
by (*rule fref-mono*[*THEN set-mp*, *OF - - - mop-polarity-st-heur-mop-polarity-wl*])
(*auto simp*: $\mathcal{L}_{all}$*-all-atms-all-lits twl-st-heur'-def map-fun-rel-def*)

lemma [*simp,iff*]: ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ⟷ *blits-in-*$\mathcal{L}_{in}$ *S*⟩
unfolding *literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* $\mathcal{L}_{all}$*-all-atms-all-lits*
by *auto*

definition *length-avdom* :: ⟨*twl-st-wl-heur* ⇒ *nat*⟩ **where**
⟨*length-avdom S* = *length* (*get-avdom S*)⟩

lemma *length-avdom-alt-def*:
⟨*length-avdom* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*). *length avdom*)⟩
by (*intro ext*) (*auto simp*: *length-avdom-def*)

definition *clause-is-learned-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool*
where
⟨*clause-is-learned-heur S C* ⟷ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED*⟩

lemma *clause-is-learned-heur-alt-def*:
⟨*clause-is-learned-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *lcount*) *C* . *arena-status N' C* = *LEARNED*)⟩
by (*intro ext*) (*auto simp*: *clause-is-learned-heur-def*)

definition *get-the-propagation-reason-heur*
:: ⟨*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat option nres*⟩
where
⟨*get-the-propagation-reason-heur S* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*)⟩

lemma *get-the-propagation-reason-heur-alt-def*:
⟨*get-the-propagation-reason-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *lcount*) *L* . *get-the-propagation-reason-pol M' L*)⟩
by (*intro ext*) (*auto simp*: *get-the-propagation-reason-heur-def*)

definition *clause-lbd-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *nat*
where
⟨*clause-lbd-heur S C* = *arena-lbd* (*get-clauses-wl-heur S*) *C*⟩

definition (**in** −) *access-length-heur* **where**
⟨*access-length-heur S i* = *arena-length* (*get-clauses-wl-heur S*) *i*⟩

lemma *access-length-heur-alt-def*:
⟨*access-length-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
lcount) *C*. *arena-length N' C*)⟩
by (*intro ext*) (*auto simp*: *access-length-heur-def arena-lbd-def*)

definition *marked-as-used-st* **where**

‹*marked-as-used-st T C* =
marked-as-used (*get-clauses-wl-heur T*) *C*›

lemma *marked-as-used-st-alt-def*:
‹*marked-as-used-st* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *lcount*) *C*.
marked-as-used N′ C)›
by (*intro ext*) (*auto simp*: *marked-as-used-st-def*)

definition *access-vdom-at* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat*› **where**
‹*access-vdom-at S i* = *get-avdom S* ! *i*›

lemma *access-vdom-at-alt-def*:
‹*access-vdom-at* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*) *i*. *avdom* ! *i*)›
by (*intro ext*) (*auto simp*: *access-vdom-at-def*)

definition *access-vdom-at-pre* **where**
‹*access-vdom-at-pre S i* ⟷ *i* < *length* (*get-avdom S*)›

definition *mark-garbage-heur* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*mark-garbage-heur C i* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*).
(*M′*, *extra-information-mark-to-delete N′ C*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *delete-index-and-swap avdom i*, *lcount* − *1*, *opts*, *old-arena*))›

definition *mark-garbage-heur2* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mark-garbage-heur2 C* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*). *do*{
let st = *arena-status N′ C* = *IRRED*;
ASSERT(¬*st* ⟶ *lcount* ≥ *1*);
RETURN (*M′*, *extra-information-mark-to-delete N′ C*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *if st then lcount else lcount* − *1*, *opts*) })›

definition *delete-index-vdom-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*›**where**
‹*delete-index-vdom-heur* = (λ*i* (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*).
(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *delete-index-and-swap avdom i*, *lcount*))›

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-unused arena C*) *C*›
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
apply *clarify*
apply (*rule-tac x*=*N* **in** *exI*)
apply (*rule-tac x*=*vdom* **in** *exI*)
by (*auto simp*: *arena-act-pre-def*
simp: *valid-arena-mark-unused*)

definition *mop-mark-garbage-heur* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mop-mark-garbage-heur C i* = (λ*S*. *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *get-learned-count S* ≥ *1* ∧ *i* < *length*

(*get-avdom S*));
RETURN (*mark-garbage-heur C i S*)
})›

definition *mark-unused-st-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*mark-unused-st-heur C* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*).
(*M′*, *arena-decr-act* (*mark-unused N′ C*) *C*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*,
lbd, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*))›

definition *mop-mark-unused-st-heur* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*mop-mark-unused-st-heur C T* = *do* {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*mark-unused-st-heur C T*)
}›

lemma *mop-mark-garbage-heur-alt-def*:
‹*mop-mark-garbage-heur C i* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur* (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*), *C*) ∧ *lcount* ≥ *1* ∧ *i* < *length avdom*);
RETURN (*M′*, *extra-information-mark-to-delete N′ C*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*,
vdom, *delete-index-and-swap avdom i*, *lcount* − *1*, *opts*, *old-arena*)
})›
unfolding *mop-mark-garbage-heur-def mark-garbage-heur-def*
by (*auto intro*!: *ext*)

lemma *mark-unused-st-heur-simp*[*simp*]:
‹*get-avdom* (*mark-unused-st-heur C T*) = *get-avdom T*›
‹*get-vdom* (*mark-unused-st-heur C T*) = *get-vdom T*›
by (*cases T*; *auto simp*: *mark-unused-st-heur-def*; *fail*)+

lemma *get-slow-ema-heur-alt-def*:
‹*RETURN o get-slow-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, -), *lcount*). *RETURN sema*)›
by *auto*

lemma *get-fast-ema-heur-alt-def*:
‹*RETURN o get-fast-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, *ccount*), *lcount*). *RETURN fema*)›
by *auto*

fun *get-conflict-count-since-last-restart-heur* :: ‹*twl-st-wl-heur* ⇒ *64 word*› **where**
‹*get-conflict-count-since-last-restart-heur* (-, -, -, -, -, -, -, -, -, -, -,
(-, -, (*ccount*, -), -), -)
= *ccount*›

lemma (**in** −) *get-counflict-count-heur-alt-def*:
‹*RETURN o get-conflict-count-since-last-restart-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*,
outl, *stats*, (-, -, (*ccount*, -), -), *lcount*). *RETURN ccount*)›

by *auto*

lemma *get-learned-count-alt-def*:
⟨*RETURN o get-learned-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, -, *vdom*, *avdom*, *lcount*, *opts*). *RETURN lcount*)⟩
by *auto*

I also played with *ema-reinit fast-ema* and *ema-reinit slow-ema*. Currently removed, to test the performance, I remove it.

definition *incr-restart-stat* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*incr-restart-stat* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*), *vdom*, *avdom*, *lcount*). *do*{
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *incr-restart stats*,
(*fast-ema*, *slow-ema*,
restart-info-restart-done res-info, *wasted*), *vdom*, *avdom*, *lcount*)
})⟩

definition *incr-lrestart-stat* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*incr-lrestart-stat* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*), *vdom*, *avdom*, *lcount*). *do*{
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *incr-lrestart stats*,
(*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*),
vdom, *avdom*, *lcount*)
})⟩

definition *incr-wasted-st* :: ⟨*64 word* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*⟩ **where**
⟨*incr-wasted-st* = (λ*waste* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*, φ), *vdom*, *avdom*, *lcount*). *do*{
(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
(*fast-ema*, *slow-ema*, *res-info*, *wasted*+*waste*, φ),
vdom, *avdom*, *lcount*)
})⟩

definition *wasted-bytes-st* :: ⟨*twl-st-wl-heur* ⇒ *64 word*⟩ **where**
⟨*wasted-bytes-st* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
res-info, *wasted*, φ), *vdom*, *avdom*, *lcount*).
wasted)⟩

definition *opts-restart-st* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*opts-restart-st* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, -). (*opts-restart opts*))⟩

definition *opts-reduction-st* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*opts-reduction-st* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*, -). (*opts-reduce opts*))⟩

definition *isasat-length-trail-st* :: ⟨*twl-st-wl-heur* ⇒ *nat*⟩ **where**
⟨*isasat-length-trail-st S* = *isa-length-trail* (*get-trail-wl-heur S*)⟩

lemma *isasat-length-trail-st-alt-def*:
⟨*isasat-length-trail-st* = (λ(*M*, -). *isa-length-trail M*)⟩
by (*auto simp*: *isasat-length-trail-st-def intro*!: *ext*)

definition *get-pos-of-level-in-trail-imp-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat nres*› **where**
‹*get-pos-of-level-in-trail-imp-st S* = *get-pos-of-level-in-trail-imp* (*get-trail-wl-heur S*)›

lemma *get-pos-of-level-in-trail-imp-alt-def*:
‹*get-pos-of-level-in-trail-imp-st* = (λ(*M*, -) *L*. *do* {*k* ← *get-pos-of-level-in-trail-imp M L*; *RETURN k*})›
by (*auto simp*: *get-pos-of-level-in-trail-imp-st-def intro*!: *ext*)

definition *mop-clause-not-marked-to-delete-heur* :: ‹- ⇒ *nat* ⇒ *bool nres*›
where
‹*mop-clause-not-marked-to-delete-heur S C* = *do* {
ASSERT(*clause-not-marked-to-delete-heur-pre* (*S*, *C*));
RETURN (*clause-not-marked-to-delete-heur S C*)
}›

definition *mop-arena-lbd-st* **where**
‹*mop-arena-lbd-st S* =
mop-arena-lbd (*get-clauses-wl-heur S*)›

lemma *mop-arena-lbd-st-alt-def*:
‹*mop-arena-lbd-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*get-clause-LBD-pre arena C*);
RETURN(*arena-lbd arena C*)
})›
unfolding *mop-arena-lbd-st-def mop-arena-lbd-def*
by (*auto intro*!: *ext*)

definition *mop-arena-status-st* **where**
‹*mop-arena-status-st S* =
mop-arena-status (*get-clauses-wl-heur S*)›

lemma *mop-arena-status-st-alt-def*:
‹*mop-arena-status-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*arena-is-valid-clause-vdom arena C*);
RETURN(*arena-status arena C*)
})›
unfolding *mop-arena-status-st-def mop-arena-status-def*
by (*auto intro*!: *ext*)

definition *mop-marked-as-used-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *bool nres*› **where**
‹*mop-marked-as-used-st S* =
mop-marked-as-used (*get-clauses-wl-heur S*)›

lemma *mop-marked-as-used-st-alt-def*:
‹*mop-marked-as-used-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*marked-as-used-pre arena C*);
RETURN(*marked-as-used arena C*)
})›
unfolding *mop-marked-as-used-st-def mop-marked-as-used-def*
by (*auto intro*!: *ext*)

definition *mop-arena-length-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length-st S* =
mop-arena-length (*get-clauses-wl-heur S*)›

lemma *mop-arena-length-st-alt-def*:
‹*mop-arena-length-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*) *C*. *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
RETURN (*arena-length arena C*)
})›
unfolding *mop-arena-length-st-def mop-arena-length-def*
by (*auto intro*!: *ext*)

definition *full-arena-length-st* :: ‹*twl-st-wl-heur* ⇒ *nat*› **where**
‹*full-arena-length-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *length arena*)›

definition (**in** −) *access-lit-in-clauses* **where**
‹*access-lit-in-clauses S i j* = (*get-clauses-wl S*) ∝ *i* ! *j*›

lemma *twl-st-heur-get-clauses-access-lit*[*simp*]:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *C* ∈# *dom-m* (*get-clauses-wl T*) ⟹
i < *length* (*get-clauses-wl T* ∝ *C*) ⟹
get-clauses-wl T ∝ *C* ! *i* = *access-lit-in-clauses-heur S C i*›
for *S T C i*
by (*cases S*; *cases T*)
(*auto simp*: *arena-lifting twl-st-heur-def access-lit-in-clauses-heur-def*)

In an attempt to avoid using $?a + ?b + ?c = ?a + (?b + ?c)$

$?a + ?b = ?b + ?a$

$?b + (?a + ?c) = ?a + (?b + ?c)$

$?a * ?b * ?c = ?a * (?b * ?c)$

$?a * ?b = ?b * ?a$

$?b * (?a * ?c) = ?a * (?b * ?c)$

$inf\ (inf\ ?a\ ?b)\ ?c = inf\ ?a\ (inf\ ?b\ ?c)$

$inf\ ?a\ ?b = inf\ ?b\ ?a$

$inf\ ?b\ (inf\ ?a\ ?c) = inf\ ?a\ (inf\ ?b\ ?c)$

$sup\ (sup\ ?a\ ?b)\ ?c = sup\ ?a\ (sup\ ?b\ ?c)$

$sup\ ?a\ ?b = sup\ ?b\ ?a$

$sup\ ?b\ (sup\ ?a\ ?c) = sup\ ?a\ (sup\ ?b\ ?c)$

$min\ (min\ ?a\ ?b)\ ?c = min\ ?a\ (min\ ?b\ ?c)$

$min\ ?a\ ?b = min\ ?b\ ?a$

$min\ ?b\ (min\ ?a\ ?c) = min\ ?a\ (min\ ?b\ ?c)$

$max\ (max\ ?a\ ?b)\ ?c = max\ ?a\ (max\ ?b\ ?c)$

$max\ ?a\ ?b = max\ ?b\ ?a$

$max\ ?b\ (max\ ?a\ ?c) = max\ ?a\ (max\ ?b\ ?c)$

$coprime\ ?b\ ?a = coprime\ ?a\ ?b$

$(?a\ dvd\ ?c - ?b) = (?a\ dvd\ ?b - ?c)$

$(?a\ @\ ?b)\ @\ ?c = ?a\ @\ ?b\ @\ ?c$

$gcd\ (gcd\ ?a\ ?b)\ ?c = gcd\ ?a\ (gcd\ ?b\ ?c)$
$gcd\ ?a\ ?b = gcd\ ?b\ ?a$
$gcd\ ?b\ (gcd\ ?a\ ?c) = gcd\ ?a\ (gcd\ ?b\ ?c)$
$lcm\ (lcm\ ?a\ ?b)\ ?c = lcm\ ?a\ (lcm\ ?b\ ?c)$
$lcm\ ?a\ ?b = lcm\ ?b\ ?a$
$lcm\ ?b\ (lcm\ ?a\ ?c) = lcm\ ?a\ (lcm\ ?b\ ?c)$
$?a \cap\# \ ?b \cap\# \ ?c = ?a \cap\# \ (?b \cap\# \ ?c)$
$?a \cap\# \ ?b = ?b \cap\# \ ?a$
$?b \cap\# \ (?a \cap\# \ ?c) = ?a \cap\# \ (?b \cap\# \ ?c)$
$?a \cup\# \ ?b \cup\# \ ?c = ?a \cup\# \ (?b \cup\# \ ?c)$
$?a \cup\# \ ?b = ?b \cup\# \ ?a$
$?b \cup\# \ (?a \cup\# \ ?c) = ?a \cup\# \ (?b \cup\# \ ?c)$
$signed.min\ (signed.min\ ?a\ ?b)\ ?c = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.min\ ?a\ ?b = signed.min\ ?b\ ?a$
$signed.min\ ?b\ (signed.min\ ?a\ ?c) = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.max\ (signed.max\ ?a\ ?b)\ ?c = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$signed.max\ ?a\ ?b = signed.max\ ?b\ ?a$
$signed.max\ ?b\ (signed.max\ ?a\ ?c) = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$(?a\ AND\ ?b)\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$?a\ AND\ ?b = ?b\ AND\ ?a$
$?b\ AND\ ?a\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$(?a\ OR\ ?b)\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$?a\ OR\ ?b = ?b\ OR\ ?a$
$?b\ OR\ ?a\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$(?a\ XOR\ ?b)\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$
$?a\ XOR\ ?b = ?b\ XOR\ ?a$
$?b\ XOR\ ?a\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$ everywhere.

lemma *all-lits-simps*[*simp*]:
‹*all-lits N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-lits N* (*NE* + *UE* + *NS* + *US*)›
‹*all-atms N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-atms N* (*NE* + *UE* + *NS* + *US*)›
by (*auto simp*: *ac-simps*)

lemma *clause-not-marked-to-delete-heur-alt-def*:
‹*RETURN* ∘∘ *clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*.
RETURN (*arena-status arena C* ≠ *DELETED*))›
unfolding *clause-not-marked-to-delete-heur-def* **by** (*auto intro*!: *ext*)

end
theory *IsaSAT-Trail-LLVM*
imports *IsaSAT-Literals-LLVM IsaSAT-Trail*
begin

type-synonym *tri-bool-assn* = *8 word*

definition *tri-bool-rel-aux* ≡ { (*0*::*nat*,*None*), (*2*,*Some True*), (*3*,*Some False*) }
definition *tri-bool-rel* ≡ *unat-rel′ TYPE*(*8*) *O tri-bool-rel-aux*

abbreviation *tri-bool-assn* ≡ *pure tri-bool-rel*
lemmas [*fcomp-norm-unfold*] = *tri-bool-rel-def*[*symmetric*]

lemma *tri-bool-UNSET-refine-aux*: (*0*,*UNSET*)∈*tri-bool-rel-aux*
  and *tri-bool-SET-TRUE-refine-aux*: (*2*,*SET-TRUE*)∈*tri-bool-rel-aux*
  and *tri-bool-SET-FALSE-refine-aux*: (*3*,*SET-FALSE*)∈*tri-bool-rel-aux*
  and *tri-bool-eq-refine-aux*: ((=),*tri-bool-eq*) ∈ *tri-bool-rel-aux*→*tri-bool-rel-aux*→*bool-rel*
  by (*auto simp*: *tri-bool-rel-aux-def tri-bool-eq-def*)

sepref-def *tri-bool-UNSET-impl* **is** [] *uncurry0* (*RETURN 0*) :: $unit\text{-}assn^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)
  apply (*annot-unat-const TYPE*(*8*))
  by *sepref*

sepref-def *tri-bool-SET-TRUE-impl* **is** [] *uncurry0* (*RETURN 2*) :: $unit\text{-}assn^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)
  apply (*annot-unat-const TYPE*(*8*))
  by *sepref*

sepref-def *tri-bool-SET-FALSE-impl* **is** [] *uncurry0* (*RETURN 3*) :: $unit\text{-}assn^k \rightarrow_a$ *unat-assn′ TYPE*(*8*)
  apply (*annot-unat-const TYPE*(*8*))
  by *sepref*

sepref-def *tri-bool-eq-impl* [*llvm-inline*] **is** [] *uncurry* (*RETURN oo* (=)) :: $(unat\text{-}assn'\ TYPE(8))^k *_a$ $(unat\text{-}assn'\ TYPE(8))^k \rightarrow_a$ *bool1-assn*
  by *sepref*

lemmas [*sepref-fr-rules*] =
  tri-bool-UNSET-impl.refine[*FCOMP tri-bool-UNSET-refine-aux*]
  tri-bool-SET-TRUE-impl.refine[*FCOMP tri-bool-SET-TRUE-refine-aux*]
  tri-bool-SET-FALSE-impl.refine[*FCOMP tri-bool-SET-FALSE-refine-aux*]
  tri-bool-eq-impl.refine[*FCOMP tri-bool-eq-refine-aux*]

type-synonym *trail-pol-fast-assn* =
  ‹*32 word array-list64* × *tri-bool-assn larray64* × *32 word larray64* ×
    64 word larray64 × *32 word* ×
    32 word array-list64›

sepref-def *DECISION-REASON-impl* **is** *uncurry0* (*RETURN DECISION-REASON*)
  :: $unit\text{-}assn^k \rightarrow_a$ *sint64-nat-assn*
  unfolding *DECISION-REASON-def* **apply** (*annot-snat-const TYPE*(*64*)) **by** *sepref*

definition *trail-pol-fast-assn* :: ‹*trail-pol* ⇒ *trail-pol-fast-assn* ⇒ *assn*› **where**
  ‹*trail-pol-fast-assn* ≡
    arl64-assn unat-lit-assn $\times_a$ *larray64-assn* (*tri-bool-assn*) $\times_a$
    larray64-assn uint32-nat-assn $\times_a$
    larray64-assn sint64-nat-assn $\times_a$ *uint32-nat-assn* $\times_a$
    arl64-assn uint32-nat-assn›

Code generation

Conversion between incomplete and complete mode **sepref-def** *count-decided-pol-impl* **is** *RETURN o count-decided-pol* :: $trail\text{-}pol\text{-}fast\text{-}assn^k \rightarrow_a$ *uint32-nat-assn*
  unfolding *trail-pol-fast-assn-def count-decided-pol-def*
  by *sepref*

sepref-def *get-level-atm-fast-code*
is ‹*uncurry* (*RETURN oo get-level-atm-pol*)›
:: ‹[*get-level-atm-pol-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *atom-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
unfolding *get-level-atm-pol-def nat-shiftr-div2*[*symmetric*]
get-level-atm-pol-pre-def trail-pol-fast-assn-def
supply [[*eta-contract* = *false*, *show-abbrevs*=*false*]]
apply (*rewrite at nth - eta-expand*)
apply (*rewrite at nth - - annot-index-of-atm*)
by *sepref*

sepref-def *get-level-fast-code*
is ‹*uncurry* (*RETURN oo get-level-pol*)›
:: ‹[*get-level-pol-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *uint32-nat-assn*›
unfolding *get-level-get-level-atm nat-shiftr-div2*[*symmetric*]
get-level-pol-pre-def get-level-pol-def
supply [[*goals-limit* = *1*]] *image-image*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
get-level-atm-pol-pre-def[*simp*]
by *sepref*

sepref-def *polarity-pol-fast-code*
is ‹*uncurry* (*RETURN oo polarity-pol*)›
:: ‹[*uncurry polarity-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *tri-bool-assn*›
unfolding *polarity-pol-def option.case-eq-if polarity-pol-pre-def*
trail-pol-fast-assn-def
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *isa-length-trail-fast-code*
is ‹*RETURN o isa-length-trail*›
:: ‹[λ-. *True*]$_a$ *trail-pol-fast-assn*^{k} $\rightarrow$ *snat-assn' TYPE*(*64*)›
unfolding *isa-length-trail-def isa-length-trail-pre-def length-uint32-nat-def*
trail-pol-fast-assn-def
by *sepref*

sepref-def *cons-trail-Propagated-tr-fast-code*
is ‹*uncurry2* (*cons-trail-Propagated-tr*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *trail-pol-fast-assn*^{d} $\rightarrow_a$ *trail-pol-fast-assn*›
unfolding *cons-trail-Propagated-tr-def cons-trail-Propagated-tr-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Propagated-tr-pre-def*
unfolding *trail-pol-fast-assn-def prod.case*
apply (*subst* (*3*)*annot-index-of-atm*)
apply (*subst* (*4*)*annot-index-of-atm*)

supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *tl-trail-tr-fast-code*
is ⟨*RETURN o tl-trailt-tr*⟩
:: ⟨[*tl-trailt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-def UNSET-def*[*symmetric*] *tl-trailt-tr-pre-def*
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const TYPE*(*32*))
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *tl-trail-proped-tr-fast-code*
is ⟨*RETURN o tl-trail-propedt-tr*⟩
:: ⟨[*tl-trail-propedt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*⟩
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trail-propedt-tr-def UNSET-def*[*symmetric*]
tl-trail-propedt-tr-pre-def
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const TYPE*(*32*))
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *lit-of-last-trail-fast-code*
is ⟨*RETURN o lit-of-last-trail-pol*⟩
:: ⟨[λ(*M*, -). *M* ≠ []]$_a$ *trail-pol-fast-assn*^{k} → *unat-lit-assn*⟩
unfolding *lit-of-last-trail-pol-def trail-pol-fast-assn-def*
by *sepref*

sepref-def *cons-trail-Decided-tr-fast-code*
is ⟨*uncurry* (*RETURN oo cons-trail-Decided-tr*)⟩
:: ⟨[*cons-trail-Decided-tr-pre*]$_a$
unat-lit-assn^k $*_a$ *trail-pol-fast-assn*^{d} → *trail-pol-fast-assn*⟩
unfolding *cons-trail-Decided-tr-def cons-trail-Decided-tr-def trail-pol-fast-assn-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Decided-tr-pre-def*

apply (*annot-unat-const TYPE*(*32*))
apply (*rewrite at* -@[⌑] **in** (-,⌑) *annot-snat-unat-downcast*[**where** ′*l*=*32*])
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *defined-atm-fast-code*
is ⟨*uncurry* (*RETURN oo defined-atm-pol*)⟩
:: ⟨[*uncurry defined-atm-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *atom-assn*^{k} → *bool1-assn*⟩
unfolding *defined-atm-pol-def UNSET-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
defined-atm-pol-pre-def trail-pol-fast-assn-def Pos-rel-def[*symmetric*]
unfolding *ins-idx-upcast64*

supply *Pos-impl.refine*[*sepref-fr-rules*]
supply *UNSET-def*[*simp del*]
by *sepref*

sepref-register *get-propagation-reason-raw-pol*
sepref-def *get-propagation-reason-fast-code*
is ‹*uncurry get-propagation-reason-raw-pol*›
:: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
unfolding *get-propagation-reason-raw-pol-def trail-pol-fast-assn-def*

by *sepref*

sepref-register *isa-trail-nth*

sepref-def *isa-trail-nth-fast-code*
is ‹*uncurry isa-trail-nth*›
:: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
unfolding *isa-trail-nth-def trail-pol-fast-assn-def*
by *sepref*

sepref-def *tl-trail-tr-no-CS-fast-code*
is ‹*RETURN o tl-trailt-tr-no-CS*›
:: ‹$[tl\text{-}trailt\text{-}tr\text{-}no\text{-}CS\text{-}pre]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^d \rightarrow trail\text{-}pol\text{-}fast\text{-}assn$›
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-no-CS-def UNSET-def*[*symmetric*] *tl-trailt-tr-no-CS-pre-def*
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const TYPE*(*32*))
supply [[*goals-limit = 1*]]
by *sepref*

sepref-def *trail-conv-back-imp-fast-code*
is ‹*uncurry trail-conv-back-imp*›
:: ‹$uint32\text{-}nat\text{-}assn^k *_a trail\text{-}pol\text{-}fast\text{-}assn^d \rightarrow_a trail\text{-}pol\text{-}fast\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *trail-conv-back-imp-def trail-pol-fast-assn-def*
apply (*rewrite at take* ⌑ *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

sepref-def *get-pos-of-level-in-trail-imp-fast-code*
is ‹*uncurry get-pos-of-level-in-trail-imp*›
:: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
unfolding *get-pos-of-level-in-trail-imp-def trail-pol-fast-assn-def*
apply (*rewrite at* ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

sepref-def *get-the-propagation-reason-fast-code*
is ‹*uncurry get-the-propagation-reason-pol*›
:: ‹$trail\text{-}pol\text{-}fast\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a snat\text{-}option\text{-}assn'\ TYPE(64)$›
unfolding *get-the-propagation-reason-pol-def trail-pol-fast-assn-def*
tri-bool-eq-def[*symmetric*]

by *sepref*

experiment begin

export-llvm
tri-bool-UNSET-impl
tri-bool-SET-TRUE-impl
tri-bool-SET-FALSE-impl
DECISION-REASON-impl
count-decided-pol-impl
get-level-atm-fast-code
get-level-fast-code
polarity-pol-fast-code
isa-length-trail-fast-code
cons-trail-Propagated-tr-fast-code
tl-trail-tr-fast-code
tl-trail-proped-tr-fast-code
lit-of-last-trail-fast-code
cons-trail-Decided-tr-fast-code
defined-atm-fast-code
get-propagation-reason-fast-code
isa-trail-nth-fast-code
tl-trail-tr-no-CS-fast-code
trail-conv-back-imp-fast-code
get-pos-of-level-in-trail-imp-fast-code
get-the-propagation-reason-fast-code

end

end
theory *IsaSAT-Lookup-Conflict-LLVM*
imports
IsaSAT-Lookup-Conflict
IsaSAT-Trail-LLVM
IsaSAT-Clauses-LLVM
LBD-LLVM
begin

sepref-decl-op *nat-lit-eq*: ‹$(=)$ :: *nat literal* ⇒ - ⇒ -› ::
‹$(Id$ :: (*nat literal* × -) *set*) → (Id :: (*nat literal* × -) *set*) → *bool-rel*› **.**

sepref-def *nat-lit-eq-impl*
is [] ‹*uncurry* (*RETURN oo* ($\lambda x\ y.\ x = y$))›
:: ‹$uint32\text{-}nat\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
by *sepref*

lemma *nat-lit-rel*: ‹($(=)$, *op-nat-lit-eq*) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
by (*auto simp*: *nat-lit-rel-def br-def split*: *if-splits*; *presburger*)

sepref-register $(=)$ :: *nat literal* ⇒ - ⇒ -
declare *nat-lit-eq-impl.refine*[*FCOMP nat-lit-rel*, *sepref-fr-rules*]

sepref-register *set-lookup-conflict-aa*
type-synonym *lookup-clause-assn* = ‹*32 word* × (*1 word*) *ptr*›

type-synonym (**in** −) *option-lookup-clause-assn* = ⟨*1 word* × *lookup-clause-assn*⟩

type-synonym (**in** −) *out-learned-assn* = ⟨*32 word array-list64*⟩

abbreviation (**in** −) *out-learned-assn* :: ⟨*out-learned* ⇒ *out-learned-assn* ⇒ *assn*⟩ **where**
⟨*out-learned-assn* ≡ *arl64-assn unat-lit-assn*⟩

definition *minimize-status-int-rel* :: ⟨(*nat* × *minimize-status*) *set*⟩ **where**
⟨*minimize-status-int-rel* = {(*0*, *SEEN-UNKNOWN*), (*1*, *SEEN-FAILED*), (*2*, *SEEN-REMOVABLE*)}⟩

abbreviation *minimize-status-ref-rel* **where**
⟨*minimize-status-ref-rel* ≡ *snat-rel′ TYPE*(*8*)⟩

abbreviation *minimize-status-ref-assn* **where**
⟨*minimize-status-ref-assn* ≡ *pure minimize-status-ref-rel*⟩

definition *minimize-status-rel* :: ⟨-⟩ **where**
⟨*minimize-status-rel* = *minimize-status-ref-rel O minimize-status-int-rel*⟩

abbreviation *minimize-status-assn* :: ⟨-⟩ **where**
⟨*minimize-status-assn* ≡ *pure minimize-status-rel*⟩

lemma *minimize-status-assn-alt-def*:
⟨*minimize-status-assn* = *pure* (*snat-rel O minimize-status-int-rel*)⟩
unfolding *minimize-status-rel-def* **..**

lemmas [*fcomp-norm-unfold*] = *minimize-status-assn-alt-def*[*symmetric*]

definition *minimize-status-rel-eq* :: ⟨*minimize-status* ⇒ *minimize-status* ⇒ *bool*⟩ **where**
[*simp*]: ⟨*minimize-status-rel-eq* = (=)⟩

lemma *minimize-status-rel-eq*:
⟨((=), *minimize-status-rel-eq*) ∈ *minimize-status-int-rel* → *minimize-status-int-rel* → *bool-rel*⟩
by (*auto simp*: *minimize-status-int-rel-def*)

sepref-def *minimize-status-rel-eq-impl*
is [] ⟨*uncurry* (*RETURN oo* (=))⟩
:: ⟨$minimize\text{-}status\text{-}ref\text{-}assn^k *_a minimize\text{-}status\text{-}ref\text{-}assn^k \rightarrow_a bool1\text{-}assn$⟩
supply [[*goals-limit=1*]]
by *sepref*

sepref-register *minimize-status-rel-eq*

lemmas [*sepref-fr-rules*] = *minimize-status-rel-eq-impl.refine*[*unfolded convert-fref*, *FCOMP minimize-status-rel-eq*]

lemma
SEEN-FAILED-rel: ⟨(*1*, *SEEN-FAILED*) ∈ *minimize-status-int-rel*⟩ **and**
SEEN-UNKNOWN-rel: ⟨(*0*, *SEEN-UNKNOWN*) ∈ *minimize-status-int-rel*⟩ **and**
SEEN-REMOVABLE-rel: ⟨(*2*, *SEEN-REMOVABLE*) ∈ *minimize-status-int-rel*⟩
by (*auto simp*: *minimize-status-int-rel-def*)

sepref-def *SEEN-FAILED-impl*
is [] ⟨*uncurry0* (*RETURN 1*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a minimize\text{-}status\text{-}ref\text{-}assn$⟩

supply [[*goals-limit=1*]]
apply (*annot-snat-const* $TYPE(8)$)
by *sepref*

sepref-def *SEEN-UNKNOWN-impl*
is [] ‹*uncurry0* (*RETURN 0*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *minimize-status-ref-assn*›
supply [[*goals-limit=1*]]
apply (*annot-snat-const* $TYPE(8)$)
by *sepref*

sepref-def *SEEN-REMOVABLE-impl*
is [] ‹*uncurry0* (*RETURN 2*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *minimize-status-ref-assn*›
supply [[*goals-limit=1*]]
apply (*annot-snat-const* $TYPE(8)$)
by *sepref*

lemmas [*sepref-fr-rules*] = *SEEN-FAILED-impl.refine*[*FCOMP SEEN-FAILED-rel*]
SEEN-UNKNOWN-impl.refine[*FCOMP SEEN-UNKNOWN-rel*]
SEEN-REMOVABLE-impl.refine[*FCOMP SEEN-REMOVABLE-rel*]

definition *option-bool-impl-rel* **where**
‹*option-bool-impl-rel* = *bool1-rel O option-bool-rel*›

abbreviation *option-bool-impl-assn* :: ‹-› **where**
‹*option-bool-impl-assn* ≡ *pure* (*option-bool-impl-rel*)›

lemma *option-bool-impl-assn-alt-def*:
‹*option-bool-impl-assn* = *hr-comp bool1-assn option-bool-rel*›
unfolding *option-bool-impl-rel-def* **by** (*simp add*: *hr-comp-pure*)

lemmas [*fcomp-norm-unfold*] = *option-bool-impl-assn-alt-def*[*symmetric*]
option-bool-impl-rel-def[*symmetric*]

lemma *Some-rel*: ‹(λ-. *True*, *ISIN*) ∈ *bool-rel* → *option-bool-rel*›
by (*auto simp*: *option-bool-rel-def*)

sepref-def *Some-impl*
is [] ‹*RETURN o* (λ-. *True*)›
:: ‹*bool1-assn*^{k} $\rightarrow_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *Some-impl.refine*[*FCOMP Some-rel*]

lemma *is-Notin-rel*: ‹($\lambda x.\ \neg x$, *is-NOTIN*) ∈ *option-bool-rel* → *bool-rel*›
by (*auto simp*: *option-bool-rel-def*)

sepref-def *is-Notin-impl*
is [] ‹*RETURN o* ($\lambda x.\ \neg x$)›
:: ‹*bool1-assn*^{k} $\rightarrow_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *is-Notin-impl.refine*[*FCOMP is-Notin-rel*]

lemma *NOTIN-rel*: ⟨(*False*, *NOTIN*) ∈ *option-bool-rel*⟩
by (*auto simp*: *option-bool-rel-def*)

sepref-def *NOTIN-impl*
is [] ⟨*uncurry0* (*RETURN False*)⟩
:: ⟨*unit-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
by *sepref*

lemmas [*sepref-fr-rules*] = *NOTIN-impl.refine*[*FCOMP NOTIN-rel*]

definition (**in** −) *lookup-clause-rel-assn*
:: ⟨*lookup-clause-rel* ⇒ *lookup-clause-assn* ⇒ *assn*⟩
where
⟨*lookup-clause-rel-assn* ≡ (*uint32-nat-assn* $\times_a$ *array-assn option-bool-impl-assn*)⟩

definition (**in** −)*conflict-option-rel-assn*
:: ⟨*conflict-option-rel* ⇒ *option-lookup-clause-assn* ⇒ *assn*⟩
where
⟨*conflict-option-rel-assn* ≡ (*bool1-assn* $\times_a$ *lookup-clause-rel-assn*)⟩

lemmas [*fcomp-norm-unfold*] = *conflict-option-rel-assn-def*[*symmetric*]
lookup-clause-rel-assn-def[*symmetric*]

definition (**in** −)*ana-refinement-fast-rel* **where**
⟨*ana-refinement-fast-rel* ≡ *snat-rel′ TYPE*(*64*) $\times_r$ *unat-rel′ TYPE*(*32*) $\times_r$ *bool1-rel*⟩

abbreviation (**in** −)*ana-refinement-fast-assn* **where**
⟨*ana-refinement-fast-assn* ≡ *sint64-nat-assn* $\times_a$ *uint32-nat-assn* $\times_a$ *bool1-assn*⟩

lemma *ana-refinement-fast-assn-def*:
⟨*ana-refinement-fast-assn* = *pure ana-refinement-fast-rel*⟩
by (*auto simp*: *ana-refinement-fast-rel-def*)

abbreviation (**in** −)*analyse-refinement-fast-assn* **where**
⟨*analyse-refinement-fast-assn* ≡
arl64-assn ana-refinement-fast-assn⟩

lemma *lookup-clause-assn-is-None-alt-def*:
⟨*RETURN o lookup-clause-assn-is-None* = (λ(*b*, -, -). *RETURN b*)⟩
unfolding *lookup-clause-assn-is-None-def* **by** *auto*

sepref-def *lookup-clause-assn-is-None-impl*
is ⟨*RETURN o lookup-clause-assn-is-None*⟩
:: ⟨*conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *lookup-clause-assn-is-None-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

lemma *size-lookup-conflict-alt-def*:
⟨*RETURN o size-lookup-conflict* = (λ(-, *b*, -). *RETURN b*)⟩
unfolding *size-lookup-conflict-def* **by** *auto*

sepref-def *size-lookup-conflict-impl*
is ‹*RETURN o size-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *size-lookup-conflict-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

sepref-def *is-in-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-lookup-conflict*)›
:: ‹$[\lambda((n, xs), L).\ atm\text{-}of\ L < length\ xs]_a$
lookup-clause-rel-assn^k $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *is-in-lookup-conflict-def is-NOTIN-alt-def*[*symmetric*]
lookup-clause-rel-assn-def
by *sepref*

lemma *lookup-clause-assn-is-empty-alt-def*:
‹*lookup-clause-assn-is-empty* = (λS. *size-lookup-conflict S* = *0*)›
by (*auto simp*: *size-lookup-conflict-def lookup-clause-assn-is-empty-def fun-eq-iff*)

sepref-def *lookup-clause-assn-is-empty-impl*
is ‹*RETURN o lookup-clause-assn-is-empty*›
:: ‹*conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *lookup-clause-assn-is-empty-alt-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

definition *the-lookup-conflict* :: ‹*conflict-option-rel* $\Rightarrow$ -› **where**
‹*the-lookup-conflict* = *snd*›

lemma *the-lookup-conflict-alt-def*:
‹*RETURN o the-lookup-conflict* = (λ(-, (*n*, *xs*)). *RETURN* (*n*, *xs*))›
by (*auto simp*: *the-lookup-conflict-def*)

sepref-def *the-lookup-conflict-impl*
is ‹*RETURN o the-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{d} $\rightarrow_a$ *lookup-clause-rel-assn*›
unfolding *the-lookup-conflict-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

definition *Some-lookup-conflict* :: ‹- $\Rightarrow$ *conflict-option-rel*› **where**
‹*Some-lookup-conflict xs* = (*False*, *xs*)›

lemma *Some-lookup-conflict-alt-def*:
‹*RETURN o Some-lookup-conflict* = (λ*xs*. *RETURN* (*False*, *xs*))›
by (*auto simp*: *Some-lookup-conflict-def*)

sepref-def *Some-lookup-conflict-impl*
is ‹*RETURN o Some-lookup-conflict*›
:: ‹*lookup-clause-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*›

unfolding *Some-lookup-conflict-alt-def conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*
sepref-register *Some-lookup-conflict*

type-synonym *cach-refinement-l-assn* = ⟨*8 word ptr* × *32 word array-list64*⟩

definition (**in** −) *cach-refinement-l-assn* :: - ⇒ *cach-refinement-l-assn* ⇒ - **where**
⟨*cach-refinement-l-assn* ≡ *array-assn minimize-status-assn* $\times_a$ *arl64-assn atom-assn*⟩

sepref-register *conflict-min-cach-l*
sepref-def *delete-from-lookup-conflict-code*
is ⟨*uncurry delete-from-lookup-conflict*⟩
:: ⟨*unat-lit-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $\rightarrow_a$ *lookup-clause-rel-assn*⟩
unfolding *delete-from-lookup-conflict-def NOTIN-def*[*symmetric*]
conflict-option-rel-assn-def
lookup-clause-rel-assn-def
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemma *arena-is-valid-clause-idx-le-uint64-max*:
⟨*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *sint64-max* ⟹
bd + *arena-length be bd* ≤ *sint64-max*⟩
⟨*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *sint64-max* ⟹
bd ≤ *sint64-max*⟩
using *arena-lifting*(*10*)[*of be - - bd*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

lemma *add-to-lookup-conflict-alt-def*:
⟨*RETURN oo add-to-lookup-conflict* = (λ*L* (*n*, *xs*). *RETURN* (*if xs* ! *atm-of L* = *NOTIN then n* + *1*
else n,
xs[*atm-of L* := *ISIN* (*is-pos L*)]))⟩
unfolding *add-to-lookup-conflict-def* **by** (*auto simp*: *fun-eq-iff*)

sepref-register *ISIN NOTIN atm-of add-to-lookup-conflict*

sepref-def *add-to-lookup-conflict-impl*
is ⟨*uncurry* (*RETURN oo add-to-lookup-conflict*)⟩
:: ⟨[λ(*L*, (*n*, *xs*)). *atm-of L* < *length xs* ∧ *n* + *1* ≤ *uint32-max*]$_a$
unat-lit-assn^k $*_a$ (*lookup-clause-rel-assn*)^d → *lookup-clause-rel-assn*⟩
unfolding *add-to-lookup-conflict-alt-def lookup-clause-rel-assn-def*
is-NOTIN-alt-def[*symmetric*] *fold-is-None NOTIN-def*
apply (*rewrite at* ⟨- + ¤⟩ *unat-const-fold*[**where** *'a* = ⟨*32*⟩])
by *sepref*

lemma *isa-lookup-conflict-merge-alt-def*:
⟨*isa-lookup-conflict-merge i0* = (λ*M N i zs clvls lbd outl*.
do {
let xs = *the-lookup-conflict zs*;
ASSERT(*arena-is-valid-clause-idx N i*);
(-, *clvls*, *zs*, *lbd*, *outl*) ← *WHILE*$_T$$^{\lambda(i::nat,\ clvls\ ::\ nat,\ zs,\ lbd,\ outl).}$ *length* (*snd zs*) = *length* (*snd xs*) ∧ Su
(λ(*j* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *j* < *i* + *arena-length N i*)
(λ(*j* :: *nat*, *clvls*, *zs*, *lbd*, *outl*). *do* {

ASSERT(j < *length N*);
ASSERT(*arena-lit-pre N j*);
ASSERT(*get-level-pol-pre* (*M*, *arena-lit N j*));
ASSERT(*get-level-pol M* (*arena-lit N j*) ≤ *Suc* (*uint32-max div 2*));
let lbd = *lbd-write lbd* (*get-level-pol M* (*arena-lit N j*));
ASSERT(*atm-of* (*arena-lit N j*) < *length* (*snd zs*));
ASSERT(¬*is-in-lookup-conflict zs* (*arena-lit N j*) ⟶ *length outl* < *uint32-max*);
let outl = *isa-outlearned-add M* (*arena-lit N j*) *zs outl*;
let clvls = *isa-clvls-add M* (*arena-lit N j*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*arena-lit N j*) *zs*;
RETURN(*Suc j*, *clvls*, *zs*, *lbd*, *outl*)
})
(*i* + *i0*, *clvls*, *xs*, *lbd*, *outl*);
RETURN (*Some-lookup-conflict zs*, *clvls*, *lbd*, *outl*)
})›
unfolding *isa-lookup-conflict-merge-def Some-lookup-conflict-def*
the-lookup-conflict-def
by (*auto simp*: *fun-eq-iff*)

sepref-def *resolve-lookup-conflict-merge-fast-code*
is ‹*uncurry6 isa-set-lookup-conflict-aa*›
:: ‹[λ((((((*M*, *N*), *i*), (-, *xs*)), -), -), *out*).
length N ≤ *sint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *conflict-option-rel-assn*^{d} $*_a$
uint32-nat-assn^k $*_a$ *lbd-assn*^{d} $*_a$ *out-learned-assn*^{d} →
conflict-option-rel-assn $\times_a$ *uint32-nat-assn* $\times_a$ *lbd-assn* $\times_a$ *out-learned-assn*›
supply
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint32-max*[*dest*]
arena-is-valid-clause-idx-le-uint64-max[*dest*]
unfolding *isa-set-lookup-conflict-aa-def lookup-conflict-merge-def*
PR-CONST-def nth-rll-def[*symmetric*]
isa-outlearned-add-def isa-clvls-add-def
isa-lookup-conflict-merge-alt-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *add-0-right*
apply (*rewrite at* ‹*RETURN* (⌑, - ,-, -)› *Suc-eq-plus1*)
apply (*rewrite at* ‹*RETURN* (- + ⌑, - ,-, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite* **in** ‹*If* - ⌑› *unat-const-fold*[**where** *'a* = ‹*32*›])
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register *isa-resolve-merge-conflict-gt2*

lemma *arena-is-valid-clause-idx-le-uint64-max2*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *sint64-max* ⟹
bd + *arena-length be bd* ≤ *sint64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *sint64-max* ⟹
bd < *sint64-max*›
using *arena-lifting*(*10*)[*of be - - bd*]
apply (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)
using *arena-lengthI*(*2*) *less-le-trans* **by** *blast*

sepref-def *resolve-merge-conflict-fast-code*
is ‹*uncurry6 isa-resolve-merge-conflict-gt2*›
:: ‹$[uncurry6\ (\lambda M\ N\ i\ (b,\ xs)\ clvls\ lbd\ outl.\ length\ N \le sint64\text{-}max)]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^d *_a$
$uint32\text{-}nat\text{-}assn^k *_a lbd\text{-}assn^d *_a out\text{-}learned\text{-}assn^d \rightarrow$
$conflict\text{-}option\text{-}rel\text{-}assn \times_a uint32\text{-}nat\text{-}assn \times_a lbd\text{-}assn \times_a out\text{-}learned\text{-}assn$›
supply
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint32-max*[*dest*]
fmap-length-rll-u-def[*simp*]
arena-is-valid-clause-idx-le-uint64-max[*intro*]
arena-is-valid-clause-idx-le-uint64-max2[*dest*]
unfolding *isa-resolve-merge-conflict-gt2-def lookup-conflict-merge-def*
PR-CONST-def nth-rll-def[*symmetric*]
isa-outlearned-add-def isa-clvls-add-def
isa-lookup-conflict-merge-alt-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *add-0-right*
apply (*rewrite at* ‹*RETURN* (¤, - ,-, -)› *Suc-eq-plus1*)
apply (*rewrite at* ‹*WHILEIT* - - - (- + ¤, - ,-, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite at* ‹*RETURN* (- + ¤, - ,-, -, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite* **in** ‹*If* - ¤› *unat-const-fold*[**where** *'a* = ‹*32*›])
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *atm-in-conflict-code*
is ‹*uncurry* (*RETURN oo atm-in-conflict-lookup*)›
:: ‹$[uncurry\ atm\text{-}in\text{-}conflict\text{-}lookup\text{-}pre]_a$
$atom\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k \rightarrow bool1\text{-}assn$›
unfolding *atm-in-conflict-lookup-def atm-in-conflict-lookup-pre-def*
is-NOTIN-alt-def[*symmetric*] *fold-is-None NOTIN-def lookup-clause-rel-assn-def*
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
by *sepref*

sepref-def *conflict-min-cach-l-code*
is ‹*uncurry* (*RETURN oo conflict-min-cach-l*)›
:: ‹$[conflict\text{-}min\text{-}cach\text{-}l\text{-}pre]_a\ cach\text{-}refinement\text{-}l\text{-}assn^k *_a atom\text{-}assn^k \rightarrow minimize\text{-}status\text{-}assn$›
unfolding *conflict-min-cach-l-def conflict-min-cach-l-pre-def cach-refinement-l-assn-def*
apply (*rewrite at nth - eta-expand*)
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
by *sepref*

lemma *conflict-min-cach-set-failed-l-alt-def*:
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *uint32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if b then sup* @ [*L*] *else sup*)
})›
unfolding *conflict-min-cach-set-failed-l-def Let-def* **by** *auto*

lemma *le-uint32-max-div2-le-uint32-max*: ‹*a2'* ≤ *Suc* (*uint32-max div 2*) ⟹ *a2'* < *uint32-max*›
by (*auto simp*: *uint32-max-def*)

sepref-def *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹*cach-refinement-l-assn*^{d} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *cach-refinement-l-assn*›
supply [[*goals-limit=1*]] *le-uint32-max-div2-le-uint32-max*[*dest*]
unfolding *conflict-min-cach-set-failed-l-alt-def*
minimize-status-rel-eq-def[*symmetric*] *cach-refinement-l-assn-def*

apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
apply (*rewrite at* ‹*list-update* - - -› *annot-index-of-atm*)
by *sepref*

lemma *conflict-min-cach-set-removable-l-alt-def*:
‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* $\leq$ *1* + *uint32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if b then sup* @ [*L*] *else sup*)
})›
unfolding *conflict-min-cach-set-removable-l-def* **by** *auto*

sepref-def *conflict-min-cach-set-removable-l-code*
is ‹*uncurry conflict-min-cach-set-removable-l*›
:: ‹*cach-refinement-l-assn*^{d} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *cach-refinement-l-assn*›
unfolding *conflict-min-cach-set-removable-l-alt-def*
minimize-status-rel-eq-def[*symmetric*] *cach-refinement-l-assn-def*
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
apply (*rewrite at* ‹*list-update* - - -› *annot-index-of-atm*)
by *sepref*

lemma *lookup-conflict-size-impl-alt-def*:
‹*RETURN o* (λ(*n*, *xs*). *n*) = (λ(*n*, *xs*). *RETURN n*)›
by *auto*

sepref-def *lookup-conflict-size-impl*
is [] ‹*RETURN o* (λ(*n*, *xs*). *n*)›
:: ‹*lookup-clause-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *lookup-clause-rel-assn-def lookup-conflict-size-impl-alt-def*
by *sepref*

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
by *auto*

sepref-register *lookup-conflict-remove1*

sepref-register *isa-lit-redundant-rec-wl-lookup*

sepref-register *isa-mark-failed-lits-stack*

sepref-register *lit-redundant-rec-wl-lookup conflict-min-cach-set-removable-l*
get-propagation-reason-pol lit-redundant-reason-stack-wl-lookup

sepref-register *isa-minimize-and-extract-highest-lookup-conflict isa-literal-redundant-wl-lookup*

lemma *set-lookup-empty-conflict-to-none-alt-def*:
⟨*RETURN o set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). *RETURN* (*True*, *n*, *xs*))⟩
by (*auto simp*: *set-lookup-empty-conflict-to-none-def*)

sepref-def *set-lookup-empty-conflict-to-none-imple*
is ⟨*RETURN o set-lookup-empty-conflict-to-none*⟩
:: ⟨*lookup-clause-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*⟩
unfolding *set-lookup-empty-conflict-to-none-alt-def*
lookup-clause-rel-assn-def conflict-option-rel-assn-def
by *sepref*

lemma *isa-mark-failed-lits-stackI*:
assumes
⟨*length ba* ≤ *Suc* (*uint32-max div 2*)⟩ **and**
⟨*a1'* < *length ba*⟩
shows ⟨*Suc a1'* ≤ *uint32-max*⟩
using *assms* **by** (*auto simp*: *uint32-max-def*)

sepref-register *conflict-min-cach-set-failed-l*
sepref-def *isa-mark-failed-lits-stack-fast-code*
is ⟨*uncurry2* (*isa-mark-failed-lits-stack*)⟩
:: ⟨[λ((*N*, -), -). *length N* ≤ *sint64-max*]$_a$
arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{k} $*_a$ *cach-refinement-l-assn*^{d} →
cach-refinement-l-assn⟩
supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*!] *image-image*[*simp*]
mark-failed-lits-stack-inv-helper1[*dest*] *mark-failed-lits-stack-inv-helper2*[*dest*]
fmap-length-rll-u-def[*simp*] *isa-mark-failed-lits-stackI*[*intro*]
arena-is-valid-clause-idx-le-uint64-max[*intro*] *le-uint32-max-div2-le-uint32-max*[*intro*]
unfolding *isa-mark-failed-lits-stack-def PR-CONST-def*
conflict-min-cach-set-failed-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*]
fmap-rll-def[*symmetric*]
arena-lit-def[*symmetric*]
minimize-status-rel-eq-def[*symmetric*]
apply (*rewrite at 1* **in** ⟨*conflict-min-cach-set-failed-l* - ⌑⟩ *snat-const-fold*[**where** *'a* = ⟨*64*⟩])
apply (*rewrite* **in** ⟨*RETURN* (- + ⌑, -)⟩ *snat-const-fold*[**where** *'a* = ⟨*64*⟩])
apply (*rewrite at 0* **in** ⟨(⌑, -)⟩ *snat-const-fold*[**where** *'a* = ⟨*64*⟩])
apply (*rewrite at* ⟨*arena-lit* - (- + ⌑ − -)⟩ *annot-unat-snat-upcast*[**where** *'l* = *64*])
by *sepref*

sepref-def *isa-get-literal-and-remove-of-analyse-wl-fast-code*
is ⟨*uncurry* (*RETURN oo isa-get-literal-and-remove-of-analyse-wl*)⟩
:: ⟨[λ(*arena*, *analyse*). *isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ∧
length arena ≤ *sint64-max*]$_a$
arena-fast-assn^k $*_a$ *analyse-refinement-fast-assn*^{d} →
unat-lit-assn $\times_a$ *analyse-refinement-fast-assn*⟩
supply [[*goals-limit=1*]] *arena-lit-pre-le2*[*dest*]
and [*dest*] = *arena-lit-implI*
unfolding *isa-get-literal-and-remove-of-analyse-wl-pre-def*
isa-get-literal-and-remove-of-analyse-wl-def
apply (*rewrite at* ⟨*length* - − ⌑⟩ *snat-const-fold*[**where** *'a*=*64*])

apply (*rewrite at* ‹*arena-lit* - (- + ⌑)› *annot-unat-snat-upcast*[**where** *'l* = *64*])
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-def *ana-lookup-conv-lookup-fast-code*
is ‹*uncurry* (*RETURN oo ana-lookup-conv-lookup*)›
:: ‹[*uncurry ana-lookup-conv-lookup-pre*]$_a$ *arena-fast-assn*^{k} $*_a$
(*ana-refinement-fast-assn*)^k
→ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*›
unfolding *ana-lookup-conv-lookup-pre-def ana-lookup-conv-lookup-def*
apply (*rewrite at* ‹(-, -, ⌑, -)› *annot-unat-snat-upcast*[**where** *'l* = *64*])
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *lit-redundant-reason-stack-wl-lookup-fast-code*
is ‹*uncurry2* (*RETURN ooo lit-redundant-reason-stack-wl-lookup*)›
:: ‹[*uncurry2 lit-redundant-reason-stack-wl-lookup-pre*]$_a$
unat-lit-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} →
ana-refinement-fast-assn›
unfolding *lit-redundant-reason-stack-wl-lookup-def lit-redundant-reason-stack-wl-lookup-pre-def*
apply (*rewrite at* ‹⌑ < -› *snat-const-fold*[**where** *'a*=*64*])
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemma *isa-lit-redundant-rec-wl-lookupI*:
assumes
‹*length ba* ≤ *Suc* (*uint32-max div 2*)›
shows ‹*length ba* < *uint32-max*›
using *assms* **by** (*auto simp*: *uint32-max-def*)

lemma *arena-lit-pre-le*: ‹
arena-lit-pre a i ⟹ *length a* ≤ *sint64-max* ⟹ *i* ≤ *sint64-max*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

lemma *get-propagation-reason-pol-get-propagation-reason-pol-raw*: ‹*do* {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *f C*
| *None* ⇒ *g*
} = *do* {
C ← *get-propagation-reason-raw-pol M* (−*L*);
if C ≠ *DECISION-REASON then f C else g*
}›
by (*cases M*) (*auto simp*: *get-propagation-reason-pol-def get-propagation-reason-raw-pol-def*)

sepref-register *atm-in-conflict-lookup*
sepref-def *lit-redundant-rec-wl-lookup-fast-code*
is ‹*uncurry5* (*isa-lit-redundant-rec-wl-lookup*)›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *analysis*), *lbd*). *length NU* ≤ *sint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ (*lookup-clause-rel-assn*)^k $*_a$
cach-refinement-l-assn^d $*_a$ *analyse-refinement-fast-assn*^{d} $*_a$ *lbd-assn*^{k} →
cach-refinement-l-assn $\times_a$ *analyse-refinement-fast-assn* $\times_a$ *bool1-assn*›

supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*] *image-image*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *nth-rll-def*[*simp*]
fmap-length-rll-u-def[*simp*]
isa-lit-redundant-rec-wl-lookupI[*intro*]
arena-lit-pre-le[*dest*] *isa-mark-failed-lits-stackI*[*intro*]

unfolding *isa-lit-redundant-rec-wl-lookup-def*
conflict-min-cach-set-removable-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*] *PR-CONST-def*
fmap-rll-u-def[*symmetric*] *minimize-status-rel-eq-def*[*symmetric*]
fmap-rll-def[*symmetric*] *length-0-conv*[*symmetric*]
apply (*subst get-propagation-reason-pol-get-propagation-reason-pol-raw*)
apply (*rewrite at* ‹*get-level-pol* - - = ⌑› *unat-const-fold*[**where** *'a*=*32*])
apply (*rewrite at* ‹(-, ⌑, -)› *annotate-assn*[**where** *A*=*analyse-refinement-fast-assn*])
apply (*annot-snat-const TYPE*(*64*))

unfolding *nth-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
fmap-rll-u-def[*symmetric*]
by *sepref*

sepref-def *delete-index-and-swap-code*
is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
:: ‹$[\lambda(xs, i).\ i < length\ xs]_a$
$(arl64\text{-}assn\ unat\text{-}lit\text{-}assn)^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow arl64\text{-}assn\ unat\text{-}lit\text{-}assn$›
unfolding *delete-index-and-swap.simps*
by *sepref*

sepref-def *lookup-conflict-upd-None-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-upd-None*)›
:: ‹$[\lambda((n, xs), i).\ i < length\ xs \wedge n > 0]_a$
$lookup\text{-}clause\text{-}rel\text{-}assn^d *_a sint32\text{-}nat\text{-}assn^k \rightarrow lookup\text{-}clause\text{-}rel\text{-}assn$›
unfolding *lookup-conflict-upd-None-RETURN-def lookup-clause-rel-assn-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemma *uint32-max-ge0*: ‹*0* < *uint32-max*› **by** (*auto simp*: *uint32-max-def*)

sepref-def *literal-redundant-wl-lookup-fast-code*
is ‹*uncurry5 isa-literal-redundant-wl-lookup*›
:: ‹$[\lambda(((((M, NU), D), cach), L), lbd).\ length\ NU \leq sint64\text{-}max]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a lookup\text{-}clause\text{-}rel\text{-}assn^k *_a$
$cach\text{-}refinement\text{-}l\text{-}assn^d *_a unat\text{-}lit\text{-}assn^k *_a lbd\text{-}assn^k \rightarrow$
$cach\text{-}refinement\text{-}l\text{-}assn \times_a analyse\text{-}refinement\text{-}fast\text{-}assn \times_a bool1\text{-}assn$›
supply [[*goals-limit*=*1*]]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*] *uint32-max-ge0*[*intro*!]

literals-are-in-$\mathcal{L}_{in}$-trail-uminus-in-lits-of-l-atms[*intro*]
unfolding *isa-literal-redundant-wl-lookup-def PR-CONST-def*
minimize-status-rel-eq-def[*symmetric*]
apply (*rewrite at* ‹(-, ⌑, -)› *al-fold-custom-empty*[**where** *'l=64*])+
unfolding *single-replicate*
apply (*rewrite at* ‹*get-level-pol* - - = ⌑› *unat-const-fold*[**where** *'a=32*])
unfolding *al-fold-custom-empty*[**where** *'l=64*]
apply (*subst get-propagation-reason-pol-get-propagation-reason-pol-raw*)
by *sepref*

sepref-def *conflict-remove1-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-remove1*)›
:: ‹[*lookup-conflict-remove1-pre*]$_a$ *unat-lit-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $\rightarrow$
lookup-clause-rel-assn›
supply [[*goals-limit=2*]]
unfolding *lookup-conflict-remove1-def lookup-conflict-remove1-pre-def lookup-clause-rel-assn-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-def *minimize-and-extract-highest-lookup-conflict-fast-code*
is ‹*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *lbd*), *outl*). *length NU* ≤ *sint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $*_a$
cach-refinement-l-assn^d $*_a$ *lbd-assn*^{k} $*_a$ *out-learned-assn*^{d} $\rightarrow$
lookup-clause-rel-assn $\times_a$ *cach-refinement-l-assn* $\times_a$ *out-learned-assn*›
supply [[*goals-limit=1*]]
literals-are-in-$\mathcal{L}_{in}$-trail-uminus-in-lits-of-l[*intro*]
minimize-and-extract-highest-lookup-conflict-inv-def[*simp*]
in-$\mathcal{L}_{all}$-less-uint32-max'[*intro*]
unfolding *isa-minimize-and-extract-highest-lookup-conflict-def*
PR-CONST-def
minimize-and-extract-highest-lookup-conflict-inv-def
apply (*rewrite at* ‹(-, ⌑, -, -)› *snat-const-fold*[**where** *'a = 64*])
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *isasat-lookup-merge-eq2-alt-def*:
‹*isasat-lookup-merge-eq2 L M N C* = (λ*zs clvls lbd outl. do* {
let zs = the-lookup-conflict zs;
ASSERT(*arena-lit-pre N C*);
ASSERT(*arena-lit-pre N* (*C+1*));
let L0 = arena-lit N C;
let L' = (*if L0 = L then arena-lit N* (*C + 1*) *else L0*);
ASSERT(*get-level-pol-pre* (*M*, *L'*));
ASSERT(*get-level-pol M L'* ≤ *Suc* (*uint32-max div 2*));
let lbd = lbd-write lbd (*get-level-pol M L'*);
ASSERT(*atm-of L'* < *length* (*snd zs*));
ASSERT(*length outl* < *uint32-max*);
let outl = isa-outlearned-add M L' zs outl;
ASSERT(*clvls* < *uint32-max*);
ASSERT(*fst zs* < *uint32-max*);
let clvls = isa-clvls-add M L' zs clvls;

let zs = add-to-lookup-conflict L′ zs;
RETURN(Some-lookup-conflict zs, clvls, lbd, outl)
})›
by (*auto simp*: *the-lookup-conflict-def Some-lookup-conflict-def Let-def*
isasat-lookup-merge-eq2-def fun-eq-iff)

sepref-def *isasat-lookup-merge-eq2-fast-code*
is ‹*uncurry7 isasat-lookup-merge-eq2*›
:: ‹$[\lambda(((((((L, M), NU), -), -), -), -), -).\ length\ NU \leq sint64\text{-}max]_a$
unat-lit-assn^k $*_a$ *trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
conflict-option-rel-assn^d $*_a$ *uint32-nat-assn*^{k} $*_a$ *lbd-assn*^{d} $*_a$ *out-learned-assn*^{d} $\rightarrow$
conflict-option-rel-assn $\times_a$ *uint32-nat-assn* $\times_a$ *lbd-assn* $\times_a$ *out-learned-assn*›
supply [[*goals-limit* = *1*]]
unfolding *isasat-lookup-merge-eq2-alt-def*
isa-outlearned-add-def isa-clvls-add-def
is-NOTIN-def[*symmetric*]
supply
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-uint32-max*[*dest*]
fmap-length-rll-u-def[*simp*] *the-lookup-conflict-def*[*simp*]
arena-is-valid-clause-idx-le-uint64-max[*dest*]
arena-lit-pre-le2[*dest*] *arena-lit-pre-le*[*dest*]
apply (*rewrite* **in** ‹*if - then - +* ⌑ *else -*› *unat-const-fold*[**where** ′*a*=*32*])
apply (*rewrite* **in** ‹*if - then arena-lit - (- +* ⌑*) else -*› *snat-const-fold*[**where** ′*a*=*64*])
by *sepref*

experiment begin

export-llvm
nat-lit-eq-impl
minimize-status-rel-eq-impl
SEEN-FAILED-impl
SEEN-UNKNOWN-impl
SEEN-REMOVABLE-impl
Some-impl
is-Notin-impl
NOTIN-impl
lookup-clause-assn-is-None-impl
size-lookup-conflict-impl
is-in-conflict-code
lookup-clause-assn-is-empty-impl
the-lookup-conflict-impl
Some-lookup-conflict-impl
delete-from-lookup-conflict-code
add-to-lookup-conflict-impl
resolve-lookup-conflict-merge-fast-code
resolve-merge-conflict-fast-code
atm-in-conflict-code
conflict-min-cach-l-code
conflict-min-cach-set-failed-l-code
conflict-min-cach-set-removable-l-code
lookup-conflict-size-impl
set-lookup-empty-conflict-to-none-imple
isa-mark-failed-lits-stack-fast-code
isa-get-literal-and-remove-of-analyse-wl-fast-code
ana-lookup-conv-lookup-fast-code

lit-redundant-reason-stack-wl-lookup-fast-code
lit-redundant-rec-wl-lookup-fast-code
delete-index-and-swap-code
lookup-conflict-upd-None-code
literal-redundant-wl-lookup-fast-code
conflict-remove1-code
minimize-and-extract-highest-lookup-conflict-fast-code
isasat-lookup-merge-eq2-fast-code

end

end
theory *IsaSAT-Setup-LLVM*
imports *IsaSAT-Setup IsaSAT-Watch-List-LLVM IsaSAT-Lookup-Conflict-LLVM Watched-Literals.WB-More-Refinement IsaSAT-Clauses-LLVM LBD-LLVM*
begin

no-notation *WB-More-Refinement.fref* ($[\text{-}]_f$ - $\to$ - [*0*,*60*,*60*] *60*)
no-notation *WB-More-Refinement.freft* (- $\to_f$ - [*60*,*60*] *60*)

abbreviation *word32-rel* ≡ *word-rel* :: (*32 word* × -) *set*
abbreviation *word64-rel* ≡ *word-rel* :: (*64 word* × -) *set*
abbreviation *word32-assn* ≡ *word-assn* :: *32 word* ⇒ -
abbreviation *word64-assn* ≡ *word-assn* :: *64 word* ⇒ -

abbreviation *stats-rel* :: ‹(*stats* × *stats*) *set*› **where**
‹*stats-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*›

abbreviation *ema-rel* :: ‹(*ema*×*ema*) *set*› **where**
‹*ema-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*›

abbreviation *ema-assn* :: ‹*ema* ⇒ *ema* ⇒ *assn*› **where**
‹*ema-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*›

abbreviation *stats-assn* :: ‹*stats* ⇒ *stats* ⇒ *assn*› **where**
‹*stats-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *ema-assn*›

lemma [*sepref-import-param*]:
(*ema-get-value*, *ema-get-value*) ∈ *ema-rel* → *word64-rel*
(*ema-bitshifting*,*ema-bitshifting*) ∈ *word64-rel*
(*ema-reinit*,*ema-reinit*) ∈ *ema-rel* → *ema-rel*
(*ema-init*,*ema-init*) ∈ *word-rel* → *ema-rel*
by *auto*

lemma *ema-bitshifting-inline*[*llvm-inline*]:
ema-bitshifting = (*0x100000000*::-::*len word*) **by** (*auto simp*: *ema-bitshifting-def*)

lemma *ema-reinit-inline*[*llvm-inline*]:
ema-reinit = (λ(*value*, α, β, *wait*, *period*).
(*value*, α, *0x100000000*::-::*len word*, *0*::- *word*, *0*:: - *word*))

by *auto*

lemmas [*llvm-inline*] = *ema-init-def*

sepref-def *ema-update-impl* **is** *uncurry* (*RETURN oo ema-update*)
:: $uint32\text{-}nat\text{-}assn^k *_a ema\text{-}assn^k \rightarrow_a ema\text{-}assn$
unfolding *ema-update-def*
apply (*rewrite at* ‹*let* - = *of-nat* ¤ ∗ - *in* -› *annot-unat-unat-upcast*[**where** *'l* = *64*])
apply (*rewrite at* ‹*let* -=- + -; -=¤ *in* -› *fold-COPY*)

apply (*annot-unat-const TYPE*(*64*))
supply [[*goals-limit* = *1*]]
by *sepref*

lemma [*sepref-import-param*]:
(*incr-propagation*,*incr-propagation*) ∈ *stats-rel* → *stats-rel*
(*incr-conflict*,*incr-conflict*) ∈ *stats-rel* → *stats-rel*
(*incr-decision*,*incr-decision*) ∈ *stats-rel* → *stats-rel*
(*incr-restart*,*incr-restart*) ∈ *stats-rel* → *stats-rel*
(*incr-lrestart*,*incr-lrestart*) ∈ *stats-rel* → *stats-rel*
(*incr-uset*,*incr-uset*) ∈ *stats-rel* → *stats-rel*
(*incr-GC*,*incr-GC*) ∈ *stats-rel* → *stats-rel*
(*add-lbd*,*add-lbd*) ∈ *word64-rel* → *stats-rel* → *stats-rel*
by *auto*

lemmas [*llvm-inline*] =
incr-propagation-def
incr-conflict-def
incr-decision-def
incr-restart-def
incr-lrestart-def
incr-uset-def
incr-GC-def

abbreviation (*input*) *restart-info-rel* ≡ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel* $\times_r$ *word64-rel*

abbreviation (*input*) *restart-info-assn* **where**
‹*restart-info-assn* ≡ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*›

lemma *restart-info-params*[*sepref-import-param*]:
(*incr-conflict-count-since-last-restart*,*incr-conflict-count-since-last-restart*) ∈
restart-info-rel → *restart-info-rel*
(*restart-info-update-lvl-avg*,*restart-info-update-lvl-avg*) ∈
word32-rel → *restart-info-rel* → *restart-info-rel*
(*restart-info-init*,*restart-info-init*) ∈ *restart-info-rel*
(*restart-info-restart-done*,*restart-info-restart-done*) ∈ *restart-info-rel* → *restart-info-rel*
by *auto*

lemmas [*llvm-inline*] =
incr-conflict-count-since-last-restart-def
restart-info-update-lvl-avg-def
restart-info-init-def
restart-info-restart-done-def

type-synonym *vmtf-node-assn* = (*64 word* × *32 word* × *32 word*)

definition *vmtf-node1-rel* ≡ { ((*a*,*b*,*c*),(*VMTF-Node a b c*)) | *a b c*. *True*}
definition *vmtf-node2-assn* ≡ *uint64-nat-assn* $\times_a$ *atom.option-assn* $\times_a$ *atom.option-assn*

definition *vmtf-node-assn* ≡ *hr-comp vmtf-node2-assn vmtf-node1-rel*
lemmas [*fcomp-norm-unfold*] = *vmtf-node-assn-def*[*symmetric*]

lemma *vmtf-node-assn-pure*[*safe-constraint-rules*]: ‹*CONSTRAINT is-pure vmtf-node-assn*›
  unfolding *vmtf-node-assn-def vmtf-node2-assn-def*
  by *solve-constraint*

lemmas [*sepref-frame-free-rules*] = *mk-free-is-pure*[*OF vmtf-node-assn-pure*[*unfolded CONSTRAINT-def*]]

lemma
    vmtf-Node-refine1: (λ*a b c*. (*a*,*b*,*c*), *VMTF-Node*) ∈ *Id* → *Id* → *Id* → *vmtf-node1-rel*
and *vmtf-stamp-refine1*: (λ(*a*,*b*,*c*). *a*, *stamp*) ∈ *vmtf-node1-rel* → *Id*
and *vmtf-get-prev-refine1*: (λ(*a*,*b*,*c*). *b*, *get-prev*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*
and *vmtf-get-next-refine1*: (λ(*a*,*b*,*c*). *c*, *get-next*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*
  by (*auto simp*: *vmtf-node1-rel-def*)

sepref-def *VMTF-Node-impl* **is** []
  uncurry2 (*RETURN ooo* (λ*a b c*. (*a*,*b*,*c*)))
  :: *uint64-nat-assn*^{k} $*_a$ (*atom.option-assn*)^k $*_a$ (*atom.option-assn*)^k $\rightarrow_a$ *vmtf-node2-assn*
  unfolding *vmtf-node2-assn-def* **by** *sepref*

sepref-def *VMTF-stamp-impl*
  is [] *RETURN o* (λ(*a*,*b*,*c*). *a*)
  :: *vmtf-node2-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*
  unfolding *vmtf-node2-assn-def*
  by *sepref*

sepref-def *VMTF-get-prev-impl*
  is [] *RETURN o* (λ(*a*,*b*,*c*). *b*)
  :: *vmtf-node2-assn*^{k} $\rightarrow_a$ *atom.option-assn*
  unfolding *vmtf-node2-assn-def*
  by *sepref*

sepref-def *VMTF-get-next-impl*
  is [] *RETURN o* (λ(*a*,*b*,*c*). *c*)
  :: *vmtf-node2-assn*^{k} $\rightarrow_a$ *atom.option-assn*
  unfolding *vmtf-node2-assn-def*
  by *sepref*

lemma *workaround-hrcomp-id-norm*[*fcomp-norm-unfold*]: *hr-comp R* (⟨*nat-rel*⟩*option-rel*) = *R* **by** *simp*

lemmas [*sepref-fr-rules*] =
  VMTF-Node-impl.refine[*FCOMP vmtf-Node-refine1*]

VMTF-stamp-impl.refine[*FCOMP vmtf-stamp-refine1*]
VMTF-get-prev-impl.refine[*FCOMP vmtf-get-prev-refine1*]
VMTF-get-next-impl.refine[*FCOMP vmtf-get-next-refine1*]

type-synonym *vmtf-assn* = ⟨*vmtf-node-assn ptr* × *64 word* × *32 word* × *32 word* × *32 word*⟩

type-synonym *vmtf-remove-assn* = ⟨*vmtf-assn* × (*32 word array-list64* × *1 word ptr*)⟩

abbreviation *vmtf-assn* :: - ⇒ *vmtf-assn* ⇒ *assn* **where**
⟨*vmtf-assn* ≡ (*array-assn vmtf-node-assn* $\times_a$ *uint64-nat-assn* $\times_a$ *atom-assn* $\times_a$ *atom-assn* $\times_a$ *atom.option-assn*)⟩

abbreviation *atoms-hash-assn* :: ⟨*bool list* ⇒ *1 word ptr* ⇒ *assn*⟩ **where**
⟨*atoms-hash-assn* ≡ *array-assn bool1-assn*⟩

abbreviation *distinct-atoms-assn* **where**
⟨*distinct-atoms-assn* ≡ *arl64-assn atom-assn* $\times_a$ *atoms-hash-assn*⟩

definition *vmtf-remove-assn*
:: ⟨*isa-vmtf-remove-int* ⇒ *vmtf-remove-assn* ⇒ *assn*⟩
where
⟨*vmtf-remove-assn* ≡ *vmtf-assn* $\times_a$ *distinct-atoms-assn*⟩

Options

type-synonym *opts-assn* = *1 word* × *1 word* × *1 word*

definition *opts-assn*
:: ⟨*opts* ⇒ *opts-assn* ⇒ *assn*⟩
where
⟨*opts-assn* ≡ *bool1-assn* $\times_a$ *bool1-assn* $\times_a$ *bool1-assn*⟩

lemma *workaround-opt-assn*: *RETURN o* (λ(*a*,*b*,*c*). *f a b c*) = (λ(*a*,*b*,*c*). *RETURN* (*f a b c*)) **by** *auto*

sepref-register *opts-restart opts-reduce opts-unbounded-mode*

sepref-def *opts-restart-impl* **is** *RETURN o opts-restart* :: $opts\text{-}assn^k \rightarrow_a bool1\text{-}assn$
unfolding *opts-restart-def workaround-opt-assn opts-assn-def*
by *sepref*

sepref-def *opts-reduce-impl* **is** *RETURN o opts-reduce* :: $opts\text{-}assn^k \rightarrow_a bool1\text{-}assn$
unfolding *opts-reduce-def workaround-opt-assn opts-assn-def*
by *sepref*

sepref-def *opts-unbounded-mode-impl* **is** *RETURN o opts-unbounded-mode* :: $opts\text{-}assn^k \rightarrow_a bool1\text{-}assn$
unfolding *opts-unbounded-mode-def workaround-opt-assn opts-assn-def*
by *sepref*

abbreviation *watchlist-fast-assn* ≡ *aal-assn′ TYPE*(*64*) *TYPE*(*64*) *watcher-fast-assn*

type-synonym *vdom-fast-assn* = ⟨*64 word array-list64*⟩
abbreviation *vdom-fast-assn* :: ⟨*vdom* ⇒ *vdom-fast-assn* ⇒ *assn*⟩ **where**
⟨*vdom-fast-assn* ≡ *arl64-assn sint64-nat-assn*⟩

type-synonym *phase-saver-assn* = *1 word larray64*
abbreviation *phase-saver-assn* :: ⟨*phase-saver* ⇒ *phase-saver-assn* ⇒ *assn*⟩ **where**
⟨*phase-saver-assn* ≡ *larray64-assn bool1-assn*⟩

type-synonym *phase-saver′-assn* = *1 word ptr*

abbreviation *phase-saver′-assn* :: ⟨*phase-saver* ⇒ *phase-saver′-assn* ⇒ *assn*⟩ **where**
⟨*phase-saver′-assn* ≡ *array-assn bool1-assn*⟩

type-synonym *arena-assn* = (*32 word*, *64*) *array-list*
type-synonym *heur-assn* = ⟨(*ema* × *ema* × *restart-info* × *64 word* ×
phase-saver-assn × *64 word* × *phase-saver′-assn* × *64 word* × *phase-saver′-assn* × *64 word* × *64 word* × *64 word*)⟩

type-synonym *twl-st-wll-trail-fast* =
⟨*trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × *vmtf-remove-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *out-learned-assn* × *stats* ×
heur-assn ×
vdom-fast-assn × *vdom-fast-assn* × *64 word* × *opts-assn* × *arena-assn*⟩

abbreviation *phase-heur-assn* **where**
⟨*phase-heur-assn* ≡ *phase-saver-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *phase-saver′-assn* $\times_a$ *sint64-nat-assn* $\times_a$
phase-saver′-assn $\times_a$ *word64-assn* $\times_a$ *word64-assn* $\times_a$ *word64-assn*⟩

definition *heuristic-assn* :: ⟨*restart-heuristics* ⇒ *heur-assn* ⇒ *assn*⟩ **where**
⟨*heuristic-assn* = *ema-assn* $\times_a$
ema-assn $\times_a$
restart-info-assn $\times_a$
word64-assn $\times_a$ *phase-heur-assn*⟩

definition *isasat-bounded-assn* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wll-trail-fast* ⇒ *assn*⟩ **where**
⟨*isasat-bounded-assn* =
trail-pol-fast-assn $\times_a$ *arena-fast-assn* $\times_a$
conflict-option-rel-assn $\times_a$
sint64-nat-assn $\times_a$
watchlist-fast-assn $\times_a$
vmtf-remove-assn $\times_a$
uint32-nat-assn $\times_a$
cach-refinement-l-assn $\times_a$
lbd-assn $\times_a$
out-learned-assn $\times_a$
stats-assn $\times_a$
heuristic-assn $\times_a$
vdom-fast-assn $\times_a$
vdom-fast-assn $\times_a$
uint64-nat-assn $\times_a$
opts-assn $\times_a$ *arena-fast-assn*⟩

sepref-register *NORMAL-PHASE QUIET-PHASE DEFAULT-INIT-PHASE*

sepref-def *NORMAL-PHASE-impl*

is ⟨*uncurry0* (*RETURN NORMAL-PHASE*)⟩
:: ⟨*unit-assn*^{k} $\rightarrow_a$ *word-assn*⟩
unfolding *NORMAL-PHASE-def*
by *sepref*

sepref-def *QUIET-PHASE-impl*
is ⟨*uncurry0* (*RETURN QUIET-PHASE*)⟩
:: ⟨*unit-assn*^{k} $\rightarrow_a$ *word-assn*⟩
unfolding *QUIET-PHASE-def*
by *sepref*

Lift Operations to State

sepref-def *get-conflict-wl-is-None-fast-code*
is ⟨*RETURN o get-conflict-wl-is-None-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *get-conflict-wl-is-None-heur-alt-def isasat-bounded-assn-def length-ll-def*[*symmetric*]
conflict-option-rel-assn-def
supply [[*goals-limit=1*]]
by *sepref*

sepref-def *isa-count-decided-st-fast-code*
is ⟨*RETURN o isa-count-decided-st*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*⟩
supply [[*goals-limit=2*]]
unfolding *isa-count-decided-st-def isasat-bounded-assn-def*
by *sepref*

sepref-def *polarity-pol-fast*
is ⟨*uncurry* (*mop-polarity-pol*)⟩
:: ⟨*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*⟩
unfolding *mop-polarity-pol-def trail-pol-fast-assn-def*
polarity-pol-def polarity-pol-pre-def
by *sepref*

sepref-def *polarity-st-heur-pol-fast*
is ⟨*uncurry* (*mop-polarity-st-heur*)⟩
:: ⟨*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *tri-bool-assn*⟩
unfolding *mop-polarity-st-heur-alt-def isasat-bounded-assn-def polarity-st-pre-def*
mop-polarity-st-heur-alt-def
supply [[*goals-limit = 1*]]
by *sepref*

8.14.1 More theorems

lemma *count-decided-st-heur-alt-def*:
⟨*count-decided-st-heur* = (λ(*M*, -). *count-decided-pol M*)⟩
by (*auto simp*: *count-decided-st-heur-def count-decided-pol-def*)

sepref-def *count-decided-st-heur-pol-fast*
is ⟨*RETURN o count-decided-st-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*⟩
unfolding *isasat-bounded-assn-def count-decided-st-heur-alt-def*
supply [[*goals-limit = 1*]]
by *sepref*

sepref-def *access-lit-in-clauses-heur-fast-code*
is ‹*uncurry2* (*RETURN ooo access-lit-in-clauses-heur*)›
:: ‹[$\lambda((S, i), j)$. *access-lit-in-clauses-heur-pre* $((S, i), j) \wedge$
length (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$
isasat-bounded-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$ *unat-lit-assn*›
supply [[*goals-limit=1*]] *arena-lit-pre-le*[*dest*]
unfolding *isasat-bounded-assn-def access-lit-in-clauses-heur-alt-def*
access-lit-in-clauses-heur-pre-def
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register ‹(=) :: *clause-status* $\Rightarrow$ *clause-status* $\Rightarrow$ -›

lemma [*def-pat-rules*]: *append-ll* $\equiv$ *op-list-list-push-back*
by (*rule eq-reflection*) (*auto simp*: *append-ll-def fun-eq-iff*)

sepref-register *rewatch-heur mop-append-ll mop-arena-length*

sepref-def *mop-append-ll-impl*
is ‹*uncurry2 mop-append-ll*›
:: ‹[$\lambda((W, i), -)$. *length* (*W* ! (*nat-of-lit i*)) < *sint64-max*]$_a$
watchlist-fast-assn^d $*_a$ *unat-lit-assn*^{k} $*_a$ *watcher-fast-assn*^{k} $\rightarrow$ *watchlist-fast-assn*›
unfolding *mop-append-ll-def*
by *sepref*

sepref-def *rewatch-heur-fast-code*
is ‹*uncurry2* (*rewatch-heur*)›
:: ‹[$\lambda((vdom, arena), W)$. ($\forall x \in$ *set vdom*. $x \leq$ *sint64-max*) $\wedge$ *length arena* $\leq$ *sint64-max* $\wedge$
length vdom $\leq$ *sint64-max*]$_a$
vdom-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *watchlist-fast-assn*^{d} $\rightarrow$ *watchlist-fast-assn*›
supply [[*goals-limit=1*]]
arena-lit-pre-le-sint64-max[*dest*] *arena-is-valid-clause-idx-le-uint64-max*[*dest*]
supply [*simp*] = *append-ll-def*
supply [*dest*] = *arena-lit-implI*(*1*)
unfolding *rewatch-heur-alt-def Let-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *if-not-swap*
FOREACH-cond-def FOREACH-body-def
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-bounded-assn^d $\rightarrow$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def rewatch-heur-st-fast-pre-def*
isasat-bounded-assn-def rewatch-heur-st-fast-def
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register *length-avdom*

sepref-def *length-avdom-fast-code*
  is ‹*RETURN o length-avdom*›
  :: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k \rightarrow_a \mathit{sint64\text{-}nat\text{-}assn}$›
  unfolding *length-avdom-alt-def isasat-bounded-assn-def*
  supply [[*goals-limit* = *1*]]
  by *sepref*

sepref-register *get-the-propagation-reason-heur*

sepref-def *get-the-propagation-reason-heur-fast-code*
  is ‹*uncurry get-the-propagation-reason-heur*›
  :: ‹$\mathit{isasat\text{-}bounded\text{-}assn}^k *_a \mathit{unat\text{-}lit\text{-}assn}^k \rightarrow_a \mathit{snat\text{-}option\text{-}assn}'\ \mathit{TYPE}(64)$›
  unfolding *get-the-propagation-reason-heur-alt-def*
    isasat-bounded-assn-def
  supply [[*goals-limit* = *1*]]
  by *sepref*

sepref-def *clause-is-learned-heur-code2*
  is ‹*uncurry* (*RETURN oo clause-is-learned-heur*)›
  :: ‹$[\lambda(S,\ C).\ \mathit{arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C]_a$
    $\mathit{isasat\text{-}bounded\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k \rightarrow \mathit{bool1\text{-}assn}$›
  supply [[*goals-limit* = *1*]]
  unfolding *clause-is-learned-heur-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-register *clause-lbd-heur*

lemma *clause-lbd-heur-alt-def*:
  ‹*clause-lbd-heur* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*,
    lcount) *C*.
    arena-lbd N' C)›
  by (*intro ext*) (*auto simp*: *clause-lbd-heur-def*)

sepref-def *clause-lbd-heur-code2*
  is ‹*uncurry* (*RETURN oo clause-lbd-heur*)›
  :: ‹$[\lambda(S,\ C).\ \mathit{get\text{-}clause\text{-}LBD\text{-}pre}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C]_a$
    $\mathit{isasat\text{-}bounded\text{-}assn}^k *_a \mathit{sint64\text{-}nat\text{-}assn}^k \rightarrow \mathit{uint32\text{-}nat\text{-}assn}$›
  unfolding *isasat-bounded-assn-def clause-lbd-heur-alt-def*
  supply [[*goals-limit* = *1*]]
  by *sepref*

sepref-register *mark-garbage-heur*

sepref-def *mark-garbage-heur-code2*
  is ‹*uncurry2* (*RETURN ooo mark-garbage-heur*)›
  :: ‹$[\lambda((C,\ i),\ S).\ \mathit{mark\text{-}garbage\text{-}pre}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S,\ C) \wedge i < \mathit{length\text{-}avdom}\ S\ \wedge$
      $\mathit{get\text{-}learned\text{-}count}\ S \geq 1]_a$

$sint64$-nat-assn^k $*_a$ $sint64$-nat-assn^k $*_a$ isasat-bounded-assn^d $\rightarrow$ isasat-bounded-assn⟩
supply [[goals-limit = 1]]
unfolding mark-garbage-heur-def isasat-bounded-assn-def delete-index-and-swap-alt-def length-avdom-def fold-tuple-optimizations
apply (annot-unat-const TYPE(64))
by sepref

sepref-register delete-index-vdom-heur

sepref-def delete-index-vdom-heur-fast-code2
is ⟨uncurry (RETURN oo delete-index-vdom-heur)⟩
:: ⟨$[\lambda(i, S).\ i < length\text{-}avdom\ S]_a$
$sint64$-nat-assn^k $*_a$ isasat-bounded-assn^d $\rightarrow$ isasat-bounded-assn⟩
supply [[goals-limit = 1]]
unfolding delete-index-vdom-heur-def isasat-bounded-assn-def delete-index-and-swap-alt-def length-avdom-def fold-tuple-optimizations
by sepref

sepref-register access-length-heur

sepref-def access-length-heur-fast-code2
is ⟨uncurry (RETURN oo access-length-heur)⟩
:: ⟨$[\lambda(S, C).\ arena\text{-}is\text{-}valid\text{-}clause\text{-}idx\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C]_a$
isasat-bounded-assn^k $*_a$ $sint64$-nat-assn^k $\rightarrow$ $sint64$-nat-assn⟩
supply [[goals-limit = 1]]
unfolding access-length-heur-alt-def isasat-bounded-assn-def fold-tuple-optimizations
by sepref

sepref-register marked-as-used-st

sepref-def marked-as-used-st-fast-code
is ⟨uncurry (RETURN oo marked-as-used-st)⟩
:: ⟨$[\lambda(S, C).\ marked\text{-}as\text{-}used\text{-}pre\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C]_a$
isasat-bounded-assn^k $*_a$ $sint64$-nat-assn^k $\rightarrow$ bool1-assn⟩
supply [[goals-limit = 1]]
unfolding marked-as-used-st-alt-def isasat-bounded-assn-def fold-tuple-optimizations
by sepref

sepref-register mark-unused-st-heur
sepref-def mark-unused-st-fast-code
is ⟨uncurry (RETURN oo mark-unused-st-heur)⟩
:: ⟨$[\lambda(C, S).\ arena\text{-}act\text{-}pre\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\ C]_a$
$sint64$-nat-assn^k $*_a$ isasat-bounded-assn^d $\rightarrow$ isasat-bounded-assn⟩
unfolding mark-unused-st-heur-def isasat-bounded-assn-def arena-act-pre-mark-used[intro!]
supply [[goals-limit = 1]]
by sepref

sepref-def get-slow-ema-heur-fast-code
is ⟨RETURN o get-slow-ema-heur⟩
:: ⟨isasat-bounded-assn^k $\rightarrow_a$ ema-assn⟩
unfolding get-slow-ema-heur-alt-def isasat-bounded-assn-def heuristic-assn-def
by sepref

sepref-def *get-fast-ema-heur-fast-code*
is ⟨*RETURN o get-fast-ema-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *ema-assn*⟩
unfolding *get-fast-ema-heur-alt-def isasat-bounded-assn-def heuristic-assn-def*
by *sepref*

sepref-def *get-conflict-count-since-last-restart-heur-fast-code*
is ⟨*RETURN o get-conflict-count-since-last-restart-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *word64-assn*⟩
unfolding *get-counflict-count-heur-alt-def isasat-bounded-assn-def heuristic-assn-def*
by *sepref*

sepref-def *get-learned-count-fast-code*
is ⟨*RETURN o get-learned-count*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint64-nat-assn*⟩
unfolding *get-learned-count-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-register *incr-restart-stat*

sepref-def *incr-restart-stat-fast-code*
is ⟨*incr-restart-stat*⟩
:: ⟨*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *incr-restart-stat-def isasat-bounded-assn-def PR-CONST-def*
heuristic-assn-def fold-tuple-optimizations
by *sepref*

sepref-register *incr-lrestart-stat*

sepref-def *incr-lrestart-stat-fast-code*
is ⟨*incr-lrestart-stat*⟩
:: ⟨*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *incr-lrestart-stat-def isasat-bounded-assn-def PR-CONST-def*
heuristic-assn-def fold-tuple-optimizations
by *sepref*

sepref-def *opts-restart-st-fast-code*
is ⟨*RETURN o opts-restart-st*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *opts-restart-st-def isasat-bounded-assn-def*
by *sepref*

sepref-def *opts-reduction-st-fast-code*
is ⟨*RETURN o opts-reduction-st*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *opts-reduction-st-def isasat-bounded-assn-def*
by *sepref*

sepref-register *opts-reduction-st opts-restart-st*

lemma *emag-get-value-alt-def*:
  ‹*ema-get-value* = (λ(*a*, *b*, *c*, *d*). *a*)›
  by *auto*
sepref-def *ema-get-value-impl*
  is ‹*RETURN o ema-get-value*›
  :: ‹$ema\text{-}assn^k \rightarrow_a word\text{-}assn$›
  unfolding *emag-get-value-alt-def*
  by *sepref*

sepref-register *isasat-length-trail-st*

sepref-def *isasat-length-trail-st-code*
  is ‹*RETURN o isasat-length-trail-st*›
  :: ‹$[isa\text{-}length\text{-}trail\text{-}pre\ o\ get\text{-}trail\text{-}wl\text{-}heur]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow sint64\text{-}nat\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *isasat-length-trail-st-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-register *get-pos-of-level-in-trail-imp-st*

sepref-def *get-pos-of-level-in-trail-imp-st-code*
  is ‹*uncurry get-pos-of-level-in-trail-imp-st*›
  :: ‹$isasat\text{-}bounded\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *get-pos-of-level-in-trail-imp-alt-def isasat-bounded-assn-def*
  apply (*rewrite* **in** - *eta-expand*[**where** *f* = *RETURN*])
  apply (*rewrite* **in** *RETURN* ¤ *annot-unat-snat-upcast*[**where** *'l=64*])
  by *sepref*

sepref-register *neq* : (*op-neq* :: *clause-status* ⇒ - ⇒ -)
lemma *status-neq-refine1*: (($\neq$),*op-neq*) ∈ *status-rel* → *status-rel* → *bool-rel*
  by (*auto simp*: *status-rel-def*)

sepref-def *status-neq-impl* **is** [] *uncurry* (*RETURN oo* ($\neq$))
  :: $(unat\text{-}assn'\ TYPE(32))^k *_a (unat\text{-}assn'\ TYPE(32))^k \rightarrow_a bool1\text{-}assn$
  by *sepref*

lemmas [*sepref-fr-rules*] = *status-neq-impl.refine*[*FCOMP status-neq-refine1*]

lemma *clause-not-marked-to-delete-heur-alt-def*:
  ‹*RETURN oo clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*.
      RETURN (*arena-status arena C* $\neq$ *DELETED*))›
  unfolding *clause-not-marked-to-delete-heur-def* **by** (*auto intro*!: *ext*)

sepref-def *clause-not-marked-to-delete-heur-fast-code*
  is ‹*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*)›
  :: ‹$[clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur\text{-}pre]_a\ isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow bool1\text{-}assn$›
  supply [[*goals-limit=1*]]
  unfolding *clause-not-marked-to-delete-heur-alt-def isasat-bounded-assn-def*
     clause-not-marked-to-delete-heur-pre-def
  by *sepref*

lemma *mop-clause-not-marked-to-delete-heur-alt-def*:

⟨*mop-clause-not-marked-to-delete-heur* = (λ(*M*, *arena*, *D*, *oth*) *C*. *do* {
ASSERT(*clause-not-marked-to-delete-heur-pre* ((*M*, *arena*, *D*, *oth*), *C*));
RETURN (*arena-status arena C* ≠ *DELETED*)
})⟩
unfolding *clause-not-marked-to-delete-heur-def mop-clause-not-marked-to-delete-heur-def*
by (*auto intro*!: *ext*)

sepref-def *mop-clause-not-marked-to-delete-heur-impl*
is ⟨*uncurry mop-clause-not-marked-to-delete-heur*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$⟩
unfolding *mop-clause-not-marked-to-delete-heur-alt-def*
clause-not-marked-to-delete-heur-pre-def prod.case isasat-bounded-assn-def
by *sepref*

sepref-def *delete-index-and-swap-code2*
is ⟨*uncurry* (*RETURN oo delete-index-and-swap*)⟩
:: ⟨$[\lambda(xs, i).\ i < length\ xs]_a$
$vdom\text{-}fast\text{-}assn^d *_a sint64\text{-}nat\text{-}assn^k \rightarrow vdom\text{-}fast\text{-}assn$⟩
unfolding *delete-index-and-swap.simps*
by *sepref*

sepref-def *mop-mark-garbage-heur-impl*
is ⟨*uncurry2 mop-mark-garbage-heur*⟩
:: ⟨$[\lambda((C, i), S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \le sint64\text{-}max]_a$
$sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *mop-mark-garbage-heur-alt-def*
clause-not-marked-to-delete-heur-pre-def prod.case isasat-bounded-assn-def
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

sepref-def *mop-mark-unused-st-heur-impl*
is ⟨*uncurry mop-mark-unused-st-heur*⟩
:: ⟨ $sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$⟩
unfolding *mop-mark-unused-st-heur-def*
by *sepref*

sepref-def *mop-arena-lbd-st-impl*
is ⟨*uncurry mop-arena-lbd-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *mop-arena-lbd-st-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-def *mop-arena-status-st-impl*
is ⟨*uncurry mop-arena-status-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a status\text{-}impl\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *mop-arena-status-st-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-def *mop-marked-as-used-st-impl*
is ⟨*uncurry mop-marked-as-used-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a bool1\text{-}assn$⟩
supply [[*goals-limit=1*]]

unfolding *mop-marked-as-used-st-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-def *mop-arena-length-st-impl*
is ‹*uncurry mop-arena-length-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *mop-arena-length-st-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-register *incr-wasted-st full-arena-length-st wasted-bytes-st*
sepref-def *incr-wasted-st-impl*
is ‹*uncurry* (*RETURN oo incr-wasted-st*)›
:: ‹$word64\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
supply[[*goals-limit=1*]]
unfolding *incr-wasted-st-def incr-wasted.simps*
isasat-bounded-assn-def heuristic-assn-def
by *sepref*

sepref-def *full-arena-length-st-impl*
is ‹*RETURN o full-arena-length-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
unfolding *full-arena-length-st-def isasat-bounded-assn-def*
by *sepref*

sepref-def *wasted-bytes-st-impl*
is ‹*RETURN o wasted-bytes-st*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a word64\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *isasat-bounded-assn-def*
heuristic-assn-def wasted-bytes-st-def
by *sepref*

lemma *set-zero-wasted-def*:
‹*set-zero-wasted* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *target*, *best*).
(*fast-ema*, *slow-ema*, *res-info*, *0*, φ, *target*, *best*))›
by (*auto intro*!: *ext*)

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted*›
:: ‹$heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$›
unfolding *heuristic-assn-def set-zero-wasted-def*
by *sepref*

lemma *mop-save-phase-heur-alt-def*:
‹*mop-save-phase-heur* = (λ *L b* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *target*, *best*). *do* {
ASSERT(*L* < *length* φ);
RETURN (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ[*L* := *b*], *target*,
best)})›
unfolding *mop-save-phase-heur-def save-phase-heur-def save-phase-heur-pre-def*
heuristic-assn-def
by (*auto intro*!: *ext*)

sepref-def *mop-save-phase-heur-impl*
is ‹*uncurry2* (*mop-save-phase-heur*)›

:: ⟨*atom-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*⟩
supply [[*goals-limit=1*]]
unfolding *mop-save-phase-heur-alt-def save-phase-heur-def save-phase-heur-pre-def heuristic-assn-def*
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *set-zero-wasted mop-save-phase-heur*

experiment begin

export-llvm
ema-update-impl
VMTF-Node-impl
VMTF-stamp-impl
VMTF-get-prev-impl
VMTF-get-next-impl
opts-restart-impl
opts-reduce-impl
opts-unbounded-mode-impl
get-conflict-wl-is-None-fast-code
isa-count-decided-st-fast-code
polarity-st-heur-pol-fast
count-decided-st-heur-pol-fast
access-lit-in-clauses-heur-fast-code
rewatch-heur-fast-code
rewatch-heur-st-fast-code
set-zero-wasted-impl

end

end
theory *IsaSAT-Inner-Propagation*
imports *IsaSAT-Setup*
IsaSAT-Clauses
begin

Chapter 9

Propagation: Inner Loop

declare *all-atms-def*[*symmetric*,*simp*]

9.1 Find replacement

lemma *literals-are-in-*$\mathcal{L}_{in}$*-nth2*:
  fixes C :: *nat*
  assumes *dom*: ‹$C \in\#$ *dom-m* (*get-clauses-wl* S)›
  shows ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* S) (*mset* (*get-clauses-wl* $S \propto C$))›
proof −
  let *?N* = ‹*get-clauses-wl* S›
  have ‹*?N* $\propto$ $C \in\#$ *ran-mf ?N*›
    using *dom* **by** (*auto simp*: *ran-m-def*)
  then have ‹*mset* (*?N* $\propto$ C) $\in\#$ *mset* ‘# (*ran-mf ?N*)›
    by *blast*
  from *all-lits-of-m-subset-all-lits-of-mmD*[*OF this*] **show** *?thesis*
    unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-in-*$\mathcal{L}_{in}$*-def literals-are-*$\mathcal{L}_{in}$*-def*
    by (*auto simp add*: *all-lits-of-mm-union all-lits-def* $\mathcal{L}_{all}$*-all-atms-all-lits*)
qed

definition *find-non-false-literal-between* **where**
  ‹*find-non-false-literal-between M a b C* =
    find-in-list-between (λL. *polarity M L* $\neq$ *Some False*) *a b C*›

definition *isa-find-unwatched-between*
 :: ‹- ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat option*) *nres*› **where**
‹*isa-find-unwatched-between P M′ NU a b C* = *do* {
  ASSERT($C+a \leq$ *length NU*);
  ASSERT($C+b \leq$ *length NU*);
  (x, -) ← $WHILE_T{}^{\lambda(found,\ i).\ True}$
    (λ(*found*, i). *found* = *None* $\wedge$ $i < C + b$)
    (λ(-, i). *do* {
      ASSERT($i < C$ + (*arena-length NU C*));
      ASSERT($i \geq C$);
      ASSERT($i < C + b$);
      ASSERT(*arena-lit-pre NU i*);
      L ← *mop-arena-lit NU i*;
      ASSERT(*polarity-pol-pre M′ L*);
      if P L then RETURN (*Some* ($i - C$), i) *else RETURN* (*None*, i+1)

})
    $(None,\ C+a)$;
  $RETURN\ x$
}
›

lemma *isa-find-unwatched-between-find-in-list-between-spec*:
  assumes ‹$a \le length\ (N \propto C)$› **and** ‹$b \le length\ (N \propto C)$› **and** ‹$a \le b$› **and**
  ‹*valid-arena arena N vdom*› **and** ‹$C \in\#\ dom\text{-}m\ N$› **and** *eq*: ‹$a' = a$› ‹$b' = b$› ‹$C' = C$› **and**
  ‹$\bigwedge L.\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow P'\ L = P\ L$› **and**
  $M'M$: ‹$(M',\ M) \in trail\text{-}pol\ \mathcal{A}$›
  assumes *lits*: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (N \propto C))$›
  shows
  ‹$isa\text{-}find\text{-}unwatched\text{-}between\ P'\ M'\ arena\ a'\ b'\ C' \le \Downarrow Id\ (find\text{-}in\text{-}list\text{-}between\ P\ a\ b\ (N \propto C))$›
proof −
  have *find-in-list-between-alt*:
    ‹$find\text{-}in\text{-}list\text{-}between\ P\ a\ b\ C = do$ {
     $(x,\ \text{-}) \leftarrow WHILE_T{}^{\lambda(found,\ i).\ i \ge a \wedge i \le length\ C \wedge i \le b \wedge (\forall j \in \{a..<i\}.\ \neg P\ (C!j)) \wedge}$ $(\forall j.\ found = Some\ j$
       $(\lambda(found,\ i).\ found = None \wedge i < b)$
       $(\lambda(\text{-},\ i).\ do$ {
         $ASSERT(i < length\ C)$;
         $let\ L = C!i$;
         $if\ P\ L\ then\ RETURN\ (Some\ i,\ i)\ else\ RETURN\ (None,\ i+1)$
       })
       $(None,\ a)$;
      $RETURN\ x$
    }› **for** $P\ a\ b\ c\ C$
   by (*auto simp*: *find-in-list-between-def*)
  have [*refine0*]: ‹$((None,\ x2m + a),\ None,\ a) \in \langle Id\rangle option\text{-}rel \times_r \{(n',\ n).\ n' = x2m + n\}$›
   for *x2m*
   by *auto*
  have [*simp*]: ‹$arena\text{-}lit\ arena\ (C + x2) \in\#\ \mathcal{L}_{all}\ \mathcal{A}$› **if** ‹$x2 < length\ (N \propto C)$› **for** *x2*
   using *that lits assms* **by** (*auto simp*: *arena-lifting*
     dest!: $literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}in\text{-}\mathcal{L}_{all}[of\ \mathcal{A}\ \text{-}\ x2]$)
  have *arena-lit-pre*: ‹*arena-lit-pre arena x2a*›
   if
    ‹$(x,\ x') \in \langle nat\text{-}rel\rangle option\text{-}rel \times_f \{(n',\ n).\ n' = C + n\}$› **and**
    ‹$case\ x\ of\ (found,\ i) \Rightarrow found = None \wedge i < C + b$› **and**
    ‹$case\ x'\ of\ (found,\ i) \Rightarrow found = None \wedge i < b$› **and**
    ‹$case\ x\ of\ (found,\ i) \Rightarrow True$› **and**
    ‹$case\ x'\ of$
    $(found,\ i) \Rightarrow$
     $a \le i \wedge$
     $i \le length\ (N \propto C) \wedge$
     $i \le b \wedge$
     $(\forall j \in \{a..<i\}.\ \neg\ P\ (N \propto C\ !\ j)) \wedge$
     $(\forall j.\ found = Some\ j \longrightarrow i = j \wedge P\ (N \propto C\ !\ j) \wedge j < b \wedge a \le j)$› **and**
    ‹$x' = (x1,\ x2)$› **and**
    ‹$x = (x1a,\ x2a)$› **and**
    ‹$x2 < length\ (N \propto C)$› **and**
    ‹$x2a < C + (arena\text{-}length\ arena\ C)$› **and**
    ‹$C \le x2a$›
   for $x\ x'\ x1\ x2\ x1a\ x2a$
  proof −
   show *?thesis*

unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
apply (*rule bex-leI*[*of - C*])
apply (*rule exI*[*of - N*])
apply (*rule exI*[*of - vdom*])
using *assms that* **by** *auto*
qed

show *?thesis*
unfolding *isa-find-unwatched-between-def find-in-list-between-alt eq*
apply (*refine-vcg mop-arena-lit*)
subgoal using *assms* **by** (*auto dest*!: *arena-lifting*(*10*))
subgoal using *assms* **by** (*auto dest*!: *arena-lifting*(*10*))
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by (*rule arena-lit-pre*)
apply (*rule assms*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal
by (*rule polarity-pol-pre*[*OF M'M*]) (*use assms* **in** ‹*auto simp*: *arena-lifting*›)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *isa-find-non-false-literal-between* **where**
‹*isa-find-non-false-literal-between M arena a b C* =
isa-find-unwatched-between (λL. *polarity-pol M L* $\neq$ *Some False*) *M arena a b C*›

definition *find-unwatched*
:: ‹(*nat literal* $\Rightarrow$ *bool*) $\Rightarrow$ (*nat*, *nat literal list* $\times$ *bool*) *fmap* $\Rightarrow$ *nat* $\Rightarrow$ (*nat option*) *nres*› **where**
‹*find-unwatched M N C* = *do* {
ASSERT($C \in\#$ *dom-m N*);
b $\leftarrow$ *SPEC*(λb::*bool*. *True*); — non-deterministic between full iteration (used in minisat), or starting in the middle (use in cadical)
if b then find-in-list-between M 2 (*length* ($N \propto C$)) ($N \propto C$)
else do {
pos $\leftarrow$ *SPEC* (λi. $i \leq$ *length* ($N \propto C$) $\wedge$ $i \geq 2$);
n $\leftarrow$ *find-in-list-between M pos* (*length* ($N \propto C$)) ($N \propto C$);
if n = *None then find-in-list-between M 2 pos* ($N \propto C$)
else RETURN n
}
}
›

definition *find-unwatched-wl-st-heur-pre* **where**
‹*find-unwatched-wl-st-heur-pre* =
(λ(*S*, *i*). *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *i*)›

definition *find-unwatched-wl-st'*

:: ⟨*nat twl-st-wl* ⇒ *nat* ⇒ *nat option nres*⟩ **where**
⟨*find-unwatched-wl-st'* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ) *i*. *do* {
find-unwatched (λ*L*. *polarity M L* ≠ *Some False*) *N i*
})⟩

definition *isa-find-unwatched*
:: ⟨(*nat literal* ⇒ *bool*) ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ (*nat option*) *nres*⟩
where
⟨*isa-find-unwatched P M' arena C* = *do* {
l ← *mop-arena-length arena C*;
b ← *RETURN*(*l* ≤ *MAX-LENGTH-SHORT-CLAUSE*);
if b then isa-find-unwatched-between P M' arena 2 l C
else do {
ASSERT(*get-saved-pos-pre arena C*);
pos ← *mop-arena-pos arena C*;
n ← *isa-find-unwatched-between P M' arena pos l C*;
if n = *None then isa-find-unwatched-between P M' arena 2 pos C*
else RETURN n
}
}
⟩

lemma *find-unwatched-alt-def*:
⟨*find-unwatched M N C* = *do* {
ASSERT(*C* ∈# *dom-m N*);
- ← *RETURN*(*length* (*N* ∝ *C*));
b ← *SPEC*(λ*b*::*bool*. *True*); — non-deterministic between full iteration (used in minisat), or starting in the middle (use in cadical)
if b then find-in-list-between M 2 (*length* (*N* ∝ *C*)) (*N* ∝ *C*)
else do {
pos ← *SPEC* (λ*i*. *i* ≤ *length* (*N* ∝ *C*) ∧ *i* ≥ *2*);
n ← *find-in-list-between M pos* (*length* (*N* ∝ *C*)) (*N* ∝ *C*);
if n = *None then find-in-list-between M 2 pos* (*N* ∝ *C*)
else RETURN n
}
}
⟩
unfolding *find-unwatched-def* **by** *auto*

lemma *isa-find-unwatched-find-unwatched*:
assumes *valid*: ⟨*valid-arena arena N vdom*⟩ **and**
⟨*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))⟩ **and**
ge2: ⟨*2* ≤ *length* (*N* ∝ *C*)⟩ **and**
M'M: ⟨(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$⟩
shows ⟨*isa-find-unwatched P M' arena C* ≤ ⇓ *Id* (*find-unwatched P N C*)⟩
proof −
have [*refine0*]:
⟨*C* ∈# *dom-m N* ⟹ (*l*, *l'*) ∈ {(*l*, *l'*). (*l*, *l'*) ∈ *nat-rel* ∧ *l'* = *length* (*N* ∝ *C*)} ⟹ *RETURN*(*l* ≤ *MAX-LENGTH-SHORT-CLAUSE*) ≤
⇓ {(*b*,*b'*). *b* = *b'* ∧ (*b* ⟷ *is-short-clause* (*N*∝*C*))}
(*SPEC* (λ-. *True*))⟩
for *l l'*
using *assms*

by (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def arena-lifting*)
have [*refine*]: ‹$C \in\# \ dom\text{-}m\ N \Longrightarrow mop\text{-}arena\text{-}length\ arena\ C \leq SPEC\ (\lambda c.\ (c,\ length\ (N \propto C)) \in \{(l,\ l').\ (l,\ l') \in nat\text{-}rel \wedge l' = length\ (N \propto C)\})$›
using *assms* **unfolding** *mop-arena-length-def*
by *refine-vcg* (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)
show *?thesis*
unfolding *isa-find-unwatched-def find-unwatched-alt-def*
apply (*refine-vcg isa-find-unwatched-between-find-in-list-between-spec*[*of* - - - - - *vdom* - - - $\mathcal{A}$ - -])
apply *assumption*
subgoal by *auto*
subgoal using *ge2* **.**
subgoal by *auto*
subgoal using *ge2* **.**
subgoal using *valid* **.**
subgoal by *fast*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
apply (*rule M'M*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **unfolding** *get-saved-pos-pre-def arena-is-valid-clause-idx-def*
by (*auto simp*: *arena-lifting*)
subgoal using *assms arena-lifting*[*OF valid*] **unfolding** *get-saved-pos-pre-def mop-arena-pos-def*
by (*auto simp*: *arena-lifting arena-pos-def*)
subgoal by (*auto simp*: *arena-pos-def*)
subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** *auto*
subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
apply (*rule M'M*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
subgoal by (*auto simp*: *arena-pos-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule M'M*)
subgoal using *assms* **by** *auto*
done
qed

definition *isa-find-unwatched-wl-st-heur*
:: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow nat \Rightarrow nat\ option\ nres$› **where**
‹$isa\text{-}find\text{-}unwatched\text{-}wl\text{-}st\text{-}heur = (\lambda(M,\ N,\ D,\ Q,\ W,\ vm,\ \varphi)\ i.\ do\ \{$
$isa\text{-}find\text{-}unwatched\ (\lambda L.\ polarity\text{-}pol\ M\ L \neq Some\ False)\ M\ N\ i$
$\})$›

```
lemma find-unwatched:
  assumes n-d: ‹no-dup M› and ‹length (N ∝ C) ≥ 2› and ‹literals-are-in-ℒin 𝒜 (mset (N ∝ C))›
  shows ‹find-unwatched (λL. polarity M L ≠ Some False) N C ≤ ⇓ Id (find-unwatched-l M N C)›
proof −
  have [refine0]: ‹find-in-list-between (λL. polarity M L ≠ Some False) 2 (length (N ∝ C)) (N ∝ C)
        ≤ SPEC
          (λfound.
              (found = None) = (∀ L∈set (unwatched-l (N ∝ C) ). − L ∈ lits-of-l M) ∧
              (∀ j. found = Some j ⟶
                    j < length (N ∝ C) ∧
                    (undefined-lit M ((N ∝ C) ! j) ∨ (N ∝ C) ! j ∈ lits-of-l M) ∧ 2 ≤ j))›
  proof −
    show ?thesis
      apply (rule order-trans)
      apply (rule find-in-list-between-spec)
      subgoal using assms by auto
      subgoal using assms by auto
      subgoal using assms by auto
      subgoal
        using n-d
        by (auto simp add: polarity-def in-set-drop-conv-nth Ball-def
          Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD)
      done
  qed
  have [refine0]: ‹find-in-list-between (λL. polarity M L ≠ Some False) xa (length (N ∝ C)) (N ∝ C)
      ≤ SPEC
        (λn. (if n = None
              then find-in-list-between (λL. polarity M L ≠ Some False) 2 xa (N ∝ C)
              else RETURN n)
              ≤ SPEC
                (λfound.
                    (found = None) =
                    (∀ L∈set (unwatched-l (N ∝ C)). − L ∈ lits-of-l M) ∧
                    (∀ j. found = Some j ⟶
                          j < length (N ∝ C) ∧
                          (undefined-lit M ((N ∝ C) ! j) ∨ (N ∝ C) ! j ∈ lits-of-l M) ∧
                          2 ≤ j)))›
    if
      ‹xa ≤ length (N ∝ C) ∧ 2 ≤ xa›
    for xa
  proof −
    show ?thesis
      apply (rule order-trans)
      apply (rule find-in-list-between-spec)
      subgoal using that by auto
      subgoal using assms by auto
      subgoal using that by auto
      subgoal
        apply (rule SPEC-rule)
        subgoal for x
          apply (cases ‹x = None›; simp only: if-True if-False refl)
        subgoal
          apply (rule order-trans)
          apply (rule find-in-list-between-spec)
          subgoal using that by auto
```

subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal
apply (*rule SPEC-rule*)
apply (*intro impI conjI iffI ballI*)
unfolding *in-set-drop-conv-nth Ball-def*
apply *normalize-goal*
subgoal for *x L xaa*
apply (*cases* ‹*xaa* $\geq$ *xa*›)
subgoal
using *n-d*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
subgoal
using *n-d*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
done
subgoal for *x*
using *n-d that assms*
apply (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*,
force)
by (*blast intro*: *dual-order.strict-trans1 dest*: *no-dup-consistentD*)
subgoal
using *n-d assms that*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib*
Decided-Propagated-in-iff-in-lits-of-l
split: *if-splits dest*: *no-dup-consistentD*)
done
done
subgoal
using *n-d that assms le-trans*
by (*auto simp add*: *polarity-def Ball-def all-conj-distrib in-set-drop-conv-nth*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
(*use le-trans no-dup-consistentD* **in** *blast*)+
done
done
done
qed

show *?thesis*
unfolding *find-unwatched-def find-unwatched-l-def*
apply (*refine-vcg*)
subgoal by *blast*
subgoal by *blast*
subgoal by *blast*
done
qed

definition *find-unwatched-wl-st-pre* **where**
‹*find-unwatched-wl-st-pre* = (λ(*S*, *i*).
i $\in\#$ *dom-m* (*get-clauses-wl S*) $\wedge$ *2* $\leq$ *length* (*get-clauses-wl S* $\propto$ *i*) $\wedge$
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* $\propto$ *i*))
)›

theorem *find-unwatched-wl-st-heur-find-unwatched-wl-s*:
$\langle$(*uncurry isa-find-unwatched-wl-st-heur*, *uncurry find-unwatched-wl-st'*)
$\in$ [*find-unwatched-wl-st-pre*]$_f$
twl-st-heur $\times_f$ *nat-rel* $\to$ $\langle$*Id*$\rangle$*nres-rel*$\rangle$
proof $-$
have [*refine0*]: $\langle((None, x2m + 2), None, 2) \in \langle Id\rangle option\text{-}rel \times_r \{(n', n).\ n' = x2m + n\}\rangle$
for *x2m*
by *auto*
have [*refine0*]:
$\langle$(*polarity M* (*arena-lit arena i'*), *polarity M'* ($N \propto C'$! *j*)) $\in \langle Id\rangle$*option-rel*$\rangle$
if $\langle\exists$ *vdom. valid-arena arena N vdom*$\rangle$ **and**
$\langle C' \in\#$ *dom-m N*$\rangle$ **and**
$\langle i' = C' + j \wedge j <$ *length* $(N \propto C')\rangle$ **and**
$\langle M = M'\rangle$
for *M arena i i' N j M' C'*
using *that* **by** (*auto simp*: *arena-lifting*)
have [*refine0*]: $\langle$*RETURN* (*arena-pos arena C*) $\leq \Downarrow \{(pos, pos').\ pos = pos' \wedge pos \geq 2 \wedge pos \leq length$ $(N \propto C)\}$
(*SPEC* ($\lambda i.\ i \leq length\ (N \propto C') \wedge 2 \leq i$))$\rangle$
if *valid*: $\langle$*valid-arena arena N vdom*$\rangle$ **and** *C*: $\langle C \in\#$ *dom-m N*$\rangle$ **and** $\langle C = C'\rangle$ **and**
$\langle$*is-long-clause* $(N \propto C')\rangle$
for *arena N vdom C C'*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff arena-pos-def*)
have [*refine0*]:
$\langle$*RETURN* (*arena-length arena C* $\leq$ *MAX-LENGTH-SHORT-CLAUSE*) $\leq \Downarrow \{(b, b').\ b = b' \wedge (b \longleftrightarrow is\text{-}short\text{-}clause\ (N \propto C))\}$
(*SPEC*(λ-. *True*))$\rangle$
if *valid*: $\langle$*valid-arena arena N vdom*$\rangle$ **and** *C*: $\langle C \in\#$ *dom-m N*$\rangle$
for *arena N vdom C*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def*)

have [*refine0*]:
$\langle C \in\#\ dom\text{-}m\ N \Longrightarrow (l, l') \in \{(l, l').\ (l, l') \in nat\text{-}rel \wedge l' = length\ (N \propto C)\} \Longrightarrow RETURN(l \leq MAX\text{-}LENGTH\text{-}SHORT\text{-}CLAUSE) \leq$
$\Downarrow \{(b,b').\ b = b' \wedge (b \longleftrightarrow is\text{-}short\text{-}clause\ (N \propto C))\}$
(*SPEC* (λ-. *True*))$\rangle$
for *l l' C N*
by (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def arena-lifting*)
have [*refine*]: $\langle C \in\#\ dom\text{-}m\ N \Longrightarrow valid\text{-}arena\ arena\ N\ vdom \Longrightarrow$
$mop\text{-}arena\text{-}length\ arena\ C \leq SPEC\ (\lambda c.\ (c, length\ (N \propto C)) \in \{(l, l').\ (l, l') \in nat\text{-}rel \wedge l' = length\ (N \propto C)\})\rangle$
for *N C arena vdom*
unfolding *mop-arena-length-def*
by *refine-vcg* (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)

have *H*: $\langle$*isa-find-unwatched P M' arena C* $\leq \Downarrow$ *Id* (*find-unwatched P' N C'*)$\rangle$
if $\langle$*valid-arena arena N vdom*$\rangle$
$\langle\bigwedge L.\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow P\ L = P'\ L\rangle$ **and**
$\langle C = C'\rangle$ **and**
$\langle 2 \leq length\ (N \propto C')\rangle$ **and** $\langle literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ (N \propto C'))\rangle$ **and**
$\langle(M', M) \in trail\text{-}pol\ \mathcal{A}\rangle$
for *arena P N C vdom P' C'* $\mathcal{A}$ *M' M*
using *that* **unfolding** *isa-find-unwatched-def find-unwatched-alt-def* **supply** [[*goals-limit=1*]]
apply (*refine-vcg isa-find-unwatched-between-find-in-list-between-spec*[*of - - - - - vdom*, **where** $\mathcal{A}$=$\mathcal{A}$])
unfolding *that* **apply** *assumption+*

subgoal by *simp*
subgoal by *auto*
subgoal using *that* **by** (*simp add*: *arena-lifting*)
subgoal using *that* **by** *auto*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting get-saved-pos-pre-def arena-is-valid-clause-idx-def*)
subgoal using *arena-lifting*[*OF* ‹*valid-arena arena N vdom*›] **unfolding** *get-saved-pos-pre-def mop-arena-pos-def*
by (*auto simp*: *arena-lifting arena-pos-def*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
subgoal using *that* **by** (*auto simp*: *arena-lifting*)
apply *assumption*
done

show *?thesis*
unfolding *isa-find-unwatched-wl-st-heur-def find-unwatched-wl-st′-def uncurry-def twl-st-heur-def find-unwatched-wl-st-pre-def*
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal for *x y*
apply (*case-tac x*, *case-tac y*)
by (*rule H*[**where** $\mathcal{A}3$ = ‹*all-atms-st* (*fst y*)›, *of* - - ‹*set* (*get-vdom* (*fst x*))›])
(*auto simp*: *polarity-pol-polarity*[*of* ‹*all-atms-st* (*fst y*)›,
unfolded option-rel-id-simp, *THEN fref-to-Down-unRET-uncurry-Id*]
all-atms-def[*symmetric*] *literals-are-in-*$\mathcal{L}_{in}$*-nth2*)
done
qed

definition *isa-save-pos* :: ‹*nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*isa-save-pos C i* = (λ(*M*, *N*, *oth*). *do* {
ASSERT(*arena-is-valid-clause-idx N C*);
if arena-length N C > *MAX-LENGTH-SHORT-CLAUSE then do* {
ASSERT(*isa-update-pos-pre* ((*C*, *i*), *N*));
RETURN (*M*, *arena-update-pos C i N*, *oth*)
} *else RETURN* (*M*, *N*, *oth*)
})
›

lemma *isa-save-pos-is-Id*:
assumes

⟨$(S, T) \in$ *twl-st-heur*⟩
⟨$C \in\#$ *dom-m* (*get-clauses-wl* T)⟩ **and**
⟨$i \leq$ *length* (*get-clauses-wl* $T \propto C$)⟩ **and**
⟨$i \geq 2$⟩
shows ⟨*isa-save-pos* C i $S \leq \Downarrow \{(S', T').\ (S', T') \in$ *twl-st-heur* $\wedge$ *length* (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* S) $\wedge$
get-watched-wl-heur S' = *get-watched-wl-heur* S $\wedge$ *get-vdom* S' = *get-vdom* $S\}$ (*RETURN* T)⟩
proof −
have ⟨*isa-update-pos-pre* ((C, i), *get-clauses-wl-heur* S)⟩ **if** ⟨*is-long-clause* (*get-clauses-wl* $T \propto C$)⟩
unfolding *isa-update-pos-pre-def*
using *assms that*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting)
then show *?thesis*
using *assms*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting
intro!: *valid-arena-update-pos ASSERT-leI*)
qed

9.2 Updates

definition *set-conflict-wl-heur-pre* **where**
⟨*set-conflict-wl-heur-pre* =
(λ(C, S). *True*)⟩

definition *set-conflict-wl-heur*
:: ⟨*nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*⟩
where
⟨*set-conflict-wl-heur* = (λC (M, N, D, Q, W, *vmtf*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*). *do* {
let $n = 0$;
ASSERT(*curry6 isa-set-lookup-conflict-aa-pre M N C D n lbd outl*);
(D, *clvls*, *lbd*, *outl*) ← *isa-set-lookup-conflict-aa M N C D n lbd outl*;
ASSERT(*isa-length-trail-pre M*);
ASSERT(*arena-act-pre N C*);
RETURN (M, *arena-incr-act N C*, D, *isa-length-trail M*, W, *vmtf*, *clvls*, *cach*, *lbd*, *outl*,
incr-conflict stats, *fema*, *sema*)})⟩

definition *update-clause-wl-code-pre* **where**
⟨*update-clause-wl-code-pre* = (λ((((((((L, C), b), j), w), i), f), S).
$w <$ *length* (*get-watched-wl-heur* S ! *nat-of-lit* L))⟩

definition *update-clause-wl-heur*
:: ⟨*nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *bool* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$
(*nat* $\times$ *nat* $\times$ *twl-st-wl-heur*) *nres*⟩
where
⟨*update-clause-wl-heur* = (λ(L::*nat literal*) C b j w i f (M, N, D, Q, W, *vm*). *do* {
K' ← *mop-arena-lit2'* (*set* (*get-vdom* (M, N, D, Q, W, *vm*))) N C f;
ASSERT($w <$ *length* N);
N' ← *mop-arena-swap* C i f N;
ASSERT(*nat-of-lit* $K' <$ *length* W);
ASSERT(*length* (W ! (*nat-of-lit* K')) $<$ *length* N);

let W = *W*[*nat-of-lit K*′:= *W* ! (*nat-of-lit K*′) @ [(*C*, *L*, *b*)]];
RETURN (*j*, *w+1*, (*M*, *N*′, *D*, *Q*, *W*, *vm*))
})›

definition *update-clause-wl-pre* **where**
‹*update-clause-wl-pre K r* = (λ(((((((*L*, *C*), *b*), *j*), *w*), *i*), *f*), *S*).
L = *K*)›
lemma *arena-lit-pre*:
‹*valid-arena NU N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *arena-lit-pre NU* (*C* + *i*)›
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of - C*], *rule exI*[*of - N*], *rule exI*[*of - vdom*]) *auto*

lemma *all-atms-swap*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *j* < *length* (*N* ∝ *C*) ⟹
all-atms (*N*(*C* ↪ *swap* (*N* ∝ *C*) *i j*)) = *all-atms N*›
unfolding *all-atms-def*
by (*auto simp del*: *all-atms-def*[*symmetric*] *simp*: *all-atms-def intro*!: *ext*)

lemma *mop-arena-swap*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C*′) ∈ *nat-rel*› ‹(*i*, *i*′) ∈ *nat-rel*› ‹(*j*, *j*′) ∈ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* ≤ ⇓{(*N*′′, *N*′). *valid-arena N*′′ *N*′ *vdom* ∧ *N*′′ = *swap-lits C*′ *i*′ *j*′ *arena*
∧ *N*′ = *op-clauses-swap N C*′ *i*′ *j*′ ∧ *all-atms N*′ = *all-atms N*} (*mop-clauses-swap N C*′ *i*′ *j*′)›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-swap-def swap-lits-pre-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits op-clauses-swap-def*)

lemma *update-clause-wl-alt-def*:
‹*update-clause-wl* = (λ(*L*::′*v literal*) *C b j w i f* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*). *do* {
ASSERT(*C* ∈# *dom-m N* ∧ *j* ≤ *w* ∧ *w* < *length* (*W L*) ∧ *correct-watching-except* (*Suc j*) (*Suc w*) *L* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
ASSERT(*L* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
K′ ← *mop-clauses-at N C f*;
ASSERT(*K*′ ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) ∧ *L* ≠ *K*′);
N′ ← *mop-clauses-swap N C i f*;
RETURN (*j*, *w+1*, (*M*, *N*′, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*(*K*′ := *W K*′ @ [(*C*, *L*, *b*)])))
})›
unfolding *update-clause-wl-def* **by** (*auto intro*!: *ext simp flip*: *all-lits-alt-def2*)

lemma *update-clause-wl-heur-update-clause-wl*:
‹(*uncurry7 update-clause-wl-heur*, *uncurry7* (*update-clause-wl*)) ∈
[*update-clause-wl-pre K r*]$_f$
Id $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up*′′ $\mathcal{D}$ *r s K* →
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up*′′ $\mathcal{D}$ *r s K*⟩*nres-rel*›
unfolding *update-clause-wl-heur-def update-clause-wl-alt-def uncurry-def*
update-clause-wl-pre-def all-lits-of-all-atms-of all-lits-of-all-atms-of
apply (*intro frefI nres-relI*, *case-tac x*, *case-tac y*)
apply (*refine-rcg*)
apply (*rule mop-arena-lit2*′)
subgoal by (*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur′*-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def

intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *bex-leI exI*)
subgoal by *auto*
subgoal by *auto*
subgoal by
(*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *bex-leI exI*)
apply (*rule-tac vdom=* ‹*set* (*get-vdom* ((λ((((((L,C),b),j),w),-),-),x). x) x))› **in** *mop-arena-swap*)
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto simp*: *twl-st-heur-def Let-def add-mset-eq-add-mset all-lits-of-all-atms-of ac-simps*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
dest: *multi-member-split simp flip*: *all-lits-def all-lits-alt-def2*
intro!: *ASSERT-refine-left valid-arena-swap-lits*)
subgoal for *x y a b c d e f g h i j k l m n p q ra t aa ba ca da ea fa ga ha ia*
ja x1 x1a x1b x1c x1d x1e x1f x2 x2a x2b x2c x2d x2e x2f x1g x2g x1h
x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x1p x1q x1r
x1s x1t x1u x2o x2p x2q x2r x2s x2t x2u x1v x2v x1w x2w x1x x2x x1y
x2y x1z x2z K' K'a N' K'a'
supply[[*goals-limit=1*]]
by (*auto dest*!: *length-watched-le2*[*of* - - - - *x2u* $\mathcal{D}$ *r K'a*])
(*simp-all add*: *twl-st-heur'-def twl-st-heur-def map-fun-rel-def ac-simps*)
subgoal
by
(*clarsimp simp*: *twl-st-heur-def Let-def*
map-fun-rel-def twl-st-heur'-def update-clause-wl-pre-def
op-clauses-swap-def)
done

definition *propagate-lit-wl-heur-pre* **where**
‹*propagate-lit-wl-heur-pre* =
(λ((L, C), S). $C \neq$ *DECISION-REASON*)›

definition *propagate-lit-wl-heur*
:: ‹*nat literal* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-heur* = ($\lambda L'$ C i (M, N, D, Q, W, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,

heur, *sema*). *do* {
  ASSERT($i \leq 1$);
  $M \leftarrow$ *cons-trail-Propagated-tr* L' C M;
  $N' \leftarrow$ *mop-arena-swap* C *0* (*1* $-$ i) N;
  let stats = *incr-propagation* (*if count-decided-pol* M = *0 then incr-uset stats else stats*);
  heur $\leftarrow$ *mop-save-phase-heur* (*atm-of* L') (*is-pos* L') *heur*;
  RETURN (M, N', D, Q, W, *vm*, *clvls*, *cach*, *lbd*, *outl*,
    stats, *heur*, *sema*)
})›

definition *propagate-lit-wl-pre* **where**
‹*propagate-lit-wl-pre* = (λ(((L, C), i), S).
  undefined-lit (*get-trail-wl* S) L $\wedge$ *get-conflict-wl* S = *None* $\wedge$
  C $\in\#$ *dom-m* (*get-clauses-wl* S) $\wedge$ L $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S) $\wedge$
  1 $-$ i < *length* (*get-clauses-wl* S $\propto$ C) $\wedge$
  0 < *length* (*get-clauses-wl* S $\propto$ C))›

lemma *isa-vmtf-consD*:
  assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *isa-vmtf* $\mathcal{A}$ M›
  shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) $\in$ *isa-vmtf* $\mathcal{A}$ (L # M)›
  using *vmtf-consD*[*of ns m fst-As lst-As next-search* - $\mathcal{A}$ M L] *assms*
  by (*auto simp*: *isa-vmtf-def*)

lemma *propagate-lit-wl-heur-propagate-lit-wl*:
  ‹(*uncurry3 propagate-lit-wl-heur*, *uncurry3* (*propagate-lit-wl*)) $\in$
  [λ-. *True*]$_f$
  Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\rightarrow$ $\langle$*twl-st-heur-up''* $\mathcal{D}$ *r s K*$\rangle$*nres-rel*›
  supply [[*goals-limit=1*]]
  unfolding *propagate-lit-wl-heur-def propagate-lit-wl-def Let-def*
  apply (*intro frefI nres-relI*) **unfolding** *uncurry-def mop-save-phase-heur-def*
    nres-monad3
  apply (*refine-rcg*)
  subgoal by *auto*
  apply (*rule-tac* $\mathcal{A}$ = ‹*all-atms-st* (*snd y*)› **in** *cons-trail-Propagated-tr2*)
  subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
      isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
      valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
    ac-simps
     intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
     dest: *multi-member-split valid-arena-DECISION-REASON*)
  subgoal
   by (*auto simp*: *twl-st-heur-def twl-st-heur'-def all-lits-def* $\mathcal{L}_{all}$*-all-atms-all-lits*
    ac-simps)
  subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
      isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
      valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
     intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
     dest: *multi-member-split valid-arena-DECISION-REASON*)
  apply (*rule-tac vdom* = ‹*set* (*get-vdom* (*snd x*))› **in** *mop-arena-swap*)
  subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
      isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
      valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
     intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
     dest: *multi-member-split valid-arena-DECISION-REASON*)
  subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*

isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def $\mathcal{L}_{all}$*-all-atms-all-lits*
all-lits-def ac-simps
intro!: *save-phase-heur-preI*)
subgoal for x y
by (*cases x*; *cases y*; *hypsubst*)
(*clarsimp simp add*: *twl-st-heur-def twl-st-heur′-def isa-vmtf-consD2*
op-clauses-swap-def ac-simps)
done

definition *propagate-lit-wl-bin-pre* **where**
‹*propagate-lit-wl-bin-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧
C ∈# *dom-m* (*get-clauses-wl S*) ∧ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›

definition *propagate-lit-wl-bin-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-lit-wl-bin-heur* = (λ*L′ C* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *sema*). *do* {
M ← *cons-trail-Propagated-tr L′ C M*;
let stats = *incr-propagation* (*if count-decided-pol M* = *0 then incr-uset stats else stats*);
heur ← *mop-save-phase-heur* (*atm-of L′*) (*is-pos L′*) *heur*;
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *sema*)
})›

lemma *propagate-lit-wl-bin-heur-propagate-lit-wl-bin*:
‹(*uncurry2 propagate-lit-wl-bin-heur*, *uncurry2* (*propagate-lit-wl-bin*)) ∈
[λ-. *True*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩*nres-rel*›
supply [[*goals-limit=1*]]
unfolding *propagate-lit-wl-bin-heur-def propagate-lit-wl-bin-def Let-def*
apply (*intro frefI nres-relI*) **unfolding** *uncurry-def mop-save-phase-heur-def nres-monad3*
apply (*refine-rcg*)
apply (*rule-tac* $\mathcal{A}$ = ‹*all-atms-st* (*snd y*)› **in** *cons-trail-Propagated-tr2*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-bin-heur-def propagate-lit-wl-bin-def*
isa-vmtf-consD twl-st-heur′-def propagate-lit-wl-bin-pre-def swap-lits-pre-def
arena-lifting phase-saving-def atms-of-def save-phase-def $\mathcal{L}_{all}$*-all-atms-all-lits*
all-lits-def ac-simps
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*

dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur′-def propagate-lit-wl-bin-pre-def swap-lits-pre-def arena-lifting phase-saving-def atms-of-def save-phase-def* $\mathcal{L}_{all}$*-all-atms-all-lits* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*
intro!: *save-phase-heur-preI*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur′-def propagate-lit-wl-bin-pre-def swap-lits-pre-def arena-lifting phase-saving-def atms-of-def save-phase-def* $\mathcal{L}_{all}$*-all-atms-all-lits all-lits-def* $\mathcal{L}_{all}$*-all-atms-all-lits* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm ac-simps*
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur′-def propagate-lit-wl-bin-pre-def swap-lits-pre-def arena-lifting phase-saving-def atms-of-def save-phase-def* $\mathcal{L}_{all}$*-all-atms-all-lits* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*
intro!: *save-phase-heur-preI*)
subgoal for *x y*
by (*cases x*; *cases y*; *hypsubst*)
(*clarsimp simp add*: *ac-simps twl-st-heur-def twl-st-heur′-def isa-vmtf-consD2 op-clauses-swap-def*)
done

definition *unit-prop-body-wl-heur-inv* **where**
‹*unit-prop-body-wl-heur-inv S j w L* ⟷
(∃ *S′*. (*S*, *S′*) ∈ *twl-st-heur* ∧ *unit-prop-body-wl-inv S′ j w L*)›

definition *unit-prop-body-wl-D-find-unwatched-heur-inv* **where**
‹*unit-prop-body-wl-D-find-unwatched-heur-inv f C S* ⟷
(∃ *S′*. (*S*, *S′*) ∈ *twl-st-heur* ∧ *unit-prop-body-wl-find-unwatched-inv f C S′*)›

definition *keep-watch-heur* **where**
‹*keep-watch-heur* = (λ*L i j* (*M*, *N*, *D*, *Q*, *W*, *vm*). *do* {
ASSERT(*nat-of-lit L* < *length W*);
ASSERT(*i* < *length* (*W* ! *nat-of-lit L*));
ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
RETURN (*M*, *N*, *D*, *Q*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*i* := *W* ! (*nat-of-lit L*) ! *j*]], *vm*)
})›

definition *update-blit-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *bool* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒
(*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where
‹*update-blit-wl-heur* = (λ(*L*::*nat literal*) *C b j w K* (*M*, *N*, *D*, *Q*, *W*, *vm*). *do* {
ASSERT(*nat-of-lit L* < *length W*);
ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
ASSERT(*j* < *length N*);
ASSERT(*w* < *length N*);
RETURN (*j+1*, *w+1*, (*M*, *N*, *D*, *Q*, *W*[*nat-of-lit L* := (*W*!*nat-of-lit L*)[*j*:= (*C*, *K*, *b*)]], *vm*))
})›

definition *pos-of-watched-heur* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*pos-of-watched-heur S C L* = *do* {
L′ ← *mop-access-lit-in-clauses-heur S C 0*;
RETURN (*if L* = *L′ then 0 else 1*)

} ›

lemma *pos-of-watched-alt*:
‹*pos-of-watched N C L* = *do* {
ASSERT(*length* (*N* ∝ *C*) > *0* ∧ *C* ∈# *dom-m N*);
let L′ = (*N* ∝ *C*) ! *0*;
RETURN (*if L′* = *L then 0 else 1*)
}›
unfolding *pos-of-watched-def Let-def* **by** *auto*

lemma *pos-of-watched-heur*:
‹(*S*, *S′*) ∈ {(*T*, *T′*). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T′*) ∈ *twl-st-heur-up′′ 𝒟 r s t*} ⟹
((*C*, *L*), (*C′*, *L′*)) ∈ *Id* $\times_r$ *Id* ⟹
pos-of-watched-heur S C L ≤ ⇓ *nat-rel* (*pos-of-watched* (*get-clauses-wl S′*) *C′ L′*)›
unfolding *pos-of-watched-heur-def pos-of-watched-alt mop-access-lit-in-clauses-heur-def*
by (*refine-rcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*)

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv0* **where**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* =
(λ(*j*, *w*, *S′*). ∃ *S*. (*S′*, *S*) ∈ *twl-st-heur* ∧ *unit-propagation-inner-loop-wl-loop-inv L* (*j*, *w*, *S*) ∧
length (*watched-by S L*) ≤ *length* (*get-clauses-wl-heur S′*) − *4*)›

definition *other-watched-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*›
where
‹*other-watched-wl-heur S L C i* = *do* {
ASSERT(*i* < *2* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C i* ∧
arena-lit (*get-clauses-wl-heur S*) (*C* + *i*) = *L* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C* (*1* − *i*));
mop-access-lit-in-clauses-heur S C (*1* − *i*)
}›

lemma *other-watched-heur*:
‹(*S*, *S′*) ∈ {(*T*, *T′*). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T′*) ∈ *twl-st-heur-up′′ 𝒟 r s t*} ⟹
((*L*, *C*, *i*), (*L′*, *C′*, *i′*)) ∈ *Id* $\times_r$ *Id* ⟹
other-watched-wl-heur S L C i ≤ ⇓ *Id* (*other-watched-wl S′ L′ C′ i′*)›
using *arena-lifting*(*5*,*7*)[*of* ‹*get-clauses-wl-heur S*› ‹*get-clauses-wl S′*› - *C i*]
unfolding *other-watched-wl-heur-def other-watched-wl-def*
mop-access-lit-in-clauses-heur-def
by (*refine-rcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*
arena-lit-pre2-def
intro!: *exI*[*of* - ‹*get-clauses-wl S′*›])

9.3 Full inner loop

definition *unit-propagation-inner-loop-body-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ (*nat* × *nat* × *twl-st-wl-heur*) *nres*›
where
‹*unit-propagation-inner-loop-body-wl-heur L j w* (*S0* :: *twl-st-wl-heur*) = *do* {
ASSERT(*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* (*j*, *w*, *S0*));
(*C*, *K*, *b*) ← *mop-watched-by-app-heur S0 L w*;
S ← *keep-watch-heur L j w S0*;
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
val-K ← *mop-polarity-st-heur S K*;
if val-K = *Some True*

```
    then RETURN (j+1, w+1, S)
    else do {
      if b then do {
        if val-K = Some False
        then do {
          S ← set-conflict-wl-heur C S;
          RETURN (j+1, w+1, S)}
        else do {
          S ← propagate-lit-wl-bin-heur K C S;
          RETURN (j+1, w+1, S)}
      }
      else do {
  — Now the costly operations:
  ASSERT(clause-not-marked-to-delete-heur-pre (S, C));
  if ¬clause-not-marked-to-delete-heur S C
  then RETURN (j, w+1, S)
  else do {
    i ← pos-of-watched-heur S C L;
          ASSERT(i ≤ 1);
    L′ ← other-watched-wl-heur S L C i;
    val-L′ ← mop-polarity-st-heur S L′;
    if val-L′ = Some True
    then update-blit-wl-heur L C b j w L′ S
    else do {
      f ← isa-find-unwatched-wl-st-heur S C;
      case f of
 None ⇒ do {
   if val-L′ = Some False
   then do {
     S ← set-conflict-wl-heur C S;
     RETURN (j+1, w+1, S)}
   else do {
     S ← propagate-lit-wl-heur L′ C i S;
     RETURN (j+1, w+1, S)}
 }
      | Some f ⇒ do {
   S ← isa-save-pos C f S;
   ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
   K ← mop-access-lit-in-clauses-heur S C f;
   val-L′ ← mop-polarity-st-heur S K;
   if val-L′ = Some True
   then update-blit-wl-heur L C b j w K S
   else do {
     update-clause-wl-heur L C b j w i f S
   }
      }
    }
        }
      }
    }
  }›
```

declare *RETURN-as-SPEC-refine*[*refine2 del*]

definition *set-conflict-wl′-pre* **where**

⟨set-conflict-wl′-pre i S ⟷
get-conflict-wl S = None ∧ i ∈# dom-m (get-clauses-wl S) ∧
literals-are-in-$\mathcal{L}_{in}$-mm (all-atms-st S) (mset '# ran-mf (get-clauses-wl S)) ∧
¬ tautology (mset (get-clauses-wl S ∝ i)) ∧
distinct (get-clauses-wl S ∝ i) ∧
literals-are-in-$\mathcal{L}_{in}$-trail (all-atms-st S) (get-trail-wl S)⟩

lemma literals-are-in-$\mathcal{L}_{in}$-mm-clauses[simp]: ⟨literals-are-in-$\mathcal{L}_{in}$-mm (all-atms-st S) (mset '# ran-mf (get-clauses-wl S))⟩
⟨literals-are-in-$\mathcal{L}_{in}$-mm (all-atms-st S) ((λx. mset (fst x)) '# ran-m (get-clauses-wl S))⟩
apply (auto simp: $\mathcal{L}_{all}$-all-atms-all-lits literals-are-in-$\mathcal{L}_{in}$-mm-def)
apply (auto simp: all-lits-def all-lits-of-mm-union)
done

lemma set-conflict-wl-alt-def:
⟨set-conflict-wl = (λC (M, N, D, NE, UE, NS, US, Q, W). do {
ASSERT(set-conflict-wl-pre C (M, N, D, NE, UE, NS, US, Q, W));
let D = Some (mset (N ∝ C));
RETURN (M, N, D, NE, UE, NS, US, {#}, W)
})⟩
unfolding set-conflict-wl-def Let-def **by** (auto simp: ac-simps)

lemma set-conflict-wl-pre-set-conflict-wl′-pre:
assumes ⟨set-conflict-wl-pre C S⟩
shows ⟨set-conflict-wl′-pre C S⟩
proof −
obtain S′ T b b′ **where**
SS′: ⟨(S, S′) ∈ state-wl-l b⟩ **and**
⟨blits-in-$\mathcal{L}_{in}$ S⟩ **and**
confl: ⟨get-conflict-l S′= None⟩ **and**
dom: ⟨C ∈# dom-m (get-clauses-l S′)⟩ **and**
tauto: ⟨¬ tautology (mset (get-clauses-l S′ ∝ C))⟩ **and**
dist: ⟨distinct (get-clauses-l S′ ∝ C)⟩ **and**
⟨get-trail-l S′ ⊨as CNot (mset (get-clauses-l S′ ∝ C))⟩ **and**
T: ⟨(set-clauses-to-update-l (clauses-to-update-l S′ + {#C#}) S′, T)
∈ twl-st-l b′⟩ **and**
struct: ⟨twl-struct-invs T⟩ **and**
⟨twl-stgy-invs T⟩
using assms
unfolding set-conflict-wl-pre-def set-conflict-l-pre-def **apply** −
by blast
have
alien: ⟨$cdcl_W$-restart-mset.no-strange-atm ($state_W$-of T)⟩
using struct **unfolding** twl-struct-invs-def $cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv-def
by fast+

have lits-trail: ⟨atm-of ' lits-of-l (get-trail T) ⊆ atms-of-mm (clause '# get-clauses T + unit-clss T +
subsumed-clauses T)⟩
using alien **unfolding** $cdcl_W$-restart-mset.no-strange-atm-def
by (cases T) (auto
simp del: all-clss-l-ran-m union-filter-mset-complement
simp: twl-st twl-st-l twl-st-wl all-lits-of-mm-union lits-of-def
convert-lits-l-def image-image in-all-lits-of-mm-ain-atms-of-iff
get-unit-clauses-wl-alt-def image-subset-iff)
moreover have ⟨atms-of-mm (clause '# get-clauses T + unit-clss T +
subsumed-clauses T) = set-mset (all-atms-st S)⟩

using SS' T **unfolding** *all-atms-st-alt-def all-lits-def*
by (*auto simp*: *mset-take-mset-drop-mset′ twl-st-l atm-of-all-lits-of-mm*)

ultimately show *?thesis*
using SS' *T dom tauto dist confl* **unfolding** *set-conflict-wl′-pre-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-atm-of twl-st-l*
mset-take-mset-drop-mset′ simp del: *all-atms-def*[*symmetric*])
qed

lemma *set-conflict-wl-heur-set-conflict-wl′*:
‹(*uncurry set-conflict-wl-heur*, *uncurry* (*set-conflict-wl*)) ∈
$[\lambda$-. *True*$]_f$
nat-rel $\times_r$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩*nres-rel*›
proof −
have *H*:
‹*isa-set-lookup-conflict-aa x y z a b c d*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ $\times_f$ (*nat-rel* $\times_f$ (*Id* $\times_f$ *Id*)))
(*set-conflict-m* x' y' z' a' b' c' d')›
if
‹(((((((x, y), z), a), b), c), d), (((((x', y'), z'), a'), b'), c'), d')
∈ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$
nat-rel $\times_f$
option-lookup-clause-rel $\mathcal{A}$ $\times_f$
nat-rel $\times_f$
Id $\times_f$
Id› **and**
‹z' ∈# *dom-m* y' ∧ a' = *None* ∧ *distinct* (y' ∝ z') ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* y') ∧
¬ *tautology* (*mset* (y' ∝ z')) ∧ b' = *0* ∧ *out-learned* x' *None* d' ∧
isasat-input-bounded $\mathcal{A}$›
for *x* x' *y* y' *z* z' *a* a' *b* b' *c* c' *d* d' *vdom* $\mathcal{A}$
by (*rule isa-set-lookup-conflict*[*THEN fref-to-Down-curry6*,
unfolded prod.case, *OF that*(*2*,*1*)])
have [*refine0*]: ‹*isa-set-lookup-conflict-aa x1h x1i x1g x1j 0 x1q x1r*
≤ ⇓ {((*C*, *n*, *lbd*, *outl*), *D*). (*C*, *D*) ∈ *option-lookup-clause-rel* (*all-atms-st x2*) ∧
n = *card-max-lvl x1a* (*the D*) ∧ *out-learned x1a D outl*}
(*RETURN* (*Some* (*mset* (*x1b* ∝ *x1*))))›
if
‹(*x*, *y*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*› **and**
‹*x2e* = (*x1f*, *x2f*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x2s* = (*x1t*, *x2t*)› **and**
‹*x2r* = (*x1s*, *x2s*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*x2p* = (*x1q*, *x2q*)› **and**
‹*x2n* = (*x1o*, *x2p*)› **and**
‹*x2m* = (*x1n*, *x2n*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*x2k* = (*x1l*, *x2l*)› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x2i* = (*x1j*, *x2j*)› **and**

‹*x2h* = (*x1i*, *x2i*)› **and**
‹*x2g* = (*x1h*, *x2h*)› **and**
‹*x* = (*x1g*, *x2g*)› **and**
‹*case y of* (*x*, *xa*) ⇒ *set-conflict-wl′-pre x xa*›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t*
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule H*[*of - - - - - - - x1a x1b x1g x1c 0 x1q x1r* ‹*all-atms-st x2*› ‹*set* (*get-vdom* (*snd x*))›])
subgoal
using *that*
by (*auto simp*: *twl-st-heur′-def twl-st-heur-def ac-simps*)
subgoal
using *that* **apply** *auto*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur′-def twl-st-heur-def set-conflict-wl′-pre-def ac-simps*)
subgoal
using *that*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur′-def twl-st-heur-def*)
done
qed
have *isa-set-lookup-conflict-aa-pre*:
‹*curry6 isa-set-lookup-conflict-aa-pre x1h x1i x1g x1j 0 x1q x1r*›
if
‹*case y of* (*x*, *xa*) ⇒ *set-conflict-wl′-pre x xa*› **and**
‹(*x*, *y*) ∈ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*› **and**
‹*x2e* = (*x1f*, *x2f*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x2s* = (*x1t*, *x2t*)› **and**
‹*x2r* = (*x1s*, *x2s*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*x2p* = (*x1q*, *x2q*)› **and**
‹*x2n* = (*x1o*, *x2p*)› **and**
‹*x2m* = (*x1n*, *x2n*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*x2k* = (*x1l*, *x2l*)› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x2i* = (*x1j*, *x2j*)› **and**
‹*x2h* = (*x1i*, *x2i*)› **and**
‹*x2g* = (*x1h*, *x2h*)› **and**
‹*x* = (*x1g*, *x2g*)›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t*
proof −
show *?thesis*
using *that* **unfolding** *isa-set-lookup-conflict-aa-pre-def set-conflict-wl′-pre-def*

twl-st-heur′-def twl-st-heur-def
by (*auto simp*: *arena-lifting*)
qed

show *?thesis*
supply [[*goals-limit=1*]]
apply (*intro nres-relI frefI*)
unfolding *uncurry-def RES-RETURN-RES4 set-conflict-wl-alt-def set-conflict-wl-heur-def*
apply (*rewrite at* ‹*let - = 0 in -*› *Let-def*)
apply (*refine-vcg*)
subgoal by (*rule isa-set-lookup-conflict-aa-pre*) (*auto dest*!: *set-conflict-wl-pre-set-conflict-wl′-pre*)
apply *assumption+*
subgoal by (*auto dest*!: *set-conflict-wl-pre-set-conflict-wl′-pre*)
subgoal for *x y*
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
by (*rule isa-length-trail-pre*)
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*)
subgoal for *x y*
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl* (*snd y*)›], *rule exI*[*of -* ‹*set* (*get-vdom* (*snd x*))›])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def set-conflict-wl′-pre-def dest*!: *set-conflict-wl-pre-set-conflict-wl′-pre*)
subgoal
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*])
(*auto simp*: *twl-st-heur′-def twl-st-heur-def counts-maximum-level-def ac-simps*
set-conflict-wl′-pre-def all-atms-def[*symmetric*] *dest*!: *set-conflict-wl-pre-set-conflict-wl′-pre*
intro!: *valid-arena-arena-incr-act valid-arena-mark-used*)
done
qed

lemma *in-Id-in-Id-option-rel*[*refine*]:
‹$(f, f') \in Id \Longrightarrow (f, f') \in \langle Id\rangle$ *option-rel*›
by *auto*

The assumption that that accessed clause is active has not been checked at this point!

definition *keep-watch-heur-pre* **where**
‹*keep-watch-heur-pre* =
(λ(((*L*, *j*), *w*), *S*).
$L \in\# \mathcal{L}_{all}$ (*all-atms-st S*))›

lemma *vdom-m-update-subset′*:
‹*fst C* ∈ *vdom-m* $\mathcal{A}$ *bh N* ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *C*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *vdom-m-update-subset*:
‹*bg* < *length* (*bh ap*) ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *bh ap* ! *bg*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *keep-watch-heur-keep-watch*:
‹(*uncurry3 keep-watch-heur*, *uncurry3* (*mop-keep-watch*)) ∈
$[\lambda\text{-}.\ True]_f$
Id $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ *r s K* → ⟨*twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩ *nres-rel*›
unfolding *keep-watch-heur-def mop-keep-watch-def uncurry-def*
$\mathcal{L}_{all}$*-all-atms-all-lits*[*symmetric*]

apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
done

This is a slightly stronger version of the previous lemma:

lemma *keep-watch-heur-keep-watch′*:
‹$((((L', j'), w'), S'), ((L, j), w), S)$
$\in$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up″* $\mathcal{D}$ r s $K \Longrightarrow$
keep-watch-heur L' j' w' $S' \le \Downarrow \{(T, T').$ *get-vdom* T = *get-vdom* $S' \wedge$
$(T, T') \in$ *twl-st-heur-up″* $\mathcal{D}$ r s $K\}$
(*mop-keep-watch* L j w S)›
unfolding *keep-watch-heur-def mop-keep-watch-def uncurry-def*
$\mathcal{L}_{all}$-*all-atms-all-lits*[*symmetric*]
apply *refine-rcg*
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def twl-st-heur-def map-fun-rel-def all-atms-def*[*symmetric*] *mop-keep-watch-def keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
done

definition *update-blit-wl-heur-pre* **where**
‹*update-blit-wl-heur-pre r K′* = (λ((((((L, C), b), j), w), K), S). L = K′)›

lemma *update-blit-wl-heur-update-blit-wl*:
‹(*uncurry6 update-blit-wl-heur*, *uncurry6 update-blit-wl*) ∈
[*update-blit-wl-heur-pre r K*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *Id* $\times_f$
twl-st-heur-up″ $\mathcal{D}$ *r s K*→
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up″* $\mathcal{D}$ *r s K*⟩ *nres-rel*›
apply (*intro frefI nres-relI*) — TODO proof
apply (*auto simp*: *update-blit-wl-heur-def update-blit-wl-def twl-st-heur′-def keep-watch-heur-pre-def*
twl-st-heur-def map-fun-rel-def update-blit-wl-heur-pre-def all-atms-def[*symmetric*]
$\mathcal{L}_{all}$*-all-atms-all-lits*
simp flip: *all-lits-alt-def2*
intro!: *ASSERT-leI ASSERT-refine-right*
simp: *vdom-m-update-subset*)
subgoal for *aa ab ac ad ae be af ag ah bf aj ak al am an bg ao bh ap aq ar bi at bl*
bm bn bo bp bq br bs bt bu bv bw bx - - - - - - - - by bz ca cb ci cj ck cl cm cn co
cq cr cs ct cv y x
apply (*subgoal-tac* ‹*vdom-m* (*all-atms co* (*cq* + *cr* + *cs* + *ct*))
(*cv*(*K* := (*cv K*)[*ck* := (*ci*, *cm*, *cj*)])) *co* ⊆
vdom-m (*all-atms co* (*cq* + *cr* + *cs* + *ct*)) *cv co*›)
apply *fast*
apply (*rule vdom-m-update-subset′*)
apply *auto*
done
subgoal for *aa ab ac ad ae be af ag ah bf ai aj ak al am an bg ao bh ap aq ar bi at*
bl bm bn bo bp bq br bs bt bu bv bw bx - - - - - - - - by bz ca cb ci cj ck cl cm cn
co cp cq cr cs ct cv x
apply (*subgoal-tac* ‹*vdom-m* (*all-atms co* (*cq* + *cr* + *cs* + *ct*))
(*cv*(*K* := (*cv K*)[*ck* := (*ci*, *cm*, *cj*)])) *co* ⊆
vdom-m (*all-atms co* (*cq* + *cr* + *cs* + *ct*)) *cv co*›)
apply *fast*
apply (*rule vdom-m-update-subset′*)
apply *auto*
done
done

lemma *mop-access-lit-in-clauses-heur*:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ (*i*, *i′*) ∈ *Id* ⟹ (*j*, *j′*) ∈ *Id* ⟹ *mop-access-lit-in-clauses-heur S i j*
≤ ⇓ *Id*
(*mop-clauses-at* (*get-clauses-wl T*) *i′ j′*)›
unfolding *mop-access-lit-in-clauses-heur-def*
by (*rule mop-arena-lit2*[**where** *vdom*=‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur-def intro*!: *mop-arena-lit2*)

lemma *isa-find-unwatched-wl-st-heur-find-unwatched-wl-st*:
‹*isa-find-unwatched-wl-st-heur x′ y′*
≤ ⇓ *Id* (*find-unwatched-l* (*get-trail-wl x*) (*get-clauses-wl x*) *y*)›
if
xy: ‹((*x′*, *y′*), *x*, *y*) ∈ *twl-st-heur* $\times_f$ *nat-rel*›
for *x y x′ y′*
proof −
have *find-unwatched-l-alt-def*: ‹*find-unwatched-l M N C* = *do* {
ASSERT(*C* ∈# *dom-m N* ∧ *length* (*N* ∝ *C*) ≥ *2* ∧ *distinct* (*N* ∝ *C*) ∧ *no-dup M*);

find-unwatched-l M N C
}⟩ **for** *M N C*
unfolding *find-unwatched-l-def* **by** (*auto simp*: *summarize-ASSERT-conv*)
have *K*: ⟨*find-unwatched-wl-st′ x y* ≤ *find-unwatched-l* (*get-trail-wl x*) (*get-clauses-wl x*) *y*⟩
unfolding *find-unwatched-wl-st′-def*
apply (*subst find-unwatched-l-alt-def*)
unfolding *le-ASSERT-iff*
apply (*cases x*)
apply *clarify*
apply (*rule order-trans*)
apply (*rule find-unwatched*[*of - - - ⟨all-atms-st x⟩*])
subgoal
by *simp*
subgoal
by *auto*
subgoal
using *literals-are-in-*$\mathcal{L}_{in}$*-nth2*[*of y x*]
by *simp*
subgoal by *auto*
done
show *?thesis*
apply (*subst find-unwatched-l-alt-def*)
apply (*intro ASSERT-refine-right*)
apply (*rule order-trans*)
apply (*rule find-unwatched-wl-st-heur-find-unwatched-wl-s*[*THEN fref-to-Down-curry*, *OF - that*(*1*)])
by (*simp-all add*: *K find-unwatched-wl-st-pre-def literals-are-in-*$\mathcal{L}_{in}$*-nth2*)
qed

lemma *unit-propagation-inner-loop-body-wl-alt-def*:
⟨*unit-propagation-inner-loop-body-wl L j w S* = *do* {
ASSERT(*unit-propagation-inner-loop-wl-loop-pre L* (*j*, *w*, *S*));
(*C*, *K*, *b*) ← *mop-watched-by-at S L w*;
S ← *mop-keep-watch L j w S*;
ASSERT(*is-nondeleted-clause-pre C L S*);
val-K ← *mop-polarity-wl S K*;
if val-K = *Some True*
then RETURN (*j+1*, *w+1*, *S*)
else do {
if b then do {
ASSERT(*propagate-proper-bin-case L K S C*);
if val-K = *Some False*
then do {*S* ← *set-conflict-wl C S*;
RETURN (*j+1*, *w+1*, *S*)}
else do {
S ← *propagate-lit-wl-bin K C S*;
RETURN (*j+1*, *w+1*, *S*)}
} — Now the costly operations:
else if C ∉# *dom-m* (*get-clauses-wl S*)
then RETURN (*j*, *w+1*, *S*)
else do {
ASSERT(*unit-prop-body-wl-inv S j w L*);
i ← *pos-of-watched* (*get-clauses-wl S*) *C L*;
ASSERT(*i* ≤ *1*);
L′ ← *other-watched-wl S L C i*;
val-L′ ← *mop-polarity-wl S L′*;

if val-L′ = Some True
then update-blit-wl L C b j w L′ S
else do {
f ← *find-unwatched-l* (*get-trail-wl S*) (*get-clauses-wl S*) *C*;
ASSERT (*unit-prop-body-wl-find-unwatched-inv f C S*);
case f of
None ⇒ *do* {
if val-L′ = Some False
then do {*S* ← *set-conflict-wl C S*;
RETURN (*j+1*, *w+1*, *S*)}
else do {*S* ← *propagate-lit-wl L′ C i S*; *RETURN* (*j+1*, *w+1*, *S*)}
}
| *Some f* ⇒ *do* {
ASSERT($C \in\#$ *dom-m* (*get-clauses-wl S*) ∧ *f* < *length* (*get-clauses-wl S* ∝ *C*) ∧ *f* ≥ *2*);
let S = S; — position saving
K ← *mop-clauses-at* (*get-clauses-wl S*) *C f*;
val-L′ ← *mop-polarity-wl S K*;
if val-L′ = Some True
then update-blit-wl L C b j w K S
else update-clause-wl L C b j w i f S
}
}
}
}
})›
unfolding *unit-propagation-inner-loop-body-wl-def Let-def* **by** *auto*

lemma *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D*:
‹(*uncurry3 unit-propagation-inner-loop-body-wl-heur*,
uncurry3 unit-propagation-inner-loop-body-wl)
∈ [λ(((*L*, *i*), *j*), *S*). *length* (*watched-by S L*) ≤ *r* − *4* ∧ *L* = *K* ∧
length (*watched-by S L*) = *s*]$_f$
nat-lit-lit-rel $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up′′* $\mathcal{D}$ *r s K* →
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up′′* $\mathcal{D}$ *r s K*⟩*nres-rel*›
proof −

have [*refine*]: ‹*clause-not-marked-to-delete-heur-pre* (*S′*, *C′*)›
if ‹*is-nondeleted-clause-pre C L S*› **and** ‹((*C′*, *S′*), (*C*, *S*)) ∈ *nat-rel* $\times_r$ *twl-st-heur*› **for** *C C′ S S′ L*
unfolding *clause-not-marked-to-delete-heur-pre-def prod.case arena-is-valid-clause-vdom-def*
by (*rule exI*[*of* - ‹*get-clauses-wl S*›], *rule exI*[*of* - ‹*set* (*get-vdom S′*)›])
(*use that* **in** ‹*force simp*: *is-nondeleted-clause-pre-def twl-st-heur-def vdom-m-def*
$\mathcal{L}_{all}$*-all-atms-all-lits dest*!: *multi-member-split*[*of L*]›)

note [*refine*] = *mop-watched-by-app-heur-mop-watched-by-at′′*[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry2*]
keep-watch-heur-keep-watch′[*of* - - - - - - - - $\mathcal{D}$ *r K s*]
mop-polarity-st-heur-mop-polarity-wl′′[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry*, *unfolded comp-def*]
set-conflict-wl-heur-set-conflict-wl′[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry*, *unfolded comp-def*]
propagate-lit-wl-bin-heur-propagate-lit-wl-bin
[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry2*, *unfolded comp-def*]
pos-of-watched-heur[*of* - - - $\mathcal{D}$ *r K s*]
mop-access-lit-in-clauses-heur
update-blit-wl-heur-update-blit-wl[*of r K* $\mathcal{D}$ *s*, *THEN fref-to-Down-curry6*]
isa-find-unwatched-wl-st-heur-find-unwatched-wl-st
propagate-lit-wl-heur-propagate-lit-wl[*of* $\mathcal{D}$ *r K s*, *THEN fref-to-Down-curry3*, *unfolded comp-def*]
isa-save-pos-is-Id

update-clause-wl-heur-update-clause-wl[*of* K r $\mathcal{D}$ s, *THEN fref-to-Down-curry7*]
other-watched-heur[*of* - - - $\mathcal{D}$ r K s]

have [*simp*]: ⟨*is-nondeleted-clause-pre x1f x1b Sa* ⟹
clause-not-marked-to-delete-pre (*Sa*, *x1f*)⟩ **for** *x1f x1b Sa*
unfolding *is-nondeleted-clause-pre-def clause-not-marked-to-delete-pre-def vdom-m-def*
$\mathcal{L}_{all}$*-all-atms-all-lits* **by** (*cases Sa*; *auto dest*!: *multi-member-split*)

show *?thesis*
supply [[*goals-limit=1*]] *twl-st-heur'-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
unfolding *unit-propagation-inner-loop-body-wl-heur-def*
unit-propagation-inner-loop-body-wl-alt-def
uncurry-def clause-not-marked-to-delete-def[*symmetric*]
watched-by-app-heur-def access-lit-in-clauses-heur-def

apply (*refine-rcg*)
subgoal unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def twl-st-heur'-def*
unit-propagation-inner-loop-wl-loop-pre-def
by *fastforce*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
apply *assumption*
subgoal by *auto*
subgoal
unfolding *Not-eq-iff*
by (*rule clause-not-marked-to-delete-rel*[*THEN fref-to-Down-unRET-Id-uncurry*])
(*simp-all add*: *clause-not-marked-to-delete-rel*[*THEN fref-to-Down-unRET-Id-uncurry*])
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal
unfolding *update-blit-wl-heur-pre-def unit-propagation-inner-loop-wl-loop-D-heur-inv0-def*
prod.case unit-propagation-inner-loop-wl-loop-pre-def
by *normalize-goal+ simp*

subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *force*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by (*simp add*: *clause-not-marked-to-delete-def*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by (*simp add*: *update-blit-wl-heur-pre-def*)
subgoal by *simp*
subgoal by (*simp add*: *update-clause-wl-pre-def*)
subgoal by *simp*
done
qed

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv* **where**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv* S_0 L =
(λ(j, w, S'). $\exists S_0{}'$ S. (S_0, $S_0{}'$) $\in$ *twl-st-heur* $\wedge$ (S', S) $\in$ *twl-st-heur* $\wedge$ *unit-propagation-inner-loop-wl-loop-inv* L (j, w, S) $\wedge$
$L \in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S) $\wedge$ *dom-m* (*get-clauses-wl* S) = *dom-m* (*get-clauses-wl* $S_0{}'$) $\wedge$
length (*get-clauses-wl-heur* S_0) = *length* (*get-clauses-wl-heur* S')))›

definition *mop-length-watched-by-int* :: ‹*twl-st-wl-heur* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat nres*› **where**
‹*mop-length-watched-by-int* S L = *do* {
ASSERT(*nat-of-lit* L < *length* (*get-watched-wl-heur* S));
RETURN (*length* (*watched-by-int* S L))
}›

lemma *mop-length-watched-by-int-alt-def*:
‹*mop-length-watched-by-int* = (λ(M, N, D, Q, W, -) L. *do* {
ASSERT(*nat-of-lit* L < *length* (W));
RETURN (*length* (W ! *nat-of-lit* L))
})›
unfolding *mop-length-watched-by-int-def* **by** (*auto intro*!: *ext*)

definition *unit-propagation-inner-loop-wl-loop-D-heur*
:: ‹*nat literal* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ (*nat* $\times$ *nat* $\times$ *twl-st-wl-heur*) *nres*›
where
‹*unit-propagation-inner-loop-wl-loop-D-heur* L S_0 = *do* {
ASSERT(*length* (*watched-by-int* S_0 L) $\leq$ *length* (*get-clauses-wl-heur* S_0));
$n \leftarrow$ *mop-length-watched-by-int* S_0 L;
$WHILE_T{}^{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv\ S_0\ L}$
(λ(j, w, S). $w < n$ $\wedge$ *get-conflict-wl-is-None-heur* S)
(λ(j, w, S). *do* {

unit-propagation-inner-loop-body-wl-heur L j w S
})
$(0, 0, S_0)$
}›

lemma *unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D*:
‹(*uncurry unit-propagation-inner-loop-wl-loop-D-heur*,
uncurry unit-propagation-inner-loop-wl-loop)
$\in [\lambda(L, S).$ *length* (*watched-by S L*) $\leq r - 4 \wedge L = K \wedge$ *length* (*watched-by S L*) $= s \wedge$
length (*watched-by S L*) $\leq r]_f$
nat-lit-lit-rel $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\rightarrow$
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*⟩*nres-rel*›
proof −
have *unit-propagation-inner-loop-wl-loop-D-heur-inv*:
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv x2a x1a xa*›
if
‹$(x, y) \in$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*› **and**
‹$y = (x1, x2)$› **and**
‹$x = (x1a, x2a)$› **and**
‹$(xa, x') \in$ *nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K*› **and**
H: ‹*unit-propagation-inner-loop-wl-loop-inv x1 x'*›
for *x y x1 x2 x1a x2a xa x'*
proof −
obtain *w S w' S' j j'* **where**
xa: ‹$xa = (j, w, S)$› **and** *x'*: ‹$x' = (j', w', S')$›
by (*cases xa*; *cases x'*) *auto*
show *?thesis*
unfolding *xa unit-propagation-inner-loop-wl-loop-D-heur-inv-def prod.case*
apply (*rule exI*[*of - x2*])
apply (*rule exI*[*of - S'*])
using *that xa x' that* **apply** −
unfolding *prod.case* **apply** *hypsubst*
apply (*auto simp*: $\mathcal{L}_{all}$*-all-atms-all-lits all-lits-def twl-st-heur'-def dest*!: *twl-struct-invs-no-alien-in-trail*[*of* - ‹−*x1*›])
unfolding *unit-propagation-inner-loop-wl-loop-inv-def unit-propagation-inner-loop-l-inv-def*
unfolding *prod.case* **apply** *normalize-goal+*
apply (*drule twl-struct-invs-no-alien-in-trail*[*of* - ‹−*x1*›])
apply (*simp-all only*: *twl-st-l* $\mathcal{L}_{all}$*-all-atms-all-lits all-lits-def multiset.map-comp comp-def*
clause-twl-clause-of twl-st-wl in-all-lits-of-mm-uminus-iff ac-simps)
done
qed
have *length*: ‹$\bigwedge$*x y x1 x2 x1a x2a*.
case y of
$(L, S) \Rightarrow$
length (*watched-by S L*) $\leq r - 4 \wedge$
$L = K \wedge$ *length* (*watched-by S L*) $= s \wedge$ *length* (*watched-by S L*) $\leq r \Longrightarrow$
$(x, y) \in$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K* $\Longrightarrow$ $y = (x1, x2) \Longrightarrow$
$x = (x1a, x2a) \Longrightarrow$
x1 $\in\#$ *all-lits-st x2* $\Longrightarrow$
length (*watched-by-int x2a x1a*) $\leq$ *length* (*get-clauses-wl-heur x2a*) $\Longrightarrow$
mop-length-watched-by-int x2a x1a
$\leq \Downarrow$ *Id* (*RETURN* (*length* (*watched-by x2 x1*)))›
unfolding *mop-length-watched-by-int-def*
by *refine-rcg*
(*auto simp*: *twl-st-heur'-def map-fun-rel-def twl-st-heur-def*
simp flip: $\mathcal{L}_{all}$*-all-atms-all-lits intro*!: *ASSERT-leI*)

note *H*[*refine*] = *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D* [*THEN fref-to-Down-curry3*] *init*
show *?thesis*
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def* *unit-propagation-inner-loop-wl-loop-def uncurry-def* *unit-propagation-inner-loop-wl-loop-inv-def*[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*)
subgoal by (*auto simp*: *twl-st-heur′-def twl-st-heur-state-simp-watched simp flip*: $\mathcal{L}_{all}$-*all-atms-all-lits*)
apply (*rule length*; *assumption*)
subgoal by *auto*
subgoal by (*rule unit-propagation-inner-loop-wl-loop-D-heur-inv*)
subgoal
by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*]) (*auto simp*: *get-conflict-wl-is-None-heur-get-conflict-wl-is-None twl-st-heur-state-simp-watched twl-st-heur′-def* *get-conflict-wl-is-None-def simp flip*: $\mathcal{L}_{all}$-*all-atms-all-lits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *cut-watch-list-heur*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cut-watch-list-heur j w L* =(λ(*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
ASSERT(*j* ≤ *length* (*W*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
RETURN (*M*, *N*, *D*, *Q*,
W[*nat-of-lit L* := *take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))], *oth*)
})›

definition *cut-watch-list-heur2*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cut-watch-list-heur2* = (λ*j w L* (*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let n = *length* (*W*!(*nat-of-lit L*));
(*j*, *w*, *W*) ← $WHILE_T$$^{\lambda(j,\ w,\ W).\ j \le w \wedge w \le n \wedge nat\text{-}of\text{-}lit\ L < length\ W}$
(λ(*j*, *w*, *W*). *w* < *n*)
(λ(*j*, *w*, *W*). *do* {
ASSERT(*w* < *length* (*W*!(*nat-of-lit L*)));
RETURN (*j+1*, *w+1*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*j* := *W*!(*nat-of-lit L*)!*w*]])
})
(*j*, *w*, *W*);
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length W*);
let W = *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*)];
RETURN (*M*, *N*, *D*, *Q*, *W*, *oth*)
})›

lemma *cut-watch-list-heur2-cut-watch-list-heur*:
shows
‹*cut-watch-list-heur2 j w L S* ≤ ⇓ *Id* (*cut-watch-list-heur j w L S*)›
proof −
obtain *M N D Q W oth* **where** *S*: ‹*S* = (*M*, *N*, *D*, *Q*, *W*, *oth*)›
by (*cases S*)
define *n* **where** *n*: ‹*n* = *length* (*W* ! *nat-of-lit L*)›
let *?R* = ‹*measure* (λ(*j′*::*nat*, *w′* :: *nat*, - :: (*nat* × *nat literal* × *bool*) *list list*). *length* (*W*!*nat-of-lit L*) − *w′*)›
define *I′* **where**
‹*I′* ≡ λ(*j′*, *w′*, *W′*). *length* (*W′* ! (*nat-of-lit L*)) = *length* (*W* ! (*nat-of-lit L*)) ∧ *j′* ≤ *w′* ∧ *w′* ≥ *w* ∧
w′ − *w* = *j′* − *j* ∧ *j′* ≥ *j* ∧
drop w′ (*W′* ! (*nat-of-lit L*)) = *drop w′* (*W* ! (*nat-of-lit L*)) ∧
w′ ≤ *length* (*W′* ! (*nat-of-lit L*)) ∧
W′[*nat-of-lit L* := *take* (*j* + *w′* − *w*) (*W′* ! *nat-of-lit L*)] =
W[*nat-of-lit L* := *take* (*j* + *w′* − *w*) ((*take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))))]›

have *cut-watch-list-heur-alt-def*:
‹*cut-watch-list-heur j w L* =(λ(*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
ASSERT(*j* ≤ *length* (*W*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let W = *W*[*nat-of-lit L* := *take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))];
RETURN (*M*, *N*, *D*, *Q*, *W*, *oth*)
})›
unfolding *cut-watch-list-heur-def* **by** *auto*
have *REC*: ‹*ASSERT* (*x1k* < *length* (*x2k* ! *nat-of-lit L*)) ⤜
(λ-. *RETURN* (*x1j* + *1*, *x1k* + *1*, *x2k* [*nat-of-lit L* := (*x2k* ! *nat-of-lit L*) [*x1j* :=
x2k ! *nat-of-lit L* !
x1k]]))
≤ *SPEC* (λ*s′*. ∀ *x1 x2 x1a x2a*. *x2* = (*x1a*, *x2a*) ⟶ *s′* = (*x1*, *x2*) ⟶
(*x1* ≤ *x1a* ∧ *nat-of-lit L* < *length x2a*) ∧ *I′ s′* ∧
(*s′*, *s*) ∈ *measure* (λ(*j′*, *w′*, -). *length* (*W* ! *nat-of-lit L*) − *w′*))›
if
‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
pre: ‹*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! *nat-of-lit L*)› **and**
I: ‹*case s of* (*j*, *w*, *W*) ⇒ *j* ≤ *w* ∧ *nat-of-lit L* < *length W*› **and**
I′: ‹*I′ s*› **and**
cond: ‹*case s of* (*j*, *w*, *W*) ⇒ *w* < *length* (*W* ! *nat-of-lit L*)› **and**
[*simp*]: ‹*x2* = (*x1k*, *x2k*)› **and**
[*simp*]: ‹*s* = (*x1j*, *x2*)›
for *s x1j x2 x1k x2k*
proof −
have [*simp*]: ‹*x1k* < *length* (*x2k* ! *nat-of-lit L*)› **and**
‹*length* (*W* ! *nat-of-lit L*) − *Suc x1k* < *length* (*W* ! *nat-of-lit L*) − *x1k*›
using *cond I I′* **unfolding** *I′-def* **by** *auto*
moreover have ‹*x1j* ≤ *x1k*› ‹*nat-of-lit L* < *length x2k*›
using *I I′* **unfolding** *I′-def* **by** *auto*
moreover have ‹*I′* (*Suc x1j*, *Suc x1k*, *x2k*
[*nat-of-lit L* := (*x2k* ! *nat-of-lit L*)[*x1j* := *x2k* ! *nat-of-lit L* ! *x1k*]])›
proof −
have *ball-leI*: ‹(⋀*x*. *x* < *A* ⟹ *P x*) ⟹ (∀ *x* < *A*. *P x*)› **for** *A P*
by *auto*
have *H*: ‹⋀*i*. *x2k*[*nat-of-lit L* := *take* (*j* + *x1k* − *w*) (*x2k* ! *nat-of-lit L*)] ! *i* = *W*
[*nat-of-lit L* :=

take (*min* (*j* + *x1k* − *w*) *j*) (*W* ! *nat-of-lit L*) @
take (*j* + *x1k* − (*w* + *min* (*length* (*W* ! *nat-of-lit L*)) *j*))
(*drop w* (*W* ! *nat-of-lit L*))] ! *i*› **and**
H′: ‹*x2k*[*nat-of-lit L* := *take* (*j* + *x1k* − *w*) (*x2k* ! *nat-of-lit L*)] = *W*
[*nat-of-lit L* :=
take (*min* (*j* + *x1k* − *w*) *j*) (*W* ! *nat-of-lit L*) @
take (*j* + *x1k* − (*w* + *min* (*length* (*W* ! *nat-of-lit L*)) *j*))
(*drop w* (*W* ! *nat-of-lit L*))]› **and**
‹*j* < *length* (*W* ! *nat-of-lit L*)› **and**
‹(*length* (*W* ! *nat-of-lit L*) − *w*) ≥ (*Suc x1k* − *w*)› **and**
‹*x1k* ≥ *w*›
‹*nat-of-lit L* < *length W*› **and**
‹*j* + *x1k* − *w* = *x1j*› **and**
‹*x1j* − *j* = *x1k* − *w*› **and**
‹*x1j* < *length* (*W* ! *nat-of-lit L*)› **and**
‹*length* (*x2k* ! *nat-of-lit L*) = *length* (*W* ! *nat-of-lit L*)› **and**
‹*drop x1k* (*x2k* ! (*nat-of-lit L*)) = *drop x1k* (*W* ! (*nat-of-lit L*))›
‹*x1j* ≥ *j*› **and**
‹*w* + *x1j* − *j* = *x1k*›
using *I I′ pre cond* **unfolding** *I′-def* **by** *auto*
have
[*simp*]: ‹*min x1j j* = *j*›
using ‹*x1j* ≥ *j*› **unfolding** *min-def* **by** *auto*
have ‹*x2k*[*nat-of-lit L* := *take* (*Suc* (*j* + *x1k*) − *w*) (*x2k*[*nat-of-lit L* := (*x2k* ! *nat-of-lit L*)
[*x1j* := *x2k* ! *nat-of-lit L* ! *x1k*]] ! *nat-of-lit L*)] =
W[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*) @ *take* (*Suc* (*j* + *x1k*) − (*w* + *min* (*length* (*W* !
nat-of-lit L)) *j*))
(*drop w* (*W* ! *nat-of-lit L*))]›
using *cond I* ‹*j* < *length* (*W* ! *nat-of-lit L*)› **and**
‹(*length* (*W* ! *nat-of-lit L*) − *w*) ≥ (*Suc x1k* − *w*)› **and**
‹*x1k* ≥ *w*›
‹*nat-of-lit L* < *length W*›
‹*j* + *x1k* − *w* = *x1j*› ‹*x1j* < *length* (*W* ! *nat-of-lit L*)›
apply (*subst list-eq-iff-nth-eq*)
apply −
apply (*intro conjI ball-leI*)
subgoal using *arg-cong*[*OF H′*, *of length*] **by** *auto*
subgoal for *k*
apply (*cases* ‹*k* ≠ *nat-of-lit L*›)
subgoal using *H*[*of k*] **by** *auto*
subgoal
using *H*[*of k*] ‹*x1j* < *length* (*W* ! *nat-of-lit L*)›
‹*length* (*x2k* ! *nat-of-lit L*) = *length* (*W* ! *nat-of-lit L*)›
arg-cong[*OF* ‹*drop x1k* (*x2k* ! (*nat-of-lit L*)) = *drop x1k* (*W* ! (*nat-of-lit L*))›,
of ‹λ*xs*. *xs* ! *0*›] ‹*x1j* ≥ *j*›
apply (*cases* ‹*Suc x1j* = *length* (*W* ! *nat-of-lit L*)›)
apply (*auto simp add*: *Suc-diff-le take-Suc-conv-app-nth* ‹*j* + *x1k* − *w* = *x1j*›
‹*x1j* − *j* = *x1k* − *w*›[*symmetric*] ‹*w* + *x1j* − *j* = *x1k*›)
apply (*metis append.assoc le-neq-implies-less list-update-id nat-in-between-eq*(*1*)
not-less-eq take-Suc-conv-app-nth take-all)
by (*metis* (*no-types*, *lifting*) ‹*x1j* < *length* (*W* ! *nat-of-lit L*)› *append.assoc*
take-Suc-conv-app-nth take-update-last)
done
done
then show *?thesis*
unfolding *I′-def prod.case*

using $I\ I'$ *cond* **unfolding** I'*-def* **by** (*auto simp*: *Cons-nth-drop-Suc*[*symmetric*])
qed
ultimately show *?thesis*
by *auto*
qed

have *step*: ‹$(s,\ W[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{take}\ j\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)\ @\ \mathit{drop}\ w\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)])$
$\in\ \{((i,\ j,\ W'),\ W).\ (W'[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{take}\ i\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)],\ W) \in \mathit{Id} \wedge$
$i \leq \mathit{length}\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L) \wedge \mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W' \wedge$
$n = \mathit{length}\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)\}$›
if
pre: ‹$j \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L) \wedge j \leq w \wedge \mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W \wedge$
$w \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$j \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L) \wedge j \leq w \wedge \mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W \wedge$
$w \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$\mathit{case}\ s\ \mathit{of}\ (j,\ w,\ W) \Rightarrow j \leq w \wedge \mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W$› **and**
‹$I'\ s$› **and**
‹$\neg\ (\mathit{case}\ s\ \mathit{of}\ (j,\ w,\ W) \Rightarrow w < \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L))$›
for *s*
proof –
obtain $j'\ w'\ W'$ **where** *s*: ‹$s = (j',\ w',\ W')$› **by** (*cases s*)
have
‹$\neg\ w' < \mathit{length}\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$j \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$j' \leq w'$› **and**
‹$\mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W'$› **and**
[*simp*]: ‹$\mathit{length}\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L) = \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$j \leq w$› **and**
‹$j' \leq w'$› **and**
‹$\mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W$› **and**
‹$w \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$w \leq w'$› **and**
‹$w' - w = j' - j$› **and**
‹$j \leq j'$› **and**
‹$\mathit{drop}\ w'\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L) = \mathit{drop}\ w'\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
‹$w' \leq \mathit{length}\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› **and**
L-le-W: ‹$\mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W$› **and**
eq: ‹$W'[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{take}\ (j + w' - w)\ (W'\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)] =$
$W[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{take}\ (j + w' - w)\ (\mathit{take}\ j\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)\ @\ \mathit{drop}\ w\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L))]$›
using *that* **unfolding** I'*-def that prod.case s*
by *blast+*
then have
j-j': ‹$j + w' - w = j'$› **and**
j-le: ‹$j + w' - w = \mathit{length}\ (\mathit{take}\ j\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)\ @\ \mathit{drop}\ w\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L))$› **and**
w': ‹$w' = \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$›
by *auto*
have [*simp*]: ‹$\mathit{length}\ W = \mathit{length}\ W'$›
using *arg-cong*[*OF eq*, *of length*] **by** *auto*
show *?thesis*
using *eq* ‹$j \leq w$› ‹$w \leq \mathit{length}\ (W\ !\ \mathit{nat\text{-}of\text{-}lit}\ L)$› ‹$j \leq j'$› ‹$w' - w = j' - j$›
‹$w \leq w'$› *w' L-le-W*
unfolding *j-j' j-le s S n*
by (*auto simp*: *min-def split*: *if-splits*)
qed

have *HHH*: ‹$X \leq \mathit{RES}\ (R^{-1}\ ``\ \{S\}) \Longrightarrow X \leq \Downarrow R\ (\mathit{RETURN}\ S)$› **for** *X S R*

by (*auto simp*: *RETURN-def conc-fun-RES*)

show *?thesis*
unfolding *cut-watch-list-heur2-def cut-watch-list-heur-alt-def prod.case S n*[*symmetric*]
apply (*rewrite at* ‹*let - = n in -*› *Let-def*)
apply (*refine-vcg WHILEIT-rule-stronger-inv-RES*[**where** $R = ?R$ **and**
$I' = I'$ **and** Φ = ‹$\{((i, j, W'), W). (W'[\mathit{nat\text{-}of\text{-}lit}\ L := \mathit{take}\ i\ (W' !\ \mathit{nat\text{-}of\text{-}lit}\ L)], W) \in Id \wedge$
$i \leq \mathit{length}\ (W' !\ \mathit{nat\text{-}of\text{-}lit}\ L) \wedge \mathit{nat\text{-}of\text{-}lit}\ L < \mathit{length}\ W' \wedge$
$n = \mathit{length}\ (W' !\ \mathit{nat\text{-}of\text{-}lit}\ L)\}^{-1}$ ‘‘ -›] *HHH*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *S*)
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *I'-def* **by** (*auto simp*: *n*)
subgoal unfolding *I'-def* **by** (*auto simp*: *n*)
subgoal unfolding *I'-def* **by** (*auto simp*: *n*)
subgoal unfolding *I'-def* **by** *auto*
subgoal unfolding *I'-def* **by** *auto*
subgoal unfolding *I'-def* **by** (*auto simp*: *n*)
subgoal using *REC* **by** (*auto simp*: *n*)
subgoal unfolding *I'-def* **by** (*auto simp*: *n*)
subgoal for *s* **using** *step*[*of* ‹*s*›] **unfolding** *I'-def* **by** (*auto simp*: *n*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *vdom-m-cut-watch-list*:
‹$\mathit{set}\ xs \subseteq \mathit{set}\ (W\ L) \Longrightarrow \mathit{vdom\text{-}m}\ \mathcal{A}\ (W(L := xs))\ d \subseteq \mathit{vdom\text{-}m}\ \mathcal{A}\ W\ d$›
by (*cases* ‹$L \in\#\ \mathcal{L}_{all}\ \mathcal{A}$›)
(*force simp*: *vdom-m-def split*: *if-splits*)+

The following order allows the rule to be used as a destruction rule, make it more useful for refinement proofs.

lemma *vdom-m-cut-watch-listD*:
‹$x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ (W(L := xs))\ d \Longrightarrow \mathit{set}\ xs \subseteq \mathit{set}\ (W\ L) \Longrightarrow x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ W\ d$›
using *vdom-m-cut-watch-list*[*of xs W L*] **by** *auto*

lemma *cut-watch-list-heur-cut-watch-list-heur*:
‹(*uncurry3 cut-watch-list-heur*, *uncurry3 cut-watch-list*) ∈
$[\lambda(((j, w), L), S).\ \mathit{True}]_f$
$\mathit{nat\text{-}rel} \times_f \mathit{nat\text{-}rel} \times_f \mathit{nat\text{-}lit\text{-}lit\text{-}rel} \times_f \mathit{twl\text{-}st\text{-}heur}''\ \mathcal{D}\ r \rightarrow \langle \mathit{twl\text{-}st\text{-}heur}''\ \mathcal{D}\ r\rangle \mathit{nres\text{-}rel}$›
unfolding *cut-watch-list-heur-def cut-watch-list-def uncurry-def*
$\mathcal{L}_{all}$*-all-atms-all-lits*[*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur'-def*
twl-st-heur-def map-fun-rel-def)
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur'-def*
twl-st-heur-def map-fun-rel-def)

subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def)
subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def)
subgoal
  by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
    twl-st-heur-def map-fun-rel-def vdom-m-cut-watch-list set-take-subset
      set-drop-subset dest!: *vdom-m-cut-watch-listD*
      dest!: *in-set-dropD in-set-takeD*)
done

definition *unit-propagation-inner-loop-wl-D-heur*
  :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
  ‹*unit-propagation-inner-loop-wl-D-heur L* S_0 = *do* {
    (*j*, *w*, *S*) ← *unit-propagation-inner-loop-wl-loop-D-heur L* S_0;
    ASSERT(*length* (*watched-by-int S L*) ≤ *length* (*get-clauses-wl-heur* S_0) − *4*);
    cut-watch-list-heur2 j w L S
  }›

lemma *unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D*:
  ‹(*uncurry unit-propagation-inner-loop-wl-D-heur*, *uncurry unit-propagation-inner-loop-wl*) ∈
    [λ(*L*, *S*). *length*(*watched-by S L*) ≤ *r*−*4*]$_f$
    nat-lit-lit-rel $\times_f$ *twl-st-heur′′* 𝒟 *r* → ⟨*twl-st-heur′′* 𝒟 *r*⟩ *nres-rel*›
proof −
  have *length-le*: ‹*length* (*watched-by x2b x1b*) ≤ *r* − *4*› **and**
    length-eq: ‹*length* (*watched-by x2b x1b*) = *length* (*watched-by* (*snd y*) (*fst y*))› **and**
    eq: ‹*x1b* = *fst y*›
    if
      ‹*case y of* (*L*, *S*) ⇒ *length* (*watched-by S L*) ≤ *r*−*4*› **and**
      ‹(*x*, *y*) ∈ *nat-lit-lit-rel* $\times_f$ *twl-st-heur′′* 𝒟 *r*› **and**
      ‹*y* = (*x1*, *x2*)› **and**
      ‹*x* = (*x1a*, *x2a*)› **and**
      ‹(*x1*, *x2*) = (*x1b*, *x2b*)›
    for *x y x1 x2 x1a x2a x1b x2b r*
      using *that* **by** *auto*
  show *?thesis*
    unfolding *unit-propagation-inner-loop-wl-D-heur-def*
      unit-propagation-inner-loop-wl-def uncurry-def
      apply (*intro frefI nres-relI*)
    apply (*refine-vcg cut-watch-list-heur-cut-watch-list-heur*[*of* 𝒟 *r*, *THEN fref-to-Down-curry3*]
 unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D[*of r - -* 𝒟,
    THEN fref-to-Down-curry])

    apply (*rule length-le*; *assumption*)
    apply (*rule eq*; *assumption*)
    apply (*rule length-eq*; *assumption*)
    subgoal by *auto*
    subgoal by (*auto simp*: *twl-st-heur′-def twl-st-heur-state-simp-watched*)
    subgoal
      by (*auto simp*: *twl-st-heur′-def twl-st-heur-state-simp-watched*
       simp flip: $\mathcal{L}_{all}$-*all-atms-all-lits*)
    apply (*rule order.trans*)
    apply (*rule cut-watch-list-heur2-cut-watch-list-heur*)
    apply (*subst Down-id-eq*)

apply (*rule cut-watch-list-heur-cut-watch-list-heur*[*of* $\mathcal{D}$, *THEN fref-to-Down-curry3*])
by *auto*
qed

definition *select-and-remove-from-literals-to-update-wl-heur*
:: ‹*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × *nat literal*) *nres*›
where
‹*select-and-remove-from-literals-to-update-wl-heur S* = *do* {
ASSERT(*literals-to-update-wl-heur S* < *length* (*fst* (*get-trail-wl-heur S*)));
ASSERT(*literals-to-update-wl-heur S* + *1* ≤ *uint32-max*);
L ← *isa-trail-nth* (*get-trail-wl-heur S*) (*literals-to-update-wl-heur S*);
RETURN (*set-literals-to-update-wl-heur* (*literals-to-update-wl-heur S* + *1*) *S*, −*L*)
}›

definition *unit-propagation-outer-loop-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur* ⇒ *bool*›
where
‹*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S' ⟷
(∃ S_0' *S*. (S_0, S_0') ∈ *twl-st-heur* ∧ (S', *S*) ∈ *twl-st-heur* ∧
unit-propagation-outer-loop-wl-inv S ∧
dom-m (*get-clauses-wl S*) = *dom-m* (*get-clauses-wl* S_0') ∧
length (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur* S_0) ∧
isa-length-trail-pre (*get-trail-wl-heur* S'))›

definition *unit-propagation-outer-loop-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*unit-propagation-outer-loop-wl-D-heur* S_0 =
$WHILE_T{}^{unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
(λ*S*. *literals-to-update-wl-heur S* < *isa-length-trail* (*get-trail-wl-heur S*))
(λ*S*. *do* {
ASSERT(*literals-to-update-wl-heur S* < *isa-length-trail* (*get-trail-wl-heur S*));
(S', *L*) ← *select-and-remove-from-literals-to-update-wl-heur S*;
ASSERT(*length* (*get-clauses-wl-heur* S') = *length* (*get-clauses-wl-heur S*));
unit-propagation-inner-loop-wl-D-heur L S'
})
S_0›

lemma *select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl*:
‹*literals-to-update-wl y* ≠ {#} ⟹
(*x*, *y*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r1* ⟹
select-and-remove-from-literals-to-update-wl-heur x
≤ ⇓{((*S*, *L*), (S', L')). ((*S*, *L*), (S', L')) ∈ *twl-st-heur''* $\mathcal{D}$*1 r1* $\times_f$ *nat-lit-lit-rel* ∧
S' = *set-literals-to-update-wl* (*literals-to-update-wl y* − {#*L*#}) *y* ∧
get-clauses-wl-heur S = *get-clauses-wl-heur x*}
(*select-and-remove-from-literals-to-update-wl y*)›
supply *RETURN-as-SPEC-refine*[*refine2*]
unfolding *select-and-remove-from-literals-to-update-wl-heur-def*
select-and-remove-from-literals-to-update-wl-def
apply (*refine-vcg*)
subgoal
by (*subst trail-pol-same-length*[*of* ‹*get-trail-wl-heur x*› ‹*get-trail-wl y*› ‹*all-atms-st y*›])
(*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff trail-pol-alt-def*)

subgoal
apply (*subst* (*asm*) *trail-pol-same-length*[*of* ‹*get-trail-wl-heur x*› ‹*get-trail-wl y*› ‹*all-atms-st y*›])
apply (*auto simp*: *twl-st-heur-def twl-st-heur′-def*; *fail*)[]
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule isa-trail-nth-rev-trail-nth*[*THEN fref-to-Down-curry*, *unfolded comp-def RETURN-def*, *unfolded conc-fun-RES*, *of* ‹*get-trail-wl y*› - - - ‹*all-atms-st y*›])
apply ((*auto simp*: *twl-st-heur-def twl-st-heur′-def*; *fail*)+)[*2*]
subgoal for *z*
apply (*cases x*; *cases y*)
by (*simp-all add*: *Cons-nth-drop-Suc*[*symmetric*] *twl-st-heur-def twl-st-heur′-def RETURN-RES-refine-iff rev-trail-nth-def*)
done
done

lemma *outer-loop-length-watched-le-length-arena*:
assumes
xa-x′: ‹(xa, x') ∈ *twl-st-heur″* $\mathcal{D}$ *r*› **and**
prop-heur-inv: ‹*unit-propagation-outer-loop-wl-D-heur-inv x xa*› **and**
prop-inv: ‹*unit-propagation-outer-loop-wl-inv x′*› **and**
xb-x′a: ‹(xb, $x'a$) ∈ {((S, L), (S', L')). ((S, L), (S', L')) ∈ *twl-st-heur″* $\mathcal{D}1$ *r* $\times_f$ *nat-lit-lit-rel* ∧ S' = *set-literals-to-update-wl* (*literals-to-update-wl x′* − {#*L*#}) *x′* ∧ *get-clauses-wl-heur S* = *get-clauses-wl-heur xa*}› **and**
st: ‹$x'a$ = ($x1$, $x2$)›
‹xb = ($x1a$, $x2a$)› **and**
x2: ‹*x2* ∈# *all-lits-st x1*› **and**
st′: ‹($x2$, $x1$) = ($x1b$, $x2b$)›
shows ‹*length* (*watched-by x2b x1b*) ≤ *r*−*4*›
proof −
have ‹*correct-watching x′*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-inv-def unit-propagation-outer-loop-wl-inv-def*
by *auto*
moreover have ‹*x2* ∈# *all-lits-st x′*›
using *x2 assms* **unfolding** *all-atms-def all-lits-def*
by (*auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
ultimately have *dist*: ‹*distinct-watched* (*watched-by x′ x2*)›
using *x2 xb-x′a* **unfolding** *all-atms-def all-lits-def*
by (*cases x′*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps ac-simps*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x′a* **unfolding** *st*
by (*cases x′*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x′*)
(*auto simp*: *twl-st-heur-def twl-st-heur′-def st*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
using *x2 xb-x′a* **unfolding** *st* $\mathcal{L}_{all}$*-all-atms-all-lits*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur xa*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x′a* **unfolding** *all-atms-def all-lits-def st*
by (*cases x′*)
(*auto simp*: *twl-st-heur′-def twl-st-heur-def*)

have ‹vdom-m (all-atms-st x1) (get-watched-wl x1) (get-clauses-wl x1) ⊆ set (get-vdom x1a)›
  using xb-x'a
  by (cases x')
    (auto simp: twl-st-heur-def twl-st-heur'-def st)
then have subset: ‹set (map fst (watched-by x1 x2)) ⊆ set (get-vdom x1a)›
  using x2 unfolding vdom-m-def st
  by (cases x1)
    (force simp: twl-st-heur'-def twl-st-heur-def
      dest!: multi-member-split)
have watched-incl: ‹mset (map fst (watched-by x1 x2)) ⊆# mset (get-vdom x1a)›
  by (rule distinct-subseteq-iff[THEN iffD1])
    (use dist[unfolded distinct-watched-alt-def] dist-vdom subset in
      ‹simp-all flip: distinct-mset-mset-distinct›)
have vdom-incl: ‹set (get-vdom x1a) ⊆ {4..< length (get-clauses-wl-heur xa)}›
  using valid-arena-in-vdom-le-arena[OF valid] arena-dom-status-iff[OF valid] by auto

have ‹length (get-vdom x1a) ≤ length (get-clauses-wl-heur xa) − 4›
  by (subst distinct-card[OF dist-vdom, symmetric])
    (use card-mono[OF - vdom-incl] in auto)
then show ?thesis
  using size-mset-mono[OF watched-incl] xb-x'a st'
  by auto
qed

theorem unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D':
‹(unit-propagation-outer-loop-wl-D-heur, unit-propagation-outer-loop-wl) ∈
  twl-st-heur'' 𝒟 r →f ⟨twl-st-heur'' 𝒟 r⟩ nres-rel›
unfolding unit-propagation-outer-loop-wl-D-heur-def
  unit-propagation-outer-loop-wl-def all-lits-alt-def2[symmetric]
apply (intro frefI nres-relI)
apply (refine-vcg
  unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D[of r 𝒟, THEN fref-to-Down-curry]
   select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl
      [of - - 𝒟 r])
subgoal for x y S T
  using isa-length-trail-pre[of ‹get-trail-wl-heur S› ‹get-trail-wl T› ‹all-atms-st T›] apply −
  unfolding unit-propagation-outer-loop-wl-D-heur-inv-def twl-st-heur'-def
  apply (rule-tac x=y in exI)
  apply (rule-tac x=T in exI)
  by (auto 5 2 simp: twl-st-heur-def twl-st-heur'-def)
subgoal for - - x y
 by (subst isa-length-trail-length-u[THEN fref-to-Down-unRET-Id, of - ‹get-trail-wl y› ‹all-atms-st y›])
    (auto simp: twl-st-heur-def twl-st-heur'-def)
subgoal by (auto simp: twl-st-heur'-def)
subgoal for x y xa x' xb x'a x1 x2 x1a x2a x1b x2b
  by (rule-tac x=x and xa=xa and 𝒟=𝒟 in outer-loop-length-watched-le-length-arena)
subgoal by (auto simp: twl-st-heur'-def)
done

lemma twl-st-heur'D-twl-st-heurD:
assumes H: ‹(⋀𝒟. f ∈ twl-st-heur' 𝒟 →f ⟨twl-st-heur' 𝒟⟩ nres-rel)›
shows ‹f ∈ twl-st-heur →f ⟨twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
proof −
obtain f1 f2 where f: ‹f = (f1, f2)›
  by (cases f) auto
show ?thesis

using *assms* **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
apply (*rule weaken-*$\Downarrow$*'*[*of* - ‹*twl-st-heur'* (*dom-m* (*get-clauses-wl y*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *watched-by-app-watched-by-app-heur*:
‹(*uncurry2* (*RETURN ooo watched-by-app-heur*), *uncurry2* (*RETURN ooo watched-by-app*)) ∈
[λ((*S*, *L*), *K*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧ *K* < *length* (*get-watched-wl S L*)]$_f$
twl-st-heur $\times_f$ *Id* $\times_f$ *Id* → ⟨*Id*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *watched-by-app-heur-def watched-by-app-def twl-st-heur-def map-fun-rel-def*)

lemma *case-tri-bool-If*:
‹(*case a of*
None ⇒ *f1*
| *Some v* ⇒
(*if v then f2 else f3*)) =
(*let b* = *a in if b* = *UNSET*
then f1
else if b = *SET-TRUE then f2 else f3*)›
by (*auto split*: *option.splits*)

definition *isa-find-unset-lit* :: ‹*trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*isa-find-unset-lit M* = *isa-find-unwatched-between* (λ*L*. *polarity-pol M L* ≠ *Some False*) *M*›

lemma *update-clause-wl-heur-pre-le-sint64*:
assumes
‹*arena-is-valid-clause-idx-and-access a1'a bf baa*› **and**
‹*length* (*get-clauses-wl-heur*
(*a1'*, *a1'a*, (*da*, *db*, *dc*), *a1'c*, *a1'd*, ((*eu*, *ev*, *ew*, *ex*, *ey*), *ez*), *fa*, *fb*,
fc, *fd*, *fe*, (*ff*, *fg*, *fh*, *fi*), *fj*, *fk*, *fl*, *fm*, *fn*)) ≤ *sint64-max*› **and**
‹*arena-lit-pre a1'a* (*bf* + *baa*)›
shows ‹*bf* + *baa* ≤ *sint64-max*›
‹*length a1'a* ≤ *sint64-max*›
using *assms*
by (*auto simp*: *arena-is-valid-clause-idx-and-access-def isasat-fast-def*
dest!: *arena-lifting*(*10*))

end
theory *IsaSAT-Inner-Propagation-LLVM*
imports *IsaSAT-Setup-LLVM*
IsaSAT-Inner-Propagation
begin

sepref-register *isa-save-pos*

sepref-def *isa-save-pos-fast-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›

supply
  [[*goals-limit=1*]]
  if-splits[*split*]
unfolding *isa-save-pos-def PR-CONST-def isasat-bounded-assn-def*
by *sepref*

lemma [*def-pat-rules*]: ‹*nth-rll* ≡ *op-list-list-idx*›
by (*auto simp*: *nth-rll-def intro*!: *ext eq-reflection*)

sepref-def *watched-by-app-heur-fast-code*
is ‹*uncurry2* (*RETURN ooo watched-by-app-heur*)›
:: ‹$[watched\text{-}by\text{-}app\text{-}heur\text{-}pre]_a$
  $isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow watcher\text{-}fast\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *watched-by-app-heur-alt-def isasat-bounded-assn-def nth-rll-def*[*symmetric*]
watched-by-app-heur-pre-def
by *sepref*

sepref-register *isa-find-unwatched-wl-st-heur isa-find-unwatched-between isa-find-unset-lit*
polarity-pol

sepref-register *0 1*

sepref-def *isa-find-unwatched-between-fast-code*
is ‹*uncurry4 isa-find-unset-lit*›
:: ‹$[\lambda((((M, N), \text{-}), \text{-}), \text{-}).\ length\ N \leq sint64\text{-}max]_a$
  $trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k$
$\rightarrow$
  $snat\text{-}option\text{-}assn'\ TYPE(64)$›
supply [[*goals-limit = 3*]]
unfolding *isa-find-unset-lit-def isa-find-unwatched-between-def SET-FALSE-def*[*symmetric*]
PR-CONST-def
apply (*rewrite* **in** ‹*if* ⌑ *then - else -*› *tri-bool-eq-def*[*symmetric*])
apply (*annot-snat-const TYPE* (*64*))
by *sepref*

sepref-register *mop-arena-pos mop-arena-lit2*
sepref-def *mop-arena-pos-impl*
is ‹*uncurry mop-arena-pos*›
:: ‹$arena\text{-}fast\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
unfolding *mop-arena-pos-def*
by *sepref*

sepref-def *swap-lits-impl* **is** *uncurry3 mop-arena-swap*
:: $sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a arena\text{-}fast\text{-}assn^d \rightarrow_a arena\text{-}fast\text{-}assn$
unfolding *mop-arena-swap-def swap-lits-pre-def*
unfolding *gen-swap*
by *sepref*

sepref-def *find-unwatched-wl-st-heur-fast-code*

is ‹*uncurry isa-find-unwatched-wl-st-heur*›
:: ‹$[(\lambda(S, C).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max)]_a$
$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow snat\text{-}option\text{-}assn'\ TYPE(64)$›
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
unfolding *isa-find-unwatched-wl-st-heur-def PR-CONST-def*
find-unwatched-def fmap-rll-def[*symmetric*] *isasat-bounded-assn-def*
length-uint32-nat-def[*symmetric*] *isa-find-unwatched-def*
case-tri-bool-If find-unwatched-wl-st-heur-pre-def
fmap-rll-u64-def[*symmetric*]
apply (*subst isa-find-unset-lit-def*[*symmetric*])
apply (*subst isa-find-unset-lit-def*[*symmetric*])
apply (*subst isa-find-unset-lit-def*[*symmetric*])
apply (*annot-snat-const TYPE* (*64*))
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register *mop-access-lit-in-clauses-heur mop-watched-by-app-heur*
sepref-def *mop-access-lit-in-clauses-heur-impl*
is ‹*uncurry2 mop-access-lit-in-clauses-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a unat\text{-}lit\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *mop-access-lit-in-clauses-heur-alt-def isasat-bounded-assn-def*
by *sepref*

lemma *other-watched-wl-heur-alt-def*:
‹$other\text{-}watched\text{-}wl\text{-}heur = (\lambda S.\ arena\text{-}other\text{-}watched\ (get\text{-}clauses\text{-}wl\text{-}heur\ S))$›
apply (*intro ext*)
unfolding *other-watched-wl-heur-def*
arena-other-watched-def
mop-access-lit-in-clauses-heur-def
by *auto argo*

lemma *other-watched-wl-heur-alt-def2*:
‹$other\text{-}watched\text{-}wl\text{-}heur = (\lambda(\text{-}, N, \text{-}).\ arena\text{-}other\text{-}watched\ N)$›
by (*auto intro*!: *ext simp*: *other-watched-wl-heur-alt-def*)

sepref-def *other-watched-wl-heur-impl*
is ‹*uncurry3 other-watched-wl-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a$
$unat\text{-}lit\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *other-watched-wl-heur-alt-def2*
isasat-bounded-assn-def
by *sepref*

sepref-register *update-clause-wl-heur*
setup ‹*map-theory-claset* (*fn ctxt* => *ctxt delSWrapper split-all-tac*)›

lemma *arena-lit-pre-le2*: ‹
$arena\text{-}lit\text{-}pre\ a\ i \Longrightarrow length\ a \leq sint64\text{-}max \Longrightarrow i < max\text{-}snat\ 64$›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
sint64-max-def max-snat-def
by *fastforce*

lemma *sint64-max-le-max-snat64*: ‹$a < sint64\text{-}max \Longrightarrow Suc\ a < max\text{-}snat\ 64$›
by (*auto simp*: *max-snat-def sint64-max-def*)

sepref-def *update-clause-wl-fast-code*
is ⟨*uncurry7 update-clause-wl-heur*⟩
:: ⟨$[\lambda(((((((L,\ C),\ b),\ j),\ w),\ i),\ f),\ S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a bool1\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k$
$*_a$
$sint64\text{-}nat\text{-}assn^k$
$*_a isasat\text{-}bounded\text{-}assn^d \rightarrow sint64\text{-}nat\text{-}assn \times_a sint64\text{-}nat\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit=1*]] *arena-lit-pre-le2*[*intro*] *swap-lits-pre-def*[*simp*]
sint64-max-le-max-snat64[*intro*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def*
fmap-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*] *fmap-swap-ll-def*[*symmetric*]
append-ll-def[*symmetric*] *update-clause-wl-code-pre-def*
fmap-rll-u64-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
fmap-swap-ll-def[*symmetric*]
PR-CONST-def mop-arena-lit2′-def
apply (*annot-snat-const TYPE* (*64*))
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register *mop-arena-swap*

sepref-def *propagate-lit-wl-fast-code*
is ⟨*uncurry3 propagate-lit-wl-heur*⟩
:: ⟨$[\lambda(((L,\ C),\ i),\ S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
unfolding *PR-CONST-def propagate-lit-wl-heur-def*
supply [[*goals-limit=1*]] *swap-lits-pre-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
save-phase-def
apply (*rewrite at* ⟨*count-decided-pol - =* ¤⟩ *unat-const-fold*[**where** *′a=32*])
apply (*annot-snat-const TYPE* (*64*))
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *propagate-lit-wl-bin-fast-code*
is ⟨*uncurry2 propagate-lit-wl-bin-heur*⟩
:: ⟨$[\lambda((L,\ C),\ S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
$unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow$
$isasat\text{-}bounded\text{-}assn$⟩
unfolding *PR-CONST-def propagate-lit-wl-heur-def*
supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
unfolding *update-clause-wl-heur-def isasat-bounded-assn-def*
propagate-lit-wl-heur-pre-def fmap-swap-ll-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
save-phase-def propagate-lit-wl-bin-heur-def
apply (*rewrite at* ⟨*count-decided-pol - =* ¤⟩ *unat-const-fold*[**where** *′a=32*])
unfolding *fold-tuple-optimizations*
by *sepref*

lemma *op-list-list-upd-alt-def*: ‹*op-list-list-upd xss i j x* = *xss*[*i* := (*xss* ! *i*)[*j* := *x*]]›
  unfolding *op-list-list-upd-def* **by** *auto*

sepref-def *update-blit-wl-heur-fast-code*
  is ‹*uncurry6 update-blit-wl-heur*›
  :: ‹[λ((((((-, -), -), -), *C*), *i*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$
      unat-lit-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
      sint64-nat-assn^k $*_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} →
    sint64-nat-assn $\times_a$ *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]] *sint64-max-le-max-snat64*[*intro*]
  unfolding *update-blit-wl-heur-def isasat-bounded-assn-def append-ll-def*[*symmetric*]
    op-list-list-upd-alt-def[*symmetric*]
  apply (*annot-snat-const TYPE* (*64*))
  by *sepref*

sepref-register *keep-watch-heur*

lemma *op-list-list-take-alt-def*: ‹*op-list-list-take xss i l* = *xss*[*i* := *take l* (*xss* ! *i*)]›
  unfolding *op-list-list-take-def* **by** *auto*

sepref-def *keep-watch-heur-fast-code*
  is ‹*uncurry3 keep-watch-heur*›
  :: ‹*unat-lit-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  supply
    [[*goals-limit=1*]]
  unfolding *keep-watch-heur-def PR-CONST-def*
  unfolding *fmap-rll-def*[*symmetric*] *isasat-bounded-assn-def*
  unfolding
    op-list-list-upd-alt-def[*symmetric*]
    nth-rll-def[*symmetric*]
    SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*]
  by *sepref*

sepref-register *isa-set-lookup-conflict-aa set-conflict-wl-heur*

sepref-register *arena-incr-act*

sepref-def *set-conflict-wl-heur-fast-code*
  is ‹*uncurry set-conflict-wl-heur*›
  :: ‹[λ(*C*, *S*).
     length (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$
    sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
  unfolding *set-conflict-wl-heur-def isasat-bounded-assn-def*
    set-conflict-wl-heur-pre-def PR-CONST-def
  apply (*annot-unat-const TYPE* (*32*))
  unfolding *fold-tuple-optimizations*
  by *sepref*

sepref-register *update-blit-wl-heur clause-not-marked-to-delete-heur*
lemma *mop-watched-by-app-heur-alt-def*:

‹mop-watched-by-app-heur = (λ(M, N, D, Q, W, vmtf, φ, clvls, cach, lbd, outl, stats, fema, sema) L K. do {
    ASSERT(K < length (W ! nat-of-lit L));
    ASSERT(nat-of-lit L < length (W));
    RETURN (W ! nat-of-lit L ! K)})›
  by (intro ext; rename-tac S L K; case-tac S)
    (auto simp: mop-watched-by-app-heur-def)

sepref-def mop-watched-by-app-heur-code
  is ‹uncurry2 mop-watched-by-app-heur›
  :: ‹$isasat\text{-}bounded\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a watcher\text{-}fast\text{-}assn$›
  unfolding mop-watched-by-app-heur-alt-def isasat-bounded-assn-def
    nth-rll-def[symmetric]
  by sepref

lemma unit-propagation-inner-loop-wl-loop-D-heur-inv0D: ‹unit-propagation-inner-loop-wl-loop-D-heur-inv0 L (j, w, S0) ⟹
   j ≤ length (get-clauses-wl-heur S0) − 4 ∧ w ≤ length (get-clauses-wl-heur S0) − 4›
  unfolding unit-propagation-inner-loop-wl-loop-D-heur-inv0-def prod.case
    unit-propagation-inner-loop-wl-loop-inv-def unit-propagation-inner-loop-l-inv-def
  apply normalize-goal+
    by (simp only: twl-st-l twl-st twl-st-wl
      $\mathcal{L}_{all}$-all-atms-all-lits) linarith

sepref-def pos-of-watched-heur-impl
  is ‹uncurry2 pos-of-watched-heur›
  :: ‹$isasat\text{-}bounded\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a unat\text{-}lit\text{-}assn^k \rightarrow_a sint64\text{-}nat\text{-}assn$›
  supply [[goals-limit=1]]
  unfolding pos-of-watched-heur-def
  apply (annot-snat-const TYPE (64))
  by sepref

sepref-def unit-propagation-inner-loop-body-wl-fast-heur-code
  is ‹uncurry3 unit-propagation-inner-loop-body-wl-heur›
  :: ‹$[\lambda((L, w), S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a$
     $unat\text{-}lit\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^d \rightarrow$
      $sint64\text{-}nat\text{-}assn \times_a sint64\text{-}nat\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
  supply [[goals-limit=1]]
   if-splits[split] sint64-max-le-max-snat64[intro] unit-propagation-inner-loop-wl-loop-D-heur-inv0D[dest!]
  unfolding unit-propagation-inner-loop-body-wl-heur-def PR-CONST-def
  unfolding fmap-rll-def[symmetric]
  unfolding option.case-eq-if is-None-alt[symmetric]
    SET-FALSE-def[symmetric] SET-TRUE-def[symmetric] tri-bool-eq-def[symmetric]
  apply (annot-snat-const TYPE (64))
  by sepref

sepref-register unit-propagation-inner-loop-body-wl-heur

lemmas [llvm-inline] =
  other-watched-wl-heur-impl-def
  pos-of-watched-heur-impl-def
  propagate-lit-wl-heur-def
  clause-not-marked-to-delete-heur-fast-code-def
  mop-watched-by-app-heur-code-def
  keep-watch-heur-fast-code-def

nat-of-lit-rel-impl-def

experiment begin

export-llvm
isa-save-pos-fast-code
watched-by-app-heur-fast-code
isa-find-unwatched-between-fast-code
find-unwatched-wl-st-heur-fast-code
update-clause-wl-fast-code
propagate-lit-wl-fast-code
propagate-lit-wl-bin-fast-code
status-neq-impl
clause-not-marked-to-delete-heur-fast-code
update-blit-wl-heur-fast-code
keep-watch-heur-fast-code
set-conflict-wl-heur-fast-code
unit-propagation-inner-loop-body-wl-fast-heur-code

end

end
theory *IsaSAT-VMTF*
imports *Watched-Literals.WB-Sort IsaSAT-Setup*
begin

Chapter 10

Decision heuristic

10.1 Code generation for the VMTF decision heuristic and the trail

definition *update-next-search* **where**
‹*update-next-search L* = (λ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
((*ns*, *m*, *fst-As*, *lst-As*, *L*), *to-remove*))›

definition *vmtf-enqueue-pre* **where**
‹*vmtf-enqueue-pre* =
(λ((*M*, *L*),(*ns*,*m*,*fst-As*,*lst-As*, *next-search*)). *L* < *length ns* ∧
(*fst-As* ≠ *None* ⟶ *the fst-As* < *length ns*) ∧
(*fst-As* ≠ *None* ⟶ *lst-As* ≠ *None*) ∧
m+*1* ≤ *uint64-max*)›

definition *isa-vmtf-enqueue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
ASSERT(*defined-atm-pol-pre M L*);
de ← *RETURN* (*defined-atm-pol M L*);
case fst-As of
None ⇒*RETURN* ((*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
(*if de then None else Some L*)))
| *Some fst-As* ⇒ *do* {
let fst-As' = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*));
RETURN (*ns*[*L* := *VMTF-Node* (*m*+*1*) *None* (*Some fst-As*), *fst-As* := *fst-As'*],
m+*1*, *L*, *the lst-As*, (*if de then next-search else Some L*))
}})›

lemma *vmtf-enqueue-alt-def*:
‹*RETURN ooo vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
let de = *defined-lit M* (*Pos L*);
case fst-As of
None ⇒ *RETURN* (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
(*if de then None else Some L*))
| *Some fst-As* ⇒
let fst-As' = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*)) *in*
RETURN (*ns*[*L* := *VMTF-Node* (*m*+*1*) *None* (*Some fst-As*), *fst-As* := *fst-As'*],
m+*1*, *L*, *the lst-As*, (*if de then next-search else Some L*))})›
unfolding *vmtf-enqueue-def Let-def*
by (*auto intro*!: *ext split*: *option.splits*)

lemma *isa-vmtf-enqueue*:
‹(*uncurry2 isa-vmtf-enqueue*, *uncurry2* (*RETURN ooo vmtf-enqueue*)) ∈
$[\lambda((M, L), \text{-}).\ L \in\# \mathcal{A}]_f$ (*trail-pol* $\mathcal{A}$) $\times_f$ *nat-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
proof −
have *defined-atm-pol*: ‹(*defined-atm-pol x1g x2f*, *defined-lit x1a* (*Pos x2*)) ∈ *Id*›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*M*, *L*) ⇒ λ-. *L* ∈# $\mathcal{A}$) *xa*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ *Id*› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1f* = (*x1g*, *x2f*)› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x2i* = (*x1j*, *x2j*)› **and**
‹*x2h* = (*x1i*, *x2i*)› **and**
‹*x2g* = (*x1h*, *x2h*)› **and**
‹*x* = (*x1f*, *x2g*)›
for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h x1i x2i x1j x2j x1k x2k*
proof −
have [*simp*]: ‹*defined-lit x1a* (*Pos x2*) ⟷ *defined-atm x1a x2*›
using *that* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *trail-pol-def defined-atm-def*)

show *?thesis*
using *undefined-atm-code*[*THEN fref-to-Down*, *unfolded uncurry-def*, *of* $\mathcal{A}$ ‹(*x1a*, *x2*)› ‹(*x1g*, *x2f*)›]
that **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *RETURN-def*)
qed

show *?thesis*
unfolding *isa-vmtf-enqueue-def vmtf-enqueue-alt-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by (*rule defined-atm-pol-pre*) *auto*
apply (*rule defined-atm-pol*; *assumption*)
apply (*rule same-in-Id-option-rel*)
subgoal for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h x1i x2i x1j x2j x1k x2k*
by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *partition-vmtf-nth* :: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*›
where
‹*partition-vmtf-nth ns* = *partition-main* (≤) (λ*n*. *stamp* (*ns* ! *n*))›

definition *partition-between-ref-vmtf* :: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*› **where**
‹*partition-between-ref-vmtf ns* = *partition-between-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*))›

definition *quicksort-vmtf-nth* :: ‹*nat-vmtf-node list* × ′*c* ⇒ *nat list* ⇒ *nat list nres*› **where**
‹*quicksort-vmtf-nth* = (λ(*ns*, -). *full-quicksort-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*)))›

definition *quicksort-vmtf-nth-ref*:: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat list nres*› **where**
‹*quicksort-vmtf-nth-ref ns a b c* =
quicksort-ref (≤) (λ*n*. *stamp* (*ns* ! *n*)) (*a*, *b*, *c*)›

lemma (**in** −) *partition-vmtf-nth-code-helper*:
assumes ‹∀ *x*∈*set ba*. *x* < *length a*› **and**
‹*b* < *length ba*› **and**
mset: ‹*mset ba* = *mset a2'*› **and**
‹*a1'* < *length a2'*›
shows ‹*a2'* ! *b* < *length a*›
using *nth-mem*[*of b a2'*] *mset-eq-setD*[*OF mset*] *mset-eq-length*[*OF mset*] *assms*
by (*auto simp del*: *nth-mem*)

lemma *partition-vmtf-nth-code-helper3*:
‹∀ *x*∈*set b*. *x* < *length a* ⟹
x'e < *length a2'* ⟹
mset a2' = *mset b* ⟹
a2' ! *x'e* < *length a*›
using *mset-eq-setD nth-mem* **by** *blast*

definition (**in** −) *isa-vmtf-en-dequeue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-en-dequeue* = (λ*M L vm*. *isa-vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *isa-vmtf-en-dequeue*:
‹(*uncurry2 isa-vmtf-en-dequeue*, *uncurry2* (*RETURN ooo vmtf-en-dequeue*)) ∈
[λ((*M*, *L*), -). *L* ∈# $\mathcal{A}$]$_f$ (*trail-pol* $\mathcal{A}$) $\times_f$ *nat-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
unfolding *isa-vmtf-en-dequeue-def vmtf-en-dequeue-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b ba ae af ag ah bb ai bc aj ak al am bd*
by (*rule order.trans*,
rule isa-vmtf-enqueue[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*,
of ai bc ‹*vmtf-dequeue bc* (*aj*, *ak*, *al*, *am*, *bd*)›])
auto
done

definition *isa-vmtf-en-dequeue-pre* :: ‹(*trail-pol* × *nat*) × *vmtf* ⇒ *bool*› **where**
‹*isa-vmtf-en-dequeue-pre* = (λ((*M*, *L*),(*ns*,*m*,*fst-As*, *lst-As*, *next-search*)).
L < *length ns* ∧ *vmtf-dequeue-pre* (*L*, *ns*) ∧
fst-As < *length ns* ∧ (*get-next* (*ns* ! *fst-As*) ≠ *None* ⟶ *get-prev* (*ns* ! *lst-As*) ≠ *None*) ∧
(*get-next* (*ns* ! *fst-As*) = *None* ⟶ *fst-As* = *lst-As*) ∧
m+*1* ≤ *uint64-max*)›

lemma *isa-vmtf-en-dequeue-preD*:
assumes ‹*isa-vmtf-en-dequeue-pre* ((*M*, *ah*), *a*, *aa*, *ab*, *ac*, *b*)›
shows ‹*ah* < *length a*› **and** ‹*vmtf-dequeue-pre* (*ah*, *a*)›
using *assms* **by** (*auto simp*: *isa-vmtf-en-dequeue-pre-def*)

lemma *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*:
‹*isa-vmtf-en-dequeue-pre* ((*M*, *L*), *a*, *st*, *fst-As*, *lst-As*, *next-search*) ⟹
vmtf-enqueue-pre ((*M*, *L*), *vmtf-dequeue L* (*a*, *st*, *fst-As*, *lst-As*, *next-search*))›
unfolding *vmtf-enqueue-pre-def*
apply *clarify*

apply (*intro conjI*)
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def ns-vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
done

lemma *insert-sort-reorder-list*:
assumes *trans*: ‹$\bigwedge$ *x y z*. ⟦*R* (*h x*) (*h y*); *R* (*h y*) (*h z*)⟧ $\Longrightarrow$ *R* (*h x*) (*h z*)› **and** *lin*: ‹$\bigwedge$*x y*. *R* (*h x*) (*h y*) $\vee$ *R* (*h y*) (*h x*)›
shows ‹(*full-quicksort-ref R h*, *reorder-list vm*) $\in$ ⟨*Id*⟩*list-rel* $\rightarrow_f$ ⟨*Id*⟩ *nres-rel*›
proof −
show *?thesis*
apply (*intro frefI nres-relI*)
apply (*rule full-quicksort-ref-full-quicksort*[*THEN fref-to-Down*, *THEN order-trans*])
using *assms* **apply** *fast*
using *assms* **apply** *fast*
apply *fast*
apply *assumption*
using *assms*
apply (*auto 5 5 simp*: *reorder-list-def intro*!: *full-quicksort-correct*[*THEN order-trans*])
done
qed

lemma *quicksort-vmtf-nth-reorder*:
‹(*uncurry quicksort-vmtf-nth*, *uncurry reorder-list*) $\in$
Id $\times_r$ ⟨*Id*⟩*list-rel* $\rightarrow_f$ ⟨*Id*⟩ *nres-rel*›
apply (*intro WB-More-Refinement.frefI nres-relI*)
subgoal for *x y*
using *insert-sort-reorder-list*[*unfolded fref-def nres-rel-def*, *of*
‹($\leq$)› ‹(λ*n*. *stamp* (*fst* (*fst y*) ! *n*) :: *nat*)› ‹*fst y*›]
by (*cases x*, *cases y*)
(*fastforce simp*: *quicksort-vmtf-nth-def uncurry-def WB-More-Refinement.fref-def*)
done

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) $\in$
nat-rel $\times_r$ *atoms-hash-rel* $\mathcal{A}$ $\rightarrow_f$ ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *atoms-hash-del-def atoms-hash-rel-def*)

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›

```
  by (auto simp: current-stamp-def intro!: ext)

lemma vmtf-rescale-alt-def:
‹vmtf-rescale = (λ(ns, m, fst-As, lst-As :: nat, next-search). do {
    (ns, m, -) ← WHILE_T^λ-. True
      (λ(ns, n, lst-As). lst-As ≠None)
      (λ(ns, n, a). do {
         ASSERT(a ≠ None);
         ASSERT(n+1 ≤ uint32-max);
         ASSERT(the a < length ns);
         let m = the a;
         let c = ns ! m;
         let nc = get-next c;
         let pc = get-prev c;
         RETURN (ns[m := VMTF-Node n pc nc], n + 1, pc)
      })
      (ns, 0, Some lst-As);
    RETURN ((ns, m, fst-As, lst-As, next-search))
  })›
  unfolding update-stamp.simps Let-def vmtf-rescale-def by auto

definition vmtf-reorder-list-raw where
  ‹vmtf-reorder-list-raw = (λvm to-remove. do {
    ASSERT(∀ x∈set to-remove. x < length vm);
    reorder-list vm to-remove
  })›


definition vmtf-reorder-list where
  ‹vmtf-reorder-list = (λ(vm, -) to-remove. do {
    vmtf-reorder-list-raw vm to-remove
  })›

definition isa-vmtf-flush-int :: ‹trail-pol ⇒ - ⇒ - nres› where
‹isa-vmtf-flush-int  = (λM (vm, (to-remove, h)). do {
    ASSERT(∀ x∈set to-remove. x < length (fst vm));
    ASSERT(length to-remove ≤ uint32-max);
    to-remove' ← vmtf-reorder-list vm to-remove;
    ASSERT(length to-remove' ≤ uint32-max);
    vm ← (if length to-remove' ≥ uint64-max − fst (snd vm)
      then vmtf-rescale vm else RETURN vm);
    ASSERT(length to-remove' + fst (snd vm) ≤ uint64-max);
   (-, vm, h) ← WHILE_T^λ(i, vm', h). i ≤ length to-remove' ∧ fst (snd vm') = i + fst (snd vm) ∧          (i < length to-remove
      (λ(i, vm, h). i < length to-remove')
      (λ(i, vm, h). do {
         ASSERT(i < length to-remove');
  ASSERT(isa-vmtf-en-dequeue-pre ((M, to-remove'!i), vm));
         vm ← isa-vmtf-en-dequeue M (to-remove'!i) vm;
  ASSERT(atoms-hash-del-pre (to-remove'!i) h);
         RETURN (i+1, vm, atoms-hash-del (to-remove'!i) h)})
      (0, vm, h);
    RETURN (vm, (emptied-list to-remove', h))
  })›
```

lemma *isa-vmtf-flush-int*:
‹(*uncurry isa-vmtf-flush-int*, *uncurry* (*vmtf-flush-int* $\mathcal{A}$)) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *Id* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
proof −
have *vmtf-flush-int-alt-def*:
‹*vmtf-flush-int* $\mathcal{A}_{in}$ = (λ*M* (*vm*, (*to-remove*, *h*)). *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length* (*fst vm*));
ASSERT(*length to-remove* ≤ *uint32-max*);
to-remove' ← *reorder-list vm to-remove*;
ASSERT(*length to-remove'* ≤ *uint32-max*);
vm ← (*if length to-remove'* + *fst* (*snd vm*) ≥ *uint64-max*
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove'* + *fst* (*snd vm*) ≤ *uint64-max*);
(-, *vm*, *h*) ← $WHILE_T{}^{\lambda(i,\ vm',\ h).\ i \le length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge\ \ (i < length\ to\text{-}remove'\ -}$
(λ(*i*, *vm*, *h*). *i* < *length to-remove'*)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove'*);
ASSERT(*to-remove'*!*i* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove'*!*i*) *h*);
vm ← *RETURN*(*vmtf-en-dequeue M* (*to-remove'*!*i*) *vm*);
RETURN (*i*+*1*, *vm*, *atoms-hash-del* (*to-remove'*!*i*) *h*)})
(*0*, *vm*, *h*);
RETURN (*vm*, (*emptied-list to-remove'*, *h*))
})› **for** $\mathcal{A}_{in}$
unfolding *vmtf-flush-int-def*
by *auto*

have *reorder-list*: ‹*vmtf-reorder-list x1d x1e*
≤ ⇓ *Id*
(*reorder-list x1a x1b*)›
if
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *Id*› **and** ‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x* = (*x1c*, *x2c*)› **and**
‹∀ *x*∈*set x1b*. *x* < *length* (*fst x1a*)› **and**
‹*length x1b* ≤ *uint32-max*› **and**
‹∀ *x*∈*set x1e*. *x* < *length* (*fst x1d*)› **and**
‹*length x1e* ≤ *uint32-max*›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *that* **unfolding** *vmtf-reorder-list-def* **by** (*cases x1a*)
(*auto intro*!: *ASSERT-leI simp*: *reorder-list-def vmtf-reorder-list-raw-def*)

have *vmtf-rescale*: ‹*vmtf-rescale x1d*
≤ ⇓ *Id*
(*vmtf-rescale x1a*)›
if
‹*True*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *Id*› **and** ‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x* = (*x1c*, *x2c*)› **and**

‹∀ x∈set x1b. x < length (fst x1a)› **and**
‹length x1b ≤ uint32-max› **and**
‹∀ x∈set x1e. x < length (fst x1d)› **and**
‹length x1e ≤ uint32-max› **and**
‹(to-remove', to-remove'a) ∈ Id› **and**
‹length to-remove'a ≤ uint32-max› **and**
‹length to-remove' ≤ uint32-max› **and**
‹uint64-max ≤ length to-remove'a + fst (snd x1a)›
for x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove' to-remove'a
using that **by** auto

have loop-rel: ‹((0, vm, x2e), 0, vma, x2b) ∈ Id›
if
‹(x, y) ∈ trail-pol $\mathcal{A}$ $\times_f$ Id› **and**
‹x2a = (x1b, x2b)› **and**
‹x2 = (x1a, x2a)› **and**
‹y = (x1, x2)› **and**
‹x2d = (x1e, x2e)› **and**
‹x2c = (x1d, x2d)› **and**
‹x = (x1c, x2c)› **and**
‹∀ x∈set x1b. x < length (fst x1a)› **and**
‹length x1b ≤ uint32-max› **and**
‹∀ x∈set x1e. x < length (fst x1d)› **and**
‹length x1e ≤ uint32-max› **and**
‹(to-remove', to-remove'a) ∈ Id› **and**
‹length to-remove'a ≤ uint32-max› **and**
‹length to-remove' ≤ uint32-max› **and**
‹(vm, vma) ∈ Id› **and**
‹length to-remove'a + fst (snd vma) ≤ uint64-max›
‹case (0, vma, x2b) of
(i, vm', h) ⇒
i ≤ length to-remove'a ∧
fst (snd vm') = i + fst (snd vma) ∧
(i < length to-remove'a ⟶
vmtf-en-dequeue-pre $\mathcal{A}$ ((x1, to-remove'a ! i), vm'))›
for x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove' to-remove'a vm
vma
using that **by** auto
have isa-vmtf-en-dequeue-pre:
‹vmtf-en-dequeue-pre $\mathcal{A}$ ((M, L), x) ⟹ isa-vmtf-en-dequeue-pre ((M', L), x)› **for** x M M' L
unfolding vmtf-en-dequeue-pre-def isa-vmtf-en-dequeue-pre-def
by auto
have isa-vmtf-en-dequeue: ‹isa-vmtf-en-dequeue x1c (to-remove' ! x1h) x1i
≤ SPEC
(λc. (c, vmtf-en-dequeue x1 (to-remove'a ! x1f) x1g)
∈ Id)›
if
‹(x, y) ∈ trail-pol $\mathcal{A}$ $\times_f$ Id› **and**
‹∀ x∈set x1b. x < length (fst x1a)› **and**
‹length x1b ≤ uint32-max› **and**
‹∀ x∈set x1e. x < length (fst x1d)› **and**
‹length x1e ≤ uint32-max› **and**
‹length to-remove'a ≤ uint32-max› **and**
‹length to-remove' ≤ uint32-max› **and**
‹length to-remove'a + fst (snd vma) ≤ uint64-max› **and**
‹case xa of (i, vm, h) ⇒ i < length to-remove'› **and**

⟨*case* x' *of* $(i, vm, h) \Rightarrow i <$ *length to-remove'a*⟩ **and**
⟨*case xa of*
$(i, vm', h) \Rightarrow$
$i \leq$ *length to-remove'* $\wedge$
fst (*snd* vm') $= i +$ *fst* (*snd vm*) $\wedge$
$(i <$ *length to-remove'* $\longrightarrow$
isa-vmtf-en-dequeue-pre ((*x1c*, *to-remove'* ! *i*), vm'))⟩ **and**
⟨*case* x' *of*
$(i, vm', h) \Rightarrow$
$i \leq$ *length to-remove'a* $\wedge$
fst (*snd* vm') $= i +$ *fst* (*snd vma*) $\wedge$
$(i <$ *length to-remove'a* $\longrightarrow$
vmtf-en-dequeue-pre $\mathcal{A}$ ((*x1*, *to-remove'a* ! *i*), vm'))⟩ **and**
⟨*isa-vmtf-en-dequeue-pre* ((*x1c*, *to-remove'* ! *x1h*), *x1i*)⟩ **and**
⟨*x1f* $<$ *length to-remove'a*⟩ **and**
⟨*to-remove'a* ! *x1f* $\in\#$ $\mathcal{A}$⟩ **and**
⟨*x1h* $<$ *length to-remove'*⟩ **and**
⟨*x2a* = (*x1b*, *x2b*)⟩ **and**
⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*y* = (*x1*, *x2*)⟩ **and**
⟨*x* = (*x1c*, *x2c*)⟩ **and**
⟨*x2d* = (*x1e*, *x2e*)⟩ **and**
⟨*x2c* = (*x1d*, *x2d*)⟩ **and**
⟨*x2f* = (*x1g*, *x2g*)⟩ **and**
⟨x' = (*x1f*, *x2f*)⟩ **and**
⟨*x2h* = (*x1i*, *x2i*)⟩ **and**
⟨*xa* = (*x1h*, *x2h*)⟩ **and**
⟨(*to-remove'*, *to-remove'a*) $\in$ *Id*⟩ **and**
⟨(*xa*, x') $\in$ *Id*⟩ **and**
⟨(*vm*, *vma*) $\in$ *Id*⟩
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove' to-remove'a vm vma xa x' x1f x2f x1g x2g x1h x2h x1i* **and** *x2i* :: ⟨*bool list*⟩
using *isa-vmtf-en-dequeue*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*, *of x1* ⟨*to-remove'a* ! *x1f*⟩ *x1g x1c* ⟨*to-remove'* ! *x1h*⟩ *x1i*] *that*
by (*auto simp*: *RETURN-def*)

show *?thesis*
unfolding *isa-vmtf-flush-int-def uncurry-def vmtf-flush-int-alt-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule reorder-list*; *assumption*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule vmtf-rescale*; *assumption*)
subgoal
by *auto*
subgoal
by *auto*
apply (*rule loop-rel*; *assumption*)
subgoal

by *auto*
subgoal
by *auto*
subgoal
by (*auto intro*!: *isa-vmtf-en-dequeue-pre*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by *auto*
apply (*rule isa-vmtf-en-dequeue*; *assumption*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove′ to-remove′a vm vma xa x′ x1f x2f x1g x2g x1h x2h x1i x2i vmb vmc*
by *auto*
subgoal
by *auto*
subgoal
by *auto*
done
qed

definition *atms-hash-insert-pre* :: ‹*nat* ⇒ *nat list* × *bool list* ⇒ *bool*› **where**
‹*atms-hash-insert-pre i* = (λ(*n*, *xs*). *i* < *length xs* ∧ (¬*xs*!*i* ⟶ *length n* < *2* + *uint32-max div 2*))›

definition *atoms-hash-insert* :: ‹*nat* ⇒ *nat list* × *bool list* ⇒ (*nat list* × *bool list*)› **where**
‹*atoms-hash-insert i* = (λ(*n*, *xs*). *if xs* ! *i then* (*n*, *xs*) *else* (*n* @ [*i*], *xs*[*i* := *True*]))›

lemma *bounded-included-le*:
assumes *bounded*: ‹*isasat-input-bounded* 𝒜› **and** ‹*distinct n*› **and**
‹*set n* ⊆ *set-mset* 𝒜 ›
shows ‹*length n* < *uint32-max*› ‹*length n* ≤ *1* + *uint32-max div 2*›
proof −
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*Pos* ‘# *mset n*)› **and**
dist: ‹*distinct n*›
using *assms*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def distinct-atoms-rel-alt-def inj-on-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset n*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset n*))›
by (*auto simp*: *tautology-decomp*)

show ‹*length n* < *uint32-max*› ‹*length n* ≤ *1* + *uint32-max div 2*›
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *uint32-max-def*)
qed

lemma *atms-hash-insert-pre*:
assumes ‹*L* ∈# 𝒜› **and** ‹(*x*, *x*′) ∈ *distinct-atoms-rel* 𝒜› **and** ‹*isasat-input-bounded* 𝒜›
shows ‹*atms-hash-insert-pre L x*›
using *assms bounded-included-le*[*OF assms*(*3*), *of* ‹*L* # *fst x*›]
by (*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def atms-hash-insert-pre-def*)

lemma *atoms-hash-del-op-set-insert*:
‹(*uncurry* (*RETURN oo atoms-hash-insert*),
uncurry (*RETURN oo insert*)) ∈
$[\lambda(i, xs).\ i \in\# \mathcal{A}_{in} \wedge \mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}]_f$
$\mathit{nat\text{-}rel} \times_r \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}_{in} \rightarrow \langle \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}_{in}\rangle \mathit{nres\text{-}rel}$›
by (*intro frefI nres-relI*)
(*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def intro*!: *ASSERT-leI*)

definition (**in** −) *atoms-hash-set-member* **where**
‹*atoms-hash-set-member i xs* = *do* {*ASSERT*(*i* < *length xs*); *RETURN* (*xs* ! *i*)}›

definition *isa-vmtf-mark-to-rescore*
:: ‹*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*›
where
‹*isa-vmtf-mark-to-rescore L* = (λ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
((*ns*, *m*, *fst-As*, *next-search*), *atoms-hash-insert L to-remove*))›

definition *isa-vmtf-mark-to-rescore-pre* **where**
‹*isa-vmtf-mark-to-rescore-pre* = (λ*L* ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
atms-hash-insert-pre L to-remove)›

lemma *isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*:
‹(*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore*), *uncurry* (*RETURN oo vmtf-mark-to-rescore*)) ∈
$[\lambda(L, vm).\ L\in\# \mathcal{A}_{in} \wedge \mathit{isasat\text{-}input\text{-}bounded}\ \mathcal{A}_{in}]_f\ \mathit{Id} \times_f (\mathit{Id} \times_r \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}_{in}) \rightarrow$
$\langle \mathit{Id} \times_r \mathit{distinct\text{-}atoms\text{-}rel}\ \mathcal{A}_{in}\rangle \mathit{nres\text{-}rel}$›
unfolding *isa-vmtf-mark-to-rescore-def vmtf-mark-to-rescore-def*
by (*intro frefI nres-relI*)
(*auto intro*!: *atoms-hash-del-op-set-insert*[*THEN fref-to-Down-unRET-uncurry*])

definition (**in** −) *isa-vmtf-unset* :: ‹*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*› **where**
‹*isa-vmtf-unset* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
(*if next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)
then ((*ns*, *m*, *fst-As*, *lst-As*, *Some L*), *to-remove*)
else ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)))›

definition *vmtf-unset-pre* **where**
‹*vmtf-unset-pre* = (λ*L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
L < *length ns* ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length ns*))›

lemma *vmtf-unset-pre-vmtf*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹*L* ∈# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›
using *assms*
by (*auto simp*: *vmtf-def vmtf-unset-pre-def atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *vmtf-unset-pre*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* ∈# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*)›

using *assms vmtf-unset-pre-vmtf*[*of ns m fst-As lst-As next-search* - $\mathcal{A}$ *M L*]
unfolding *isa-vmtf-def vmtf-unset-pre-def*
by *auto*

lemma *vmtf-unset-pre′*:
assumes
‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and**
‹*L* ∈# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L vm*›
using *assms* **by** (*cases vm*) (*auto dest*: *vmtf-unset-pre*)

definition *isa-vmtf-mark-to-rescore-and-unset* :: ‹*nat* ⇒ *isa-vmtf-remove-int* ⇒ *isa-vmtf-remove-int*›
where
‹*isa-vmtf-mark-to-rescore-and-unset L M* = *isa-vmtf-mark-to-rescore L* (*isa-vmtf-unset L M*)›

definition *isa-vmtf-mark-to-rescore-and-unset-pre* **where**
‹*isa-vmtf-mark-to-rescore-and-unset-pre* = (λ(*L*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*)).
vmtf-unset-pre L ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *tor*) ∧
atms-hash-insert-pre L tor)›

lemma *size-conflict-int-size-conflict*:
‹(*RETURN o size-conflict-int*, *RETURN o size-conflict*) ∈ $[\lambda D.\ D \neq None]_f$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *size-conflict-int-def size-conflict-def option-lookup-clause-rel-def*
lookup-clause-rel-def)

definition *rescore-clause*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *vmtf-remove-int* ⇒
(*vmtf-remove-int*) *nres*›
where
‹*rescore-clause* $\mathcal{A}$ *C M vm* = *SPEC* (λ(*vm′*). *vm′* ∈ *vmtf* $\mathcal{A}$ *M*)›

lemma *isa-vmtf-unset-vmtf-unset*:
‹(*uncurry* (*RETURN oo isa-vmtf-unset*), *uncurry* (*RETURN oo vmtf-unset*)) ∈
nat-rel $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\rightarrow_f$
⟨(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)⟩*nres-rel*›
unfolding *vmtf-unset-def isa-vmtf-unset-def uncurry-def*
by (*intro frefI nres-relI*) *auto*

lemma *isa-vmtf-unset-isa-vmtf*:
assumes ‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and** ‹*L* ∈# $\mathcal{A}$›
shows ‹*isa-vmtf-unset L vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*›
proof −
obtain *vm0 to-remove to-remove′* **where**
vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
vm0: ‹(*vm0*, *to-remove′*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹(*to-remove*, *to-remove′*) ∈ *distinct-atoms-rel* $\mathcal{A}$›
using *assms* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def*)

then show *?thesis*
using *assms*

isa-vmtf-unset-vmtf-unset[*of* $\mathcal{A}$, *THEN fref-to-Down-unRET-uncurry*, *of L vm L* ‹(*vm0*, *to-remove*′)›]
abs-vmtf-ns-unset-vmtf-unset[*of* ‹*fst vm0*› ‹*fst* (*snd vm0*)› ‹*fst* (*snd* (*snd vm0*))›
‹*fst* (*snd* (*snd* (*snd vm0*)))› ‹*snd* (*snd* (*snd* (*snd vm0*)))› *to-remove*′ $\mathcal{A}$ *M L to-remove*′]
by (*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *intro*: *isa-vmtfI elim*!: *prod-relE*)
qed

lemma *isa-vmtf-tl-isa-vmtf*:
assumes ‹*vm* ∈ *isa-vmtf* $\mathcal{A}$ *M*› **and** ‹*M* ≠ []› **and** ‹*lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*L* = (*atm-of* (*lit-of* (*hd M*)))›
shows ‹*isa-vmtf-unset L vm* ∈ *isa-vmtf* $\mathcal{A}$ (*tl M*)›
proof −
let *?L* = ‹*atm-of* (*lit-of* (*hd M*))›
obtain *vm0 to-remove to-remove*′ **where**
vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
vm0: ‹(*vm0*, *to-remove*′) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹(*to-remove*, *to-remove*′) ∈ *distinct-atoms-rel* $\mathcal{A}$›
using *assms* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def*)

then show *?thesis*
using *assms*
isa-vmtf-unset-vmtf-unset[*of* $\mathcal{A}$, *THEN fref-to-Down-unRET-uncurry*, *of ?L vm ?L* ‹(*vm0*, *to-remove*′)›]
vmtf-unset-vmtf-tl[*of* ‹*fst vm0*› ‹*fst* (*snd vm0*)› ‹*fst* (*snd* (*snd vm0*))›
‹*fst* (*snd* (*snd* (*snd vm0*)))› ‹*snd* (*snd* (*snd* (*snd vm0*)))› *to-remove*′ $\mathcal{A}$ *M*]
by (*cases M*)
(*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *intro*: *isa-vmtfI elim*!: *prod-relE*)
qed

definition *isa-vmtf-find-next-undef* :: ‹*isa-vmtf-remove-int* ⇒ *trail-pol* ⇒ (*nat option*) *nres*› **where**
‹*isa-vmtf-find-next-undef* = (λ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) *M*. *do* {
$WHILE_T$^*λnext-search. next-search ≠ None ⟶ defined-atm-pol-pre M (the next-search)*
(λ*next-search*. *next-search* ≠ *None* ∧ *defined-atm-pol M* (*the next-search*))
(λ*next-search*. *do* {
ASSERT(*next-search* ≠ *None*);
let n = *the next-search*;
ASSERT (*n* < *length ns*);
RETURN (*get-next* (*ns*!*n*))
}
)
next-search
})›

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
‹(*uncurry isa-vmtf-find-next-undef*, *uncurry* (*vmtf-find-next-undef* $\mathcal{A}$)) ∈
(*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\times_r$ *trail-pol* $\mathcal{A}$ $\rightarrow_f$ ⟨⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
unfolding *isa-vmtf-find-next-undef-def vmtf-find-next-undef-def uncurry-def*
defined-atm-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by (*rule defined-atm-pol-pre*) (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
subgoal
by (*auto simp*: *undefined-atm-code*[*THEN fref-to-Down-unRET-uncurry-Id*])
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
done

10.2 Bumping

definition *vmtf-rescore-body*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf-remove-int* ⇒
(*nat* × *vmtf-remove-int*) *nres*›
where
‹*vmtf-rescore-body* $\mathcal{A}_{in}$ *C* - *vm* = *do* {
  $WHILE_T{}^{\lambda(i,\ vm).\ i \leq length\ C\ \wedge\ (\forall c \in set\ C.\ atm\text{-}of\ c < length\ (fst\ (fst\ vm)))}$
    (λ(*i*, *vm*). *i* < *length C*)
    (λ(*i*, *vm*). *do* {
      ASSERT(*i* < *length C*);
      ASSERT(*atm-of* (*C*!*i*) ∈# $\mathcal{A}_{in}$);
      let vm′ = *vmtf-mark-to-rescore* (*atm-of* (*C*!*i*)) *vm*;
      RETURN(*i*+*1*, *vm′*)
    })
    (*0*, *vm*)
}›

definition *vmtf-rescore*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf-remove-int* ⇒
(*vmtf-remove-int*) *nres*›
where
‹*vmtf-rescore* $\mathcal{A}_{in}$ *C M vm* = *do* {
  (-, *vm*) ← *vmtf-rescore-body* $\mathcal{A}_{in}$ *C M vm*;
  RETURN (*vm*)
}›

find-theorems *isa-vmtf-mark-to-rescore*

definition *isa-vmtf-rescore-body*
:: ‹*nat clause-l* ⇒ *trail-pol* ⇒ *isa-vmtf-remove-int* ⇒
(*nat* × *isa-vmtf-remove-int*) *nres*›
where
‹*isa-vmtf-rescore-body C* - *vm* = *do* {
  $WHILE_T{}^{\lambda(i,\ vm).\ i \leq length\ C\ \wedge\ (\forall c \in set\ C.\ atm\text{-}of\ c < length\ (fst\ (fst\ vm)))}$
    (λ(*i*, *vm*). *i* < *length C*)
    (λ(*i*, *vm*). *do* {
      ASSERT(*i* < *length C*);
      ASSERT(*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*C*!*i*)) *vm*);
      let vm′ = *isa-vmtf-mark-to-rescore* (*atm-of* (*C*!*i*)) *vm*;
      RETURN(*i*+*1*, *vm′*)
    })
    (*0*, *vm*)
}›

definition *isa-vmtf-rescore*
:: ‹*nat clause-l* ⇒ *trail-pol* ⇒ *isa-vmtf-remove-int* ⇒
(*isa-vmtf-remove-int*) *nres*›
where
‹*isa-vmtf-rescore C M vm* = *do* {
  (-, *vm*) ← *isa-vmtf-rescore-body C M vm*;
  RETURN (*vm*)

}›

lemma *vmtf-rescore-score-clause*:
‹(*uncurry2* (*vmtf-rescore* $\mathcal{A}$), *uncurry2* (*rescore-clause* $\mathcal{A}$)) ∈
[λ((*C*, *M*), *vm*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*]$_f$
(⟨*Id*⟩*list-rel* $\times_f$ *Id* $\times_f$ *Id*) → ⟨*Id*⟩ *nres-rel*›
proof −
have *H*: ‹*vmtf-rescore-body* $\mathcal{A}$ *C M vm* ≤
SPEC (λ(*n* :: *nat*, *vm'*). *vm'* ∈ *vmtf* $\mathcal{A}$ *M*)›
if *M*: ‹*vm* ∈ *vmtf* $\mathcal{A}$ *M*› **and** *C*: ‹∀ *c*∈*set C*. *atm-of c* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
for *C vm φ M*
unfolding *vmtf-rescore-body-def vmtf-mark-to-rescore-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*i*, -). *length C* − *i*)› **and**
I' = ‹λ(*i*, *vm'*). *vm'* ∈ *vmtf* $\mathcal{A}$ *M*›])
subgoal by *auto*
subgoal by *auto*
subgoal using *C M* **by** (*auto simp*: *vmtf-def phase-saving-def*)
subgoal using *C M* **by** *auto*
subgoal using *M* **by** *auto*
subgoal using *C* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal using *C* **by** *auto*
subgoal using *C* **by** *auto*
subgoal using *C* **by** (*auto simp*: *vmtf-append-remove-iff'*)
subgoal by *auto*
done
have *K*: ‹((*a*, *b*),(*a'*, *b'*)) ∈ *A* $\times_f$ *B* ⟷ (*a*, *a'*) ∈ *A* ∧ (*b*, *b'*) ∈ *B*› **for** *a b a' b' A B*
by *auto*
show *?thesis*
unfolding *vmtf-rescore-def rescore-clause-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule bind-refine-spec*)
prefer *2*
apply (*subst* (*asm*) *K*)
apply (*rule H*; *auto*)
subgoal
by (*meson atm-of-lit-in-atms-of contra-subsetD in-all-lits-of-m-ain-atms-of-iff*
in-multiset-in-set literals-are-in-$\mathcal{L}_{in}$*-def*)
subgoal by *auto*
done
qed

lemma *isa-vmtf-rescore-body*:
‹(*uncurry2* (*isa-vmtf-rescore-body*), *uncurry2* (*vmtf-rescore-body* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
(*Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)) → ⟨*Id* $\times_r$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)⟩ *nres-rel*›
proof −
show *?thesis*
unfolding *isa-vmtf-rescore-body-def vmtf-rescore-body-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x1g x2g*
by (*cases x2g*) *auto*
subgoal by *auto*
subgoal by *auto*

subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e x1g x2g*
  unfolding *isa-vmtf-mark-to-rescore-pre-def*
  by (*cases x2e*)
    (*auto intro*!: *atms-hash-insert-pre*)
subgoal
by (*auto intro*!: *isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
done
qed

lemma *isa-vmtf-rescore*:
‹(*uncurry2* (*isa-vmtf-rescore*), *uncurry2* (*vmtf-rescore* $\mathcal{A}$)) $\in$ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
  (*Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)) $\rightarrow$ $\langle$(*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)$\rangle$ *nres-rel*›
proof −
show *?thesis*
  unfolding *isa-vmtf-rescore-def vmtf-rescore-def uncurry-def*
  apply (*intro frefI nres-relI*)
  apply (*refine-rcg isa-vmtf-rescore-body*[*THEN fref-to-Down-curry2*])
  subgoal by *auto*
  subgoal by *auto*
  done
qed

definition *vmtf-mark-to-rescore-clause* **where**
‹*vmtf-mark-to-rescore-clause* $\mathcal{A}_{in}$ *arena C vm* = *do* {
  ASSERT(*arena-is-valid-clause-idx arena C*);
  nfoldli
    ([*C*..<*C* + (*arena-length arena C*)])
    (λ-. *True*)
    (λ*i vm*. *do* {
      ASSERT(*i* < *length arena*);
      ASSERT(*arena-lit-pre arena i*);
      ASSERT(*atm-of* (*arena-lit arena i*) $\in\#$ $\mathcal{A}_{in}$);
      RETURN (*vmtf-mark-to-rescore* (*atm-of* (*arena-lit arena i*)) *vm*)
    })
    vm
}›

definition *isa-vmtf-mark-to-rescore-clause* **where**
‹*isa-vmtf-mark-to-rescore-clause arena C vm* = *do* {
  ASSERT(*arena-is-valid-clause-idx arena C*);
  nfoldli
    ([*C*..<*C* + (*arena-length arena C*)])
    (λ-. *True*)
    (λ*i vm*. *do* {
      ASSERT(*i* < *length arena*);
      ASSERT(*arena-lit-pre arena i*);
      ASSERT(*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*arena-lit arena i*)) *vm*);
      RETURN (*isa-vmtf-mark-to-rescore* (*atm-of* (*arena-lit arena i*)) *vm*)
    })
    vm
}›

lemma *isa-vmtf-mark-to-rescore-clause-vmtf-mark-to-rescore-clause*:

‹(*uncurry2 isa-vmtf-mark-to-rescore-clause*, *uncurry2* (*vmtf-mark-to-rescore-clause* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
Id $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) → ⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
unfolding *isa-vmtf-mark-to-rescore-clause-def vmtf-mark-to-rescore-clause-def*
uncurry-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg nfoldli-refine*[**where** R = ‹*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$› **and** S = *Id*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c xi xa si s*
by (*cases s*)
(*auto simp*: *isa-vmtf-mark-to-rescore-pre-def*
intro!: *atms-hash-insert-pre*)
subgoal
by (*rule isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
auto
done

lemma *vmtf-mark-to-rescore-clause-spec*:
‹*vm* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹
(∀ *C* ∈ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
vmtf-mark-to-rescore-clause $\mathcal{A}$ *arena C vm* ≤ *RES* (*vmtf* $\mathcal{A}$ *M*)›
unfolding *vmtf-mark-to-rescore-clause-def*
apply (*subst RES-SPEC-conv*)
apply (*refine-vcg nfoldli-rule*[**where** I = ‹λ- - *vm*. *vm* ∈ *vmtf* $\mathcal{A}$ *M*›])
subgoal
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-def*
apply (*rule exI*[*of - N*])
apply (*rule exI*[*of - vdom*])
apply (*fastforce simp*: *arena-lifting*)
done
subgoal for *x it σ*
using *arena-lifting*(*7*)[*of arena N vdom C* ‹*x* − *C*›]
by (*auto simp*: *arena-lifting*(*1*−*6*) *dest*!: *in-list-in-setD*)
subgoal for *x it σ*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
apply (*rule exI*[*of - C*])
apply (*intro conjI*)
apply (*solves* ‹*auto dest*: *in-list-in-setD*›)
apply (*rule exI*[*of - N*])
apply (*rule exI*[*of - vdom*])
apply (*fastforce simp*: *arena-lifting dest*: *in-list-in-setD*)
done
subgoal for *x it σ*
by *fastforce*
subgoal for *x it - σ*
by (*cases σ*)
(*auto intro*!: *vmtf-mark-to-rescore simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
dest: *in-list-in-setD*)
done

definition *vmtf-mark-to-rescore-also-reasons*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *arena* ⇒ *nat literal list* ⇒ - ⇒-› **where**
‹*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$ *M arena outl vm* = *do* {
  ASSERT(*length outl* ≤ *uint32-max*);
  nfoldli
   ([*0*..<*length outl*])
   (λ-. *True*)
   (λ*i vm*. *do* {
    ASSERT(*i* < *length outl*); *ASSERT*(*length outl* ≤ *uint32-max*);
    ASSERT(−*outl* ! *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
    C ← *get-the-propagation-reason M* (−(*outl* ! *i*));
    case C of
     None ⇒ *RETURN* (*vmtf-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*)
    | *Some C* ⇒ *if C* = *0 then RETURN vm else vmtf-mark-to-rescore-clause* $\mathcal{A}$ *arena C vm*
   })
   vm
}›

definition *isa-vmtf-mark-to-rescore-also-reasons*
:: ‹*trail-pol* ⇒ *arena* ⇒ *nat literal list* ⇒ - ⇒-› **where**
‹*isa-vmtf-mark-to-rescore-also-reasons M arena outl vm* = *do* {
  ASSERT(*length outl* ≤ *uint32-max*);
  nfoldli
   ([*0*..<*length outl*])
   (λ-. *True*)
   (λ*i vm*. *do* {
    ASSERT(*i* < *length outl*); *ASSERT*(*length outl*≤ *uint32-max*);
    C ← *get-the-propagation-reason-pol M* (−(*outl* ! *i*));
    case C of
     None ⇒ *do* {
      ASSERT (*isa-vmtf-mark-to-rescore-pre* (*atm-of* (*outl* ! *i*)) *vm*);
      RETURN (*isa-vmtf-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*)
 }
    | *Some C* ⇒ *if C* = *0 then RETURN vm else isa-vmtf-mark-to-rescore-clause arena C vm*
   })
   vm
}›

lemma *isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons*:
‹(*uncurry3 isa-vmtf-mark-to-rescore-also-reasons*, *uncurry3* (*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$)) ∈
 [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
 trail-pol $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) → ⟨*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
unfolding *isa-vmtf-mark-to-rescore-also-reasons-def vmtf-mark-to-rescore-also-reasons-def*
 uncurry-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg nfoldli-refine*[**where** *R* = ‹*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$› **and** *S* = *Id*]
 get-the-propagation-reason-pol[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*]
 isa-vmtf-mark-to-rescore-clause-vmtf-mark-to-rescore-clause[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply *assumption*

subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xi xa si s xb x'*
  by (*cases s*)
   (*auto simp*: *isa-vmtf-mark-to-rescore-pre-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*
     intro!: *atms-hash-insert-pre*[*of* - $\mathcal{A}$])
subgoal
  by (*rule isa-vmtf-mark-to-rescore-vmtf-mark-to-rescore*[*THEN fref-to-Down-unRET-uncurry*])
   (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
subgoal by *auto*
subgoal by *auto*
done

lemma *vmtf-mark-to-rescore'*:
‹$L \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\Longrightarrow$ *vm* $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *vmtf-mark-to-rescore L vm* $\in$ *vmtf* $\mathcal{A}$ *M*›
by (*cases vm*) (*auto intro*: *vmtf-mark-to-rescore*)

lemma *vmtf-mark-to-rescore-also-reasons-spec*:
‹*vm* $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *valid-arena arena N vdom* $\Longrightarrow$ *length outl* $\leq$ *uint32-max* $\Longrightarrow$
 ($\forall L \in$ *set outl*. $L \in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$) $\Longrightarrow$
 ($\forall L \in$ *set outl*. $\forall$ *C*. (*Propagated* ($-L$) *C* $\in$ *set M* $\longrightarrow$ $C \neq 0$ $\longrightarrow$ ($C \in\#$ *dom-m N* $\wedge$
   ($\forall C \in$ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$)))) $\Longrightarrow$
 vmtf-mark-to-rescore-also-reasons $\mathcal{A}$ *M arena outl vm* $\leq$ *RES* (*vmtf* $\mathcal{A}$ *M*)›
unfolding *vmtf-mark-to-rescore-also-reasons-def*
apply (*subst RES-SPEC-conv*)
apply (*refine-vcg nfoldli-rule*[**where** *I* = ‹λ- - *vm*. *vm* $\in$ *vmtf* $\mathcal{A}$ *M*›])
subgoal by (*auto dest*: *in-list-in-setD*)
subgoal for *x l1 l2 σ*
  unfolding *all-set-conv-nth*
  by (*auto simp*: *uminus-$\mathcal{A}_{in}$-iff dest*!: *in-list-in-setD*)
subgoal for *x l1 l2 σ*
  unfolding *get-the-propagation-reason-def*
  apply (*rule SPEC-rule*)
  apply (*rename-tac reason*, *case-tac reason*; *simp only*: *option.simps RES-SPEC-conv*[*symmetric*])
  subgoal
    by (*auto simp*: *vmtf-mark-to-rescore'*
     in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff[*symmetric*])
  apply (*rename-tac D*, *case-tac* ‹*D* = *0*›; *simp*)
  subgoal
    by (*rule vmtf-mark-to-rescore-clause-spec*, *assumption*, *assumption*)
     fastforce+
  done
done

10.3 Backtrack level for Restarts

We here find out how many decisions can be reused. Remark that since VMTF does not reuse many levels anyway, the implementation might be mostly useless, but I was not aware of that when I implemented it.

definition *find-decomp-w-ns-pre* **where**
‹*find-decomp-w-ns-pre* $\mathcal{A}$ = (λ((*M*, *highest*), *vm*).
   no-dup M $\wedge$
   highest < *count-decided M* $\wedge$
   isasat-input-bounded $\mathcal{A}$ $\wedge$
   literals-are-in-$\mathcal{L}_{in}$-trail $\mathcal{A}$ *M* $\wedge$
   vm $\in$ *vmtf* $\mathcal{A}$ *M*)›

definition *find-decomp-wl-imp*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf-remove-int* ⇒
((*nat*, *nat*) *ann-lits* × *vmtf-remove-int*) *nres*›
where
‹*find-decomp-wl-imp* $\mathcal{A}$ = (λM_0 *lev vm*. *do* {
let k = *count-decided* M_0;
let M_0 = *trail-conv-to-no-CS* M_0;
let n = *length* M_0;
pos ← *get-pos-of-level-in-trail* M_0 *lev*;
ASSERT((*n* − *pos*) ≤ *uint32-max*);
ASSERT(*n* ≥ *pos*);
let target = *n* − *pos*;
(-, *M*, *vm'*) ←
$WHILE_T{}^{\lambda(j,\ M,\ vm').\ j \leq target\ \wedge\ M = drop\ j\ M_0\ \wedge\ target \leq length\ M_0\ \wedge\ vm' \in vmtf\ \mathcal{A}\ M\ \wedge\ literals\text{-}ar}$
(λ(*j*, *M*, *vm*). *j* < *target*)
(λ(*j*, *M*, *vm*). *do* {
ASSERT(*M* ≠ []);
ASSERT(*Suc j* ≤ *uint32-max*);
let L = *atm-of* (*lit-of-hd-trail M*);
ASSERT(*L* ∈# $\mathcal{A}$);
RETURN (*j* + *1*, *tl M*, *vmtf-unset L vm*)
})
(*0*, M_0, *vm*);
ASSERT(*lev* = *count-decided M*);
let M = *trail-conv-back lev M*;
RETURN (*M*, *vm'*)
})›

definition *isa-find-decomp-wl-imp*
:: ‹*trail-pol* ⇒ *nat* ⇒ *isa-vmtf-remove-int* ⇒ (*trail-pol* × *isa-vmtf-remove-int*) *nres*›
where
‹*isa-find-decomp-wl-imp* = (λM_0 *lev vm*. *do* {
let k = *count-decided-pol* M_0;
let M_0 = *trail-pol-conv-to-no-CS* M_0;
ASSERT(*isa-length-trail-pre* M_0);
let n = *isa-length-trail* M_0;
pos ← *get-pos-of-level-in-trail-imp* M_0 *lev*;
ASSERT((*n* − *pos*) ≤ *uint32-max*);
ASSERT(*n* ≥ *pos*);
let target = *n* − *pos*;
(-, *M*, *vm'*) ←
$WHILE_T{}^{\lambda(j,\ M,\ vm').\ j \leq target}$
(λ(*j*, *M*, *vm*). *j* < *target*)
(λ(*j*, *M*, *vm*). *do* {
ASSERT(*Suc j* ≤ *uint32-max*);
ASSERT(*case M of* (*M*, -) ⇒ *M* ≠ []);
ASSERT(*tl-trailt-tr-no-CS-pre M*);
let L = *atm-of* (*lit-of-last-trail-pol M*);
ASSERT(*vmtf-unset-pre L vm*);
RETURN (*j* + *1*, *tl-trailt-tr-no-CS M*, *isa-vmtf-unset L vm*)
})
(*0*, M_0, *vm*);
M ← *trail-conv-back-imp lev M*;
RETURN (*M*, *vm'*)

})›

abbreviation *find-decomp-w-ns-prop* **where**
‹*find-decomp-w-ns-prop* $\mathcal{A}$ ≡
(λ(*M*::(*nat*, *nat*) *ann-lits*) *highest* -.
(λ(*M1*, *vm*). ∃ *K M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
get-level M K = *Suc highest* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M1*))›

definition *find-decomp-w-ns* **where**
‹*find-decomp-w-ns* $\mathcal{A}$ =
(λ(*M*::(*nat*, *nat*) *ann-lits*) *highest vm*.
SPEC(*find-decomp-w-ns-prop* $\mathcal{A}$ *M highest vm*))›

lemma *isa-find-decomp-wl-imp-find-decomp-wl-imp*:
‹(*uncurry2 isa-find-decomp-wl-imp*, *uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$)) ∈
[λ((*M*, *lev*), *vm*). *lev* < *count-decided M*]$_f$ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)
→
⟨*trail-pol* $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)⟩*nres-rel*›
proof −
have [*intro*]: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ (*M'*, *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$› **for** *M' M*
by (*auto simp*: *trail-pol-def trail-pol-no-CS-def control-stack-length-count-dec*[*symmetric*])

have [*refine0*]: ‹((*0*, *trail-pol-conv-to-no-CS x1c*, *x2c*),
0, *trail-conv-to-no-CS x1a*, *x2a*)
∈ *nat-rel* $\times_r$ *trail-pol-no-CS* $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$)›
if
‹*case y of*
(*x*, *xa*) ⇒ (*case x of* (*M*, *lev*) ⇒ λ-. *lev* < *count-decided M*) *xa*› **and**
‹(*x*, *y*)
∈ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1b* = (*x1c*, *x2b*)› **and**
‹*x* = (*x1b*, *x2c*)› **and**
‹*isa-length-trail-pre* (*trail-pol-conv-to-no-CS x1c*)› **and**
‹(*pos*, *posa*) ∈ *nat-rel*› **and**
‹*length* (*trail-conv-to-no-CS x1a*) − *posa* ≤ *uint32-max*› **and**
‹*isa-length-trail* (*trail-pol-conv-to-no-CS x1c*) − *pos* ≤ *uint32-max*› **and**
‹*case* (*0*, *trail-conv-to-no-CS x1a*, *x2a*) *of*
(*j*, *M*, *vm'*) ⇒
j ≤ *length* (*trail-conv-to-no-CS x1a*) − *posa* ∧
M = *drop j* (*trail-conv-to-no-CS x1a*) ∧
length (*trail-conv-to-no-CS x1a*) − *posa*
≤ *length* (*trail-conv-to-no-CS x1a*) ∧
vm' ∈ *vmtf* $\mathcal{A}$ *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset M*)›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa*
proof −
show *?thesis*
supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
using *that* **by** *auto*
qed
have *trail-pol-empty*: ‹((([], *x2g*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ ⟹ *M* = []› **for** *M x2g*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def*)

have *isa-vmtf*: ‹(*x2c*, *x2a*) ∈ *Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$ ⟹
(((*aa*, *ab*, *ac*, *ad*, *ba*), *baa*, *ca*), *x2e*) ∈ *Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$ ⟹

x2e ∈ *vmtf* $\mathcal{A}$ (*drop x1d x1a*) $\Longrightarrow$
((*aa*, *ab*, *ac*, *ad*, *ba*), *baa*, *ca*) ∈ *isa-vmtf* $\mathcal{A}$ (*drop x1d x1a*)›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f x1g x1h x2g x2h aa ab ac ad ba baa ca*
by (*cases x2e*)
(*auto 6 6 simp*: *isa-vmtf-def Image-iff converse-iff prod-rel-iff*
intro!: *bexI*[*of - x2e*])
have *trail-pol-no-CS-last-hd*:
‹((*x1h*, *t*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ $\Longrightarrow$ *M* ≠ [] $\Longrightarrow$ (*last x1h*) = *lit-of* (*hd M*)›
for *x1h t M*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def last-map last-rev*)

have *trail-conv-back*: ‹*trail-conv-back-imp x2b x1g*
≤ *SPEC*
(λ*c*. (*c*, *trail-conv-back x2 x1e*)
∈ *trail-pol* $\mathcal{A}$)›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*M*, *lev*) ⇒ λ*vm*. *lev* < *count-decided M*) *xa*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$)› **and**
‹*x1* = (*x1a*, *x2*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1b* = (*x1c*, *x2b*)› **and**
‹*x* = (*x1b*, *x2c*)› **and**
‹*isa-length-trail-pre* (*trail-pol-conv-to-no-CS x1c*)› **and**
‹(*pos*, *posa*) ∈ *nat-rel*› **and**
‹*length* (*trail-conv-to-no-CS x1a*) − *posa* ≤ *uint32-max*› **and**
‹*isa-length-trail* (*trail-pol-conv-to-no-CS x1c*) − *pos* ≤ *uint32-max*› **and**
‹(*xa*, *x'*) ∈ *nat-rel* $\times_f$ (*trail-pol-no-CS* $\mathcal{A}$ $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel* $\mathcal{A}$))› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x'* = (*x1d*, *x2d*)› **and**
‹*x2f* = (*x1g*, *x2g*)› **and**
‹*xa* = (*x1f*, *x2f*)› **and**
‹*x2* = *count-decided x1e*›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f x1g x2g*
apply (*rule trail-conv-back*[*THEN fref-to-Down-curry*, *THEN order-trans*])
using *that* **by** (*auto simp*: *conc-fun-RETURN*)

show *?thesis*
supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
unfolding *isa-find-decomp-wl-imp-def find-decomp-wl-imp-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
id-trail-conv-to-no-CS[*THEN fref-to-Down*, *unfolded comp-def*]
get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*])
subgoal
by (*rule isa-length-trail-pre*) *auto*
subgoal
by (*auto simp*: *get-pos-of-level-in-trail-pre-def*)
subgoal
by *auto*
subgoal
by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
subgoal
by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*

apply (*assumption*+)[*10*]
subgoal
by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
subgoal
by (*subst isa-length-trail-length-u-no-CS*[*THEN fref-to-Down-unRET-Id*]) *auto*
subgoal
by (*auto dest*!: *trail-pol-empty*)
subgoal
by (*auto dest*!: *trail-pol-empty*)
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa*
by (*rule tl-trailt-tr-no-CS-pre*) *auto*
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x′ x1d x2d x1e x2e x1f x2f x1g x1h x2g x2h*
by (*cases x1g*, *cases x2h*)
(*auto intro*!: *vmtf-unset-pre*[*of - - - - - - 𝒜* ‹*drop x1d x1a*›] *isa-vmtf*
simp: *lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def*)
subgoal
by (*auto simp*: *lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def*
intro!: *tl-trail-tr-no-CS*[*THEN fref-to-Down-unRET*]
isa-vmtf-unset-vmtf-unset[*THEN fref-to-Down-unRET-uncurry*])
apply (*rule trail-conv-back*; *assumption*)
subgoal
by *auto*
done
qed

definition (**in** −) *find-decomp-wl-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *nat twl-st-wl nres*› **where**
‹*find-decomp-wl-st* = (λ*L* (*M*, *N*, *D*, *oth*). *do*{
M′ ← *find-decomp-wl′ M* (*the D*) *L*;
RETURN (*M′*, *N*, *D*, *oth*)
})›

definition *find-decomp-wl-st-int* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*find-decomp-wl-st-int* = (λ*highest* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*). *do*{
(*M′*, *vm*) ← *isa-find-decomp-wl-imp M highest vm*;
RETURN (*M′*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *stats*)
})›

lemma
assumes
vm: ‹*vm* ∈ *vmtf* 𝒜 M_0› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* 𝒜 M_0› **and**
target: ‹*highest* < *count-decided* M_0› **and**
n-d: ‹*no-dup* M_0› **and**
bounded: ‹*isasat-input-bounded* 𝒜›
shows
find-decomp-wl-imp-le-find-decomp-wl′:
‹*find-decomp-wl-imp* 𝒜 M_0 *highest vm* ≤ *find-decomp-w-ns* 𝒜 M_0 *highest vm*›
(**is** *?decomp*)
proof −
have *length-M0*: ‹*length* M_0 ≤ *uint32-max div 2* + *1*›
using *length-trail-uint32-max-div2*[*of* 𝒜 M_0, *OF bounded*]
n-d literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of* 𝒜, *OF lits*]
by (*auto simp*: *lits-of-def*)

have *1*: ‹((*count-decided x1g*, *x1g*), *count-decided x1*, *x1*) ∈ *Id*›
  if ‹*x1g* = *x1*› **for** *x1g x1* :: ‹(*nat*, *nat*) *ann-lits*›
  using *that* **by** *auto*
have [*simp*]: ‹∃ *M'a*. *M'* @ *x2g* = *M'a* @ *tl x2g*› **for** *M' x2g* :: ‹(*nat*, *nat*) *ann-lits*›
  by (*rule exI*[*of* - ‹*M'* @ (*if x2g* = [] *then* [] *else* [*hd x2g*])›]) *auto*
have *butlast-nil-iff*: ‹*butlast xs* = [] ⟷ *xs* = [] ∨ (∃ *a*. *xs* = [*a*])› **for** *xs* :: ‹(*nat*, *nat*) *ann-lits*›
  by (*cases xs*) *auto*
have *butlast1*: ‹*tl x2g* = *drop* (*Suc* (*length x1*) − *length x2g*) *x1*› (**is** ‹*?G1*›)
  if ‹*x2g* = *drop* (*length x1* − *length x2g*) *x1*› **for** *x2g x1* :: ‹*'a list*›
proof −
  have [*simp*]: ‹*Suc* (*length x1* − *length x2g*) = *Suc* (*length x1*) − *length x2g*›
    by (*metis Suc-diff-le diff-le-mono2 diff-zero length-drop that zero-le*)
  show *?G1*
    by (*subst that*) (*auto simp*: *butlast-conv-take tl-drop-def*)[]
qed
have *butlast2*: ‹*tl x2g* = *drop* (*length x1* − (*length x2g* − *Suc 0*)) *x1*› (**is** ‹*?G2*›)
  if ‹*x2g* = *drop* (*length x1* − *length x2g*) *x1*› **and** *x2g*: ‹*x2g* ≠ []› **for** *x2g x1* :: ‹*'a list*›
proof −
  have [*simp*]: ‹*Suc* (*length x1* − *length x2g*) = *Suc* (*length x1*) − *length x2g*›
    by (*metis Suc-diff-le diff-le-mono2 diff-zero length-drop that*(*1*) *zero-le*)
  have [*simp*]: ‹*Suc* (*length x1*) − *length x2g* = *length x1* − (*length x2g* − *Suc 0*)›
    using *x2g* **by** *auto*
  show *?G2*
    by (*subst that*) (*auto simp*: *butlast-conv-take tl-drop-def*)[]
qed
note *butlast* = *butlast1 butlast2*

have *count-decided-not-Nil*[*simp*]: ‹*0* < *count-decided M* ⟹ *M* ≠ []› **for** *M* :: ‹(*nat*, *nat*) *ann-lits*›
  by *auto*
have *get-lev-last*: ‹*get-level* (*M'* @ *M*) (*lit-of* (*last M'*)) = *Suc* (*count-decided M*)›
  if ‹M_0 = *M'* @ *M*› **and** ‹*M'* ≠ []› **and** ‹*is-decided* (*last M'*)› **for** *M' M*
  apply (*cases M' rule*: *rev-cases*)
  using *that* **apply** (*solves simp*)
  using *n-d that* **by** *auto*

have *atm-of-N*:
 ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset aa*) ⟹ *aa* ≠ [] ⟹ *atm-of* (*lit-of* (*hd aa*)) ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
  for *aa*
  by (*cases aa*) (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
have *Lin-drop-tl*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset* (*drop b* M_0)) ⟹
    literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset* (*tl* (*drop b* M_0)))› **for** *b*
  apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-mono*)
   apply *assumption*
  by (*cases* ‹*drop b* M_0›) *auto*

have *highest*: ‹*highest* = *count-decided M*› **and**
   ex-decomp: ‹∃ *K M2*.
     (*Decided K* # *M*, *M2*)
     ∈ *set* (*get-all-ann-decomposition* M_0) ∧
     get-level M_0 *K* = *Suc highest* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*›
  if
   pos: ‹*pos* < *length* M_0 ∧ *is-decided* (*rev* M_0 ! *pos*) ∧ *get-level* M_0 (*lit-of* (*rev* M_0 ! *pos*)) =
      highest + *1*› **and**
   ‹*length* M_0 − *pos* ≤ *uint32-max*› **and**
   inv: ‹*case s of* (*j*, *M*, *vm'*) ⇒
      j ≤ *length* M_0 − *pos* ∧

M = *drop j* M_0 ∧
length M_0 − *pos* ≤ *length* M_0 ∧
vm′ ∈ *vmtf* $\mathcal{A}$ *M* ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset M*)› **and**
cond: ‹¬ (*case s of*
(*j*, *M*, *vm*) ⇒ *j* < *length* M_0 − *pos*)› **and**
s: ‹*s* = (*j*, *s′*)› ‹*s′* = (*M*, *vm*)›
for *pos s j s′ M vm*
proof −
have
‹*j* = *length* M_0 − *pos*› **and**
M: ‹*M* = *drop* (*length* M_0 − *pos*) M_0› **and**
vm: ‹*vm* ∈ *vmtf* $\mathcal{A}$ (*drop* (*length* M_0 − *pos*) M_0)› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of* ‘# *mset* (*drop* (*length* M_0 − *pos*) M_0))›
using *cond inv* **unfolding** *s*
by *auto*
define *M2* **and** *L* **where** ‹*M2* = *take* (*length* M_0 − *Suc pos*) M_0› **and** ‹*L* = *rev* M_0 ! *pos*›
have *le-Suc-pos*: ‹*length* M_0 − *pos* = *Suc* (*length* M_0 − *Suc pos*)›
using *pos* **by** *auto*
have *1*: ‹*take* (*length* M_0 − *pos*) M_0 = *take* (*length* M_0 − *Suc pos*) M_0 @ [*rev* M_0 ! *pos*]›
unfolding *le-Suc-pos*
apply (*subst take-Suc-conv-app-nth*)
using *pos* **by** (*auto simp*: *rev-nth*)
have M_0: ‹M_0 = *M2* @ *L* # *M*›
apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* M_0 − *pos*›])
unfolding *M L-def M2-def 1*
by *auto*
have *L′*: ‹*Decided* (*lit-of L*) = *L*›
using *pos* **unfolding** *L-def*[*symmetric*] **by** (*cases L*) *auto*
then have M_0*′*: ‹M_0 = *M2* @ *Decided* (*lit-of L*) # *M*›
unfolding M_0 **by** *auto*

have ‹*highest* = *count-decided M*› **and** ‹*get-level* M_0 (*lit-of L*) = *Suc highest*› **and** ‹*is-decided L*›
using *n-d pos* **unfolding** *L-def*[*symmetric*] **unfolding** M_0
by (*auto simp*: *get-level-append-if split*: *if-splits*)
then show
‹∃ *K M2*.
(*Decided K* # *M*, *M2*)
∈ *set* (*get-all-ann-decomposition* M_0) ∧
get-level M_0 *K* = *Suc highest* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*›
using *get-all-ann-decomposition-ex*[*of* ‹*lit-of L*› *M M2*] *vm* **unfolding** M_0*′*[*symmetric*] *M*[*symmetric*]
by *blast*
show ‹*highest* = *count-decided M*›
using ‹*highest* = *count-decided M*› **.**
qed
show *?decomp*
unfolding *find-decomp-wl-imp-def Let-def find-decomp-w-ns-def trail-conv-to-no-CS-def*
get-pos-of-level-in-trail-def trail-conv-back-def
apply (*refine-vcg 1 WHILEIT-rule*[**where** *R*=‹*measure* (λ(-, *M*, -). *length M*)›])
subgoal using *length-M0* **unfolding** *uint32-max-def* **by** *simp*
subgoal by *auto*
subgoal by *auto*
subgoal using *target* **by** (*auto simp*: *count-decided-ge-get-maximum-level*)
subgoal by *auto*
subgoal by *auto*
subgoal using *vm* **by** *auto*

subgoal using *lits* **unfolding** *literals-are-in-$\mathcal{L}_{in}$-trail-lit-of-mset* **by** *auto*
subgoal for *target s j b M vm* **by** *simp*
subgoal using *length-M0* **unfolding** *uint32-max-def* **by** *simp*
subgoal for *x s a ab aa bb*
by (*cases* ‹*drop a* M_0›)
(*auto simp*: *lit-of-hd-trail-def literals-are-in-$\mathcal{L}_{in}$-add-mset*)
subgoal by *auto*
subgoal by (*auto simp*: *drop-Suc drop-tl*)
subgoal by *auto*
subgoal for *s a b aa ba vm x2 x1a x2a*
by (*cases vm*)
(*auto intro*!: *vmtf-unset-vmtf-tl atm-of-N drop-tl simp*: *lit-of-hd-trail-def*)
subgoal for *s a b aa ba x1 x2 x1a x2a*
using *lits* **by** (*auto intro*: *Lin-drop-tl*)
subgoal by *auto*
subgoal by (*rule highest*)
subgoal by (*rule ex-decomp*) (*assumption*+, *auto*)
done
qed

lemma *find-decomp-wl-imp-find-decomp-wl′*:
‹(*uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$), *uncurry2* (*find-decomp-w-ns* $\mathcal{A}$)) $\in$
[*find-decomp-w-ns-pre* $\mathcal{A}$]$_f$ *Id* $\times_f$ *Id* $\times_f$ *Id* $\rightarrow$ $\langle$*Id* $\times_f$ *Id*$\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *find-decomp-w-ns-pre-def simp del*: *twl-st-of-wl.simps*
intro!: *find-decomp-wl-imp-le-find-decomp-wl′*)

lemma *find-decomp-wl-imp-code-conbine-cond*:
‹(λ((*b*, *a*), *c*). *find-decomp-w-ns-pre* $\mathcal{A}$ ((*b*, *a*), *c*) $\wedge$ *a* < *count-decided b*) = (λ((*b*, *a*), *c*).
find-decomp-w-ns-pre $\mathcal{A}$ ((*b*, *a*), *c*))›
by (*auto intro*!: *ext simp*: *find-decomp-w-ns-pre-def*)

end
theory *IsaSAT-Sorting*
imports *IsaSAT-Setup*
begin

Chapter 11

Sorting of clauses

We use the sort function developped by Peter Lammich.

definition *clause-score-ordering* **where**
‹$clause\text{-}score\text{-}ordering = (\lambda(lbd, act)\ (lbd', act').\ lbd < lbd' \lor (lbd = lbd' \land act < act'))$›

definition (**in** −) *clause-score-extract* :: ‹*arena* ⇒ *nat* ⇒ *nat* × *nat*› **where**
‹*clause-score-extract arena C* = (
  if arena-status arena C = *DELETED*
  then (*uint32-max*, *0*) — deleted elements are the largest possible
  else
    let lbd = *arena-lbd arena C in*
    let act = *arena-act arena C in*
    (*lbd*, *act*)
)›

definition *valid-sort-clause-score-pre-at* **where**
‹*valid-sort-clause-score-pre-at arena C* ⟷
  (∃ *i vdom*. *C* = *vdom* ! *i* ∧ *arena-is-valid-clause-vdom arena* (*vdom*!*i*) ∧
      (*arena-status arena* (*vdom*!*i*) ≠ *DELETED* ⟶
        (*get-clause-LBD-pre arena* (*vdom*!*i*) ∧ *arena-act-pre arena* (*vdom*!*i*)))
      ∧ *i* < *length vdom*)›

definition (**in** −)*valid-sort-clause-score-pre* **where**
‹*valid-sort-clause-score-pre arena vdom* ⟷
  (∀ *C* ∈ *set vdom*. *arena-is-valid-clause-vdom arena C* ∧
    (*arena-status arena C* ≠ *DELETED* ⟶
      (*get-clause-LBD-pre arena C* ∧ *arena-act-pre arena C*)))›

definition *clause-score-less* :: *arena* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
clause-score-less arena i j ⟷
  clause-score-ordering (*clause-score-extract arena i*) (*clause-score-extract arena j*)

definition *idx-cdom* :: *arena* ⇒ *nat set* **where**
idx-cdom arena ≡ {*i*. *valid-sort-clause-score-pre-at arena i*}

definition *mop-clause-score-less* **where**
‹*mop-clause-score-less arena i j* = *do* {
  ASSERT(*valid-sort-clause-score-pre-at arena i*);
  ASSERT(*valid-sort-clause-score-pre-at arena j*);
  RETURN (*clause-score-ordering* (*clause-score-extract arena i*) (*clause-score-extract arena j*))

}›

end
theory *IsaSAT-Sorting-LLVM*
imports *IsaSAT-Sorting IsaSAT-Setup-LLVM*
Isabelle-LLVM.Sorting-Introsort
begin

no-notation *WB-More-Refinement.fref* ($[-]_f$ - → - [0,60,60] 60)
no-notation *WB-More-Refinement.freft* (- $\rightarrow_f$ - [60,60] 60)
declare *α-butlast*[*simp del*]

locale *pure-eo-adapter* =
fixes *elem-assn* :: *'a ⇒ 'ai::llvm-rep ⇒ assn*
and *wo-assn* :: *'a list ⇒ 'oi::llvm-rep ⇒ assn*
and *wo-get-impl* :: *'oi ⇒ 'size::len2 word ⇒ 'ai llM*
and *wo-set-impl* :: *'oi ⇒ 'size::len2 word ⇒ 'ai ⇒ 'oi llM*
assumes *pure*[*safe-constraint-rules*]: *is-pure elem-assn*
and *get-hnr*: (*uncurry wo-get-impl*,*uncurry mop-list-get*) ∈ $wo\text{-}assn^k *_a snat\text{-}assn^k \rightarrow_a elem\text{-}assn$
and *set-hnr*: (*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ $wo\text{-}assn^d *_a snat\text{-}assn^k *_a elem\text{-}assn^k \rightarrow_{ad}$ (λ- ((*ai*,-),-). *cnc-assn* (λ*x*. *x*=*ai*) *wo-assn*)
begin

lemmas [*sepref-fr-rules*] = *get-hnr set-hnr*

definition *only-some-rel* ≡ {(*a*, *Some a*) | *a*. *True*} ∪ {(*x*, *None*) | *x*. *True*}

definition *eo-assn* ≡ *hr-comp wo-assn* (⟨*only-some-rel*⟩*list-rel*)

definition *eo-extract1 p i* ≡ *doN* { *r* ← *mop-list-get p i*; *RETURN* (*r*,*p*) }
sepref-definition *eo-extract-impl* **is** *uncurry eo-extract1*
:: $wo\text{-}assn^d *_a (snat\text{-}assn'\ TYPE('size))^k \rightarrow_a elem\text{-}assn \times_a wo\text{-}assn$
unfolding *eo-extract1-def*
by *sepref*

lemma *mop-eo-extract-aux*: *mop-eo-extract p i* = *doN* { *r* ← *mop-list-get p i*; *ASSERT* (*r*≠*None* ∧ *i*<*length p*); *RETURN* (*the r*, *p*[*i*:=*None*]) }
by (*auto simp*: *pw-eq-iff refine-pw-simps*)

lemma *assign-none-only-some-list-rel*:
assumes *SR*[*param*]: (*a*, *a'*) ∈ ⟨*only-some-rel*⟩*list-rel* **and** *L*: *i* < *length a'*
shows (*a*, *a'*[*i* := *None*]) ∈ ⟨*only-some-rel*⟩*list-rel*
proof −
have (*a*[*i* := *a*!*i*], *a'*[*i* := *None*]) ∈ ⟨*only-some-rel*⟩*list-rel*
apply (*parametricity*)
by (*auto simp*: *only-some-rel-def*)
also from *L list-rel-imp-same-length*[*OF SR*] **have** *a*[*i* := *a*!*i*] = *a* **by** *auto*
finally show *?thesis* **.**
qed

lemma *eo-extract1-refine*: (*eo-extract1*, *mop-eo-extract*) ∈ ⟨*only-some-rel*⟩*list-rel* → *nat-rel* → ⟨$Id \times_r$ ⟨*only-some-rel*⟩*list-rel*⟩*nres-rel*
unfolding *eo-extract1-def mop-eo-extract-aux*
supply *R* = *mop-list-get.fref*[*THEN frefD*, *OF TrueI prod-relI*, *unfolded uncurry-apply*, *THEN*

nres-relD]
apply (*refine-rcg R*)
apply *assumption*
apply (*clarsimp simp*: *assign-none-only-some-list-rel*)
by (*auto simp*: *only-some-rel-def*)

lemma *eo-list-set-refine*: (*mop-list-set*, *mop-eo-set*) ∈ ⟨*only-some-rel*⟩*list-rel* → *Id* → *Id* → ⟨⟨*only-some-rel*⟩*list-rel*⟩*nres-*
unfolding *mop-list-set-alt mop-eo-set-alt*
apply *refine-rcg*
apply (*simp add*: *list-rel-imp-same-length*)
apply *simp*
apply *parametricity*
apply (*auto simp*: *only-some-rel-def*)
done

lemma *set-hnr'*: (*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ *wo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{k} $\rightarrow_a$ *wo-assn*
apply (*rule hfref-cons*[*OF set-hnr*])
apply (*auto simp*: *cnc-assn-def entails-lift-extract-simps sep-algebra-simps*)
done

context
notes [*fcomp-norm-unfold*] = *eo-assn-def*[*symmetric*]
begin
lemmas *eo-extract-refine-aux* = *eo-extract-impl.refine*[*FCOMP eo-extract1-refine*]

lemma *eo-extract-refine*: (*uncurry eo-extract-impl*, *uncurry mop-eo-extract*) ∈ *eo-assn*^{d} $*_a$ *snat-assn*^{k} $\rightarrow_{ad}$ (λ- (*ai*,-). *elem-assn* $\times_a$ *cnc-assn* (λ*x*. *x*=*ai*) *eo-assn*)
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
unfolding *cnc-assn-prod-conv*
apply (*rule hnr-ceq-assnI*)
subgoal
supply *R* = *eo-extract-refine-aux*[*to-hnr*, *unfolded APP-def*]
apply (*rule hn-refine-cons*[*OF - R*])
apply (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def*)
done
subgoal
unfolding *eo-extract-impl-def mop-eo-extract-def hn-ctxt-def eo-assn-def hr-comp-def*
supply *R* = *get-hnr*[*to-hnr*, *THEN hn-refineD*, *unfolded APP-def hn-ctxt-def*]
thm *R*
supply [*vcg-rules*] = *R*
supply [*simp*] = *refine-pw-simps list-rel-imp-same-length*
apply (*vcg*)
done
done

lemmas *eo-set-refine-aux* = *set-hnr'*[*FCOMP eo-list-set-refine*]

lemma *pure-part-cnc-imp-eq*: *pure-part* (*cnc-assn* (λ*x*. *x* = *cc*) *wo-assn a c*) ⟹ *c*=*cc*
by (*auto simp*: *pure-part-def cnc-assn-def pred-lift-extract-simps*)

lemma *pure-entails-empty*: *is-pure A* $\Longrightarrow$ *A a c* $\vdash \square$
by (*auto simp*: *is-pure-def sep-algebra-simps entails-lift-extract-simps*)

lemma *eo-set-refine*: (*uncurry2 wo-set-impl*, *uncurry2 mop-eo-set*) $\in$ *eo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{d} $\rightarrow_{ad}$ (λ- ((*ai*, -), -). *cnc-assn* (λx. x = *ai*) *eo-assn*)
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
apply (*rule hnr-ceq-assnI*)
subgoal
supply *R* = *eo-set-refine-aux*[*to-hnr*, *unfolded APP-def*]
apply (*rule hn-refine-cons*[*OF* - *R*])
apply (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def pure-entails-empty*[*OF pure*])
done
subgoal
unfolding *hn-ctxt-def eo-assn-def hr-comp-def*
supply *R* = *set-hnr*[*to-hnr*, *THEN hn-refineD*, *unfolded APP-def hn-ctxt-def*]
supply [*vcg-rules*] = *R*
supply [*simp*] = *refine-pw-simps list-rel-imp-same-length pure-part-cnc-imp-eq*
apply (*vcg'*)
done
done

end

lemma *id-Some-only-some-rel*: (*id*, *Some*) $\in$ *Id* $\rightarrow$ *only-some-rel*
by (*auto simp*: *only-some-rel-def*)

lemma *map-some-only-some-rel-iff*: (*xs*, *map Some ys*) $\in$ $\langle$*only-some-rel*$\rangle$*list-rel* $\longleftrightarrow$ *xs*=*ys*
apply (*rule iffI*)
subgoal
apply (*induction xs map Some ys arbitrary*: *ys rule*: *list-rel-induct*)
apply (*auto simp*: *only-some-rel-def*)
done
subgoal
apply (*rewrite* **in** (⌑,-) *list.map-id*[*symmetric*])
apply (*parametricity add*: *id-Some-only-some-rel*)
by *simp*
done

lemma *wo-assn-conv*: *wo-assn xs ys* = *eo-assn* (*map Some xs*) *ys*
unfolding *eo-assn-def hr-comp-def*
by (*auto simp*: *pred-lift-extract-simps sep-algebra-simps fun-eq-iff map-some-only-some-rel-iff*)

lemma *to-eo-conv-refine*: (*return*, *mop-to-eo-conv*) $\in$ *wo-assn*^{d} $\rightarrow_{ad}$ (λ- *ai*. *cnc-assn* (λx. x = *ai*) *eo-assn*)
unfolding *mop-to-eo-conv-def cnc-assn-def*
apply *sepref-to-hoare*
apply (*rewrite wo-assn-conv*)
apply *vcg*
done

lemma *None* $\notin$ *set xs* $\longleftrightarrow$ ($\exists$ *ys*. *xs* = *map Some ys*) using *None-not-in-set-conv* **by** *auto*

lemma *to-wo-conv-refine*: (*return*, *mop-to-wo-conv*) $\in$ *eo-assn*^{d} $\rightarrow_{ad}$ (λ- *ai*. *cnc-assn* (λx. $x = ai$) *wo-assn*)
unfolding *mop-to-wo-conv-def cnc-assn-def eo-assn-def hr-comp-def*
apply *sepref-to-hoare*
apply (*auto simp add*: *refine-pw-simps map-some-only-some-rel-iff elim*!: *None-not-in-set-conv*)
by *vcg*

lemma *random-access-iterator*: *random-access-iterator wo-assn eo-assn elem-assn*
return return
eo-extract-impl
wo-set-impl
apply *unfold-locales*
using *to-eo-conv-refine to-wo-conv-refine eo-extract-refine eo-set-refine*
apply *blast*+
done

sublocale *random-access-iterator wo-assn eo-assn elem-assn*
return return
eo-extract-impl
wo-set-impl
by (*rule random-access-iterator*)

end

lemma *al-pure-eo*: *is-pure A* $\Longrightarrow$ *pure-eo-adapter A* (*al-assn A*) *arl-nth arl-upd*
apply *unfold-locales*
apply *assumption*
apply (*rule al-nth-hnr-mop*; *simp*)
subgoal
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
apply (*rule hnr-ceq-assnI*)
subgoal
supply R = *al-upd-hnr-mop*[*to-hnr*, *unfolded APP-def*, *of A*]
apply (*rule hn-refine-cons*[*OF - R*])
apply (*auto simp*: *hn-ctxt-def pure-def invalid-assn-def sep-algebra-simps entails-lift-extract-simps*)
done
subgoal
unfolding *hn-ctxt-def al-assn-def hr-comp-def pure-def in-snat-rel-conv-assn*
apply (*erule is-pureE*)
apply (*simp add*: *refine-pw-simps*)
supply [*simp*] = *list-rel-imp-same-length*
by *vcg*
done
done

end
theory *IsaSAT-VMTF-LLVM*
imports *Watched-Literals.WB-Sort IsaSAT-VMTF IsaSAT-Setup-LLVM*
Isabelle-LLVM.Sorting-Introsort
IsaSAT-Sorting-LLVM
begin

definition *valid-atoms* :: *nat-vmtf-node list* ⇒ *nat set* **where**
valid-atoms xs ≡ {*i*. *i* < *length xs*}

definition *VMTF-score-less* **where**
‹*VMTF-score-less xs i j* ⟷ *stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*)›

definition *mop-VMTF-score-less* **where**
‹*mop-VMTF-score-less xs i j* = *do* {
ASSERT(*i* < *length xs*);
ASSERT(*j* < *length xs*);
RETURN (*stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*))
}›

sepref-register *VMTF-score-less*

sepref-def (**in** −) *mop-VMTF-score-less-impl*
is ‹*uncurry2* (*mop-VMTF-score-less*)›
:: ‹$(array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a atom\text{-}assn^k *_a atom\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
supply [[*goals-limit* = *1*]]
unfolding *mop-VMTF-score-less-def*
apply (*rewrite at* ‹*stamp* (- ! ⌑)› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹*stamp* (- ! ⌑)› **in** ‹- < ⌑› *value-of-atm-def*[*symmetric*])
unfolding *index-of-atm-def*[*symmetric*]
by *sepref*

interpretation *VMTF*: *weak-ordering-on-lt* **where**
C = *valid-atoms vs* **and**
less = *VMTF-score-less vs*
by *unfold-locales*
(*auto simp*: *VMTF-score-less-def split*: *if-splits*)

interpretation *VMTF*: *parameterized-weak-ordering valid-atoms VMTF-score-less*
mop-VMTF-score-less
by *unfold-locales*
(*auto simp*: *mop-VMTF-score-less-def*
valid-atoms-def VMTF-score-less-def)

global-interpretation *VMTF*: *parameterized-sort-impl-context*
woarray-assn atom-assn eoarray-assn atom-assn atom-assn
return return
eo-extract-impl
array-upd
valid-atoms VMTF-score-less mop-VMTF-score-less mop-VMTF-score-less-impl
array-assn vmtf-node-assn
defines
VMTF-is-guarded-insert-impl = *VMTF.is-guarded-param-insert-impl*
and *VMTF-is-unguarded-insert-impl* = *VMTF.is-unguarded-param-insert-impl*

and *VMTF-unguarded-insertion-sort-impl* = *VMTF.unguarded-insertion-sort-param-impl*
and *VMTF-guarded-insertion-sort-impl* = *VMTF.guarded-insertion-sort-param-impl*
and *VMTF-final-insertion-sort-impl* = *VMTF.final-insertion-sort-param-impl*

and *VMTF-pcmpo-idxs-impl* = *VMTF.pcmpo-idxs-impl*
and *VMTF-pcmpo-v-idx-impl* = *VMTF.pcmpo-v-idx-impl*
and *VMTF-pcmpo-idx-v-impl* = *VMTF.pcmpo-idx-v-impl*
and *VMTF-pcmp-idxs-impl* = *VMTF.pcmp-idxs-impl*

and *VMTF-mop-geth-impl* = *VMTF.mop-geth-impl*
and *VMTF-mop-seth-impl* = *VMTF.mop-seth-impl*
and *VMTF-sift-down-impl* = *VMTF.sift-down-impl*
and *VMTF-heapify-btu-impl* = *VMTF.heapify-btu-impl*
and *VMTF-heapsort-impl* = *VMTF.heapsort-param-impl*
and *VMTF-qsp-next-l-impl* = *VMTF.qsp-next-l-impl*
and *VMTF-qsp-next-h-impl* = *VMTF.qsp-next-h-impl*
and *VMTF-qs-partition-impl* = *VMTF.qs-partition-impl*

and *VMTF-partition-pivot-impl* = *VMTF.partition-pivot-impl*
and *VMTF-introsort-aux-impl* = *VMTF.introsort-aux-param-impl*
and *VMTF-introsort-impl* = *VMTF.introsort-param-impl*
and *VMTF-move-median-to-first-impl* = *VMTF.move-median-to-first-param-impl*

apply *unfold-locales*
apply (*rule eo-hnr-dep*)+
unfolding *GEN-ALGO-def refines-param-relp-def*
supply[[*unify-trace-failure*]]
by (*rule mop-VMTF-score-less-impl.refine*)

global-interpretation
VMTF-it: *pure-eo-adapter atom-assn arl64-assn atom-assn arl-nth arl-upd*
defines *VMTF-it-eo-extract-impl* = *VMTF-it.eo-extract-impl*
apply (*rule al-pure-eo*)
by (*simp add*: *safe-constraint-rules*)

global-interpretation *VMTF-it*: *parameterized-sort-impl-context*
where
wo-assn = ‹*arl64-assn atom-assn*›
and *eo-assn* = *VMTF-it.eo-assn*
and *elem-assn* = *atom-assn*
and *to-eo-impl* = *return*
and *to-wo-impl* = *return*
and *extract-impl* = *VMTF-it-eo-extract-impl*
and *set-impl* = *arl-upd*
and *cdom* = *valid-atoms*
and *pless* = *VMTF-score-less*
and *pcmp* = *mop-VMTF-score-less*
and *pcmp-impl* = *mop-VMTF-score-less-impl*
and *cparam-assn* = ‹*array-assn vmtf-node-assn*›
defines
VMTF-it-is-guarded-insert-impl = *VMTF-it.is-guarded-param-insert-impl*

and *VMTF-it-is-unguarded-insert-impl* = *VMTF-it.is-unguarded-param-insert-impl*
and *VMTF-it-unguarded-insertion-sort-impl* = *VMTF-it.unguarded-insertion-sort-param-impl*
and *VMTF-it-guarded-insertion-sort-impl* = *VMTF-it.guarded-insertion-sort-param-impl*
and *VMTF-it-final-insertion-sort-impl* = *VMTF-it.final-insertion-sort-param-impl*

and *VMTF-it-pcmpo-idxs-impl* = *VMTF-it.pcmpo-idxs-impl*
and *VMTF-it-pcmpo-v-idx-impl* = *VMTF-it.pcmpo-v-idx-impl*
and *VMTF-it-pcmpo-idx-v-impl* = *VMTF-it.pcmpo-idx-v-impl*
and *VMTF-it-pcmp-idxs-impl* = *VMTF-it.pcmp-idxs-impl*

and *VMTF-it-mop-geth-impl* = *VMTF-it.mop-geth-impl*
and *VMTF-it-mop-seth-impl* = *VMTF-it.mop-seth-impl*
and *VMTF-it-sift-down-impl* = *VMTF-it.sift-down-impl*
and *VMTF-it-heapify-btu-impl* = *VMTF-it.heapify-btu-impl*
and *VMTF-it-heapsort-impl* = *VMTF-it.heapsort-param-impl*
and *VMTF-it-qsp-next-l-impl* = *VMTF-it.qsp-next-l-impl*
and *VMTF-it-qsp-next-h-impl* = *VMTF-it.qsp-next-h-impl*
and *VMTF-it-qs-partition-impl* = *VMTF-it.qs-partition-impl*

and *VMTF-it-partition-pivot-impl* = *VMTF-it.partition-pivot-impl*
and *VMTF-it-introsort-aux-impl* = *VMTF-it.introsort-aux-param-impl*
and *VMTF-it-introsort-impl* = *VMTF-it.introsort-param-impl*
and *VMTF-it-move-median-to-first-impl* = *VMTF-it.move-median-to-first-param-impl*

apply *unfold-locales*
unfolding *GEN-ALGO-def refines-param-relp-def*
apply (*rule mop-VMTF-score-less-impl.refine*)
done

lemmas [*llvm-inline*] = *VMTF-it.eo-extract-impl-def*[*THEN meta-fun-cong*, *THEN meta-fun-cong*]

print-named-simpset *llvm-inline*
export-llvm
VMTF-heapsort-impl :: - ⇒ - ⇒ -
VMTF-introsort-impl :: - ⇒ - ⇒ -

definition *VMTF-sort-scores-raw* :: ‹-› **where**
‹*VMTF-sort-scores-raw* = *pslice-sort-spec valid-atoms VMTF-score-less*›

definition *VMTF-sort-scores* :: ‹-› **where**
‹*VMTF-sort-scores xs ys* = *VMTF-sort-scores-raw xs ys 0* (*length ys*)›

lemmas *VMTF-introsort*[*sepref-fr-rules*] =
VMTF-it.introsort-param-impl-correct[*unfolded VMTF-sort-scores-raw-def*[*symmetric*] *PR-CONST-def*]

sepref-register *VMTF-sort-scores-raw vmtf-reorder-list-raw*

lemma *VMTF-sort-scores-vmtf-reorder-list-raw*:
‹(*VMTF-sort-scores*, *vmtf-reorder-list-raw*) ∈ *Id* → *Id* → ⟨*Id*⟩*nres-rel*›
unfolding *VMTF-sort-scores-def VMTF-sort-scores-raw-def pslice-sort-spec-def*
vmtf-reorder-list-raw-def
apply (*refine-rcg*)
subgoal by (*auto simp*: *valid-atoms-def*)
subgoal for *vm vm′ arr arr′*

by (*auto intro*!: *slice-sort-spec-refine-sort*[*THEN order-trans*, *of - arr′ arr′*]
simp: *valid-atoms-def slice-rel-def br-def reorder-list-def conc-fun-RES sort-spec-def*
eq-commute[*of* ‹*length -*› ‹*length arr′*›])
done

sepref-def *VMTF-sort-scores-raw-impl*
is ‹*uncurry VMTF-sort-scores*›
:: ‹$(IICF\text{-}Array.array\text{-}assn\ vmtf\text{-}node\text{-}assn)^k *_a VMTF\text{-}it.arr\text{-}assn^d \rightarrow_a VMTF\text{-}it.arr\text{-}assn$›
unfolding *VMTF-sort-scores-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemmas[*sepref-fr-rules*] =
VMTF-sort-scores-raw-impl.refine[*FCOMP VMTF-sort-scores-vmtf-reorder-list-raw*]

sepref-def *VMTF-sort-scores-impl*
is ‹*uncurry vmtf-reorder-list*›
:: ‹$(vmtf\text{-}assn)^k *_a VMTF\text{-}it.arr\text{-}assn^d \rightarrow_a VMTF\text{-}it.arr\text{-}assn$›
unfolding *vmtf-reorder-list-def*
by *sepref*

sepref-def *atoms-hash-del-code*
is ‹*uncurry* (*RETURN oo atoms-hash-del*)›
:: ‹$[uncurry\ atoms\text{-}hash\text{-}del\text{-}pre]_a\ atom\text{-}assn^k *_a (atoms\text{-}hash\text{-}assn)^d \rightarrow atoms\text{-}hash\text{-}assn$›
unfolding *atoms-hash-del-def atoms-hash-del-pre-def*
apply *annot-all-atm-idxs*
by *sepref*

sepref-def *atoms-hash-insert-code*
is ‹*uncurry* (*RETURN oo atoms-hash-insert*)›
:: ‹$[uncurry\ atms\text{-}hash\text{-}insert\text{-}pre]_a$
$atom\text{-}assn^k *_a (distinct\text{-}atoms\text{-}assn)^d \rightarrow distinct\text{-}atoms\text{-}assn$›
unfolding *atoms-hash-insert-def atms-hash-insert-pre-def*
supply [[*goals-limit=1*]]
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *find-decomp-wl-imp*
sepref-register *rescore-clause vmtf-flush*
sepref-register *vmtf-mark-to-rescore*
sepref-register *vmtf-mark-to-rescore-clause*

sepref-register *vmtf-mark-to-rescore-also-reasons get-the-propagation-reason-pol*

sepref-register *find-decomp-w-ns*

sepref-def *update-next-search-impl*
is ‹*uncurry* (*RETURN oo update-next-search*)›
:: ‹$(atom.option\text{-}assn)^k *_a vmtf\text{-}remove\text{-}assn^d \rightarrow_a vmtf\text{-}remove\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *update-next-search-def vmtf-remove-assn-def*
by *sepref*

lemma *case-option-split*:

‹(*case a of None* ⇒ *x* | *Some y* ⇒ *f y*) =
(*if is-None a then x else let y* = *the a in f y*)›
by (*auto split*: *option.splits*)

sepref-def *ns-vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo ns-vmtf-dequeue*)›
:: ‹$[vmtf\text{-}dequeue\text{-}pre]_a$
$atom\text{-}assn^k *_a (array\text{-}assn\ vmtf\text{-}node\text{-}assn)^d \rightarrow array\text{-}assn\ vmtf\text{-}node\text{-}assn$›
supply [[*goals-limit* = *1*]]
supply *option.splits*[*split*] *if-splits*[*split*]
unfolding *ns-vmtf-dequeue-def vmtf-dequeue-pre-alt-def case-option-split atom.fold-option*
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *get-next get-prev stamp*
lemma *eq-Some-iff*: $x = Some\ b \longleftrightarrow (\neg is\text{-}None\ x \wedge the\ x = b)$
by (*cases x*) *auto*

lemma *hfref-refine-with-pre*:
assumes $\bigwedge x.\ P\ x \Longrightarrow g'\ x \leq g\ x$
assumes $(f,g') \in [P]_{ad}\ A \rightarrow R$
shows $(f,g) \in [P]_{ad}\ A \rightarrow R$
using *assms*(*2*)[*THEN hfrefD*] *assms*(*1*)
by (*auto intro*!: *hfrefI intro*: *hn-refine-ref*)

lemma *isa-vmtf-en-dequeue-preI*:
assumes *isa-vmtf-en-dequeue-pre* ((*M*,*L*),(*ns*, *m*, *fst-As*, *lst-As*, *next-search*))
shows *fst-As* < *length ns L* < *length ns Suc m* < *max-unat 64*
and *get-next* (*ns*!*L*) = *Some i* ⟶ *i* < *length ns*
and *fst-As* ≠ *lst-As* ⟶ *get-prev* (*ns* ! *lst-As*) ≠ *None*
and *get-next* (*ns* ! *fst-As*) ≠ *None* ⟶ *get-prev* (*ns* ! *lst-As*) ≠ *None*
using *assms*
unfolding *isa-vmtf-en-dequeue-pre-def vmtf-dequeue-pre-def*
apply (*auto simp*: *max-unat-def uint64-max-def sint64-max-def*)
done

find-theorems - ≠ *None* ⟷ -

lemma *isa-vmtf-en-dequeue-alt-def2*:
‹*isa-vmtf-en-dequeue-pre x* ⟹ *uncurry2* (λ*M L vm*.
case vm of (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ⇒ *doN* {
ASSERT(*L*<*length ns*);
nsL ← *mop-list-get ns* (*index-of-atm L*);
let fst-As = (*if fst-As* = *L then get-next nsL else* (*Some fst-As*));

let next-search = (*if next-search* = (*Some L*) *then get-next nsL*
else next-search);
let lst-As = (*if lst-As* = *L then get-prev nsL else* (*Some lst-As*));
ASSERT (*vmtf-dequeue-pre* (*L*,*ns*));
let ns = *ns-vmtf-dequeue L ns*;
ASSERT (*defined-atm-pol-pre M L*);

let de = (*defined-atm-pol M L*);
ASSERT (*Suc m* < *max-unat 64*);
case fst-As of
None ⇒ *RETURN*
(*ns*[*L* := *VMTF-Node m fst-As None*], *m* + *1*, *L*, *L*,
if de then None else Some L)
| *Some fst-As* ⇒ *doN* {
ASSERT (*L* < *length ns* ∧ *fst-As* < *length ns* ∧ *lst-As* ≠ *None*);
let fst-As' =
VMTF-Node (*stamp* (*ns* ! *fst-As*)) (*Some L*)
(*get-next* (*ns* ! *fst-As*));
RETURN (
ns[*L* := *VMTF-Node* (*m* + *1*) *None* (*Some fst-As*),
fst-As := *fst-As'*],
m + *1*, *L*, *the lst-As*,
if de then next-search else Some L)
}
}) *x*
≤ *uncurry2* (*isa-vmtf-en-dequeue*) *x*
⟩
unfolding *isa-vmtf-en-dequeue-def vmtf-dequeue-def isa-vmtf-enqueue-def*
annot-unat-snat-upcast[*symmetric*] *ASSN-ANNOT-def*
apply (*cases x*; *simp add*: *Let-def*)
apply (*simp*
only: *pw-le-iff refine-pw-simps*
split: *prod.splits*
)
supply *isa-vmtf-en-dequeue-preD*[*simp*]
apply (*auto*
split!: *if-splits option.splits*
simp: *refine-pw-simps isa-vmtf-en-dequeue-preI dest*: *isa-vmtf-en-dequeue-preI*
simp del: *not-None-eq*
)
done

sepref-register *1 0*

lemma *vmtf-en-dequeue-fast-codeI*:
assumes *isa-vmtf-en-dequeue-pre* ((*M*, *L*),(*ns*,*m*,*fst-As*, *lst-As*, *next-search*))
shows *Suc m* < *max-unat 64*
using *assms*
unfolding *isa-vmtf-en-dequeue-pre-def max-unat-def uint64-max-def*
by *auto*

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: *MK-FREE trail-pol-fast-assn ?fr*
unfolding *trail-pol-fast-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

sepref-def *vmtf-en-dequeue-fast-code*
is ⟨*uncurry2 isa-vmtf-en-dequeue*⟩
:: ⟨[*isa-vmtf-en-dequeue-pre*]$_a$
trail-pol-fast-assn^k $*_a$ *atom-assn*^{k} $*_a$ *vmtf-assn*^{d} → *vmtf-assn*⟩

apply (*rule hfref-refine-with-pre*[*OF isa-vmtf-en-dequeue-alt-def2*], *assumption*)

supply [[*goals-limit* = *1*]]
unfolding *isa-vmtf-en-dequeue-alt-def2 case-option-split eq-Some-iff*
apply (*rewrite* **in** *if* ¤ *then get-next - else - short-circuit-conv*)
apply *annot-all-atm-idxs*
apply (*annot-unat-const TYPE*(*64*))
unfolding *atom.fold-option*
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-register *vmtf-rescale*
sepref-def *vmtf-rescale-code*
is ⟨*vmtf-rescale*⟩
:: ⟨$vmtf\text{-}assn^d \rightarrow_a vmtf\text{-}assn$⟩
supply [[*goals-limit* = *1*]]
supply *vmtf-en-dequeue-pre-def*[*simp*]
unfolding *vmtf-rescale-alt-def update-stamp.simps*
unfolding *atom.fold-option*
apply (*annot-unat-const TYPE*(*64*))
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *partition-between-ref*

sepref-register *isa-vmtf-enqueue*

lemma *emptied-list-alt-def*: ⟨*emptied-list xs* = *take 0 xs*⟩
by (*auto simp*: *emptied-list-def*)

sepref-def *current-stamp-impl*
is ⟨*RETURN o current-stamp*⟩
:: ⟨$vmtf\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$⟩
unfolding *current-stamp-alt-def*
by *sepref*

sepref-register *isa-vmtf-en-dequeue*

sepref-def *isa-vmtf-flush-fast-code*
is ⟨*uncurry isa-vmtf-flush-int*⟩
:: ⟨$trail\text{-}pol\text{-}fast\text{-}assn^k *_a (vmtf\text{-}remove\text{-}assn)^d \rightarrow_a$
vmtf-remove-assn⟩
supply [[*goals-limit* = *1*]]
unfolding *vmtf-flush-def PR-CONST-def isa-vmtf-flush-int-def*
current-stamp-def[*symmetric*] *emptied-list-alt-def*
vmtf-remove-assn-def
apply (*rewrite at* ⟨*If* (- − - ≤ ¤) - -⟩ *annot-snat-unat-conv*)
apply (*rewrite at* ⟨*WHILEIT* - (λ(-, -, -).- < ¤)⟩ *annot-snat-unat-conv*)
apply (*rewrite at* ⟨*isa-vmtf-en-dequeue* - (- ! ¤)⟩ *annot-unat-snat-conv*)
apply (*rewrite at* ⟨*atoms-hash-del* (- ! ¤)⟩ *annot-unat-snat-conv*)

apply (*rewrite at* ⟨*take* ¤ -⟩ *snat-const-fold*[**where** ′*a*=*64*])
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

sepref-register *isa-vmtf-mark-to-rescore*
sepref-def *isa-vmtf-mark-to-rescore-code*
is ⟨*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore*)⟩
:: ⟨[*uncurry isa-vmtf-mark-to-rescore-pre*]$_a$
atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*⟩
supply [[*goals-limit*=*1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
neq-NilE[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
unfolding *isa-vmtf-mark-to-rescore-pre-def isa-vmtf-mark-to-rescore-def vmtf-remove-assn-def*
by *sepref*

sepref-register *isa-vmtf-unset*
sepref-def *isa-vmtf-unset-code*
is ⟨*uncurry* (*RETURN oo isa-vmtf-unset*)⟩
:: ⟨[*uncurry vmtf-unset-pre*]$_a$
atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*⟩
supply [[*goals-limit*=*1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
neq-NilE[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
unfolding *isa-vmtf-unset-def vmtf-unset-pre-def vmtf-remove-assn-def atom.fold-option*
apply (*rewrite* **in** ⟨*If* (- ∨ -)⟩ *short-circuit-conv*)
apply *annot-all-atm-idxs*
by *sepref*

lemma *isa-vmtf-mark-to-rescore-and-unsetI*: ⟨
atms-hash-insert-pre ak (*ad*, *ba*) ⟹
isa-vmtf-mark-to-rescore-pre ak ((*a*, *aa*, *ab*, *ac*, *Some ak*′), *ad*, *ba*)⟩
by (*auto simp*: *isa-vmtf-mark-to-rescore-pre-def*)

sepref-def *vmtf-mark-to-rescore-and-unset-code*
is ⟨*uncurry* (*RETURN oo isa-vmtf-mark-to-rescore-and-unset*)⟩
:: ⟨[*isa-vmtf-mark-to-rescore-and-unset-pre*]$_a$
atom-assn^k $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*⟩
supply *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*iff*] *in-diffD*[*dest*] *option.splits*[*split*]
if-splits[*split*] *isa-vmtf-unset-def*[*simp*] *isa-vmtf-mark-to-rescore-and-unsetI*[*intro*!]
supply [[*goals-limit*=*1*]]
unfolding *isa-vmtf-mark-to-rescore-and-unset-def isa-vmtf-mark-to-rescore-and-unset-pre-def*
save-phase-def isa-vmtf-mark-to-rescore-and-unset-pre-def
by *sepref*

sepref-def *find-decomp-wl-imp-fast-code*
is ⟨*uncurry2* (*isa-find-decomp-wl-imp*)⟩
:: ⟨[λ((*M*, *lev*), *vm*). *True*]$_a$ *trail-pol-fast-assn*^{d} $*_a$ *uint32-nat-assn*^{k} $*_a$ *vmtf-remove-assn*^{d}
→ *trail-pol-fast-assn* $\times_a$ *vmtf-remove-assn*⟩
unfolding *isa-find-decomp-wl-imp-def get-maximum-level-remove-def*[*symmetric*] *PR-CONST-def*
trail-pol-conv-to-no-CS-def
supply *trail-conv-to-no-CS-def*[*simp*] *lit-of-hd-trail-def*[*simp*]
supply [[*goals-limit*=*1*]] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
supply *vmtf-unset-pre-def*[*simp*]
apply (*rewrite at* ⟨*let* - = - − ¤ *in* -⟩ *annot-unat-snat-upcast*[**where** ′*l*=*64*])

apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *vmtf-rescore-fast-code*
is ‹*uncurry2 isa-vmtf-rescore*›
:: ‹*clause-ll-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$
vmtf-remove-assn›
unfolding *isa-vmtf-rescore-body-def*[*abs-def*] *PR-CONST-def isa-vmtf-rescore-def*
supply [[*goals-limit* = *1*]] *fold-is-None*[*simp*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *find-decomp-wl-imp′-fast-code*
is ‹*uncurry find-decomp-wl-st-int*›
:: ‹*uint32-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$
isasat-bounded-assn›
unfolding *find-decomp-wl-st-int-def PR-CONST-def isasat-bounded-assn-def*
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

lemma (**in** −) *arena-is-valid-clause-idx-le-uint64-max*:
‹*arena-is-valid-clause-idx be bd* $\Longrightarrow$
length be $\leq$ *sint64-max* $\Longrightarrow$
bd + *arena-length be bd* < *max-snat 64*›
‹*arena-is-valid-clause-idx be bd* $\Longrightarrow$ *length be* $\leq$ *sint64-max* $\Longrightarrow$
bd < *max-snat 64*›
using *arena-lifting*(*10*)[*of be - - bd*] **unfolding** *max-snat-def sint64-max-def*
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

sepref-def *vmtf-mark-to-rescore-clause-fast-code*
is ‹*uncurry2* (*isa-vmtf-mark-to-rescore-clause*)›
:: ‹[λ((*N*, -), -). *length N* $\leq$ *sint64-max*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} → *vmtf-remove-assn*›
supply [[*goals-limit=1*]] *arena-is-valid-clause-idx-le-uint64-max*[*intro*]
unfolding *isa-vmtf-mark-to-rescore-clause-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *nres-monad3*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *vmtf-mark-to-rescore-also-reasons-fast-code*
is ‹*uncurry3* (*isa-vmtf-mark-to-rescore-also-reasons*)›
:: ‹[λ(((-, *N*), -), -). *length N* $\leq$ *sint64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *out-learned-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} →
vmtf-remove-assn›
supply *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*iff*] *in-diffD*[*dest*] *option.splits*[*split*]
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$[*simp*]
supply [[*goals-limit=1*]]
unfolding *isa-vmtf-mark-to-rescore-also-reasons-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]

apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const TYPE*(*64*))
unfolding *nres-monad3 case-option-split*
by *sepref*

experiment begin

export-llvm
ns-vmtf-dequeue-code
atoms-hash-del-code
atoms-hash-insert-code
update-next-search-impl
ns-vmtf-dequeue-code
vmtf-en-dequeue-fast-code
vmtf-rescale-code
current-stamp-impl
isa-vmtf-flush-fast-code
isa-vmtf-mark-to-rescore-code
isa-vmtf-unset-code
vmtf-mark-to-rescore-and-unset-code
find-decomp-wl-imp-fast-code
vmtf-rescore-fast-code
find-decomp-wl-imp′-fast-code
vmtf-mark-to-rescore-clause-fast-code
vmtf-mark-to-rescore-also-reasons-fast-code

end

end
theory *IsaSAT-Show*
imports
Show.Show-Instances
IsaSAT-Setup
begin

Chapter 12

Printing information about progress

We provide a function to print some information about the state. This is mostly meant to ease extracting statistics and printing information during the run. Remark that this function is basically an FFI (to follow Andreas Lochbihler words) and is not unsafe (since printing has not side effects), but we do not need any correctness theorems.

However, it seems that the PolyML as targeted by *export-code checking* does not support that print function. Therefore, we cannot provide the code printing equations by default.

For the LLVM version code equations are not supported and hence we replace the function by hand.

definition *println-string* :: ‹*String.literal* ⇒ *unit*› **where**
‹*println-string* - = ()›

definition *print-c* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-c* - = ()›

definition *print-char* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-char* - = ()›

definition *print-uint64* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-uint64* - = ()›

12.0.1 Print Information for IsaSAT

Printing the information slows down the solver by a huge factor.

definition *isasat-banner-content* **where**
‹*isasat-banner-content* =
''*c conflicts decisions restarts uset avg-lbd*
'' @
''*c propagations reductions GC Learnt*
'' @
''*c clauses* ''›

definition *isasat-information-banner* :: ‹- ⇒ *unit nres*› **where**
‹*isasat-information-banner* - =
RETURN (*println-string* (*String.implode* (*show isasat-banner-content*)))›

definition *zero-some-stats* :: ‹*stats* ⇒ *stats*› **where**
‹*zero-some-stats* = (λ(*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *lbds*).

(*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *0*))›

definition *print-open-colour* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-open-colour* - = ()›

definition *print-close-colour* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-close-colour* - = ()›

definition *isasat-current-information* :: ‹*64 word* ⇒ *stats* ⇒ - ⇒ *stats*› **where**
‹*isasat-current-information* =
(λ*curr-phase* (*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *lbds*) *lcount*.
if confl AND 8191 = 8191 — (*8191*::′*b*) = (*8192*::′*b*) − (*1*::′*b*), i.e., we print when all first bits are 1.
then do{
let - = *print-c propa*;
- = *if curr-phase = 1 then print-open-colour 33 else* ();
- = *print-uint64 propa*;
- = *print-uint64 confl*;
- = *print-uint64 frestarts*;
- = *print-uint64 lrestarts*;
- = *print-uint64 uset*;
- = *print-uint64 gcs*;
- = *print-uint64 lbds*;
- = *print-close-colour 0*
in
zero-some-stats (*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *lbds*)}
else (*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *lbds*)
)›

definition *isasat-current-status* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*isasat-current-status* =
(λ(*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *avdom*,
vdom, *lcount*, *opts*, *old-arena*).
let curr-phase = *current-restart-phase heur*;
stats = (*isasat-current-information curr-phase stats lcount*)
in RETURN (*M*′, *N*′, *D*′, *j*, *W*′, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *avdom*,
vdom, *lcount*, *opts*, *old-arena*))›

lemma *isasat-current-status-id*:
‹(*isasat-current-status*, *RETURN o id*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*} $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*}⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def isasat-current-status-def*)

definition *isasat-print-progress* :: ‹*64 word* ⇒ *64 word* ⇒ *stats* ⇒ - ⇒ *unit*› **where**
‹*isasat-print-progress c curr-phase* =
(λ(*propa*, *confl*, *decs*, *frestarts*, *lrestarts*, *uset*, *gcs*, *lbds*) *lcount*.
let
- = *print-c propa*;
- = *if curr-phase = 1 then print-open-colour 33 else* ();
- = *print-char c*;
- = *print-uint64 propa*;

- = *print-uint64 confl*;
- = *print-uint64 frestarts*;
- = *print-uint64 lrestarts*;
- = *print-uint64 uset*;
- = *print-uint64 gcs*;
- = *print-close-colour 0*

in

())›

definition *isasat-current-progress* :: ‹*64 word* ⇒ *twl-st-wl-heur* ⇒ *unit nres*› **where**
‹*isasat-current-progress* =
  (λ*c* (*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
    heur, *avdom*,
    vdom, *lcount*, *opts*, *old-arena*).
  let
    curr-phase = *current-restart-phase heur*;
    - = *isasat-print-progress c curr-phase stats lcount*
  in RETURN ())›

end
theory *IsaSAT-Rephase*
  imports *IsaSAT-Setup IsaSAT-Show*
begin

Chapter 13

Rephasing

We implement the idea in CaDiCaL of rephasing:

- We remember the best model found so far. It is used as base.
- We flip the phase saving heuristics between *True*, *False*, and random.

definition *rephase-init* :: ‹*bool* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*rephase-init b* φ = *do* {
  let n = *length* φ;
  nfoldli [*0*..<*n*]
    (λ-. *True*)
    (λ *a* φ. *do* {
        ASSERT(*a* < *length* φ);
        RETURN (φ[*a* := *b*])
    })
    φ
 }›

lemma *rephase-init-spec*:
  ‹*rephase-init b* $\varphi \leq$ *SPEC*($\lambda\psi$. *length* ψ = *length* φ)›
proof −
  show *?thesis*
  unfolding *rephase-init-def Let-def*
  apply (*rule nfoldli-rule*[**where** *I* = ‹λ- - ψ. *length* φ = *length* ψ›])
  apply (*auto dest*: *in-list-in-setD*)
  done
qed

definition *copy-phase* :: ‹*bool list* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*copy-phase* φ φ' = *do* {
  ASSERT(*length* φ = *length* φ');
  let n = *length* φ';
  nfoldli [*0*..<*n*]
    (λ-. *True*)
    (λ *a* φ'. *do* {
        ASSERT(*a* < *length* φ);
        ASSERT(*a* < *length* φ');
        RETURN (φ'[*a* := φ!*a*])
    })

    φ'
  }›

lemma *copy-phase-alt-def*:
‹*copy-phase* φ φ' = *do* {
  ASSERT(*length* φ = *length* φ');
  let *n* = *length* φ;
  nfoldli [*0*..<*n*]
    (λ-. *True*)
    (λ *a* φ'. *do* {
        ASSERT(*a* < *length* φ);
        ASSERT(*a* < *length* φ');
        RETURN (φ'[*a* := φ!*a*])
    })
    φ'
  }›
  unfolding *copy-phase-def*
  by (*auto simp*: *ASSERT-same-eq-conv*)

lemma *copy-phase-spec*:
  ‹*length* φ = *length* φ' $\Longrightarrow$ *copy-phase* φ φ' $\leq$ *SPEC*($\lambda\psi$. *length* ψ = *length* φ)›
  unfolding *copy-phase-def Let-def*
  apply (*intro ASSERT-leI*)
  subgoal by *auto*
  apply (*rule nfoldli-rule*[**where** *I* = ‹λ- - ψ. *length* φ = *length* ψ›])
  apply (*auto dest*: *in-list-in-setD*)
  done

definition *rephase-random* :: ‹*64 word* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*rephase-random* *b* φ = *do* {
  let *n* = *length* φ;
  (-, φ) ← *nfoldli* [*0*..<*n*]
      (λ-. *True*)
      (λ*a* (*state*, φ). *do* {
        ASSERT(*a* < *length* φ);
        let state = *state* ∗ *6364136223846793005* + *1442695040888963407*;
        RETURN (*state*, φ[*a* := (*state* < *2147483648*)])
      })
      (*b*, φ);
  RETURN φ
  }›

lemma *rephase-random-spec*:
  ‹*rephase-random* *b* φ $\leq$ *SPEC*($\lambda\psi$. *length* ψ = *length* φ)›
  unfolding *rephase-random-def Let-def*
  apply (*refine-vcg nfoldli-rule*[**where** *I* = ‹λ- - (-, ψ). *length* φ = *length* ψ›])
  apply (*auto dest*: *in-list-in-setD*)
  done

definition *phase-rephase* :: ‹*64 word* ⇒ *phase-save-heur* ⇒ *phase-save-heur nres*› **where**
‹*phase-rephase* = (λ*b* (φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*).
    if *b* = *0*
    then do {

```
      if curr-phase = 0
      then do {
        φ ← rephase-init False φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 1,
length-phase)
      }
      else if curr-phase = 1
      then do {
        φ ← copy-phase best φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 2,
length-phase)
      }
      else if curr-phase = 2
      then do {
        φ ← rephase-init True φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 3,
length-phase)
      }
      else if curr-phase = 3
      then do {
        φ ← rephase-random end-of-phase φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 4,
length-phase)
      }
      else do {
        φ ← copy-phase best φ;
         RETURN (φ, target-assigned, target, best-assigned, best, (1+length-phase)∗100+end-of-phase,
0,
           length-phase+1)
      }
    }
    else do {
      if curr-phase = 0
      then do {
        φ ← rephase-init False φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 1,
length-phase)
      }
      else if curr-phase = 1
      then do {
        φ ← copy-phase best φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 2,
length-phase)
      }
      else if curr-phase = 2
      then do {
        φ ← rephase-init True φ;
          RETURN (φ, target-assigned, target, best-assigned, best, length-phase∗100+end-of-phase, 3,
length-phase)
      }
      else do {
        φ ← copy-phase best φ;
         RETURN (φ, target-assigned, target, best-assigned, best, (1+length-phase)∗100+end-of-phase,
0,
          length-phase+1)
     }
```

})›

lemma *phase-rephase-spec*:
  assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
  shows ‹*phase-rephase* b φ $\leq$ $\Downarrow$*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
proof −
  obtain φ' *target-assigned target best-assigned best end-of-phase curr-phase* **where**
    φ: ‹φ = (φ', *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)›
    by (*cases* φ) *auto*
  then have [*simp*]: ‹*length* φ' = *length best*›
    using *assms* **by** (*auto simp*: *phase-save-heur-rel-def*)
  have *1*: ‹$\Downarrow$*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$)) $\geq$
    $\Downarrow$*Id*((λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*).
      if b = 0
      then do {
        if curr-phase = 0 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*,*length-phase*∗*100*+*end-of-phase*, *1*, *length-phase*)
        }
        else if curr-phase = 1 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*100*+*end-of-phase*, *2*, *length-phase*)
        }
        else if curr-phase = 2 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*100*+*end-of-phase*, *3*, *length-phase*)
        }
        else if curr-phase = 3 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*100*+*end-of-phase*, *4*, *length-phase*)
        }
        else do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
          RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, (*1*+*length-phase*)∗*100*+*end-of-phase*, *0*, *length-phase*+*1*)
        }
      }
      else do {
        if curr-phase = 0 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*,*length-phase*∗*100*+*end-of-phase*, *1*, *length-phase*)
        }
        else if curr-phase = 1 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*100*+*end-of-phase*, *2*, *length-phase*)
        }
        else if curr-phase = 2 then do {
          φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
            RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*100*+*end-of-phase*, *3*, *length-phase*)

}
else do {
φ' ← *SPEC* ($\lambda\varphi'$. *length* φ = *length* φ');
RETURN (φ', *target-assigned*, *target*, *best-assigned*, *best*, (*1*+*length-phase*)∗*100*+*end-of-phase*, *0*,
length-phase+*1*)
}
}
) φ)›
using *assms*
by (*cases* φ)
(*auto simp*: *phase-save-heur-rel-def phase-saving-def RES-RETURN-RES*)

show *?thesis*
unfolding *phase-rephase-def* φ
apply (*simp only*: *prod.case*)
apply (*rule order-trans*)
defer
apply (*rule 1*)
apply (*simp only*: *prod.case* φ)
apply (*refine-vcg if-mono rephase-init-spec copy-phase-spec rephase-random-spec*)
apply (*auto simp*: *phase-rephase-def*)
done
qed

definition *rephase-heur* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*rephase-heur* = (λ*b* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ).
do {
φ ← *phase-rephase b* φ;
RETURN (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ)
})›

lemma *rephase-heur-spec*:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *rephase-heur b heur* ≤ ⇓*Id* (*SPEC*(*heuristic-rel* $\mathcal{A}$))›
unfolding *rephase-heur-def*
apply (*refine-vcg phase-rephase-spec*[*THEN order-trans*])
apply (*auto simp*: *heuristic-rel-def*)
done

definition *rephase-heur-st* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*rephase-heur-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
let b = *current-restart-phase heur*;
heur ← *rephase-heur b heur*;
let - = *isasat-print-progress* (*current-rephasing-phase heur*) *b stats lcount*;
RETURN (*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*)
})›

lemma *rephase-heur-st-spec*:
‹(*S*, *S'*) ∈ *twl-st-heur* ⟹ *rephase-heur-st S* ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur*)›
unfolding *rephase-heur-st-def*
apply (*cases S'*)
apply (*refine-vcg rephase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
apply (*simp-all add*: *twl-st-heur-def*)
done

definition *phase-save-phase* :: ‹*nat* ⇒ *phase-save-heur* ⇒ *phase-save-heur nres*› **where**
‹*phase-save-phase* = (λ*n* (φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*). *do* {
target ← (*if n* > *target-assigned*
then copy-phase φ *target else RETURN target*);
target-assigned ← (*if n* > *target-assigned*
then RETURN n else RETURN target-assigned);
best ← (*if n* > *best-assigned*
then copy-phase φ *best else RETURN best*);
best-assigned ← (*if n* > *best-assigned*
then RETURN n else RETURN best-assigned);
RETURN (φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)
})›

lemma *phase-save-phase-spec*:
assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
shows ‹*phase-save-phase n* φ ≤ ⇓*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
proof −
obtain φ′ *target-assigned target best-assigned best end-of-phase curr-phase* **where**
φ: ‹φ = (φ′, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)›
by (*cases* φ) *auto*
then have [*simp*]: ‹*length* φ′ = *length best*› ‹*length target* = *length best*›
using *assms* **by** (*auto simp*: *phase-save-heur-rel-def*)
have *1*: ‹⇓*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$)) ≥
⇓*Id*((λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*). *do* {
target ← (*if n* > *target-assigned*
then SPEC (λφ′. *length* φ = *length* φ′) *else RETURN target*);
target-assigned ← (*if n* > *target-assigned*
then RETURN n else RETURN target-assigned);
best ← (*if n* > *best-assigned*
then SPEC (λφ′. *length* φ = *length* φ′) *else RETURN best*);
best-assigned ← (*if n* > *best-assigned*
then RETURN n else RETURN best-assigned);
RETURN (φ′, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)
}) φ)›
using *assms*
by (*auto simp*: *phase-save-heur-rel-def phase-saving-def RES-RETURN-RES* φ *RES-RES-RETURN-RES*)

show *?thesis*
unfolding *phase-save-phase-def* φ
apply (*simp only*: *prod.case*)
apply (*rule order-trans*)
defer
apply (*rule 1*)
apply (*simp only*: *prod.case* φ)
apply (*refine-vcg if-mono rephase-init-spec copy-phase-spec rephase-random-spec*)
apply (*auto simp*: *phase-rephase-def*)
done
qed

definition *save-rephase-heur* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*save-rephase-heur* = (λ*n* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ).
do {
φ ← *phase-save-phase n* φ;
RETURN (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ)
})›

lemma *save-phase-heur-spec*:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *save-rephase-heur n heur* $\leq$ $\Downarrow$*Id* (*SPEC*(*heuristic-rel* $\mathcal{A}$))›
unfolding *save-rephase-heur-def*
apply (*refine-vcg phase-save-phase-spec*[*THEN order-trans*])
apply (*auto simp*: *heuristic-rel-def*)
done

definition *save-phase-st* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*save-phase-st* = (λ(*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
ASSERT(*isa-length-trail-pre M'*);
let n = *isa-length-trail M'*;
heur ← *save-rephase-heur n heur*;
RETURN (*M'*, *arena*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*)
})›

lemma *save-phase-st-spec*:
‹(*S*, *S'*) ∈ *twl-st-heur* $\Longrightarrow$ *save-phase-st S* ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur*)›
unfolding *save-phase-st-def*
apply (*cases S'*)
apply (*refine-vcg save-phase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
apply (*simp-all add*: *twl-st-heur-def isa-length-trail-pre*)
apply (*rule isa-length-trail-pre*)
apply *blast*
done

end
theory *IsaSAT-Backtrack*
imports *IsaSAT-Setup IsaSAT-VMTF IsaSAT-Rephase*
begin

Chapter 14

Backtrack

The backtrack function is highly complicated and tricky to maintain.

14.1 Backtrack with direct extraction of literal if highest level

Empty conflict **definition** (**in** −) *empty-conflict-and-extract-clause*
:: ‹(*nat*,*nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat clause-l* ⇒
(*nat clause option* × *nat clause-l* × *nat*) *nres*›
where
‹*empty-conflict-and-extract-clause M D outl* =
$SPEC(\lambda(D,\ C,\ n).\ D = None \wedge mset\ C = mset\ outl \wedge C!0 = outl!0 \wedge$
$(length\ C > 1 \longrightarrow highest\text{-}lit\ M\ (mset\ (tl\ C))\ (Some\ (C!1,\ get\text{-}level\ M\ (C!1)))) \wedge$
$(length\ C > 1 \longrightarrow n = get\text{-}level\ M\ (C!1)) \wedge$
$(length\ C = 1 \longrightarrow n = 0)$
)›

definition *empty-conflict-and-extract-clause-heur-inv* **where**
‹*empty-conflict-and-extract-clause-heur-inv M outl* =
$(\lambda(E,\ C,\ i).\ mset\ (take\ i\ C) = mset\ (take\ i\ outl) \wedge$
$length\ C = length\ outl \wedge C\ !\ 0 = outl\ !\ 0 \wedge i \geq 1 \wedge i \leq length\ outl \wedge$
$(1 < length\ (take\ i\ C) \longrightarrow$
$highest\text{-}lit\ M\ (mset\ (tl\ (take\ i\ C)))$
$(Some\ (C\ !\ 1,\ get\text{-}level\ M\ (C\ !\ 1)))))$›

definition *empty-conflict-and-extract-clause-heur* ::
nat multiset ⇒ (*nat*, *nat*) *ann-lits*
⇒ *lookup-clause-rel*
⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*
where
‹*empty-conflict-and-extract-clause-heur* $\mathcal{A}$ *M D outl* = *do* {
let $C = replicate\ (length\ outl)\ (outl!0)$;
$(D,\ C,\ \text{-}) \leftarrow WHILE_T{}^{empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\text{-}heur\text{-}inv\ M\ outl}$
$(\lambda(D,\ C,\ i).\ i < length\text{-}uint32\text{-}nat\ outl)$
$(\lambda(D,\ C,\ i).$ *do* {
$ASSERT(i < length\ outl)$;
$ASSERT(i < length\ C)$;
$ASSERT(lookup\text{-}conflict\text{-}remove1\text{-}pre\ (outl\ !\ i,\ D))$;
let $D = lookup\text{-}conflict\text{-}remove1\ (outl\ !\ i)\ D$;
let $C = C[i := outl\ !\ i]$;
$ASSERT(C!i \in\#\ \mathcal{L}_{all}\ \mathcal{A} \wedge C!1 \in\#\ \mathcal{L}_{all}\ \mathcal{A} \wedge 1 < length\ C)$;
let $C = (if\ get\text{-}level\ M\ (C!i) > get\text{-}level\ M\ (C!1)\ then\ swap\ C\ 1\ i\ else\ C)$;

$ASSERT(i+1 \leq uint32\text{-}max)$;
$RETURN\ (D,\ C,\ i+1)$
})
$(D,\ C,\ 1)$;
$ASSERT(length\ outl \neq 1 \longrightarrow length\ C > 1)$;
$ASSERT(length\ outl \neq 1 \longrightarrow C!1 \in\#\ \mathcal{L}_{all}\ \mathcal{A})$;
$RETURN\ ((True,\ D),\ C,\ if\ length\ outl = 1\ then\ 0\ else\ get\text{-}level\ M\ (C!1))$
}›

lemma *empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause*:
assumes
D: ‹$D = mset\ (tl\ outl)$› **and**
outl: ‹$outl \neq []$› **and**
dist: ‹*distinct outl*› **and**
lits: ‹$literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ \mathcal{A}\ (mset\ outl)$› **and**
DD': ‹$(D',\ D) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}$› **and**
consistent: ‹$\neg\ tautology\ (mset\ outl)$› **and**
bounded: ‹$isasat\text{-}input\text{-}bounded\ \mathcal{A}$›
shows
‹$empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\text{-}heur\ \mathcal{A}\ M\ D'\ outl \leq \Downarrow (option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \times_r Id \times_r Id)$
$(empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\ M\ D\ outl)$›
proof −
have *size-out*: ‹$size\ (mset\ outl) \leq 1 + uint32\text{-}max\ div\ 2$›
using *simple-clss-size-upper-div2*[*OF bounded lits - consistent*]
‹*distinct outl*› **by** *auto*
have *empty-conflict-and-extract-clause-alt-def*:
‹$empty\text{-}conflict\text{-}and\text{-}extract\text{-}clause\ M\ D\ outl = do$ {
$(D',\ outl') \leftarrow SPEC\ (\lambda(E,\ F).\ E = \{\#\} \wedge mset\ F = D)$;
$SPEC$
$(\lambda(D,\ C,\ n).$
$D = None \wedge$
$mset\ C = mset\ outl \wedge$
$C\ !\ 0 = outl\ !\ 0 \wedge$
$(1 < length\ C \longrightarrow$
$highest\text{-}lit\ M\ (mset\ (tl\ C))\ (Some\ (C\ !\ 1,\ get\text{-}level\ M\ (C\ !\ 1)))) \wedge$
$(1 < length\ C \longrightarrow n = get\text{-}level\ M\ (C\ !\ 1)) \wedge (length\ C = 1 \longrightarrow n = 0))$
}› **for** *M D outl*
unfolding *empty-conflict-and-extract-clause-def RES-RES2-RETURN-RES*
by (*auto simp*: *ex-mset*)
define *I* **where**
‹$I \equiv \lambda(E,\ C,\ i).\ mset\ (take\ i\ C) = mset\ (take\ i\ outl) \wedge$
$(E,\ D - mset\ (take\ i\ outl)) \in lookup\text{-}clause\text{-}rel\ \mathcal{A} \wedge$
$length\ C = length\ outl \wedge C\ !\ 0 = outl\ !\ 0 \wedge i \geq 1 \wedge i \leq length\ outl \wedge$
$(1 < length\ (take\ i\ C) \longrightarrow$
$highest\text{-}lit\ M\ (mset\ (tl\ (take\ i\ C)))$
$(Some\ (C\ !\ 1,\ get\text{-}level\ M\ (C\ !\ 1))))$›
have *I0*: ‹$I\ (D',\ replicate\ (length\ outl)\ (outl\ !\ 0),\ 1)$›
using *assms* **by** (*cases outl*) (*auto simp*: *I-def*)

have [*simp*]: ‹$ba \geq 1 \Longrightarrow mset\ (tl\ outl) - mset\ (take\ ba\ outl) = mset\ ((drop\ ba\ outl))$›
for *ba*
apply (*subst append-take-drop-id*[*of* ‹$ba - 1$›, *symmetric*])
using *dist*
unfolding *mset-append*
by (*cases outl*; *cases ba*)
(*auto simp*: *take-tl drop-Suc*[*symmetric*] *remove-1-mset-id-iff-notin dest*: *in-set-dropD*)

have *empty-conflict-and-extract-clause-heur-inv*:
‹*empty-conflict-and-extract-clause-heur-inv M outl*
(*D'*, *replicate* (*length outl*) (*outl* ! *0*), *1*)›
using *assms*
unfolding *empty-conflict-and-extract-clause-heur-inv-def*
by (*cases outl*) *auto*
have *I0*: ‹*I* (*D'*, *replicate* (*length outl*) (*outl* ! *0*), *1*)›
using *assms*
unfolding *I-def*
by (*cases outl*) *auto*
have
C1-L: ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› (**is** *?A1inL*) **and**
ba-le: ‹*ba* + *1* ≤ *uint32-max*› (**is** *?ba-le*) **and**
I-rec: ‹*I* (*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*],
ba + *1*)› (**is** *?I*) **and**
inv: ‹*empty-conflict-and-extract-clause-heur-inv M outl*
(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*],
ba + *1*)› (**is** *?inv*)
if
‹*empty-conflict-and-extract-clause-heur-inv M outl s*› **and**
I: ‹*I s*› **and**
‹*case s of* (*D*, *C*, *i*) ⇒ *i* < *length-uint32-nat outl*› **and**
st:
‹*s* = (*a*, *b*)›
‹*b* = (*aa*, *ba*)› **and**
ba-le: ‹*ba* < *length outl*› **and**
‹*ba* < *length aa*› **and**
‹*lookup-conflict-remove1-pre* (*outl* ! *ba*, *a*)›
for *s a b aa ba*
proof −
have
mset-aa: ‹*mset* (*take ba aa*) = *mset* (*take ba outl*)› **and**
aD: ‹(*a*, *D* − *mset* (*take ba outl*)) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
l-aa-outl: ‹*length aa* = *length outl*› **and**
aa0: ‹*aa* ! *0* = *outl* ! *0*› **and**
ba-ge1: ‹*1* ≤ *ba*› **and**
ba-lt: ‹*ba* ≤ *length outl*› **and**
highest: ‹*1* < *length* (*take ba aa*) ⟶
highest-lit M (*mset* (*tl* (*take ba aa*)))
(*Some* (*aa* ! *1*, *get-level M* (*aa* ! *1*)))›
using *I* **unfolding** *st I-def prod.case*
by *auto*
have *set-aa-outl*: ‹*set* (*take ba aa*) = *set* (*take ba outl*)›
using *mset-aa* **by** (*blast dest*: *mset-eq-setD*)
show *?ba-le*
using *ba-le assms size-out*
by (*auto simp*: *uint32-max-def*)
have *ba-ge1-aa-ge*: ‹*ba* > *1* ⟹ *aa* ! *1* ∈ *set* (*take ba aa*)›

using *ba-ge1 ba-le l-aa-outl*
by (*auto simp*: *in-set-take-conv-nth intro*!: *bex-lessI*[*of* - ‹*Suc 0*›])
then have ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈ *set outl*›
using *ba-le l-aa-outl ba-ge1*
unfolding *mset-aa in-multiset-in-set*[*symmetric*]
by (*cases* ‹*ba* > *1*›)
(*auto simp*: *mset-aa dest*: *in-set-takeD*)
then show *?A1inL*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*OF lits*, *of* ‹*aa*[*ba* := *outl* ! *ba*] ! *1*›] **by** *auto*

define *aa2* **where** ‹*aa2* ≡ *tl* (*tl* (*take ba aa*))›
have *tl-take-nth-con*: ‹*tl* (*take ba aa*) = *aa* ! *Suc 0* # *aa2*› **if** ‹*ba* > *Suc 0*›
using *ba-le ba-ge1 that l-aa-outl* **unfolding** *aa2-def*
by (*cases aa*; *cases* ‹*tl aa*›; *cases ba*; *cases* ‹*ba* − *1*›)
auto
have *no-tauto-nth*: ‹ *i* < *length outl* ⟹ − *outl* ! *ba* = *outl* ! *i* ⟹ *False*› **for** *i*
using *consistent ba-le nth-mem*
by (*force simp*: *tautology-decomp′ uminus-lit-swap*)
have *outl-ba--L*: ‹*outl* ! *ba* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *ba-le literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*OF lits*, *of* ‹*outl* ! *ba*›] **by** *auto*
have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
remove1-mset (*outl* ! *ba*) (*D* −(*mset* (*take ba outl*)))) ∈ *lookup-clause-rel* $\mathcal{A}$›
by (*rule lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*])
(*use ba-ge1 ba-le aD outl-ba--L* **in**
‹*auto simp*: *D in-set-drop-conv-nth image-image dest*: *no-tauto-nth*
intro!: *bex-geI*[*of* - *ba*]›)
then have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
D − *mset* (*take* (*Suc ba*) *outl*))
∈ *lookup-clause-rel* $\mathcal{A}$›
using *aD ba-le ba-ge1 ba-ge1-aa-ge aa0*
by (*auto simp*: *take-Suc-conv-app-nth*)
moreover have ‹*1* < *length*
(*take* (*ba* + *1*)
(*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*])) ⟶
highest-lit M
(*mset*
(*tl* (*take* (*ba* + *1*)
(*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*]))))
(*Some*
((*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*]) !
1,
get-level M
((*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*]) !
1)))›

```
      using highest ba-le ba-ge1
      by (cases ‹ba = Suc 0›)
        (auto simp: highest-lit-def take-Suc-conv-app-nth l-aa-outl
          get-maximum-level-add-mset swap-nth-relevant max-def take-update-swap
          swap-only-first-relevant tl-update-swap mset-update nth-tl
          get-maximum-level-remove-non-max-lvl tl-take-nth-con
          aa2-def[symmetric])
    moreover have ‹mset
      (take (ba + 1)
        (if get-level M (aa[ba := outl ! ba] ! 1)
            < get-level M (aa[ba := outl ! ba] ! ba)
          then swap (aa[ba := outl ! ba]) 1 ba
          else aa[ba := outl ! ba])) =
      mset (take (ba + 1) outl)›
      using ba-le ba-ge1 ba-ge1-aa-ge aa0
      unfolding mset-aa
      by (cases ‹ba = 1›)
        (auto simp: take-Suc-conv-app-nth l-aa-outl
          take-swap-relevant swap-only-first-relevant mset-aa set-aa-outl
          mset-update add-mset-remove-trivial-If)
    ultimately show ?I
      using ba-ge1 ba-le
      unfolding I-def prod.simps
      by (auto simp: l-aa-outl aa0)

    then show ?inv
      unfolding empty-conflict-and-extract-clause-heur-inv-def I-def
      by (auto simp: l-aa-outl aa0)
  qed
  have mset-tl-out: ‹mset (tl outl) − mset outl = {#}›
    by (cases outl) auto
  have H1: ‹WHILE_T^(empty-conflict-and-extract-clause-heur-inv M outl)
     (λ(D, C, i). i < length-uint32-nat outl)
     (λ(D, C, i). do {
           - ← ASSERT (i < length outl);
           - ← ASSERT (i < length C);
           - ← ASSERT (lookup-conflict-remove1-pre (outl ! i, D));
           - ← ASSERT
                (C[i := outl ! i] ! i ∈# ℒ_all 𝒜 ∧
                 C[i := outl ! i] ! 1 ∈# ℒ_all 𝒜 ∧
                1 < length (C[i := outl ! i]));
           - ← ASSERT (i + 1 ≤ uint32-max);
           RETURN
            (lookup-conflict-remove1 (outl ! i) D,
             if get-level M (C[i := outl ! i] ! 1)
                < get-level M (C[i := outl ! i] ! i)
             then swap (C[i := outl ! i]) 1 i
             else C[i := outl ! i],
             i + 1)
         })
     (D′, replicate (length outl) (outl ! 0), 1)
    ≤ ⇓ {((E, C, n), (E′, F′)). (E, E′) ∈ lookup-clause-rel 𝒜 ∧ mset C = mset outl ∧
             C ! 0 = outl ! 0 ∧
            (1 < length C ⟶
              highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
            n = length outl ∧
```

I (*E*, *C*, *n*)}
(*SPEC* (λ(*E*, *F*). *E* = {#} ∧ *mset F* = *D*))›
unfolding *conc-fun-RES*
apply (*refine-vcg WHILEIT-rule-stronger-inv-RES*[**where** *R* = ‹*measure* (λ(-, -, *i*). *length outl* − *i*)› **and**
I' = ‹*I*›])
subgoal by *auto*
subgoal by (*rule empty-conflict-and-extract-clause-heur-inv*)
subgoal by (*rule I0*)
subgoal using *assms* **by** (*cases outl*; *auto*)
subgoal
by (*auto simp*: *I-def*)
subgoal for *s a b aa ba*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
unfolding *lookup-conflict-remove1-pre-def prod.simps*
by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
unfolding *lookup-conflict-remove1-pre-def prod.simps*
by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
by (*rule C1-L*)
subgoal for *s a b aa ba*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *lits*
unfolding *lookup-conflict-remove1-pre-def prod.simps*
by (*auto simp*: *I-def empty-conflict-and-extract-clause-heur-inv-def*
lookup-clause-rel-def D atms-of-def)
subgoal for *s a b aa ba*
by (*rule ba-le*)
subgoal
by (*rule inv*)
subgoal
by (*rule I-rec*)
subgoal
by *auto*
subgoal for *s*
unfolding *prod.simps*
apply (*cases s*)
apply *clarsimp*
apply (*intro conjI*)
subgoal
by (*rule ex-mset*)
subgoal
using *assms*
by (*auto simp*: *empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out*)
subgoal
using *assms*
by (*auto simp*: *empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out*)
subgoal
using *assms*
by (*auto simp*: *empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out*)
subgoal
using *assms*
by (*auto simp*: *empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out*)

```
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    done
  done
have x1b-lall: ‹x1b ! 1 ∈# ℒ_all 𝒜›
  if
    inv: ‹(x, x')
    ∈ {((E, C, n), E', F').
        (E, E') ∈ lookup-clause-rel 𝒜 ∧
        mset C = mset outl ∧
        C ! 0 = outl ! 0 ∧
        (1 < length C ⟶
        highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
          n = length outl  ∧
        I (E, C, n)}› and
    ‹x' ∈ {(E, F). E = {#} ∧ mset F = D}› and
    st:
    ‹x' = (x1, x2)›
    ‹x2a = (x1b, x2b)›
    ‹x = (x1a, x2a)› and
    ‹length outl ≠ 1 ⟶ 1 < length x1b› and
    ‹length outl ≠ 1›
  for x x' x1 x2 x1a x2a x1b x2b
proof −
  have
    ‹(x1a, x1) ∈ lookup-clause-rel 𝒜› and
    ‹mset x1b = mset outl› and
    ‹x1b ! 0 = outl ! 0› and
    ‹Suc 0 < length x1b ⟶
    highest-lit M (mset (tl x1b))
      (Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))› and
    mset-aa: ‹mset (take x2b x1b) = mset (take x2b outl)› and
    ‹(x1a, D − mset (take x2b outl)) ∈ lookup-clause-rel 𝒜› and
    l-aa-outl: ‹length x1b = length outl› and
    ‹x1b ! 0 = outl ! 0› and
    ba-ge1: ‹Suc 0 ≤ x2b› and
    ba-le: ‹x2b ≤ length outl› and
    ‹Suc 0 < length x1b ∧ Suc 0 < x2b ⟶
   highest-lit M (mset (tl (take x2b x1b)))
    (Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))›
    using inv unfolding st I-def prod.case st
    by auto

  have set-aa-outl: ‹set (take x2b x1b) = set (take x2b outl)›
    using mset-aa by (blast dest: mset-eq-setD)
  have ba-ge1-aa-ge: ‹x2b > 1 ⟹ x1b ! 1 ∈ set (take x2b x1b)›
    using ba-ge1 ba-le l-aa-outl
    by (auto simp: in-set-take-conv-nth intro!: bex-lessI[of - ‹Suc 0›])
  then have ‹x1b ! 1 ∈ set outl›
    using ba-le l-aa-outl ba-ge1 that
    unfolding mset-aa in-multiset-in-set[symmetric]
    by (cases ‹x2b > 1›)
      (auto simp: mset-aa dest: in-set-takeD)
  then show ?thesis
    using literals-are-in-ℒ_in-in-mset-ℒ_all[OF lits, of ‹x1b ! 1›] by auto
```

qed

show *?thesis*
unfolding *empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-alt-def Let-def I-def*[*symmetric*]
apply (*subst empty-conflict-and-extract-clause-alt-def*)
unfolding *conc-fun-RES*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** R = ‹*measure* (λ(-, -, *i*). *length outl* − *i*)› **and** I' = ‹*I*›] *H1*)
subgoal using *assms* **by** (*auto simp*: *I-def*)
subgoal by (*rule x1b-lall*)
subgoal using *assms*
by (*auto intro*!: *RETURN-RES-refine simp*: *option-lookup-clause-rel-def I-def*)
done
qed

definition *isa-empty-conflict-and-extract-clause-heur* ::
trail-pol ⇒ *lookup-clause-rel* ⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*
where
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(*D*, *C*, -) ← $WHILE_T$
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*get-level-pol-pre* (*M*, *C*!*i*));
ASSERT(*get-level-pol-pre* (*M*, *C*!*1*));
ASSERT(*1* < *length C*);
let C = (*if get-level-pol M* (*C*!*i*) > *get-level-pol M* (*C*!*1*) *then swap C 1 i else C*);
ASSERT(*i*+*1* ≤ *uint32-max*);
RETURN (*D*, *C*, *i*+*1*)
})
(*D*, *C*, *1*);
ASSERT(*length outl* ≠ *1* ⟶ *length C* > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (*M*, *C*!*1*));
RETURN ((*True*, *D*), *C*, *if length outl* = *1 then 0 else get-level-pol M* (*C*!*1*))
}›

lemma *isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur*:
‹(*uncurry2 isa-empty-conflict-and-extract-clause-heur*, *uncurry2* (*empty-conflict-and-extract-clause-heur* $\mathcal{A}$)) ∈
trail-pol $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id* $\rightarrow_f$ ⟨*Id*⟩*nres-rel* ›
proof −
have [*refine0*]: ‹((*x2b*, *replicate* (*length x2c*) (*x2c* ! *0*), *1*), *x2*, *replicate* (*length x2a*) (*x2a* ! *0*), *1*)
∈ *Id* $\times_f$ *Id* $\times_f$ *Id*›
if
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ $\times_f$ *Id* $\times_f$ *Id*› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1b* = (*x1c*, *x2b*)› **and**
‹*x* = (*x1b*, *x2c*)›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c*

using *that* **by** *auto*

show *?thesis*
supply [[*goals-limit=1*]]
unfolding *isa-empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-heur-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
apply (*assumption+*)[*5*]
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*rule get-level-pol-pre*) *auto*
subgoal
by (*rule get-level-pol-pre*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*auto simp*: *get-level-get-level-pol*[*of - -* $\mathcal{A}$])
subgoal by *auto*
subgoal
by (*rule get-level-pol-pre*) *auto*
subgoal by (*auto simp*: *get-level-get-level-pol*[*of - -* $\mathcal{A}$])
done
qed

definition *extract-shorter-conflict-wl-nlit* **where**
⟨*extract-shorter-conflict-wl-nlit K M NU D NE UE* =
SPEC(λ*D'*. *D'* ≠ *None* ∧ *the D'* ⊆# *the D* ∧ *K* ∈# *the D'* ∧
mset '# *ran-mf NU* + *NE* + *UE* ⊨*pm the D'*)⟩

definition *extract-shorter-conflict-wl-nlit-st*
:: ⟨'*v twl-st-wl* ⇒ '*v twl-st-wl nres*⟩
where
⟨*extract-shorter-conflict-wl-nlit-st* =
(λ(*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*). *do* {
let K = −*lit-of* (*hd M*);
D ← *extract-shorter-conflict-wl-nlit K M N D NE UE*;
RETURN (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*)})⟩

definition *empty-lookup-conflict-and-highest*
:: ⟨'*v twl-st-wl* ⇒ ('*v twl-st-wl* × *nat*) *nres*⟩
where
⟨*empty-lookup-conflict-and-highest* =
(λ(*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*). *do* {
let K = −*lit-of* (*hd M*);
let n = *get-maximum-level M* (*remove1-mset K* (*the D*));
RETURN ((*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*), *n*)})⟩

definition *backtrack-wl-D-heur-inv* **where**
⟨*backtrack-wl-D-heur-inv S* ⟷ (∃ *S'*. (*S*, *S'*) ∈ *twl-st-heur-conflict-ana* ∧ *backtrack-wl-inv S'*)⟩

definition *extract-shorter-conflict-heur* **where**

‹extract-shorter-conflict-heur = (λM NU NUE C outl. do {
  let K = lit-of (hd M);
  let C = Some (remove1-mset (−K) (the C));
  C ← iterate-over-conflict (−K) M NU NUE (the C);
  RETURN (Some (add-mset (−K) C))
})›

definition (**in** −) *empty-cach* **where**
‹empty-cach cach = (λ-. SEEN-UNKNOWN)›

definition *empty-conflict-and-extract-clause-pre*
:: ‹(((nat,nat) ann-lits × nat clause) × nat clause-l) ⇒ bool› **where**
‹empty-conflict-and-extract-clause-pre =
  (λ((M, D), outl). D = mset (tl outl) ∧ outl ≠ [] ∧ distinct outl ∧
  ¬tautology (mset outl) ∧ length outl ≤ uint32-max)›

definition (**in** −) *empty-cach-ref* **where**
‹empty-cach-ref = (λ(cach, support). (replicate (length cach) SEEN-UNKNOWN, []))›

definition *empty-cach-ref-set-inv* **where**
‹empty-cach-ref-set-inv cach0 support =
  (λ(i, cach). length cach = length cach0 ∧
      (∀ L ∈ set (drop i support). L < length cach) ∧
      (∀ L ∈ set (take i support). cach ! L = SEEN-UNKNOWN) ∧
      (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set (drop i support)))›

definition *empty-cach-ref-set* **where**
‹empty-cach-ref-set = (λ(cach0, support). do {
  let n = length support;
  ASSERT(n ≤ Suc (uint32-max div 2));
  (-, cach) ← $WHILE_T$$^{\text{empty-cach-ref-set-inv cach0 support}}$
    (λ(i, cach). i < length support)
    (λ(i, cach). do {
      ASSERT(i < length support);
      ASSERT(support ! i < length cach);
      RETURN(i+1, cach[support ! i := SEEN-UNKNOWN])
    })
    (0, cach0);
  RETURN (cach, emptied-list support)
})›

lemma *empty-cach-ref-set-empty-cach-ref*:
‹(empty-cach-ref-set, RETURN o empty-cach-ref) ∈
  [λ(cach, supp). (∀ L ∈ set supp. L < length cach) ∧ length supp ≤ Suc (uint32-max div 2) ∧
    (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set supp)]$_f$
  Id → ⟨Id⟩ nres-rel›
proof −
  have *H*: ‹$WHILE_T$$^{\text{empty-cach-ref-set-inv cach0 support}'}$ (λ(i, cach). i < length support')
      (λ(i, cach).
          ASSERT (i < length support') ⤜
          (λ-. ASSERT (support' ! i < length cach) ⤜
          (λ-. RETURN (i + 1, cach[support' ! i := SEEN-UNKNOWN]))))
      (0, cach0) ⤜
    (λ(-, cach). RETURN (cach, emptied-list support'))

$\leq \Downarrow$ *Id* (*RETURN* (*replicate* (*length cach0*) *SEEN-UNKNOWN*, []))›
if
‹∀ *L*∈*set support'*. *L* < *length cach0*› **and**
‹∀ *L*<*length cach0*. *cach0* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set support'*›
for *cach support cach0 support'*
proof −
have *init*: ‹*empty-cach-ref-set-inv cach0 support'* (*0*, *cach0*)›
using *that* **unfolding** *empty-cach-ref-set-inv-def*
by *auto*
have *valid-length*:
‹*empty-cach-ref-set-inv cach0 support' s* ⟹ *case s of* (*i*, *cach*) ⇒ *i* < *length support'* ⟹
s = (*cach'*, *sup'*) ⟹ *support'* ! *cach'* < *length sup'*› **for** *s cach' sup'*
using *that* **unfolding** *empty-cach-ref-set-inv-def*
by *auto*
have *set-next*: ‹*empty-cach-ref-set-inv cach0 support'* (*i* + *1*, *cach'*[*support'* ! *i* := *SEEN-UNKNOWN*])›
if
inv: ‹*empty-cach-ref-set-inv cach0 support' s*› **and**
cond: ‹*case s of* (*i*, *cach*) ⇒ *i* < *length support'*› **and**
s: ‹*s* = (*i*, *cach'*)› **and**
valid[*simp*]: ‹*support'* ! *i* < *length cach'*›
for *s i cach'*
proof −
have
le-cach-cach0: ‹*length cach'* = *length cach0*› **and**
le-length: ‹∀ *L*∈*set* (*drop i support'*). *L* < *length cach'*› **and**
UNKNOWN: ‹∀ *L*∈*set* (*take i support'*). *cach'* ! *L* = *SEEN-UNKNOWN*› **and**
support: ‹∀ *L*<*length cach'*. *cach'* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set* (*drop i support'*)› **and**
[*simp*]: ‹*i* < *length support'*›
using *inv cond* **unfolding** *empty-cach-ref-set-inv-def s prod.case*
by *auto*

show *?thesis*
unfolding *empty-cach-ref-set-inv-def*
unfolding *prod.case*
apply (*intro conjI*)
subgoal by (*simp add*: *le-cach-cach0*)
subgoal using *le-length* **by** (*simp add*: *Cons-nth-drop-Suc*[*symmetric*])
subgoal using *UNKNOWN* **by** (*auto simp add*: *take-Suc-conv-app-nth*)
subgoal using *support* **by** (*auto simp add*: *Cons-nth-drop-Suc*[*symmetric*])
done
qed
have *final*: ‹((*cach'*, *emptied-list support'*), *replicate* (*length cach0*) *SEEN-UNKNOWN*, []) ∈ *Id*›
if
inv: ‹*empty-cach-ref-set-inv cach0 support' s*› **and**
cond: ‹¬ (*case s of* (*i*, *cach*) ⇒ *i* < *length support'*)› **and**
s: ‹*s* = (*i*, *cach'*)›
for *s cach' i*
proof −
have
le-cach-cach0: ‹*length cach'* = *length cach0*› **and**
le-length: ‹∀ *L*∈*set* (*drop i support'*). *L* < *length cach'*› **and**
UNKNOWN: ‹∀ *L*∈*set* (*take i support'*). *cach'* ! *L* = *SEEN-UNKNOWN*› **and**
support: ‹∀ *L*<*length cach'*. *cach'* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set* (*drop i support'*)› **and**
i: ‹¬*i* < *length support'*›
using *inv cond* **unfolding** *empty-cach-ref-set-inv-def s prod.case*
by *auto*

have ‹∀ L<*length cach'*. *cach'* ! L = *SEEN-UNKNOWN*›
using *support i* **by** *auto*
then have [*dest*]: ‹L ∈ *set cach'* ⟹ L = *SEEN-UNKNOWN*› **for** L
by (*metis in-set-conv-nth*)
then have [*dest*]: ‹*SEEN-UNKNOWN* ∉ *set cach'* ⟹ *cach0* = [] ∧ *cach'* = []›
using *le-cach-cach0* **by** (*cases cach'*) *auto*
show *?thesis*
by (*auto simp*: *emptied-list-def list-eq-replicate-iff le-cach-cach0*)
qed
show *?thesis*
unfolding *conc-Id id-def*
apply (*refine-vcg WHILEIT-rule*[**where** R = ‹*measure* (λ(i, -). *length support'* − i)›])
subgoal by *auto*
subgoal by (*rule init*)
subgoal by *auto*
subgoal by (*rule valid-length*)
subgoal by (*rule set-next*)
subgoal by *auto*
subgoal using *final* **by** *simp*
done
qed
show *?thesis*
unfolding *empty-cach-ref-set-def empty-cach-ref-def Let-def comp-def*
by (*intro frefI nres-relI ASSERT-leI*) (*clarify intro*!: *H ASSERT-leI*)

qed

lemma *empty-cach-ref-empty-cach*:
‹*isasat-input-bounded* $\mathcal{A}$ ⟹ (*RETURN o empty-cach-ref*, *RETURN o empty-cach*) ∈ *cach-refinement* $\mathcal{A} \rightarrow_f$ ⟨*cach-refinement* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *empty-cach-def empty-cach-ref-def cach-refinement-alt-def cach-refinement-list-def map-fun-rel-def intro*: *bounded-included-le*)

definition *empty-cach-ref-pre* **where**
‹*empty-cach-ref-pre* = (λ(*cach* :: *minimize-status list*, *supp* :: *nat list*).
(∀ L∈*set supp*. L < *length cach*) ∧
length supp ≤ *Suc* (*uint32-max div 2*) ∧
(∀ L<*length cach*. *cach* ! L ≠ *SEEN-UNKNOWN* ⟶ L ∈ *set supp*))›

Minimisation of the conflict

definition *extract-shorter-conflict-list-heur-st*
:: ‹*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × - × -) *nres*›
where
‹*extract-shorter-conflict-list-heur-st* = (λ(M, N, (-, D), Q', W', *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *ccont*, *vdom*). *do* {
ASSERT(*fst* M ≠ []);
let K = *lit-of-last-trail-pol* M;
ASSERT(*0* < *length outl*);
ASSERT(*lookup-conflict-remove1-pre* (−K, D));
let D = *lookup-conflict-remove1* (−K) D;
let outl = *outl*[*0* := −K];
vm ← *isa-vmtf-mark-to-rescore-also-reasons* M N *outl vm*;
(D, *cach*, *outl*) ← *isa-minimize-and-extract-highest-lookup-conflict* M N D *cach lbd outl*;
ASSERT(*empty-cach-ref-pre cach*);

let cach = *empty-cach-ref cach*;
ASSERT(*outl* ≠ [] ∧ *length outl* ≤ *uint32-max*);
(*D*, *C*, *n*) ← *isa-empty-conflict-and-extract-clause-heur M D outl*;
RETURN ((*M*, *N*, *D*, *Q*′, *W*′, *vm*, *clvls*, *cach*, *lbd*, *take 1 outl*, *stats*, *ccont*, *vdom*), *n*, *C*)
})›

lemma *the-option-lookup-clause-assn*:
‹(*RETURN o snd*, *RETURN o the*) ∈ [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* 𝒜 → ⟨*lookup-clause-rel* 𝒜⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def*)

definition *update-heuristics* **where**
‹*update-heuristics* = (λ*glue* (*fema*, *sema*, *res-info*, *wasted*).
(*ema-update glue fema*, *ema-update glue sema*,
incr-conflict-count-since-last-restart res-info, *wasted*))›

lemma *heuristic-rel-update-heuristics*[*intro*!]:
‹*heuristic-rel* 𝒜 *heur* ⟹ *heuristic-rel* 𝒜 (*update-heuristics glue heur*)›
by (*auto simp*: *heuristic-rel-def phase-save-heur-rel-def phase-saving-def*
update-heuristics-def)

definition *propagate-bt-wl-D-heur*
:: ‹*nat literal* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*length vdom* ≤ *length N0*);
ASSERT(*length avdom* ≤ *length N0*);
ASSERT(*nat-of-lit* (*C*!*1*) < *length W0* ∧ *nat-of-lit* (−*L*) < *length W0*);
ASSERT(*length C* > *1*);
let L′ = *C*!*1*;
ASSERT(*length C* ≤ *uint32-max div 2* + *1*);
vm ← *isa-vmtf-rescore C M vm0*;
glue ← *get-LBD lbd*;
let b = *False*;
let b′ = (*length C* = *2*);
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*) ⟶ *append-and-length-fast-code-pre* ((*b*, *C*), *N0*));
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*) ⟶ *lcount* < *sint64-max*);
(*N*, *i*) ← *fm-add-new b C N0*;
ASSERT(*update-lbd-pre* ((*i*, *glue*), *N*));
let N = *update-lbd i glue N*;
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*) ⟶ *length-ll W0* (*nat-of-lit* (−*L*)) < *sint64-max*);
let W = *W0*[*nat-of-lit* (− *L*) := *W0* ! *nat-of-lit* (− *L*) @ [(*i*, *L*′, *b*′)]];
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*) ⟶ *length-ll W* (*nat-of-lit L*′) < *sint64-max*);
let W = *W*[*nat-of-lit L*′ := *W*!*nat-of-lit L*′ @ [(*i*, −*L*, *b*′)]];
lbd ← *lbd-empty lbd*;
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
ASSERT(*i* ≠ *DECISION-REASON*);
ASSERT(*cons-trail-Propagated-tr-pre* ((−*L*, *i*), *M*));
M ← *cons-trail-Propagated-tr* (− *L*) *i M*;
vm ← *isa-vmtf-flush-int M vm*;

heur ← *mop-save-phase-heur* (*atm-of* L') (*is-neg* L') *heur*;
RETURN (*M*, *N*, *D*, *j*, *W*, *vm*, *0*,
cach, *lbd*, *outl*, *add-lbd* (*of-nat glue*) *stats*, *update-heuristics glue heur*, *vdom* @ [*i*],
avdom @ [*i*],
lcount + *1*, *opts*)
})›

definition (**in** −) *lit-of-hd-trail-st-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal nres*› **where**
‹*lit-of-hd-trail-st-heur S* = *do* {*ASSERT* (*fst* (*get-trail-wl-heur S*) ≠ []); *RETURN* (*lit-of-last-trail-pol* (*get-trail-wl-heur S*))}›

definition *remove-last*
:: ‹*nat literal* ⇒ *nat clause option* ⇒ *nat clause option nres*›
where
‹*remove-last* - - = *SPEC*((=) *None*)›

definition *propagate-unit-bt-wl-D-int*
:: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*propagate-unit-bt-wl-D-int* = (λ*L* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*). *do* {
vm ← *isa-vmtf-flush-int M vm*;
glue ← *get-LBD lbd*;
lbd ← *lbd-empty lbd*;
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
ASSERT(*0* ≠ *DECISION-REASON*);
ASSERT(*cons-trail-Propagated-tr-pre* ((− *L*, *0*::*nat*), *M*));
M ← *cons-trail-Propagated-tr* (− *L*) *0 M*;
let stats = *incr-uset stats*;
RETURN (*M*, *N*, *D*, *j*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
(*update-heuristics glue heur*), *vdom*)})›

Full function **definition** *backtrack-wl-D-nlit-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*backtrack-wl-D-nlit-heur* S_0 =
do {
ASSERT(*backtrack-wl-D-heur-inv* S_0);
ASSERT(*fst* (*get-trail-wl-heur* S_0) ≠ []);
L ← *lit-of-hd-trail-st-heur* S_0;
(*S*, *n*, *C*) ← *extract-shorter-conflict-list-heur-st* S_0;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);
S ← *find-decomp-wl-st-int n S*;

ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);
if size C > *1*
then do {
S ← *propagate-bt-wl-D-heur L C S*;
save-phase-st S
}
else do {
propagate-unit-bt-wl-D-int L S
}
}›

lemma *get-all-ann-decomposition-get-level*:
assumes
L': ⟨*L' = lit-of (hd M')*⟩ **and**
nd: ⟨*no-dup M'*⟩ **and**
decomp: ⟨*(Decided K # a, M2)* ∈ *set (get-all-ann-decomposition M')*⟩ **and**
lev-K: ⟨*get-level M' K = Suc (get-maximum-level M' (remove1-mset (− L') y))*⟩ **and**
L: ⟨*L* ∈# *remove1-mset (− lit-of (hd M')) y*⟩
shows ⟨*get-level a L = get-level M' L*⟩
proof −
obtain *M3* **where** *M3*: ⟨*M' = M3 @ M2 @ Decided K # a*⟩
using *decomp* **by** *blast*
from *lev-K* **have** *lev-L*: ⟨*get-level M' L < get-level M' K*⟩
using *get-maximum-level-ge-get-level[OF L, of M']* **unfolding** *L'[symmetric]* **by** *auto*
have *[simp]*: ⟨*get-level (M3 @ M2 @ Decided K # a) K = Suc (count-decided a)*⟩
using *nd* **unfolding** *M3* **by** *auto*
have *undef*:⟨*undefined-lit (M3 @ M2) L*⟩
using *lev-L get-level-skip-end[of* ⟨*M3 @ M2*⟩ *L* ⟨*Decided K # a*⟩*]* **unfolding** *M3*
by *auto*
then have ⟨*atm-of L* ≠ *atm-of K*⟩
using *lev-L* **unfolding** *M3* **by** *auto*
then show *?thesis*
using *undef* **unfolding** *M3* **by** *(auto simp: get-level-cons-if)*
qed

definition *del-conflict-wl* :: ⟨*'v twl-st-wl* ⇒ *'v twl-st-wl*⟩ **where**
⟨*del-conflict-wl* = (λ(*M, N, D, NE, UE, Q, W*). (*M, N, None, NE, UE, Q, W*))⟩

lemma *[simp]*:
⟨*get-clauses-wl (del-conflict-wl S) = get-clauses-wl S*⟩
by *(cases S) (auto simp: del-conflict-wl-def)*

lemma *lcount-add-clause[simp]*: ⟨*i* ∉# *dom-m N* ⟹
size (learned-clss-l (fmupd i (C, False) N)) = Suc (size (learned-clss-l N))⟩
by *(simp add: learned-clss-l-mapsto-upd-notin)*

lemma *length-watched-le*:
assumes
prop-inv: ⟨*correct-watching x1*⟩ **and**
xb-x'a: ⟨*(x1a, x1)* ∈ *twl-st-heur-conflict-ana*⟩ **and**
x2: ⟨*x2* ∈# $\mathcal{L}_{all}$ *(all-atms-st x1)*⟩
shows ⟨*length (watched-by x1 x2)* ≤ *length (get-clauses-wl-heur x1a) − 2*⟩
proof −
have ⟨*correct-watching x1*⟩
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ⟨*distinct-watched (watched-by x1 x2)*⟩
using *x2* **unfolding** *all-atms-def[symmetric] all-lits-alt-def[symmetric]*
by *(cases x1; auto simp:* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*
$\mathcal{L}_{all}$*-all-atms-all-lits*
simp flip: all-lits-alt-def2 all-lits-def all-atms-def)
then have *dist*: ⟨*distinct-watched (watched-by x1 x2)*⟩
using *xb-x'a*
by *(cases x1; auto simp:* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps)*
have *dist-vdom*: ⟨*distinct (get-vdom x1a)*⟩
using *xb-x'a*

by (*cases x1*)
  (*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def*)
have *x2*: ‹*x2* $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st x1*)›
  using *x2 xb-x'a* **unfolding** *all-atms-def*
  by *auto*

have
  valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
  using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
  by (*cases x1*)
  (*auto simp*: *twl-st-heur'-def twl-st-heur-conflict-ana-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) $\subseteq$ *set* (*get-vdom x1a*)›
  using *xb-x'a*
  by (*cases x1*)
  (*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def all-atms-def*[*symmetric*])
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) $\subseteq$ *set* (*get-vdom x1a*)›
  using *x2* **unfolding** *vdom-m-def*
  by (*cases x1*)
  (*force simp*: *twl-st-heur'-def twl-st-heur-def simp flip*: *all-atms-def*
  dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) $\subseteq\#$ *mset* (*get-vdom x1a*)›
  by (*rule distinct-subseteq-iff*[*THEN iffD1*])
  (*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
  ‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) $\subseteq$ {*4* ..< *length* (*get-clauses-wl-heur x1a*)}›
  using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) $\leq$ *length* (*get-clauses-wl-heur x1a*) $-$ *4*›
  by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
  (*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show *?thesis*
  using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
  by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

definition *single-of-mset* **where**
‹*single-of-mset D* = *SPEC*(λ*L*. *D* = *mset* [*L*])›

lemma *length-list-ge2*: ‹*length S* $\geq$ *2* $\longleftrightarrow$ ($\exists$ *a b S'*. *S* = [*a*, *b*] @ *S'*)›
  apply (*cases S*)
  apply (*simp*; *fail*)
  apply (*rename-tac a S'*)
  apply (*case-tac S'*)
  by *simp-all*

lemma *backtrack-wl-D-nlit-backtrack-wl-D*:
‹(*backtrack-wl-D-nlit-heur*, *backtrack-wl*) $\in$
{(*S*, *T*). (*S*, *T*) $\in$ *twl-st-heur-conflict-ana* $\wedge$ *length* (*get-clauses-wl-heur S*) = *r*} $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) $\in$ *twl-st-heur* $\wedge$ *length* (*get-clauses-wl-heur S*) $\leq$ *6* + *r* + *uint32-max div 2*}⟩*nres-rel*›
(**is** ‹- $\in$ *?R* $\rightarrow_f$ ⟨*?S*⟩*nres-rel*›)
proof $-$
  have *backtrack-wl-D-nlit-heur-alt-def*: ‹*backtrack-wl-D-nlit-heur* S_0 =
  do {

ASSERT(*backtrack-wl-D-heur-inv* S_0);
ASSERT(*fst* (*get-trail-wl-heur* S_0) ≠ []);
L ← *lit-of-hd-trail-st-heur* S_0;
(*S*, *n*, *C*) ← *extract-shorter-conflict-list-heur-st* S_0;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);
S ← *find-decomp-wl-st-int n S*;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur* S_0);

if size C > *1*
then do {
let - = *C* ! *1*;
S ← *propagate-bt-wl-D-heur L C S*;
save-phase-st S
}
else do {
propagate-unit-bt-wl-D-int L S
}
}› **for** S_0
unfolding *backtrack-wl-D-nlit-heur-def Let-def*
by *auto*
have *inv*: ‹*backtrack-wl-D-heur-inv S'*›
if
‹*backtrack-wl-inv S*› **and**
‹(*S'*, *S*) ∈ *?R*›
for *S S'*
using *that* **unfolding** *backtrack-wl-D-heur-inv-def*
by (*cases S*; *cases S'*) (*blast intro*: *exI*[*of - S'*])
have *shorter*:
‹*extract-shorter-conflict-list-heur-st S'*
≤ ⇓ {((*T'*, *n*, *C*), *T*). (*T'*, *del-conflict-wl T*) ∈ *twl-st-heur-bt* ∧
n = *get-maximum-level* (*get-trail-wl T*)
(*remove1-mset* (−*lit-of*(*hd* (*get-trail-wl T*))) (*the* (*get-conflict-wl T*))) ∧
mset C = *the* (*get-conflict-wl T*) ∧
get-conflict-wl T ≠ *None*∧
equality-except-conflict-wl T S ∧
get-clauses-wl-heur T' = *get-clauses-wl-heur S'* ∧
(*1* < *length C* ⟶
highest-lit (*get-trail-wl T*) (*mset* (*tl C*))
(*Some* (*C* ! *1*, *get-level* (*get-trail-wl T*) (*C* ! *1*)))) ∧
C ≠ [] ∧ *hd C* = −*lit-of*(*hd* (*get-trail-wl T*)) ∧
mset C ⊆# *the* (*get-conflict-wl S*) ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*)) ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st T*) (*get-trail-wl T*) ∧
get-conflict-wl S ≠ *None* ∧
− *lit-of* (*hd* (*get-trail-wl S*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st T*) *T* ∧
n < *count-decided* (*get-trail-wl T*) ∧
get-trail-wl T ≠ [] ∧
¬ *tautology* (*mset C*) ∧
correct-watching S ∧ *length* (*get-clauses-wl-heur T'*) = *length* (*get-clauses-wl-heur S'*)}
(*extract-shorter-conflict-wl S*)›
(**is** ‹- ≤ ⇓ *?shorter* -›)
if
inv: ‹*backtrack-wl-inv S*› **and**
S'-S: ‹(*S'*, *S*) ∈ *?R*›

for *S S'*
proof −
obtain *M N D NE UE NS US Q W* **where**
S: ‹*S* = (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)›
by (*cases S*)
obtain *M' W' vm clvls cach lbd outl stats heur avdom vdom lcount D' arena b Q' opts* **where**
S': ‹*S'* = (*M'*, *arena*, (*b*, *D'*), *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*)›
using *S'-S* **by** (*cases S'*) (*auto simp*: *twl-st-heur-conflict-ana-def S*)
have
M'-M: ‹(*M'*, *M*) ∈ *trail-pol* (*all-atms-st S*)› **and**
‹(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st S*))› **and**
vm: ‹*vm* ∈ *isa-vmtf* (*all-atms-st S*) *M*› **and**
n-d: ‹*no-dup M*› **and**
‹*clvls* ∈ *counts-maximum-level M D*› **and**
cach-empty: ‹*cach-refinement-empty* (*all-atms-st S*) *cach*› **and**
outl: ‹*out-learned M D outl*› **and**
lcount: ‹*lcount* = *size* (*learned-clss-l N*)› **and**
‹*vdom-m* (*all-atms-st S*) *W N* ⊆ *set vdom*› **and**
D': ‹((*b*, *D'*), *D*) ∈ *option-lookup-clause-rel* (*all-atms-st S*)› **and**
arena: ‹*valid-arena arena N* (*set vdom*)› **and**
avdom: ‹*mset avdom* ⊆# *mset vdom*› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st S*)›
using *S'-S* **unfolding** *S S' twl-st-heur-conflict-ana-def*
by (*auto simp*: *S all-atms-def*[*symmetric*])
obtain *T U* **where**
$\mathcal{L}_{in}$:‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S*› **and**
S-T: ‹(*S*, *T*) ∈ *state-wl-l None*› **and**
corr: ‹*correct-watching S*› **and**
T-U: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
trail-nempty: ‹*get-trail-l T* ≠ []› **and**
not-none: ‹*get-conflict-l T* ≠ *None*› **and**
struct-invs: ‹*twl-struct-invs U*› **and**
stgy-invs: ‹*twl-stgy-invs U*› **and**
list-invs: ‹*twl-list-invs T*› **and**
not-empty: ‹*get-conflict-l T* ≠ *Some* {#}› **and**
uL-D: ‹− *lit-of* (*hd* (*get-trail-wl S*)) ∈# *the* (*get-conflict-wl S*)› **and**
nss: ‹*no-step cdcl*$_W$*-restart-mset.skip* (*state*$_W$*-of U*)› **and**
nsr: ‹*no-step cdcl*$_W$*-restart-mset.resolve* (*state*$_W$*-of U*)›
using *inv* **unfolding** *backtrack-wl-inv-def backtrack-wl-inv-def backtrack-l-inv-def backtrack-inv-def*
apply −
apply *normalize-goal*+ **by** *simp*
have *D-none*: ‹*D* ≠ *None*›
using *S-T not-none* **by** (*auto simp*: *S*)
have *b*: ‹¬*b*›
using *D' not-none S-T* **by** (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def*)
have *all-struct*:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of U*)›
using *struct-invs*
by (*auto simp*: *twl-struct-invs-def*)
have
‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)› **and**
lev-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of U*)› **and**
‹∀ *s*∈#*learned-clss* (*state*$_W$*-of U*). ¬ *tautology s*› **and**
dist: ‹*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of U*)› **and**
confl: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of U*)› **and**

⟨*all-decomposition-implies-m* (*cdcl*$_W$*-restart-mset.clauses* (*state*$_W$*-of* *U*))
(*get-all-ann-decomposition* (*trail* (*state*$_W$*-of* *U*)))⟩ **and**
learned: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*state*$_W$*-of* *U*)⟩
using *all-struct* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast*+
have *n-d*: ⟨*no-dup M*⟩
using *lev-inv S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *twl-st S*)
have *M-*$\mathcal{L}_{in}$: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)⟩
apply (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*[*OF S-T struct-invs T-U* $\mathcal{L}_{in}$]) **.**
have *dist-D*: ⟨*distinct-mset* (*the* (*get-conflict-wl S*))⟩
using *dist not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def S*
by (*auto simp*: *twl-st*)
have ⟨*the* (*conflicting* (*state*$_W$*-of* *U*)) =
add-mset (− *lit-of* (*cdcl*$_W$*-restart-mset.hd-trail* (*state*$_W$*-of* *U*)))
{#*L* ∈# *the* (*conflicting* (*state*$_W$*-of* *U*)). *get-level* (*trail* (*state*$_W$*-of* *U*)) *L*
< *backtrack-lvl* (*state*$_W$*-of* *U*)#}⟩
apply (*rule cdcl*$_W$*-restart-mset.no-skip-no-resolve-single-highest-level*)
subgoal using *nss* **unfolding** *S* **by** *simp*
subgoal using *nsr* **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *twl-struct-invs-def S* **by** *simp*
subgoal using *stgy-invs* **unfolding** *twl-stgy-invs-def S* **by** *simp*
subgoal using *not-none S-T T-U* **by** (*simp add*: *twl-st*)
subgoal using *not-empty not-none S-T T-U* **by** (*auto simp add*: *twl-st*)
done
then have *D-filter*: ⟨*the D* = *add-mset* (− *lit-of* (*hd M*)) {#*L* ∈# *the D*. *get-level M L* < *count-decided M*#}⟩
using *trail-nempty S-T T-U* **by** (*simp add*: *twl-st S*)
have *tl-outl-D*: ⟨*mset* (*tl* (*outl*[*0* := − *lit-of* (*hd M*)])) = *remove1-mset* (*outl*[*0* := − *lit-of* (*hd M*)] ! *0*) (*the D*)⟩
using *outl S-T T-U not-none*
apply (*subst D-filter*)
by (*cases outl*) (*auto simp*: *out-learned-def S*)
let *?D* = ⟨*remove1-mset* (− *lit-of* (*hd M*)) (*the D*)⟩
have $\mathcal{L}_{in}$*-S*: ⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*))⟩
apply (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-in-*$\mathcal{L}_{in}$*-conflict*[*OF S-T - T-U*])
using $\mathcal{L}_{in}$ *not-none struct-invs not-none S-T T-U* **by** (*auto simp*: *S*)
then have $\mathcal{L}_{in}$*-D*: ⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) *?D*⟩
unfolding *S* **by** (*auto intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)
have $\mathcal{L}_{in}$*-NU*: ⟨*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* '# *ran-mf* (*get-clauses-wl S*))⟩

by (*auto simp*: *all-atms-def all-lits-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*)
(*simp add*: *all-lits-of-mm-union*)
have *tauto-confl*: ⟨¬ *tautology* (*the* (*get-conflict-wl S*))⟩
apply (*rule conflict-not-tautology*[*OF S-T - T-U*])
using *struct-invs not-none S-T T-U* **by** (*auto simp*: *twl-st*)
from *not-tautology-mono*[*OF - this*, *of ?D*] **have** *tauto-D*: ⟨¬ *tautology ?D*⟩
by (*auto simp*: *S*)
have *entailed*:
⟨*mset* '# *ran-mf* (*get-clauses-wl S*) + (*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) +
(*get-subsumed-init-clauses-wl S* + *get-subsumed-learned-clauses-wl S*)⊨*pm*
add-mset (− *lit-of* (*hd* (*get-trail-wl S*)))
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl S*))) (*the* (*get-conflict-wl S*)))⟩
using *uL-D learned not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause-alt-def*
by (*auto simp*: *ac-simps twl-st get-unit-clauses-wl-alt-def*)

define *cach′* **where** ‹*cach′* = (λ-::*nat*. *SEEN-UNKNOWN*)›
have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) (*get-trail-wl S*) (*get-clauses-wl S*)
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf* (*get-clauses-wl S*))
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*))
?D)›
apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*of S T U*
‹*outl* [*0* := − *lit-of* (*hd M*)]›
‹*remove1-mset* - (*the D*)› ‹*all-atms-st S*› *cach′* ‹−*lit-of* (*hd M*)› *lbd*])
subgoal using *S-T* **.**
subgoal using *T-U* **.**
subgoal using ‹*out-learned M D outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *confl not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
by (*auto simp*: *true-annot-CNot-diff twl-st S*)
subgoal
using *dist not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def*
by (*auto simp*: *twl-st S*)
subgoal using *tauto-D* **.**
subgoal using *M-*$\mathcal{L}_{in}$ **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *S* **by** *simp*
subgoal using *list-invs* **unfolding** *S* **by** *simp*
subgoal using *M-*$\mathcal{L}_{in}$ *cach-empty S-T T-U*
unfolding *cach-refinement-empty-def conflict-min-analysis-inv-def*
by (*auto dest*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms simp*: *cach′-def twl-st S*)
subgoal using *entailed* **unfolding** *S* **by** (*simp add*: *ac-simps*)
subgoal using $\mathcal{L}_{in}$*-D* **.**
subgoal using $\mathcal{L}_{in}$*-NU* **.**
subgoal using ‹*out-learned M D outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using ‹*out-learned M D outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *bounded* **unfolding** *all-atms-def* **by** (*simp add*: *S*)
done
then have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf N*)
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* +
get-subsumed-init-clauses-wl S)) *?D*)›
unfolding *S* **by** *auto*
have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}

(*SPEC* (*λD′*. *D′* ⊆# *?D* ∧ *mset* ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* +
get-subsumed-init-clauses-wl S)) ⊨*pm add-mset* (− *lit-of* (*hd M*)) *D′*))›
apply (*rule order.trans*)
apply (*rule mini*)
apply (*rule ref-two-step′*)
apply (*rule order.trans*)
apply (*rule iterate-over-conflict-spec*)
subgoal using *entailed* **by** (*auto simp*: *S ac-simps*)
subgoal
using *dist not-none S-T T-U* **unfolding** *S* $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
by (*auto simp*: *twl-st*)
subgoal by (*auto simp*: *ac-simps*)
done
have *uM-*$\mathcal{L}_{all}$: ‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
using *M-*$\mathcal{L}_{in}$ *trail-nempty S-T T-U* **by** (*cases M*)
(*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons uminus-*$\mathcal{A}_{in}$*-iff twl-st S*)

have *L-D*: ‹*lit-of* (*hd M*) ∉# *the D*› **and**
tauto-confl′: ‹¬*tautology* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*))›
using *uL-D tauto-confl*
by (*auto dest*!: *multi-member-split simp*: *S add-mset-eq-add-mset tautology-add-mset*)
then have *pre1*: ‹*D* ≠ *None* ∧ *delete-from-lookup-conflict-pre* (*all-atms-st S*) (− *lit-of* (*hd M*), *the D*)›
using *not-none uL-D uM-*$\mathcal{L}_{all}$ *S-T T-U* **unfolding** *delete-from-lookup-conflict-pre-def*
by (*auto simp*: *twl-st S*)
have *pre2*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*) ≡ *True*›
and *lits-N*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*)›
using *M-*$\mathcal{L}_{in}$ *S-T T-U not-none* $\mathcal{L}_{in}$
unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*-def all-atms-def all-lits-def*
by (*auto simp*: *twl-st S all-lits-of-mm-union*)
have ‹*0* < *length outl*›
using ‹*out-learned M D outl*›
by (*auto simp*: *out-learned-def*)
have *trail-nempty*: ‹*M* ≠ []›
using *trail-nempty S-T T-U*
by (*auto simp*: *twl-st S*)

have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of* (*hd M*), *D′*)›
using *D′ not-none not-empty S-T uM-*$\mathcal{L}_{all}$
unfolding *lookup-conflict-remove1-pre-def*
by (*cases* ‹*the D*›)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def S*
state-wl-l-def atms-of-def)
then have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of-last-trail-pol M′*, *D′*)›
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of M M′*])
(*use M′-M trail-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def*›)

have ‹− *lit-of* (*hd M*) ∈# (*the D*)›
using *uL-D* **by** (*auto simp*: *S*)
then have *extract-shorter-conflict-wl-alt-def*:
‹*extract-shorter-conflict-wl* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) = *do* {
let K = *lit-of* (*hd M*);
let D = (*remove1-mset* (−*K*) (*the D*));

- ← *RETURN* (); ~~*vmtf rescoring*~~
E′ ← (*SPEC*
(λ(*E′*). *E′* ⊆# *add-mset* (−*K*) *D* ∧ − *lit-of* (*hd M*) :# *E′* ∧
mset ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* +
get-subsumed-init-clauses-wl S)) ⊨*pm E′*));
D ← *RETURN* (*Some E′*);
RETURN (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)
}›
unfolding *extract-shorter-conflict-wl-def*
by (*auto simp*: *RES-RETURN-RES image-iff mset-take-mset-drop-mset′ S union-assoc*
Un-commute Let-def Un-assoc sup-left-commute)

have *lookup-clause-rel-unique*: ‹(*D′*, *a*) ∈ *lookup-clause-rel* 𝒜 ⟹ (*D′*, *b*) ∈ *lookup-clause-rel* 𝒜 ⟹ *a* = *b*›
for *a b* 𝒜
by (*auto simp*: *lookup-clause-rel-def mset-as-position-right-unique*)
have *isa-minimize-and-extract-highest-lookup-conflict*:
‹*isa-minimize-and-extract-highest-lookup-conflict M′ arena*
(*lookup-conflict-remove1* (−*lit-of* (*hd M*)) *D′*) *cach lbd* (*outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*).
(*E*, *mset* (*tl outl*)) ∈ *lookup-clause-rel* (*all-atms-st S*) ∧
mset outl = *E′* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧
E′ ⊆# *the D* ∧ *outl* ≠ [] ∧ *distinct outl* ∧ *literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset outl*) ∧
¬*tautology* (*mset outl*) ∧
(∃ *cach′*. (*s*, *cach′*) ∈ *cach-refinement* (*all-atms-st S*))}
(*SPEC* (λ*E′*. *E′* ⊆# *add-mset* (− *lit-of* (*hd M*)) (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*)) ∧
− *lit-of* (*hd M*) ∈# *E′* ∧
mset ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* +
get-subsumed-init-clauses-wl S)) ⊨*pm*
E′))›
(**is** ‹- ≤ ⇓ *?minimize* (*RES ?E*)›)
apply (*rule order-trans*)
apply (*rule*
isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict
[*THEN fref-to-Down-curry5*,
of ‹*all-atms-st S*› *M N* ‹*remove1-mset* (−*lit-of* (*hd M*)) (*the D*)› *cach′ lbd* ‹*outl*[*0* := − *lit-of* (*hd M*)]›
- - - - - - ‹*set vdom*›])
subgoal using *bounded* **by** (*auto simp*: *S all-atms-def*)
subgoal using *tauto-confl′ pre2* **by** *auto*
subgoal using *D′ not-none arena S-T uL-D uM-*$\mathcal{L}_{all}$ *not-empty D′ L-D b cach-empty M′-M*
unfolding *all-atms-def*
by (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def image-image cach-refinement-empty-def cach′-def*
intro!: *lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*]
dest: *multi-member-split lookup-clause-rel-unique*)
apply (*rule order-trans*)
apply (*rule mini*[*THEN ref-two-step′*])
subgoal
using *uL-D dist-D tauto-D* $\mathcal{L}_{in}$*-S* $\mathcal{L}_{in}$*-D tauto-D L-D*
by (*fastforce simp*: *conc-fun-chain conc-fun-RES image-iff S union-assoc insert-subset-eq-iff*

neq-Nil-conv literals-are-in-$\mathcal{L}_{in}$-add-mset tautology-add-mset
intro: *literals-are-in-$\mathcal{L}_{in}$-mono*
dest: *distinct-mset-mono not-tautology-mono*
dest!: *multi-member-split*)
done

have *empty-conflict-and-extract-clause-heur*: ‹*isa-empty-conflict-and-extract-clause-heur* M' *x1 x2a*
$\leq \Downarrow$ ({((*E*, *outl*, *n*), E').
(*E*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st S*) ∧
mset outl = *the* E' ∧
outl ! *0* = − *lit-of* (*hd M*) ∧
the E' ⊆# *the D* ∧ *outl* ≠ [] ∧ E' ≠ *None* ∧
(*1* < *length outl* ⟶
highest-lit M (*mset* (*tl outl*)) (*Some* (*outl* ! *1*, *get-level M* (*outl* ! *1*)))) ∧
(*1* < *length outl* ⟶ *n* = *get-level M* (*outl* ! *1*)) ∧ (*length outl* = *1* ⟶ *n* = *0*)}) (*RETURN*
(*Some* E'))›
(**is** ‹- ≤ ⇓ *?empty-conflict* -›)
if
‹*M* ≠ []› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)› **and**
‹*0* < *length outl*› **and**
‹*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), D')› **and**
‹(*x*, E') ∈ *?minimize*› **and**
‹E' ∈ *?E*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x* = (*x1*, *x2*)›
for *x* :: ‹(*nat* × *bool option list*) × (*minimize-status list* × *nat list*) × *nat literal list*› **and**
E' :: ‹*nat literal multiset*› **and**
x1 :: ‹*nat* × *bool option list*› **and**
x2 :: ‹(*minimize-status list* × *nat list*) × *nat literal list*› **and**
x1a :: ‹*minimize-status list* × *nat list*› **and**
x2a :: ‹*nat literal list*›
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur*
[*THEN fref-to-Down-curry2*, *of - - - M x1 x2a* ‹*all-atms-st S*›])
apply *fast*
subgoal using *M'-M* **by** *auto*
apply (*subst Down-id-eq*)
apply (*rule order.trans*)
apply (*rule empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause*[*of* ‹*mset* (*tl*
x2a)›])
subgoal by *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal using *bounded* **unfolding** *S all-atms-def* **by** *simp*
subgoal unfolding *empty-conflict-and-extract-clause-def*
using *that*
by (*auto simp*: *conc-fun-RES RETURN-def*)
done
qed

have *final*: ‹((((M', *arena*, *x1b*, Q', W', vm', *clvls*, *empty-cach-ref x1a*, *lbd*, *take 1 x2a*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*),
x2c, *x1c*),
M, *N*, *Da*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)
∈ *?shorter*›
if
‹$M \neq []$› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)› **and**
‹*0* < *length outl*› **and**
‹*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), D')› **and**
mini: ‹(*x*, E') ∈ *?minimize*› **and**
‹E' ∈ *?E*› **and**
‹(*xa*, *Da*) ∈ *?empty-conflict*› **and**
st[*simp*]:
‹*x2b* = (*x1c*, *x2c*)›
‹*x2* = (*x1a*, *x2a*)›
‹*x* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
vm': ‹(vm', *uu*) ∈ {(*c*, *uu*). *c* ∈ *isa-vmtf* (*all-atms-st S*) *M*}›
for *x* E' *x1 x2 x1a x2a xa Da x1b x2b x1c x2c* vm' *uu*
proof −
have *x1b-None*: ‹(*x1b*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st S*)›
using *that* **apply** (*auto simp*: *S simp flip*: *all-atms-def*)
done
have *cach*[*simp*]: ‹*cach-refinement-empty* (*all-atms-st S*) (*empty-cach-ref x1a*)›
using *empty-cach-ref-empty-cach*[*of* ‹*all-atms-st S*›, *THEN fref-to-Down-unRET*, *of x1a*]
mini bounded
by (*auto simp add*: *cach-refinement-empty-def empty-cach-def* $cach'$*-def S*
simp flip: *all-atms-def*)

have
out: ‹*out-learned M None* (*take* (*Suc 0*) *x2a*)› **and**
x1c-Da: ‹*mset x1c* = *the Da*› **and**
Da-None: ‹*Da* ≠ *None*› **and**
Da-D: ‹*the Da* ⊆# *the D*› **and**
x1c-D: ‹*mset x1c* ⊆# *the D*› **and**
x1c: ‹*x1c* ≠ []› **and**
hd-x1c: ‹*hd x1c* = − *lit-of* (*hd M*)› **and**
highest: ‹*Suc 0* < *length x1c* ⟹ *x2c* = *get-level M* (*x1c* ! *1*) ∧
highest-lit M (*mset* (*tl x1c*))
(*Some* (*x1c* ! *Suc 0*, *get-level M* (*x1c* ! *Suc 0*)))› **and**
highest2: ‹*length x1c* = *Suc 0* ⟹ *x2c* = *0*› **and**
‹E' = *mset x2a*› **and**
‹− *lit-of* (*M* ! *0*) ∈ *set x2a*› **and**
‹(λ*x*. *mset* (*fst x*)) ‘ *set-mset* (*ran-m N*) ∪
(*set-mset* (*get-unit-learned-clss-wl S*) ∪
set-mset (*get-unit-init-clss-wl S*)) ∪
(*set-mset* (*get-subsumed-learned-clauses-wl S*) ∪
set-mset (*get-subsumed-init-clauses-wl S*)) ⊨*p*
mset x2a› **and**
‹*x2a* ! *0* = − *lit-of* (*M* ! *0*)› **and**
‹*x1c* ! *0* = − *lit-of* (*M* ! *0*)› **and**
‹*mset x2a* ⊆# *the D*› **and**
‹*mset x1c* ⊆# *the D*› **and**
‹*x2a* ≠ []› **and**
x1c-nempty: ‹*x1c* ≠ []› **and**

⟨*distinct x2a*⟩ **and**
Da: ⟨*Da* = *Some* (*mset x1c*)⟩ **and**
⟨*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset x2a*)⟩ **and**
⟨¬ *tautology* (*mset x2a*)⟩
using *that*
unfolding *out-learned-def*
by (*auto simp add*: *hd-conv-nth S ac-simps simp flip*: *all-atms-def*)
have *Da-D′*: ⟨*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*) ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*)⟩
using *Da-D mset-le-subtract* **by** *blast*

have *K*: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*state*$_W$*-of U*)⟩
using *stgy-invs* **unfolding** *twl-stgy-invs-def* **by** *fast*
have ⟨*get-maximum-level M* {#*L* ∈# *the D*. *get-level M L* < *count-decided M*#}
< *count-decided M*⟩
using *cdcl*$_W$*-restart-mset.no-skip-no-resolve-level-get-maximum-lvl-le*[*OF nss nsr all-struct K*]
not-none not-empty confl trail-nempty S-T T-U
unfolding *get-maximum-level-def* **by** (*auto simp*: *twl-st S*)
then have
⟨*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*)) < *count-decided M*⟩
by (*subst D-filter*) *auto*
then have *max-lvl-le*:
⟨*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*)) < *count-decided M*⟩
using *get-maximum-level-mono*[*OF Da-D′*, *of M*] **by** *auto*
have ⟨((*M′*, *arena*, *x1b*, *Q′*, *W′*, *vm′*, *clvls*, *empty-cach-ref x1a*, *lbd*, *take* (*Suc 0*) *x2a*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*),
del-conflict-wl (*M*, *N*, *Da*, *NE*, *UE*, *NS*, *US*, *Q*, *W*))
∈ *twl-st-heur-bt*⟩
using *S′-S x1b-None cach out vm′* **unfolding** *twl-st-heur-bt-def*
by (*auto simp*: *twl-st-heur-def del-conflict-wl-def S S′ twl-st-heur-bt-def*
twl-st-heur-conflict-ana-def S simp flip: *all-atms-def*)
moreover have *x2c*: ⟨*x2c* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*))⟩
using *highest highest2 x1c-nempty hd-x1c*
by (*cases* ⟨*length x1c* = *Suc 0*⟩; *cases x1c*)
(*auto simp*: *highest-lit-def Da mset-tl*)
moreover have ⟨*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*M*, *N*, *Some* (*mset x1c*), *NE*, *UE*, *NS*, *US*, *Q*, *W*)⟩
using $\mathcal{L}_{in}$
by (*auto simp*: *S x2c literals-are-*$\mathcal{L}_{in}$*-def blits-in-*$\mathcal{L}_{in}$*-def simp flip*: *all-atms-def*)
moreover have ⟨¬*tautology* (*mset x1c*)⟩
using *tauto-confl not-tautology-mono*[*OF x1c-D*]
by (*auto simp*: *S x2c S′*)
ultimately show *?thesis*
using $\mathcal{L}_{in}$*-S x1c-Da Da-None dist-D D-none x1c-D x1c hd-x1c highest uM-*$\mathcal{L}_{all}$ *vm′ M-*$\mathcal{L}_{in}$
max-lvl-le corr trail-nempty **unfolding** *literals-are-*$\mathcal{L}_{in}$*-def*
by (*simp add*: *S x2c S′*)
qed
have *hd-M′-M*: ⟨*lit-of-last-trail-pol M′* = *lit-of* (*hd M*)⟩
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of M M′*])
(*use M′-M trail-nempty* **in** ⟨*auto simp*: *lit-of-hd-trail-def*⟩)

have *vmtf-mark-to-rescore-also-reasons*:
⟨*isa-vmtf-mark-to-rescore-also-reasons M′ arena* (*outl*[*0* := − *lit-of* (*hd M*)]) *vm*
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* ∈ *isa-vmtf* (*all-atms-st S*) *M*})⟩
if
⟨*M* ≠ []⟩ **and**

⟨*literals-are-in-$\mathcal{L}_{in}$-trail* (*all-atms-st S*) *M*⟩ **and**
⟨− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)⟩ **and**
⟨*0* < *length outl*⟩ **and**
⟨*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), *D′*)⟩
proof −
have *outl-hd-tl*: ⟨*outl*[*0* := − *lit-of* (*hd M*)] = − *lit-of* (*hd M*) # *tl* (*outl*[*0* := − *lit-of* (*hd M*)])⟩ **and**
[*simp*]: ⟨*outl* ≠ []⟩
using *outl* **unfolding** *out-learned-def*
by (*cases outl*; *auto*; *fail*)+
have *uM-D*: ⟨− *lit-of* (*hd M*) ∈# *the D*⟩
by (*subst D-filter*) *auto*
have *mset-outl-D*: ⟨*mset* (*outl*[*0* := − *lit-of* (*hd M*)]) = (*the D*)⟩
by (*subst outl-hd-tl*, *subst mset.simps*, *subst tl-outl-D*, *subst D-filter*)
(*use uM-D D-filter*[*symmetric*] **in** *auto*)
from *arg-cong*[*OF this*, *of set-mset*] **have** *set-outl-D*: ⟨*set* (*outl*[*0* := − *lit-of* (*hd M*)]) = *set-mset* (*the D*)⟩
by *auto*

have *outl-Lall*: ⟨∀ *L*∈*set* (*outl*[*0* := − *lit-of* (*hd M*)]). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)⟩
using $\mathcal{L}_{in}$-*S* **unfolding** *set-outl-D*
by (*auto simp*: *S all-lits-of-m-add-mset*
all-atms-def literals-are-in-$\mathcal{L}_{in}$-def literals-are-in-$\mathcal{L}_{in}$-in-mset-$\mathcal{L}_{all}$
dest: *multi-member-split*)
have ⟨*distinct* (*outl*[*0* := − *lit-of* (*hd M*)])⟩ **using** *dist-D* **by**(*auto simp*: *S mset-outl-D*[*symmetric*])
then have *length-outl*: ⟨*length outl* ≤ *uint32-max*⟩
using *bounded tauto-confl* $\mathcal{L}_{in}$-*S simple-clss-size-upper-div2*[*OF bounded*, *of* ⟨*mset* (*outl*[*0* := − *lit-of* (*hd M*)])⟩]
by (*auto simp*: *out-learned-def S mset-outl-D*[*symmetric*] *uint32-max-def simp flip*: *all-atms-def*)
have *lit-annots*: ⟨∀ *L*∈*set* (*outl*[*0* := − *lit-of* (*hd M*)]).
∀ *C*. *Propagated* (− *L*) *C* ∈ *set M* ⟶
C ≠ *0* ⟶
C ∈# *dom-m N* ∧
(∀ *C*∈*set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))⟩
unfolding *set-outl-D*
apply (*intro ballI allI impI conjI*)
subgoal
using *list-invs S-T* **unfolding** *twl-list-invs-def*
by (*auto simp*: *S*)
subgoal for *L C i*
using *list-invs S-T arena lits-N literals-are-in-$\mathcal{L}_{in}$-mm-in-$\mathcal{L}_{all}$*[*of* ⟨(*all-atms-st S*)⟩ *N C* ⟨*i* − *C*⟩]
unfolding *twl-list-invs-def*
by (*auto simp*: *S arena-lifting all-atms-def*[*symmetric*])
done
obtain *vm0* **where**
vm-vm0: ⟨(*vm*, *vm0*) ∈ *Id* $\times_f$ *distinct-atoms-rel* (*all-atms-st S*)⟩ **and**
vm0: ⟨*vm0* ∈ *vmtf* (*all-atms-st S*) *M*⟩
using *vm* **by** (*cases vm*) (*auto simp*: *isa-vmtf-def S simp flip*: *all-atms-def*)
show *?thesis*
apply (*cases vm*)
apply (*rule order.trans*,
rule isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons[*of* ⟨*all-atms-st S*⟩,
THEN fref-to-Down-curry3,
of - - - *vm M arena* ⟨*outl*[*0* := − *lit-of* (*hd M*)]⟩ *vm0*])
subgoal using *bounded S* **by** (*auto simp*: *all-atms-def*)
subgoal using *vm arena M′-M vm-vm0* **by** (*auto simp*: *isa-vmtf-def*)[]

apply (*rule order.trans*, *rule ref-two-step'*)
apply (*rule vmtf-mark-to-rescore-also-reasons-spec*[*OF vm0 arena - outl-Lall lit-annots*])
subgoal using *length-outl* **by** *auto*
by (*auto simp*: *isa-vmtf-def conc-fun-RES S all-atms-def*)
qed

show *?thesis*
unfolding *extract-shorter-conflict-list-heur-st-def*
empty-conflict-and-extract-clause-def S S' prod.simps hd-M'-M
apply (*rewrite at* ‹*let - = list-update - - - in -* ›*Let-def*)
apply (*rewrite at* ‹*let - = empty-cach-ref - in -* › *Let-def*)
apply (*subst extract-shorter-conflict-wl-alt-def*)
apply (*refine-vcg isa-minimize-and-extract-highest-lookup-conflict*
empty-conflict-and-extract-clause-heur)
subgoal using *trail-nempty* **using** *M'-M* **by** (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)
subgoal using ‹*0 < length outl*› **.**
subgoal unfolding *hd-M'-M*[*symmetric*] **by** (*rule lookup-conflict-remove1-pre*)
apply (*rule vmtf-mark-to-rescore-also-reasons*; *assumption?*)
subgoal using *trail-nempty* **.**
subgoal using *pre2* **by** (*auto simp*: *S all-atms-def*)
subgoal using *uM-*$\mathcal{L}_{all}$ **by** (*auto simp*: *S all-atms-def*)
subgoal premises *p*
using *bounded p*(*5*,*7*−) **by** (*auto simp*: *S empty-cach-ref-pre-def cach-refinement-alt-def*
intro!: *IsaSAT-Lookup-Conflict.bounded-included-le simp*: *all-atms-def simp del*: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal using *bounded pre2*
by (*auto dest*!: *simple-clss-size-upper-div2 simp*: *uint32-max-def S all-atms-def*[*symmetric*]
simp del: *isasat-input-bounded-def*)
subgoal using *trail-nempty* **by** *fast*
subgoal using *uM-*$\mathcal{L}_{all}$ **.**
apply *assumption+*
subgoal
using *trail-nempty uM-*$\mathcal{L}_{all}$
unfolding *S*[*symmetric*] *S'*[*symmetric*]
by (*rule final*)
done
qed

have *find-decomp-wl-nlit*: ‹*find-decomp-wl-st-int n T*
≤ ⇓ {(*U*, *U''*). (*U*, *U''*) ∈ *twl-st-heur-bt* ∧ *equality-except-trail-wl U'' T'* ∧
(∃ *K M2*. (*Decided K* # (*get-trail-wl U''*), *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T'*))
∧
get-level (*get-trail-wl T'*) *K* = *get-maximum-level* (*get-trail-wl T'*) (*the* (*get-conflict-wl T'*) −
{#−*lit-of* (*hd* (*get-trail-wl T'*))#}) + *1* ∧
get-clauses-wl-heur U = *get-clauses-wl-heur S*) ∧
(*get-trail-wl U''*, *get-vmtf-heur U*) ∈ (*Id* $\times_f$ (*Id* $\times_f$ (*distinct-atoms-rel* (*all-atms-st T'*))$^{-1}$)) “
(*Collect* (*find-decomp-w-ns-prop* (*all-atms-st T'*) (*get-trail-wl T'*) *n* (*get-vmtf-heur T*)))}
(*find-decomp-wl LK' T'*)›
(**is** ‹*-* ≤ ⇓ *?find-decomp -*›)
if
‹(*S*, *S'*) ∈ *?R*› **and**
‹*backtrack-wl-inv S'*› **and**
‹*backtrack-wl-D-heur-inv S*› **and**
TT': ‹(*TnC*, *T'*) ∈ *?shorter S' S*› **and**
[*simp*]: ‹*nC* = (*n*, *C*)› **and**
[*simp*]: ‹*TnC* = (*T*, *nC*)› **and**

KK′: ‹$(LK, LK') \in \{(L, L').\ L = L' \wedge L = \textit{lit-of}\ (\textit{hd}\ (\textit{get-trail-wl}\ S'))\}$›
for *S S′ TnC T′ T nC n C LK LK′*
proof −
obtain *M N D NE UE NS US Q W* **where**
T′: ‹$T' = (M, N, D, NE, UE, NS, US, Q, W)$›
by (*cases T′*)
obtain *M′ W′ vm clvls cach lbd outl stats arena D′ Q′* **where**
T: ‹$T = (M', \textit{arena}, D', Q', W', \textit{vm}, \textit{clvls}, \textit{cach}, \textit{lbd}, \textit{outl}, \textit{stats})$›
using *TT′* **by** (*cases T*) (*auto simp*: *twl-st-heur-bt-def T′ del-conflict-wl-def*)
have
vm: ‹*vm* ∈ *isa-vmtf* (*all-atms-st T′*) *M*› **and**
M′M: ‹(*M′*, *M*) ∈ *trail-pol* (*all-atms-st T′*)› **and**
lits-trail: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st T′*) (*get-trail-wl T′*)›
using *TT′* **by** (*auto simp*: *twl-st-heur-bt-def del-conflict-wl-def*
all-atms-def[*symmetric*] *T T′*)

obtain *vm0* **where**
vm: ‹(*vm*, *vm0*) ∈ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st T′*)› **and**
vm0: ‹*vm0* ∈ *vmtf* (*all-atms-st T′*) *M*›
using *vm* **unfolding** *isa-vmtf-def* **by** (*cases vm*) *auto*

have [*simp*]:
‹*LK′* = *lit-of* (*hd* (*get-trail-wl T′*))›
‹*LK* = *LK′*›
using *KK′ TT′* **by** (*auto simp*: *equality-except-conflict-wl-get-trail-wl*)

have *n*: ‹*n* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*mset C*))› **and**
eq: ‹*equality-except-conflict-wl T′ S′*› **and**
‹*the D* = *mset C*› ‹*D* ≠ *None*› **and**
clss-eq: ‹*get-clauses-wl-heur S* = *arena*› **and**
n: ‹*n* < *count-decided* (*get-trail-wl T′*)› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st T′*)› **and**
T-T′: ‹(*T*, *del-conflict-wl T′*) ∈ *twl-st-heur-bt*› **and**
n2: ‹*n* = *get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*))›
using *TT′ KK′* **by** (*auto simp*: *T T′ twl-st-heur-bt-def del-conflict-wl-def simp flip*: *all-atms-def*
simp del: *isasat-input-bounded-def*)
have [*simp*]: ‹*get-trail-wl S′* = *M*›
using *eq* ‹*the D* = *mset C*› ‹*D* ≠ *None*› **by** (*cases S′*; *auto simp*: *T′*)
have [*simp*]: ‹*get-clauses-wl-heur S* = *arena*›
using *TT′* **by** (*auto simp*: *T T′*)

have *n-d*: ‹*no-dup M*›
using *M′M* **unfolding** *trail-pol-def* **by** *auto*

have [*simp*]: ‹*NO-MATCH* [] *M* ⟹ *out-learned M None ai* ⟷ *out-learned* [] *None ai*› **for** *M ai*
by (*auto simp*: *out-learned-def*)

show *?thesis*
unfolding *T′ find-decomp-wl-st-int-def prod.case T*
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*, *of M n vm0*
- - - ‹*all-atms-st T′*›])
subgoal using *n* **by** (*auto simp*: *T′*)
subgoal using *M′M vm* **by** *auto*

apply (*rule order.trans*)
apply (*rule ref-two-step′*)
apply (*rule find-decomp-wl-imp-le-find-decomp-wl′*)
subgoal using *vm0* **.**
subgoal using *lits-trail* **by** (*auto simp*: *T′*)
subgoal using *n* **by** (*auto simp*: *T′*)
subgoal using *n-d* **.**
subgoal using *bounded* **.**
unfolding *find-decomp-w-ns-def conc-fun-RES*
apply (*rule order.refl*)
using *T-T′ n-d*
apply (*cases* ‹*get-vmtf-heur T*›)
apply (*auto simp*: *find-decomp-wl-def twl-st-heur-bt-def T T′ del-conflict-wl-def*
dest: *no-dup-appendD*
simp flip: *all-atms-def n2*
intro!: *RETURN-RES-refine*
intro: *isa-vmtfI*)
apply (*rule-tac x=an* **in** *exI*)
apply (*auto dest*: *no-dup-appendD intro*: *isa-vmtfI simp*: *T′*)
apply (*auto simp*: *Image-iff T′*)
done
qed

have *fst-find-lit-of-max-level-wl*: ‹*RETURN* (*C* ! *1*)
≤ ⇓ *Id*
(*find-lit-of-max-level-wl U′ LK′*)›
if
‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-inv S′*› **and**
‹*backtrack-wl-D-heur-inv S*› **and**
TT′: ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› **and**
[*simp*]: ‹*nC* = (*n*, *C*)› **and**
[*simp*]: ‹*TnC* = (*T*, *nC*)› **and**
find-decomp: ‹(*U*, *U′*) ∈ *?find-decomp S T′ n*› **and**
size-C: ‹*1* < *length C*› **and**
size-conflict-U′: ‹*1* < *size* (*the* (*get-conflict-wl U′*))› **and**
KK′: ‹(*LK*, *LK′*) ∈ {(*L*, *L′*). *L* = *L′* ∧ *L* = *lit-of* (*hd* (*get-trail-wl S′*))}›
for *S S′ TnC T′ T nC n C U U′ LK LK′*
proof −
obtain *M N NE UE Q W NS US* **where**
T′: ‹*T′* = (*M*, *N*, *Some* (*mset C*), *NE*, *UE*, *NS*, *US*, *Q*, *W*)› **and**
‹*C* ≠ []›
using ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› ‹*1* < *length C*› *find-decomp*
apply (*cases U′*; *cases T′*; *cases S′*)
by (*auto simp*: *find-lit-of-max-level-wl-def*)

obtain *M′ K M2* **where**
U′: ‹*U′* = (*M′*, *N*, *Some* (*mset C*), *NE*, *UE*, *NS*, *US*, *Q*, *W*)› **and**
decomp: ‹(*Decided K* # *M′*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
lev-K: ‹*get-level M K* = *Suc* (*get-maximum-level M* (*remove1-mset* (− *lit-of* (*hd M*)) (*the* (*Some* (*mset C*)))))›
using ‹(*TnC*, *T′*) ∈ *?shorter S′ S*› ‹*1* < *length C*› *find-decomp*
by (*cases U′*; *cases S′*)
(*auto simp*: *find-lit-of-max-level-wl-def T′*)

have [*simp*]:

⟨$LK' = $ *lit-of* (*hd* (*get-trail-wl* T'))⟩
⟨$LK = LK'$⟩
using KK' TT' **by** (*auto simp*: *equality-except-conflict-wl-get-trail-wl*)

have *n-d*: ⟨*no-dup* (*get-trail-wl* S')⟩
using ⟨(S, S') ∈ *?R*⟩
by (*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-def*)

have [*simp*]: ⟨*get-trail-wl* S' = *get-trail-wl* T'⟩
using ⟨(*TnC*, T') ∈ *?shorter* S' S⟩ ⟨*1* < *length* C⟩ *find-decomp*
by (*cases* T'; *cases* S'; *auto simp*: *find-lit-of-max-level-wl-def* U'; *fail*)+
have [*simp*]: ⟨*remove1-mset* (− *lit-of* (*hd* M)) (*mset* C) = *mset* (*tl* C)⟩
apply (*subst mset-tl*)
using ⟨(*TnC*, T') ∈ *?shorter* S' S⟩
by (*auto simp*: *find-lit-of-max-level-wl-def* U' *highest-lit-def* T')

have *n-d*: ⟨*no-dup* M⟩
using ⟨(*TnC*, T') ∈ *?shorter* S' S⟩ *n-d* **unfolding** T'
by (*cases* S') *auto*

have *nempty*[*iff*]: ⟨*remove1-mset* (− *lit-of* (*hd* M)) (*the* (*Some*(*mset* C))) ≠ {#}⟩
using U' T' *find-decomp size-C* **by** (*cases* C) (*auto simp*: *remove1-mset-empty-iff*)
have *H*[*simp*]: ⟨*aa* ∈# *remove1-mset* (− *lit-of* (*hd* M)) (*the* (*Some*(*mset* C))) ⟹
get-level M' *aa* = *get-level* M *aa*⟩ **for** *aa*
apply (*rule get-all-ann-decomposition-get-level*[*of* ⟨*lit-of* (*hd* M)⟩ - K - *M2* ⟨*the* (*Some*(*mset* C))⟩])
subgoal ..
subgoal by (*rule n-d*)
subgoal by (*rule decomp*)
subgoal by (*rule lev-K*)
subgoal by *simp*
done

have ⟨*get-maximum-level* M (*remove1-mset* (− *lit-of* (*hd* M)) (*mset* C)) =
get-maximum-level M' (*remove1-mset* (− *lit-of* (*hd* M)) (*mset* C))⟩
by (*rule get-maximum-level-cong*) *auto*
then show *?thesis*
using ⟨(*TnC*, T') ∈ *?shorter* S' S⟩ ⟨*1* < *length* C⟩ *hd-conv-nth*[*OF* ⟨C ≠ []⟩, *symmetric*]
by (*auto simp*: *find-lit-of-max-level-wl-def* U' *highest-lit-def* T')
qed

have *propagate-bt-wl-D-heur*: ⟨*propagate-bt-wl-D-heur* *LK* C U
≤ ⇓ *?S* (*propagate-bt-wl* LK' L' U')⟩
if
SS': ⟨(S, S') ∈ *?R*⟩ **and**
⟨*backtrack-wl-inv* S'⟩ **and**
⟨*backtrack-wl-D-heur-inv* S⟩ **and**
⟨(*TnC*, T') ∈ *?shorter* S' S⟩ **and**
[*simp*]: ⟨*nC* = (n, C)⟩ **and**
[*simp*]: ⟨*TnC* = (T, *nC*)⟩ **and**
find-decomp: ⟨(U, U') ∈ *?find-decomp* S T' n⟩ **and**
le-C: ⟨*1* < *length* C⟩ **and**
⟨*1* < *size* (*the* (*get-conflict-wl* U'))⟩ **and**
C-L': ⟨(C ! *1*, L') ∈ *nat-lit-lit-rel*⟩ **and**
KK': ⟨(*LK*, LK') ∈ {(L, L'). $L = L' \wedge L$ = *lit-of* (*hd* (*get-trail-wl* S'))}⟩
for S S' *TnC* T' T *nC* n C U U' L' *LK* LK'
proof −

have
TT': ‹(*T*, *del-conflict-wl T'*) ∈ *twl-st-heur-bt*› **and**
n: ‹*n* = *get-maximum-level* (*get-trail-wl T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T'*))) (*mset C*))› **and**
T-C: ‹*get-conflict-wl T'* = *Some* (*mset C*)› **and**
T'S': ‹*equality-except-conflict-wl T' S'*› **and**
C-nempty: ‹*C* ≠ []› **and**
hd-C: ‹*hd C* = − *lit-of* (*hd* (*get-trail-wl T'*))› **and**
highest: ‹*highest-lit* (*get-trail-wl T'*) (*mset* (*tl C*))
(*Some* (*C* ! *Suc 0*, *get-level* (*get-trail-wl T'*) (*C* ! *Suc 0*)))› **and**
incl: ‹*mset C* ⊆# *the* (*get-conflict-wl S'*)› **and**
dist-S': ‹*distinct-mset* (*the* (*get-conflict-wl S'*))› **and**
list-confl-S': ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*the* (*get-conflict-wl S'*))› **and**
‹*get-conflict-wl S'* ≠ *None*› **and**
uM-$\mathcal{L}_{all}$: ‹−*lit-of* (*hd* (*get-trail-wl S'*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st S'*)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st T'*) *T'*› **and**
tr-nempty: ‹*get-trail-wl T'* ≠ []› **and**
r: ‹*length* (*get-clauses-wl-heur S*) = *r*› ‹*length* (*get-clauses-wl-heur T*) = *r*› **and**
corr: ‹*correct-watching S'*›
using ‹(*TnC*, *T'*) ∈ *?shorter S' S*› ‹*1* < *length C*› ‹(*S*, *S'*) ∈ *?R*›
by *auto*

obtain *K M2* **where**
UU': ‹(*U*, *U'*) ∈ *twl-st-heur-bt*› **and**
U'U': ‹*equality-except-trail-wl U' T'*› **and**
lev-K: ‹*get-level* (*get-trail-wl T'*) *K* = *Suc* (*get-maximum-level* (*get-trail-wl T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl T'*)))
(*the* (*get-conflict-wl T'*))))› **and**
decomp: ‹(*Decided K* # *get-trail-wl U'*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T'*))›
and
r': ‹*length* (*get-clauses-wl-heur U*) = *r*› **and**
S-arena: ‹*get-clauses-wl-heur U* = *get-clauses-wl-heur S*›
using *find-decomp r*
by *auto*

obtain *M N NE UE Q NS US W* **where**
T': ‹*T'* = (*M*, *N*, *Some* (*mset C*), *NE*, *UE*, *NS*, *US*, *Q*, *W*)› **and**
‹*C* ≠ []›
using *TT' T-C* ‹*1* < *length C*›
apply (*cases T'*; *cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def*)
obtain *D* **where**
S': ‹*S'* = (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)›
using *T'S'* ‹*1* < *length C*›
apply (*cases S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T' del-conflict-wl-def*)

obtain *M1* **where**
U': ‹*U'* = (*M1*, *N*, *Some* (*mset C*), *NE*, *UE*, *NS*, *US*, *Q*, *W*)› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*mset C*)› **and**
‹*mset C* ⊆# *the* (*get-conflict-wl S'*)› **and**
tauto: ‹¬ *tautology* (*mset C*)›
using ‹(*TnC*, *T'*) ∈ *?shorter S' S*› ‹*1* < *length C*› *find-decomp*
apply (*cases U'*)
by (*auto simp*: *find-lit-of-max-level-wl-def T' intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)

obtain *M1′ vm′ W′ clvls cach lbd outl stats heur avdom vdom lcount arena D′ Q′ opts*
where
U: ‹*U* = (*M1′*, *arena*, *D′*, *Q′*, *W′*, *vm′*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, [])›
using *UU′ find-decomp* **by** (*cases U*) (*auto simp*: *U′ T′ twl-st-heur-bt-def all-atms-def*[*symmetric*])

have [*simp*]:
‹*LK′* = *lit-of* (*hd M*)›
‹*LK* = *LK′*›
using *KK′ SS′* **by** (*auto simp*: *equality-except-conflict-wl-get-trail-wl S′*)
have
M1′-M1: ‹(*M1′*, *M1*) ∈ *trail-pol* (*all-atms-st U′*)› **and**
W′W: ‹(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st U′*))› **and**
vmtf: ‹*vm′* ∈ *isa-vmtf* (*all-atms-st U′*) *M1*› **and**
n-d-M1: ‹*no-dup M1*› **and**
empty-cach: ‹*cach-refinement-empty* (*all-atms-st U′*) *cach*› **and**
‹*length outl* = *Suc 0*› **and**
outl: ‹*out-learned M1 None outl*› **and**
vdom: ‹*vdom-m* (*all-atms-st U′*) *W N* ⊆ *set vdom*› **and**
lcount: ‹*lcount* = *size* (*learned-clss-l N*)› **and**
vdom-m: ‹*vdom-m* (*all-atms-st U′*) *W N* ⊆ *set vdom*› **and**
D′: ‹(*D′*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st U′*)› **and**
valid: ‹*valid-arena arena N* (*set vdom*)› **and**
avdom: ‹*mset avdom* ⊆# *mset vdom*› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st U′*)› **and**
nempty: ‹*isasat-input-nempty* (*all-atms-st U′*)› **and**
dist-vdom: ‹*distinct vdom*› **and**
heur: ‹*heuristic-rel* (*all-atms-st U′*) *heur*›
using *UU′* **by** (*auto simp*: *out-learned-def twl-st-heur-bt-def U U′ all-atms-def*[*symmetric*])
have [*simp*]: ‹*C* ! *1* = *L′*› ‹*C* ! *0* = − *lit-of* (*hd M*)› **and**
n-d: ‹*no-dup M*›
using *SS′ C-L′ hd-C* ‹*C* ≠ []› **by** (*auto simp*: *S′ U′ T′ twl-st-heur-conflict-ana-def hd-conv-nth*)
have *undef*: ‹*undefined-lit M1* (*lit-of* (*hd M*))›
using *decomp n-d*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend simp*: *T′ hd-append U′ neq-Nil-conv split*: *if-splits*)
have *C-1-neq-hd*: ‹*C* ! *Suc 0* ≠ − *lit-of* (*hd M*)›
using *distinct-mset-mono*[*OF incl dist-S′*] *C-L′* ‹*1* < *length C*› ‹*C* ! *0* = − *lit-of* (*hd M*)›
by (*cases C*; *cases* ‹*tl C*›) (*auto simp del*: ‹*C* ! *0* = − *lit-of* (*hd M*)›)
have *H*: ‹(*RES* ((λ*i*. (*fmupd i* (*C*, *False*) *N*, *i*)) ‘ {*i*. *0* < *i* ∧ *i* ∉# *dom-m N*}) ⤜
(λ(*N*, *i*). *ASSERT* (*i* ∈# *dom-m N*) ⤜ (λ-. *f N i*))) =
(*RES* ((λ*i*. (*fmupd i* (*C*, *False*) *N*, *i*)) ‘ {*i*. *0* < *i* ∧ *i* ∉# *dom-m N*}) ⤜
(λ(*N*, *i*). *f N i*))› (**is** ‹*?A* = *?B*›) **for** *f C N*
proof −
have ‹*?B* ≤ *?A*›
by (*force intro*: *ext complete-lattice-class.Sup-subset-mono simp*: *intro-spec-iff bind-RES*)
moreover have ‹*?A* ≤ *?B*›
by (*force intro*: *ext complete-lattice-class.Sup-subset-mono simp*: *intro-spec-iff bind-RES*)
ultimately show *?thesis* **by** *auto*
qed

have *propagate-bt-wl-D-heur-alt-def*:
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*,

vdom, avdom, lcount, opts). do {
ASSERT(length vdom ≤ length N0);
ASSERT(length avdom ≤ length N0);
ASSERT(nat-of-lit (C!1) < length W0 ∧ nat-of-lit (−L) < length W0);
ASSERT(length C > 1);
let L′ = C!1;
ASSERT (length C ≤ uint32-max div 2 + 1);
vm ← isa-vmtf-rescore C M vm0;
glue ← get-LBD lbd;
let - = C;
let b = False;
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
vdom, avdom, lcount, opts) ⟶ append-and-length-fast-code-pre ((b, C), N0));
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
vdom, avdom, lcount, opts) ⟶ lcount < sint64-max);
(N, i) ← fm-add-new b C N0;
ASSERT(update-lbd-pre ((i, glue), N));
let N = update-lbd i glue N;
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
vdom, avdom, lcount, opts) ⟶ length-ll W0 (nat-of-lit (−L)) < sint64-max);
let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [(i, L′, length C = 2)]];
ASSERT(isasat-fast (M, N0, D, Q, W0, vm0, y, cach, lbd, outl, stats, heur,
vdom, avdom, lcount, opts) ⟶ length-ll W (nat-of-lit L′) < sint64-max);
let W = W[nat-of-lit L′ := W!nat-of-lit L′ @ [(i, −L, length C = 2)]];
lbd ← lbd-empty lbd;
ASSERT(isa-length-trail-pre M);
let j = isa-length-trail M;
ASSERT(i ≠ DECISION-REASON);
ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
M ← cons-trail-Propagated-tr (− L) i M;
vm ← isa-vmtf-flush-int M vm;
heur ← mop-save-phase-heur (atm-of L′) (is-neg L′) heur;
RETURN (M, N, D, j, W, vm, 0,
cach, lbd, outl, add-lbd (of-nat glue) stats, update-heuristics glue heur, vdom @ [i],
avdom @ [i], Suc lcount, opts)
})›
unfolding *propagate-bt-wl-D-heur-def Let-def*
by *auto*
have *find-new-alt*: ‹*SPEC*
(λ(N′, i). N′ = fmupd i (D″, False) N ∧ 0 < i ∧
i ∉# dom-m N ∧
(∀L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE) + (NS + US)).
i ∉ fst ‘ set (W L))) = do {

i ← SPEC
(λi. 0 < i ∧
i ∉# dom-m N ∧
(∀L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE) + (NS + US)).
i ∉ fst ‘ set (W L)));
N′ ← RETURN (fmupd i (D″, False) N);
RETURN (N′, i)
}› **for** *D″*
by (*auto simp*: *RETURN-def RES-RES-RETURN-RES2*
RES-RES-RETURN-RES)
have *propagate-bt-wl-D-alt-def*:
‹*propagate-bt-wl LK′ L′ U′ = do {*

ASSERT (*propagate-bt-wl-pre LK′ L′* (*M1*, *N*, *Some* (*mset C*), *NE*, *UE*, *NS*, *US*, *Q*, *W*));
- ← *RETURN* (); ~~*phase-saving*~~
- ← *RETURN* (); ~~*LBD*~~
D″ ←
list-of-mset2 (− *LK′*) *L′*
(*the* (*Some* (*mset C*)));
(*N*, *i*) ← *SPEC*
(λ(*N′*, *i*). *N′* = *fmupd i* (*D″*, *False*) *N* ∧ *0* < *i* ∧
i ∉# *dom-m N* ∧
(∀ *L*∈#*all-lits-of-mm* (*mset* ‘# *ran-mf N* + (*NE* + *UE*) + (*NS* + *US*)).
i ∉ *fst* ‘ *set* (*W L*)));
- ← *RETURN* (); ~~*lbd-empty*~~
- ← *RETURN* (); ~~*lbd-empty*~~
M2 ← *cons-trail-propagate-l* (− *LK′*) *i M1*;
- ← *RETURN* (); ~~*vmtf-flush*~~
- ← *RETURN* (); ~~*heur*~~
RETURN
(*M2*,
N, *None*, *NE*, *UE*, *NS*, *US*, {#*LK′*#},
W(− *LK′* := *W* (− *LK′*) @ [(*i*, *L′*, *length D″* = *2*)],
L′ := *W L′* @ [(*i*, − *LK′*, *length D″* = *2*)]))
}⟩
unfolding *propagate-bt-wl-def Let-def find-new-alt nres-monad3*
U U′ H get-fresh-index-wl-def prod.case
propagate-bt-wl-def Let-def rescore-clause-def
by (*auto simp*: *U′ RES-RES2-RETURN-RES RES-RETURN-RES uminus-*$\mathcal{A}_{in}$*-iff*
uncurry-def RES-RES-RETURN-RES length-list-ge2 C-1-neq-hd
get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 list-of-mset2-def
cons-trail-propagate-l-def
intro!: *bind-cong*[*OF refl*]
simp flip: *all-lits-alt-def2 all-lits-alt-def all-lits-def*)

have [*refine0*]: ⟨*SPEC* (λ(*vm′*). *vm′* ∈ *vmtf* $\mathcal{A}$ *M1*)
≤ ⇓{((*vm′*), ()). *vm′* ∈ *vmtf* $\mathcal{A}$ *M1* } (*RETURN* ())⟩ **for** $\mathcal{A}$
by (*auto intro*!: *RES-refine simp*: *RETURN-def*)

obtain *vm0* **where**
vm: ⟨(*vm′*, *vm0*) ∈ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st U′*)⟩ **and**
vm0: ⟨*vm0* ∈ *vmtf* (*all-atms-st U′*) *M1*⟩
using *vmtf* **unfolding** *isa-vmtf-def* **by** (*cases vm′*) *auto*
have [*refine0*]:
⟨*isa-vmtf-rescore C M1′ vm′* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ {((*vm*), -).
vm ∈ *isa-vmtf* (*all-atms-st U′*) *M1*})⟩
apply (*rule order.trans*)
apply (*rule isa-vmtf-rescore*[*of* ⟨*all-atms-st U′*⟩, *THEN fref-to-Down-curry2*, *of* - - - *C M1 vm0*])
subgoal using *bounded* **by** *auto*
subgoal using *vm M1′-M1* **by** *auto*
apply (*rule order.trans*)
apply (*rule ref-two-step′*)
apply (*rule vmtf-rescore-score-clause*[*THEN fref-to-Down-curry2*, *of* ⟨*all-atms-st U′*⟩ *C M1 vm0*])
subgoal using *vm0 lits-confl* **by** (*auto simp*: *S′ U′*)
subgoal using *vm M1′-M1* **by** *auto*
subgoal by (*auto simp*: *rescore-clause-def conc-fun-RES intro*!: *isa-vmtfI*)
done

have [*refine0*]: ⟨*isa-vmtf-flush-int Ma vm* ≤

SPEC(λ*c*. (*c*, ()) $\in$ {(*vm'*, -). *vm'* $\in$ *isa-vmtf* (*all-atms-st U'*) *M2*})›
if *vm*: ‹*vm* $\in$ *isa-vmtf* (*all-atms-st U'*) *M1*› **and**
Ma: ‹(*Ma*, *M2*)
$\in$ {(*M0*, *M0''*).
(*M0*, *M0''*) $\in$ *trail-pol* (*all-atms-st U'*) $\wedge$
M0'' = *Propagated* (− *L*) *i* # *M1* $\wedge$
no-dup M0''}›
for *vm i L Ma M2*
proof −
let *?M1'* = ‹*cons-trail-Propagated-tr L i M1'*›
let *?M1* = ‹*Propagated* (−*L*) *i* # *M1*›

have *M1'-M1*: ‹(*Ma*, *M2*) $\in$ *trail-pol* (*all-atms-st U'*)›
using *Ma* **by** *auto*

have *vm*: ‹*vm* $\in$ *isa-vmtf* (*all-atms-st U'*) *?M1*›
using *vm* **by** (*auto simp*: *isa-vmtf-def dest*: *vmtf-consD*)
obtain *vm0* **where**
vm: ‹(*vm*, *vm0*) $\in$ *Id* $\times_r$ *distinct-atoms-rel* (*all-atms-st U'*)› **and**
vm0: ‹*vm0* $\in$ *vmtf* (*all-atms-st U'*) *?M1*›
using *vm* **unfolding** *isa-vmtf-def* **by** (*cases vm*) *auto*
show *?thesis*
apply (*rule order.trans*)
apply (*rule isa-vmtf-flush-int*[*THEN fref-to-Down-curry*, *of - - ?M1 vm*])
apply ((*solves* ‹*use M1'-M1 Ma in auto*›)+)[*2*]
apply (*subst Down-id-eq*)
apply (*rule order.trans*)
apply (*rule vmtf-change-to-remove-order'*[*THEN fref-to-Down-curry*, *of* ‹*all-atms-st U'*› *?M1 vm0 ?M1 vm*])
subgoal using *vm0 bounded nempty* **by** *auto*
subgoal using *vm* **by** *auto*
subgoal using *Ma* **by** (*auto simp*: *vmtf-flush-def conc-fun-RES RETURN-def intro*: *isa-vmtfI*)
done
qed

have [*refine0*]: ‹(*isa-length-trail M1'*, ()) $\in$ {(*j*, -). *j* = *length M1*}›
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*, *OF - M1'-M1*]) *auto*
have [*refine0*]: ‹*get-LBD lbd* $\leq \Downarrow$ {(-, -). *True*}(*RETURN* ())›
unfolding *get-LBD-def* **by** (*auto intro*!: *RES-refine simp*: *RETURN-def*)
have [*refine0*]: ‹*RETURN C*
$\leq \Downarrow$ *Id*
(*list-of-mset2* (− *LK'*) *L'*
(*the* (*Some* (*mset C*))))›
using *that*
by (*auto simp*: *list-of-mset2-def S'*)
have [*simp*]: ‹*0* < *header-size D''*› **for** *D''*
by (*auto simp*: *header-size-def*)
have [*simp*]: ‹*length arena* + *header-size D''* $\notin$ *set vdom*›
‹*length arena* + *header-size D''* $\notin$ *vdom-m* (*all-atms-st U'*) *W N*›
‹*length arena* + *header-size D''* $\notin$# *dom-m N*› **for** *D''*
using *valid-arena-in-vdom-le-arena*(*1*)[*OF valid*] *vdom*
by (*auto 5 1 simp*: *vdom-m-def*)
have *add-new-alt-def*: ‹(*SPEC*
(λ(*N'*, *i*).
N' = *fmupd i* (*D''*, *False*) *N* $\wedge$
0 < *i* $\wedge$

$i \notin\# \ dom\text{-}m\ N \wedge$
$(\forall L{\in}\#all\text{-}lits\text{-}of\text{-}mm\ (mset\ `\#\ ran\text{-}mf\ N + (NE + UE) + (NS + US)).$
$i \notin fst\ `\ set\ (W\ L)))) =$
$(SPEC$
$(\lambda(N', i).$
$N' = fmupd\ i\ (D'', False)\ N \wedge$
$0 < i \wedge$
$i \notin vdom\text{-}m\ (all\text{-}atms\text{-}st\ U')\ W\ N))\rangle$ **for** D''
using *lits*
by (*auto simp*: T' *vdom-m-def literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* U' *all-atms-def all-lits-def ac-simps*)
have [*refine0*]: $\langle fm\text{-}add\text{-}new\ False\ C\ arena$
$\leq \Downarrow \{((arena', i), (N', i')).\ valid\text{-}arena\ arena'\ N'\ (insert\ i\ (set\ vdom)) \wedge i = i' \wedge$
$i \notin\#\ dom\text{-}m\ N \wedge i \notin set\ vdom \wedge length\ arena' = length\ arena + header\text{-}size\ D'' + length\ D''\}$
$(SPEC$
$(\lambda(N', i).$
$N' = fmupd\ i\ (D'', False)\ N \wedge$
$0 < i \wedge$
$i \notin\#\ dom\text{-}m\ N \wedge$
$(\forall L{\in}\#all\text{-}lits\text{-}of\text{-}mm\ (mset\ `\#\ ran\text{-}mf\ N + (NE + UE) + (NS + US)).$
$i \notin fst\ `\ set\ (W\ L))))\rangle$
if $\langle(C, D'') \in Id\rangle$ **for** D''
apply (*subst add-new-alt-def*)
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *that valid le-C vdom*
by (*auto simp*: *intro*!: *RETURN-RES-refine valid-arena-append-clause*)
have [*refine0*]:
$\langle lbd\text{-}empty\ lbd \leq SPEC\ (\lambda c.\ (c, ())) \in \{(c, \text{-}).\ c = replicate\ (length\ lbd)\ False\})\rangle$
by (*auto simp*: *lbd-empty-def*)

have $\langle literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ (all\text{-}atms\text{-}st\ S')\ (mset\ C)\rangle$
using *incl list-confl-*S' *literals-are-in-*$\mathcal{L}_{in}$*-mono* **by** *blast*
then have *C-Suc1-in*: $\langle C\ !\ Suc\ 0 \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S')\rangle$
using $\langle 1 < length\ C\rangle$
by (*cases* C; *cases* $\langle tl\ C\rangle$) (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
then have $\langle nat\text{-}of\text{-}lit\ (C\ !\ Suc\ 0) < length\ W'\rangle$ $\langle nat\text{-}of\text{-}lit\ (-\ lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ S'))) < length\ W'\rangle$ **and**
W'*-eq*: $\langle W'\ !\ (nat\text{-}of\text{-}lit\ (C\ !\ Suc\ 0)) = W\ (C!\ Suc\ 0)\rangle$
$\langle W'\ !\ (nat\text{-}of\text{-}lit\ (-\ lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ S')))) = W\ (-\ lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ S')))\rangle$
using *uM-*$\mathcal{L}_{all}$ $W'W$ **unfolding** *map-fun-rel-def* **by** (*auto simp*: *image-image* S' U')
have *le-C-ge*: $\langle length\ C \leq uint32\text{-}max\ div\ 2 + 1\rangle$
using *clss-size-uint32-max*[*OF bounded*, *of* $\langle mset\ C\rangle$] $\langle literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\ (all\text{-}atms\text{-}st\ S')\ (mset\ C)\rangle$ *list-confl-*S'
dist-S' *incl size-mset-mono*[*OF incl*] *distinct-mset-mono*[*OF incl*]
simple-clss-size-upper-div2[*OF bounded - - tauto*]
by (*auto simp*: *uint32-max-def* S' U' *all-atms-def*[*symmetric*])
have *tr-SS'*: $\langle(get\text{-}trail\text{-}wl\text{-}heur\ S, M) \in trail\text{-}pol\ (all\text{-}atms\text{-}st\ S')\rangle$
using $\langle(S, S') \in ?R\rangle$ **unfolding** *twl-st-heur-conflict-ana-def*
by (*auto simp*: *all-atms-def* S')
have *All-atms-rew*: $\langle set\text{-}mset\ (all\text{-}atms\ (fmupd\ x'\ (C', b)\ N)\ (NE + UE + NS + US)) =$
$set\text{-}mset\ (all\text{-}atms\ N\ (NE + UE + NS + US))\rangle$ (**is** *?A*)
$\langle trail\text{-}pol\ (all\text{-}atms\ (fmupd\ x'\ (C', b)\ N)\ (NE + UE + NS + US)) =$
$trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE + NS + US))\rangle$ (**is** *?B*)
$\langle isa\text{-}vmtf\ (all\text{-}atms\ (fmupd\ x'\ (C', b)\ N)\ (NE + UE + NS + US)) =$

isa-vmtf (*all-atms N* (*NE* + *UE* + *NS* + *US*))› (**is** *?C*)
‹*option-lookup-clause-rel* (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*)) =
option-lookup-clause-rel (*all-atms N* (*NE* + *UE* + *NS* + *US*))› (**is** *?D*)
‹⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?E*)
‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*)))›
‹*phase-saving* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
phase-saving ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?F*)
‹*cach-refinement-empty* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
cach-refinement-empty ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?G*)
‹*cach-refinement-nonull* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
cach-refinement-nonull ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?G2*)
‹*vdom-m* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
vdom-m ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?H*)
‹*isasat-input-bounded* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
isasat-input-bounded ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?I*)
‹*isasat-input-nempty* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
isasat-input-nempty ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?J*)
‹*vdom-m* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *W* (*fmupd x'* (*C'*, *b*) *N*) =
insert x' (*vdom-m* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *W N*)› (**is** *?K*)
‹*heuristic-rel* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
heuristic-rel (*all-atms N* (*NE* + *UE* + *NS* + *US*))› (**is** *?L*)
if ‹*x'* ∉# *dom-m N*› **and** *C*: ‹*C'* = *C*› **for** *b x' C'*
proof −
show *A*: *?A*
using ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*mset C*)› *that*
by (*auto simp*: *all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*
U' S' in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literals-are-in-*$\mathcal{L}_{in}$*-def ac-simps*)
have *A2*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x'* (*C*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*)))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
then show ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) (*NE* + *UE* + *NS* + *US*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*)))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
have *A3*: ‹*set-mset* (*all-atms* (*fmupd x'* (*C*, *b*) *N*) (*NE* + *UE* + *NS* + *US*)) =
set-mset (*all-atms N* (*NE* + *UE* + *NS* + *US*))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)

show *?B* **and** *?C* **and** *?D* **and** *?E* **and** *?F* **and** *?G* **and** *?G2* **and** *?H* **and** *?I* **and** *?J* **and** *?L*
unfolding *trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def*
isa-vmtf-def vmtf-def distinct-atoms-rel-def vmtf-$\mathcal{L}_{all}$*-def atms-of-def*
distinct-hash-atoms-rel-def
atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
cach-refinement-def heuristic-rel-def
cach-refinement-list-def vdom-m-def
isasat-input-bounded-def
isasat-input-nempty-def cach-refinement-nonull-def
heuristic-rel-def phase-save-heur-rel-def
unfolding *trail-pol-def*[*symmetric*] *ann-lits-split-reasons-def*[*symmetric*]
isasat-input-bounded-def[*symmetric*]
vmtf-def[*symmetric*]
isa-vmtf-def[*symmetric*]
distinct-atoms-rel-def[*symmetric*]
vmtf-$\mathcal{L}_{all}$*-def*[*symmetric*] *atms-of-def*[*symmetric*]

distinct-hash-atoms-rel-def[*symmetric*]
atoms-hash-rel-def[*symmetric*]
option-lookup-clause-rel-def[*symmetric*]
lookup-clause-rel-def[*symmetric*]
phase-saving-def[*symmetric*] *cach-refinement-empty-def*[*symmetric*]
cach-refinement-def[*symmetric*] *cach-refinement-nonull-def*[*symmetric*]
cach-refinement-list-def[*symmetric*]
vdom-m-def[*symmetric*]
isasat-input-bounded-def[*symmetric*]
isasat-input-nempty-def[*symmetric*]
heuristic-rel-def[*symmetric*]
heuristic-rel-def[*symmetric*] *phase-save-heur-rel-def*[*symmetric*]
apply *auto*
done
show *?K*
using *that*
by (*auto simp*: *vdom-m-simps5 vdom-m-def*)
qed

have [*refine0*]: ‹*mop-save-phase-heur* (*atm-of* (*C* ! *1*)) (*is-neg* (*C* ! *1*)) *heur*
≤ *SPEC*
(λ*c*. (*c*, ())
∈ {(*c*, -). *heuristic-rel* (*all-atms-st U′*) *c*})›
using *heur uM-*$\mathcal{L}_{all}$ *lits-confl le-C*
literals-are-in-$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*of* ‹*all-atms-st S′*› ‹*mset C*› ‹*C*!*1*›]
unfolding *mop-save-phase-heur-def*
by (*auto intro*!: *ASSERT-leI save-phase-heur-preI simp*: *U′ S′*)

have *arena-le*: ‹*length arena* + *header-size C* + *length C* ≤ *6* + *r* + *uint32-max div 2*›
using *r r′ le-C-ge* **by** (*auto simp*: *uint32-max-def header-size-def S′ U*)
have *vm*: ‹*vm* ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) *M1* ⟹
vm ∈ *isa-vmtf* (*all-atms N* (*NE* + *UE*)) (*Propagated* (− *lit-of* (*hd M*)) *x2a* # *M1*)› **for** *x2a vm*
by (*cases vm*)
(*auto intro*!: *vmtf-consD simp*: *isa-vmtf-def*)
then show *?thesis*
supply [[*goals-limit=1*]]
using *empty-cach n-d-M1 C-L′ W′W outl vmtf undef* ‹*1* < *length C*› *lits*
uM-$\mathcal{L}_{all}$ *vdom lcount vdom-m dist-vdom heur*
apply (*subst propagate-bt-wl-D-alt-def*)
unfolding *U U′ H get-fresh-index-wl-def prod.case*
propagate-bt-wl-D-heur-alt-def rescore-clause-def
apply (*rewrite* **in** ‹*let* - = -!*1 in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *update-lbd* - - - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *False in* -› *Let-def*)
apply (*refine-rcg cons-trail-Propagated-tr2*[*of* - - - - - - ‹*all-atms-st U′*›])
subgoal using *valid* **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using *valid size-mset-mono*[*OF avdom*] **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using ‹*nat-of-lit* (*C* ! *Suc 0*) < *length W′*› **by** *simp*
subgoal using ‹*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S′*))) < *length W′*›
by (*simp add*: *S′ lit-of-hd-trail-def*)
subgoal using *le-C-ge* **.**
subgoal by (*auto simp*: *append-and-length-fast-code-pre-def isasat-fast-def*
sint64-max-def uint32-max-def)
subgoal

using D' *C-1-neq-hd vmtf avdom M1'-M1 size-learned-clss-dom-m*[*of N*] *valid-arena-size-dom-m-le-arena*[*OF valid*]
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def phase-saving-def atms-of-def* S' U' *lit-of-hd-trail-def all-atms-def*[*symmetric*] *isasat-fast-def sint64-max-def uint32-max-def*)
subgoal for *x uu x1 x2 vm uua- glue uub* D'' *xa* x'
by (*auto simp*: *update-lbd-pre-def arena-is-valid-clause-idx-def*)
subgoal using *length-watched-le*[*of* S' *S* ⟨−*lit-of-hd-trail M*⟩] *corr* SS' *uM*-$\mathcal{L}_{all}$ W'*-eq S-arena*
by (*auto simp*: *isasat-fast-def length-ll-def* S' *U lit-of-hd-trail-def simp flip*: *all-atms-def*)
subgoal using *length-watched-le*[*of* S' *S* ⟨*C* ! *Suc 0*⟩] *corr* SS' W'*-eq S-arena C-1-neq-hd C-Suc1-in*
by (*auto simp*: *length-ll-def* S' *U lit-of-hd-trail-def isasat-fast-def simp flip*: *all-atms-def*)
subgoal
using *M1'-M1* **by** (*rule isa-length-trail-pre*)
subgoal using D' *C-1-neq-hd vmtf avdom*
by (*auto simp*: *DECISION-REASON-def*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
subgoal
using *M1'-M1*
by (*rule cons-trail-Propagated-tr-pre*)
(*use undef uM*-$\mathcal{L}_{all}$ **in** ⟨*auto simp*: *lit-of-hd-trail-def* S' U' *all-atms-def*[*symmetric*]⟩)
subgoal using *M1'-M1* **by** (*auto simp*: *lit-of-hd-trail-def* S' U' *all-atms-def*[*symmetric*])
subgoal using *uM*-$\mathcal{L}_{all}$ **by** (*auto simp*: S' U' *uminus*-$\mathcal{A}_{in}$*-iff lit-of-hd-trail-def*)
subgoal
using D' *C-1-neq-hd vmtf avdom*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def*
intro!: *ASSERT-refine-left ASSERT-leI RES-refine exI*[*of - C*] *valid-arena-update-lbd*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
apply *assumption*
subgoal
supply *All-atms-rew*[*simp*]
unfolding *twl-st-heur-def*
using D' *C-1-neq-hd vmtf avdom M1'-M1 bounded nempty r arena-le*
by (*clarsimp simp add*: *propagate-bt-wl-D-heur-def twl-st-heur-def*
Let-def T' U' *U rescore-clause-def* S' *map-fun-rel-def*
list-of-mset2-def vmtf-flush-def RES-RES2-RETURN-RES RES-RETURN-RES uminus-$\mathcal{A}_{in}$*-iff*
get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 lit-of-hd-trail-def
RES-RES-RETURN-RES lbd-empty-def get-LBD-def DECISION-REASON-def
all-atms-def[*symmetric*] *All-atms-rew*
intro!: *valid-arena-update-lbd*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*
dest: *valid-arena-one-notin-vdomD*)
(*intro conjI*, *clarsimp-all*
intro!: *valid-arena-update-lbd*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*
dest: *valid-arena-one-notin-vdomD*, *auto simp*:
dest: *valid-arena-one-notin-vdomD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
done
qed

have *propagate-unit-bt-wl-D-int*: ⟨*propagate-unit-bt-wl-D-int LK U*
≤ ⇓ *?S*
(*propagate-unit-bt-wl* LK' U')⟩
if

SS': ‹$(S, S') \in$?R› **and**
‹*backtrack-wl-inv* S'› **and**
‹*backtrack-wl-D-heur-inv* S› **and**
‹$(TnC, T') \in$?*shorter* S' S› **and**
[*simp*]: ‹$nC = (n, C)$› **and**
[*simp*]: ‹$TnC = (T, nC)$› **and**
find-decomp: ‹$(U, U') \in$?*find-decomp* S T' n› **and**
‹$\neg$ *1* $<$ *length* C› **and**
‹$\neg$ *1* $<$ *size* (*the* (*get-conflict-wl* U'))› **and**
KK': ‹$(LK, LK') \in \{(L, L'). L = L' \wedge L =$ *lit-of* (*hd* (*get-trail-wl* S'))$\}$›
for S S' TnC T' T nC n C U U' LK LK'
proof −
have
TT': ‹$(T,$ *del-conflict-wl* $T') \in$ *twl-st-heur-bt*› **and**
n: ‹$n =$ *get-maximum-level* (*get-trail-wl* T')
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl* T'))) (*mset* C))› **and**
T-C: ‹*get-conflict-wl* T' = *Some* (*mset* C)› **and**
$T'S'$: ‹*equality-except-conflict-wl* T' S'› **and**
‹$C \neq []$› **and**
hd-C: ‹*hd* C = − *lit-of* (*hd* (*get-trail-wl* T'))› **and**
incl: ‹*mset* $C \subseteq\#$ *the* (*get-conflict-wl* S')› **and**
dist-S': ‹*distinct-mset* (*the* (*get-conflict-wl* S'))› **and**
list-confl-S': ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* S') (*the* (*get-conflict-wl* S'))› **and**
‹*get-conflict-wl* $S' \neq$ *None*› **and**
‹$C \neq []$› **and**
uL-M: ‹− *lit-of* (*hd* (*get-trail-wl* S')) $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S')› **and**
tr-nempty: ‹*get-trail-wl* $T' \neq []$›
using ‹$(TnC, T') \in$?*shorter* S' S› ‹~*1* $<$ *length* C›
by (*auto*)
obtain K *M2* **where**
UU': ‹$(U, U') \in$ *twl-st-heur-bt*› **and**
$U'U'$: ‹*equality-except-trail-wl* U' T'› **and**
lev-K: ‹*get-level* (*get-trail-wl* T') K = *Suc* (*get-maximum-level* (*get-trail-wl* T')
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl* T')))
(*the* (*get-conflict-wl* T'))))› **and**
decomp: ‹(*Decided* K # *get-trail-wl* U', *M2*) $\in$ *set* (*get-all-ann-decomposition* (*get-trail-wl* T'))›
and
r: ‹*length* (*get-clauses-wl-heur* S) = r›
using *find-decomp* SS'
by (*auto*)

obtain M N NE UE NS US Q W **where**
T': ‹$T' = (M, N,$ *Some* (*mset* C), $NE, UE, NS, US, Q, W)$›
using TT' *T-C* ‹$\neg$*1* $<$ *length* C›
apply (*cases* T'; *cases* S')
by (*auto simp*: *find-lit-of-max-level-wl-def*)
obtain D' **where**
S': ‹$S' = (M, N, D', NE, UE, NS, US, Q, W)$›
using $T'S'$
apply (*cases* S')
by (*auto simp*: *find-lit-of-max-level-wl-def* T' *del-conflict-wl-def*)

obtain *M1* **where**
U': ‹$U' = (M1, N,$ *Some* (*mset* C), $NE, UE, NS, US, Q, W)$›
using ‹$(TnC, T') \in$?*shorter* S' S› *find-decomp*
apply (*cases* U')

by (*auto simp*: *find-lit-of-max-level-wl-def T'*)
have [*simp*]:
⟨*LK' = lit-of* (*hd* (*get-trail-wl T'*))⟩
⟨*LK = LK'*⟩
using *KK' SS' S'* **by** (*auto simp*: *T'*)
obtain *vm' W' clvls cach lbd outl stats heur vdom avdom lcount arena D' Q' opts M1'*
where
U: ⟨*U* = (*M1'*, *arena*, *D'*, *Q'*, *W'*, *vm'*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, [])⟩ **and**
avdom: ⟨*mset avdom* ⊆# *mset vdom*⟩ **and**
r': ⟨*length* (*get-clauses-wl-heur U*) = *r*⟩
using *UU' find-decomp r* **by** (*cases U*) (*auto simp*: *U' T' twl-st-heur-bt-def*)
have
M'M: ⟨(*M1'*, *M1*) ∈ *trail-pol* (*all-atms-st U'*)⟩ **and**
W'W: ⟨(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st U'*))⟩ **and**
vmtf: ⟨*vm'* ∈ *isa-vmtf* (*all-atms-st U'*) *M1*⟩ **and**
n-d-M1: ⟨*no-dup M1*⟩ **and**
empty-cach: ⟨*cach-refinement-empty* (*all-atms-st U'*) *cach*⟩ **and**
⟨*length outl = Suc 0*⟩ **and**
outl: ⟨*out-learned M1 None outl*⟩ **and**
lcount: ⟨*lcount = size* (*learned-clss-l N*)⟩ **and**
vdom: ⟨*vdom-m* (*all-atms-st U'*) *W N* ⊆ *set vdom*⟩ **and**
valid: ⟨*valid-arena arena N* (*set vdom*)⟩ **and**
D': ⟨(*D'*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st U'*)⟩ **and**
bounded: ⟨*isasat-input-bounded* (*all-atms-st U'*)⟩ **and**
nempty: ⟨*isasat-input-nempty* (*all-atms-st U'*)⟩ **and**
dist-vdom: ⟨*distinct vdom*⟩ **and**
heur: ⟨*heuristic-rel* (*all-atms-st U'*) *heur*⟩
using *UU'* **by** (*auto simp*: *out-learned-def twl-st-heur-bt-def U U' all-atms-def*[*symmetric*])
have [*simp*]: ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩ **and**
n-d: ⟨*no-dup M*⟩
using *SS' hd-C* ⟨*C* ≠ []⟩ **by** (*auto simp*: *S' U' T' twl-st-heur-conflict-ana-def hd-conv-nth*)
have *undef*: ⟨*undefined-lit M1* (*lit-of* (*hd M*))⟩
using *decomp n-d*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend simp*: *T' hd-append U' neq-Nil-conv split*: *if-splits*)
have *C*: ⟨*C* = [− *lit-of* (*hd M*)]⟩
using ⟨*C* ≠ []⟩ ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩ ⟨¬*1* < *length C*⟩
by (*cases C*) (*auto simp del*: ⟨*C* ! *0* = − *lit-of* (*hd M*)⟩)
have *propagate-unit-bt-wl-alt-def*:
⟨*propagate-unit-bt-wl* = (λ*L* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*). *do* {
ASSERT(*L* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
ASSERT(*propagate-unit-bt-wl-pre L* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*));
- ← *RETURN* ();
- ← *RETURN* ();
- ← *RETURN* ();
- ← *RETURN* ();
M ← *cons-trail-propagate-l* (−*L*) *0 M*;
RETURN (*M*, *N*, *None*, *NE*, *add-mset* (*the D*) *UE*, *NS*, *US*, {#*L*#}, *W*)
})⟩
unfolding *propagate-unit-bt-wl-def Let-def* **by** (*auto intro*!: *ext bind-cong*[*OF refl*]
simp: *propagate-unit-bt-wl-pre-def propagate-unit-bt-l-pre-def single-of-mset-def RES-RETURN-RES image-iff*)

have [*refine0*]:

⟨lbd-empty lbd ≤ SPEC (λc. (c, ()) ∈ {(c, -). c = replicate (length lbd) False})⟩
by (auto simp: lbd-empty-def)
have [refine0]: ⟨(isa-length-trail M1′, ()) ∈ {(j, -). j = length M1}⟩
by (subst isa-length-trail-length-u[THEN fref-to-Down-unRET-Id, OF - M′M]) auto

have [refine0]: ⟨isa-vmtf-flush-int M1′ vm′ ≤
SPEC(λc. (c, ()) ∈ {(vm′, -). vm′ ∈ isa-vmtf (all-atms-st U′) M1})⟩
for vm i L
proof −
obtain vm0 **where**
vm: ⟨(vm′, vm0) ∈ Id $\times_r$ distinct-atoms-rel (all-atms-st U′)⟩ **and**
vm0: ⟨vm0 ∈ vmtf (all-atms-st U′) M1⟩
using vmtf **unfolding** isa-vmtf-def **by** (cases vm′) auto
show ?thesis
apply (rule order.trans)
apply (rule isa-vmtf-flush-int[THEN fref-to-Down-curry, of - - M1 vm′])
apply ((solves ⟨use M′M in auto⟩)+)[2]
apply (subst Down-id-eq)
apply (rule order.trans)
apply (rule vmtf-change-to-remove-order′[THEN fref-to-Down-curry, of ⟨all-atms-st U′⟩ M1 vm0 M1 vm′])
subgoal using vm0 bounded nempty **by** auto
subgoal using vm **by** auto
subgoal by (auto simp: vmtf-flush-def conc-fun-RES RETURN-def intro: isa-vmtfI)
done
qed
have [refine0]: ⟨get-LBD lbd ≤ SPEC(λc. (c, ()) ∈ UNIV)⟩
by (auto simp: get-LBD-def)

have tr-S: ⟨(get-trail-wl-heur S, M) ∈ trail-pol (all-atms-st S′)⟩
using SS′ **by** (auto simp: S′ twl-st-heur-conflict-ana-def all-atms-def)

have hd-SM: ⟨lit-of-last-trail-pol (get-trail-wl-heur S) = lit-of (hd M)⟩
unfolding lit-of-hd-trail-def lit-of-hd-trail-st-heur-def
by (subst lit-of-last-trail-pol-lit-of-last-trail[THEN fref-to-Down-unRET-Id])
(use M′M tr-S tr-nempty **in** ⟨auto simp: lit-of-hd-trail-def T′ S′⟩)
have uL-M: ⟨− lit-of (hd (get-trail-wl S′)) ∈# $\mathcal{L}_{all}$ (all-atms-st U′)⟩
using uL-M **by** (simp add: S′ U′)
let ?NE = ⟨add-mset {#− lit-of (hd M)#} (NE + UE + NS + US)⟩
have All-atms-rew: ⟨set-mset (all-atms (N) (?NE)) =
set-mset (all-atms N (NE + UE + NS + US))⟩ (**is** ?A)
⟨trail-pol (all-atms (N) (?NE)) =
trail-pol (all-atms N (NE + UE + NS + US))⟩ (**is** ?B)
⟨isa-vmtf (all-atms (N) (?NE)) =
isa-vmtf (all-atms N (NE + UE + NS + US))⟩ (**is** ?C)
⟨option-lookup-clause-rel (all-atms (N) (?NE)) =
option-lookup-clause-rel (all-atms N (NE + UE + NS + US))⟩ (**is** ?D)
⟨⟨Id⟩map-fun-rel (D_0 (all-atms (N) (?NE))) =
⟨Id⟩map-fun-rel (D_0 (all-atms N (NE + UE + NS + US)))⟩ (**is** ?E)
⟨set-mset ($\mathcal{L}_{all}$ (all-atms (N) (?NE))) =
set-mset ($\mathcal{L}_{all}$ (all-atms N (NE + UE + NS + US)))⟩
⟨phase-saving ((all-atms (N) (?NE))) =
phase-saving ((all-atms N (NE + UE + NS + US)))⟩ (**is** ?F)
⟨cach-refinement-empty ((all-atms (N) (?NE))) =
cach-refinement-empty ((all-atms N (NE + UE + NS + US)))⟩ (**is** ?G)
⟨vdom-m ((all-atms (N) (?NE))) =

vdom-m ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?H*)
‹*isasat-input-bounded* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-bounded ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?I*)
‹*isasat-input-nempty* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-nempty ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?J*)
‹*vdom-m* (*all-atms N ?NE*) *W* (*N*) =
(*vdom-m* (*all-atms N* (*NE* + *UE* + *NS* + *US*)) *W N*)› (**is** *?K*)
‹*heuristic-rel* ((*all-atms* (*N*) (*?NE*))) =
heuristic-rel ((*all-atms N* (*NE* + *UE* + *NS* + *US*)))› (**is** *?L*)
for b x' C'
proof −
show *A*: *?A*
using *uL-M*
apply (*cases* ‹*hd M*›)
by (*auto simp*: *all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset* U' S' *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literals-are-in-*$\mathcal{L}_{in}$*-def atm-of-eq-atm-of all-lits-of-m-add-mset ac-simps lits-of-def*)
have *A2*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms N* (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*)))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
then show ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*N*) (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*)))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
have *A3*: ‹*set-mset* (*all-atms N* (*?NE*)) =
set-mset (*all-atms N* (*NE* + *UE* + *NS* + *US*))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)

show *?B* **and** *?C* **and** *?D* **and** *?E* **and** *?F* **and** *?G* **and** *?H* **and** *?I* **and** *?J* **and** *?K* **and** *?L*
unfolding *trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def*
isa-vmtf-def vmtf-def distinct-atoms-rel-def vmtf-$\mathcal{L}_{all}$*-def atms-of-def*
distinct-hash-atoms-rel-def
atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
cach-refinement-def
cach-refinement-list-def vdom-m-def
isasat-input-bounded-def heuristic-rel-def
isasat-input-nempty-def cach-refinement-nonull-def vdom-m-def
phase-save-heur-rel-def phase-saving-def
unfolding *trail-pol-def*[*symmetric*] *ann-lits-split-reasons-def*[*symmetric*]
isasat-input-bounded-def[*symmetric*]
vmtf-def[*symmetric*]
isa-vmtf-def[*symmetric*]
distinct-atoms-rel-def[*symmetric*]
vmtf-$\mathcal{L}_{all}$*-def*[*symmetric*] *atms-of-def*[*symmetric*]
distinct-hash-atoms-rel-def[*symmetric*]
atoms-hash-rel-def[*symmetric*]
option-lookup-clause-rel-def[*symmetric*]
lookup-clause-rel-def[*symmetric*]
phase-saving-def[*symmetric*] *cach-refinement-empty-def*[*symmetric*]
cach-refinement-def[*symmetric*]
cach-refinement-list-def[*symmetric*]
vdom-m-def[*symmetric*]
isasat-input-bounded-def[*symmetric*] *cach-refinement-nonull-def*[*symmetric*]
isasat-input-nempty-def[*symmetric*] *heuristic-rel-def*[*symmetric*]
phase-save-heur-rel-def[*symmetric*] *phase-saving-def*[*symmetric*]
apply *auto*

done
qed

show *?thesis*
using *empty-cach n-d-M1 W′W outl vmtf C undef uL-M vdom lcount valid D′ avdom*
unfolding *U U′ propagate-unit-bt-wl-D-int-def prod.simps hd-SM propagate-unit-bt-wl-alt-def*
apply (*rewrite at* ‹*let - = incr-uset - in -*› *Let-def*)
apply (*refine-rcg cons-trail-Propagated-tr2*[**where** $\mathcal{A}$ = ‹*all-atms-st U′*›])
subgoal using *M′M* **by** (*rule isa-length-trail-pre*)
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal
using *M′M* **by** (*rule cons-trail-Propagated-tr-pre*)
(*use undef uL-M* **in** ‹*auto simp*: *hd-SM all-atms-def*[*symmetric*] *T′ lit-of-hd-trail-def S′*›)
subgoal
using *M′M* **by** (*auto simp*: *U U′ lit-of-hd-trail-st-heur-def RETURN-def single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def RES-RES2-RETURN-RES RES-RETURN-RES S′ uminus-*$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES DECISION-REASON-def hd-SM lit-of-hd-trail-st-heur-def*
intro!: *ASSERT-refine-left RES-refine exI*[*of -* ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
by (*auto simp*: *U U′ lit-of-hd-trail-st-heur-def RETURN-def single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def RES-RES2-RETURN-RES RES-RETURN-RES S′ uminus-*$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES DECISION-REASON-def hd-SM T′*
intro!: *ASSERT-refine-left RES-refine exI*[*of -* ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
using *bounded nempty dist-vdom r′ heur*
by (*auto simp*: *U U′ lit-of-hd-trail-st-heur-def RETURN-def single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def RES-RES2-RETURN-RES RES-RETURN-RES S′ uminus-*$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES DECISION-REASON-def hd-SM All-atms-rew all-atms-def*[*symmetric*]
intro!: *ASSERT-refine-left RES-refine exI*[*of -* ‹−*lit-of* (*hd M*)›]
intro!: *isa-vmtf-consD2*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
done
qed

have *trail-nempty*: ‹*fst* (*get-trail-wl-heur S*) ≠ []›
if
‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-inv S′*›
for *S S′*
proof −
show *?thesis*
using *that* **unfolding** *backtrack-wl-inv-def backtrack-wl-D-heur-inv-def backtrack-l-inv-def backtrack-inv-def backtrack-l-inv-def* **apply** −
by *normalize-goal*+
(*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-def ann-lits-split-reasons-def*)
qed

have [*refine*]: ‹$\bigwedge$*x y*. (*x*, *y*)
$\in$ {(*S*, *T*).
(*S*, *T*) $\in$ *twl-st-heur-conflict-ana* $\wedge$
length (*get-clauses-wl-heur S*) = *r*} $\Longrightarrow$
lit-of-hd-trail-st-heur x
$\leq$ $\Downarrow$ {(*L*, *L'*). *L* = *L'* $\wedge$ *L* = *lit-of* (*hd* (*get-trail-wl y*))} (*mop-lit-hd-trail-wl y*)›
unfolding *mop-lit-hd-trail-wl-def lit-of-hd-trail-st-heur-def*
apply *refine-rcg*
subgoal unfolding *mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def mop-lit-hd-trail-pre-def*
by (*auto simp*: *twl-st-heur-conflict-ana-def mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def trail-pol-alt-def*
mop-lit-hd-trail-pre-def state-wl-l-def twl-st-l-def lit-of-hd-trail-def RETURN-RES-refine-iff)
subgoal for *x y*
apply *simp-all*
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of* ‹*get-trail-wl y*› ‹*get-trail-wl-heur x*› ‹*all-atms-st y*›])
(*auto simp*: *twl-st-heur-conflict-ana-def mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def*
mop-lit-hd-trail-pre-def state-wl-l-def twl-st-l-def lit-of-hd-trail-def RETURN-RES-refine-iff)
done
have *backtrack-wl-alt-def*:
‹*backtrack-wl S* =
do {
ASSERT(*backtrack-wl-inv S*);
L ← *mop-lit-hd-trail-wl S*;
S ← *extract-shorter-conflict-wl S*;
S ← *find-decomp-wl L S*;

if size (*the* (*get-conflict-wl S*)) > *1*
then do {
L' ← *find-lit-of-max-level-wl S L*;
S ← *propagate-bt-wl L L' S*;
RETURN S
}
else do {
propagate-unit-bt-wl L S
}
}› **for** *S*
unfolding *backtrack-wl-def while.imonad2*
by *auto*

have *save-phase-st*: ‹(*xb*, *x'*) $\in$ *?S* $\Longrightarrow$
save-phase-st xb
$\leq$ *SPEC*
(λ*c*. (*c*, *x'*)
$\in$ {(*S*, *T*).
(*S*, *T*) $\in$ *twl-st-heur* $\wedge$
length (*get-clauses-wl-heur S*)
$\leq$ *6* + *r* + *uint32-max div 2*})› **for** *xb x'*
unfolding *save-phase-st-def*
apply (*refine-vcg save-phase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st x'*›])
subgoal
by (*rule isa-length-trail-pre*[*of* - ‹*get-trail-wl x'*› ‹*all-atms-st x'*›])
(*auto simp*: *twl-st-heur-def*)
subgoal
by (*auto simp*: *twl-st-heur-def*)

subgoal
by (*auto simp*: *twl-st-heur-def*)
done
show *?thesis*
supply [[*goals-limit=1*]]
apply (*intro frefI nres-relI*)
unfolding *backtrack-wl-D-nlit-heur-alt-def backtrack-wl-alt-def*
apply (*refine-rcg shorter*)
subgoal by (*rule inv*)
subgoal by (*rule trail-nempty*)
subgoal for *x y xa S x1 x2 x1a x2a*
by (*auto simp*: *twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl*)
apply (*rule find-decomp-wl-nlit*; *assumption*)
subgoal by (*auto simp*: *twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl equality-except-trail-wl-get-clauses-wl*)
subgoal for *x y L La xa S x1 x2 x1a x2a Sa Sb*
by (*auto simp*: *twl-st-heur-state-simp equality-except-trail-wl-get-conflict-wl*)
apply (*rule fst-find-lit-of-max-level-wl*; *solves assumption*)
apply (*rule propagate-bt-wl-D-heur*; *assumption*)
apply (*rule save-phase-st*; *assumption*)
apply (*rule propagate-unit-bt-wl-D-int*; *assumption*)
done
qed

14.2 Backtrack with direct extraction of literal if highest level

lemma *le-uint32-max-div-2-le-uint32-max*: ‹$a \leq$ *uint32-max div 2* + *1* $\Longrightarrow a \leq$ *uint32-max*›
by (*auto simp*: *uint32-max-def sint64-max-def*)

lemma *propagate-bt-wl-D-heur-alt-def*:
‹*propagate-bt-wl-D-heur* = (λ*L C* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*length vdom* $\leq$ *length N0*);
ASSERT(*length avdom* $\leq$ *length N0*);
ASSERT(*nat-of-lit* (*C*!*1*) < *length W0* $\wedge$ *nat-of-lit* ($-L$) < *length W0*);
ASSERT(*length C* > *1*);
let L′ = *C*!*1*;
ASSERT(*length C* $\leq$ *uint32-max div 2* + *1*);
vm $\leftarrow$ *isa-vmtf-rescore C M vm0*;
glue $\leftarrow$ *get-LBD lbd*;
let b = *False*;
let b′ = (*length C* = *2*);
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*) $\longrightarrow$ *append-and-length-fast-code-pre* ((*b*, *C*), *N0*));
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*) $\longrightarrow$ *lcount* < *sint64-max*);
(*N*, *i*) $\leftarrow$ *fm-add-new-fast b C N0*;
ASSERT(*update-lbd-pre* ((*i*, *glue*), *N*));
let N = *update-lbd i glue N*;
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*) $\longrightarrow$ *length-ll W0* (*nat-of-lit* ($-L$)) < *sint64-max*);
let W = *W0*[*nat-of-lit* ($-$ *L*) := *W0* ! *nat-of-lit* ($-$ *L*) @ [(*i*, *L′*, *b′*)]];
ASSERT(*isasat-fast* (*M*, *N0*, *D*, *Q*, *W0*, *vm0*, *y*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*) $\longrightarrow$ *length-ll W* (*nat-of-lit L′*) < *sint64-max*);

let W = W[nat-of-lit L′ := W!nat-of-lit L′ @ [(i, −L, b′)]];
lbd ← lbd-empty lbd;
ASSERT(isa-length-trail-pre M);
let j = isa-length-trail M;
ASSERT(i ≠ DECISION-REASON);
ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
M ← cons-trail-Propagated-tr (− L) i M;
vm ← isa-vmtf-flush-int M vm;
heur ← mop-save-phase-heur (atm-of L′) (is-neg L′) heur;
RETURN (M, N, D, j, W, vm, 0,
cach, lbd, outl, add-lbd (of-nat glue) stats, update-heuristics glue heur, vdom @ [i],
avdom @ [i],
lcount + 1, opts)
})›
unfolding *propagate-bt-wl-D-heur-def Let-def* **by** (*auto intro*!: *ext*)

lemma *propagate-bt-wl-D-fast-code-isasat-fastI2*: ‹*isasat-fast b* ⟹
b = (*a1′*, *a2′*) ⟹
a2′ = (*a1′a*, *a2′a*) ⟹
a < *length a1′a* ⟹ *a* ≤ *sint64-max*›
by (*cases b*) (*auto simp*: *isasat-fast-def*)

lemma *propagate-bt-wl-D-fast-code-isasat-fastI3*: ‹*isasat-fast b* ⟹
b = (*a1′*, *a2′*) ⟹
a2′ = (*a1′a*, *a2′a*) ⟹
a ≤ *length a1′a* ⟹ *a* < *sint64-max*›
by (*cases b*) (*auto simp*: *isasat-fast-def sint64-max-def uint32-max-def*)

lemma *lit-of-hd-trail-st-heur-alt-def*:
‹*lit-of-hd-trail-st-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, φ). *do* {*ASSERT* (*fst M* ≠ []); *RETURN* (*lit-of-last-trail-pol M*)})›
by (*auto simp*: *lit-of-hd-trail-st-heur-def lit-of-hd-trail-def intro*!: *ext*)

end
theory *IsaSAT-Show-LLVM*
imports
IsaSAT-Show
IsaSAT-Setup-LLVM
begin

sepref-register *isasat-current-information print-c print-uint64*

sepref-def *print-c-impl*
is ‹*RETURN o print-c*›
:: ‹$word\text{-}assn^k \rightarrow_a unit\text{-}assn$›
unfolding *print-c-def*
by *sepref*

sepref-def *print-uint64-impl*
is ‹*RETURN o print-uint64*›
:: ‹$word\text{-}assn^k \rightarrow_a unit\text{-}assn$›
unfolding *print-uint64-def*
by *sepref*

sepref-def *print-open-colour-impl*
is ‹*RETURN o print-open-colour*›
:: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-open-colour-def*
by *sepref*

sepref-def *print-close-colour-impl*
is ‹*RETURN o print-close-colour*›
:: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-close-colour-def*
by *sepref*

sepref-def *print-char-impl*
is ‹*RETURN o print-char*›
:: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-char-def*
by *sepref*

sepref-def *zero-some-stats-impl*
is ‹*RETURN o zero-some-stats*›
:: ‹*stats-assn*^{d} $\rightarrow_a$ *stats-assn*›
unfolding *zero-some-stats-def*
by *sepref*

sepref-def *isasat-current-information-impl* [*llvm-code*]
is ‹*uncurry2* (*RETURN ooo isasat-current-information*)›
:: ‹*word-assn*^{k} $*_a$ *stats-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *stats-assn*›
unfolding *isasat-current-information-def*
isasat-current-information-def
by *sepref*

declare *isasat-current-information-impl.refine*[*sepref-fr-rules*]

lemma *current-restart-phase-alt-def*:
‹*current-restart-phase* =
(λ(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ).
restart-phase)›
by (*auto intro*!: *ext*)

sepref-def *current-restart-phase-impl*
is ‹*RETURN o current-restart-phase*›
:: ‹*heuristic-assn*^{k} $\rightarrow_a$ *word-assn*›
unfolding *current-restart-phase-alt-def heuristic-assn-def*
by *sepref*

sepref-def *isasat-current-status-fast-code*
is ‹*isasat-current-status*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-bounded-assn-def isasat-current-status-def*
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *isasat-print-progress-impl*
is ‹*uncurry3* (*RETURN oooo isasat-print-progress*)›

:: ‹$word\text{-}assn^k *_a word\text{-}assn^k *_a stats\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k \rightarrow_a unit\text{-}assn$›
unfolding *isasat-print-progress-def*
by *sepref*

term *isasat-current-progress*

sepref-def *isasat-current-progress-impl*
is ‹*uncurry isasat-current-progress*›
:: ‹$word\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^k \rightarrow_a unit\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *isasat-bounded-assn-def isasat-current-progress-def*
unfolding *fold-tuple-optimizations*
by *sepref*

end
theory *IsaSAT-Rephase-LLVM*
imports *IsaSAT-Rephase IsaSAT-Show-LLVM*
begin

sepref-def *rephase-random-impl*
is ‹*uncurry rephase-random*›
:: ‹$word\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *rephase-random-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *rephase-init-impl*
is ‹*uncurry rephase-init*›
:: ‹$bool1\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
unfolding *rephase-init-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *copy-phase-impl*
is ‹*uncurry copy-phase*›
:: ‹$phase\text{-}saver\text{-}assn^k *_a phase\text{-}saver'\text{-}assn^d \rightarrow_a phase\text{-}saver'\text{-}assn$›
unfolding *copy-phase-alt-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *simp-thms*(*21*) — remove $a \wedge True$ from condition
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

definition *copy-phase2* **where**
‹*copy-phase2* = *copy-phase*›

sepref-def *copy-phase-impl2*
is ‹*uncurry copy-phase2*›
:: ‹$phase\text{-}saver'\text{-}assn^k *_a phase\text{-}saver\text{-}assn^d \rightarrow_a phase\text{-}saver\text{-}assn$›
unfolding *copy-phase-def copy-phase2-def*
while-eq-nfoldli[*symmetric*]

apply (*subst while-upt-while-direct*, *simp*)
unfolding *simp-thms*(*21*) — remove $a \wedge True$ from condition
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *rephase-init rephase-random copy-phase*

sepref-def *phase-save-phase-impl*
is ‹*uncurry phase-save-phase*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a phase\text{-}heur\text{-}assn^d \rightarrow_a phase\text{-}heur\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *phase-save-phase-def*
by *sepref*

sepref-def *save-phase-heur-impl*
is ‹*uncurry save-rephase-heur*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *save-rephase-heur-def heuristic-assn-def*
by *sepref*

sepref-def *save-phase-heur-st*
is *save-phase-st*
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *save-phase-st-def isasat-bounded-assn-def*
by *sepref*

sepref-def *phase-save-rephase-impl*
is ‹*uncurry phase-rephase*›
:: ‹$word\text{-}assn^k *_a phase\text{-}heur\text{-}assn^d \rightarrow_a phase\text{-}heur\text{-}assn$›
unfolding *phase-rephase-def copy-phase2-def*[*symmetric*]
by *sepref*

sepref-def *rephase-heur-impl*
is ‹*uncurry rephase-heur*›
:: ‹$word\text{-}assn^k *_a heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$›
unfolding *rephase-heur-def heuristic-assn-def*
by *sepref*

lemma *current-rephasing-phase-alt-def*:
‹*RETURN o current-rephasing-phase* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*,
(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*)).
RETURN curr-phase)›
unfolding *current-rephasing-phase-def*
phase-current-rephasing-phase-def
by (*auto intro*!: *ext*)

sepref-def *current-rephasing-phase*
is ‹*RETURN o current-rephasing-phase*›

:: ‹*heuristic-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *current-rephasing-phase-alt-def heuristic-assn-def*
by *sepref*

sepref-register *rephase-heur*
sepref-def *rephase-heur-st-impl*
is *rephase-heur-st*
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *rephase-heur-st-def isasat-bounded-assn-def*
by *sepref*

experiment
begin
export-llvm *rephase-heur-st-impl*
save-phase-heur-st
end

end
theory *IsaSAT-Backtrack-LLVM*
imports *IsaSAT-Backtrack IsaSAT-VMTF-LLVM IsaSAT-Lookup-Conflict-LLVM*
IsaSAT-Rephase-LLVM
begin

lemma *isa-empty-conflict-and-extract-clause-heur-alt-def*:
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(*D*, *C*, -) ← $WHILE_T$
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*get-level-pol-pre* (*M*, *C*!*i*));
ASSERT(*get-level-pol-pre* (*M*, *C*!*1*));
ASSERT(*1* < *length C*);
let L1 = *C*!*i*;
let L2 = *C*!*1*;
let C = (*if get-level-pol M L1* > *get-level-pol M L2 then swap C 1 i else C*);
ASSERT(*i*+*1* ≤ *uint32-max*);
RETURN (*D*, *C*, *i*+*1*)
})
(*D*, *C*, *1*);
ASSERT(*length outl* ≠ *1* ⟶ *length C* > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (*M*, *C*!*1*));
RETURN ((*True*, *D*), *C*, *if length outl* = *1 then 0 else get-level-pol M* (*C*!*1*))
}›
unfolding *isa-empty-conflict-and-extract-clause-heur-def*
by *auto*

sepref-def *empty-conflict-and-extract-clause-heur-fast-code*
is ‹*uncurry2* (*isa-empty-conflict-and-extract-clause-heur*)›
:: ‹[λ((*M*, *D*), *outl*). *outl* ≠ [] ∧ *length outl* ≤ *uint32-max*]$_a$
trail-pol-fast-assn^k $*_a$ *lookup-clause-rel-assn*^{d} $*_a$ *out-learned-assn*^{k} →

($conflict$-$option$-rel-$assn$) $\times_a$ $clause$-ll-$assn$ $\times_a$ $uint32$-nat-$assn$⟩
supply [[$goals$-$limit$=1]] $image$-$image$[$simp$]
supply [$simp$] = max-$snat$-def $uint32$-max-def
unfolding isa-$empty$-$conflict$-and-$extract$-$clause$-$heur$-alt-def
$larray$-$fold$-$custom$-$replicate$ $length$-$uint32$-nat-def $conflict$-$option$-rel-$assn$-def
apply ($rewrite$ at ⟨¤⟩ **in** ⟨- !1⟩ $snat$-$const$-$fold$[**where** $'a$=64])+
apply ($rewrite$ at ⟨¤⟩ **in** ⟨- !0⟩ $snat$-$const$-$fold$[**where** $'a$=64])
apply ($rewrite$ at ⟨$swap$ - ¤ -⟩ $snat$-$const$-$fold$[**where** $'a$=64])
apply ($rewrite$ at ⟨¤⟩ **in** ⟨(-, -, - + 1)⟩ $snat$-$const$-$fold$[**where** $'a$=64])
apply ($rewrite$ at ⟨¤⟩ **in** ⟨(-, -, 1)⟩ $snat$-$const$-$fold$[**where** $'a$=64])
apply ($rewrite$ at ⟨¤⟩ **in** ⟨If ($length$ - = ¤)⟩ $snat$-$const$-$fold$[**where** $'a$=64])
apply ($annot$-$unat$-$const$ $TYPE$(32))
unfolding gen-$swap$ $convert$-$swap$
by $sepref$

lemma $emptied$-$list$-alt-def: ⟨$emptied$-$list$ xs = $take$ 0 xs⟩
by ($auto$ $simp$: $emptied$-$list$-def)

sepref-def $empty$-$cach$-$code$
is ⟨$empty$-$cach$-ref-set⟩
:: ⟨$cach$-$refinement$-l-$assn^d$ $\rightarrow_a$ $cach$-$refinement$-l-$assn$⟩
supply [[$goals$-$limit$=1]]
unfolding $empty$-$cach$-ref-set-def $comp$-def $cach$-$refinement$-l-$assn$-def $emptied$-$list$-alt-def
apply ($annot$-$snat$-$const$ $TYPE$(64))
apply ($rewrite$ at ⟨-[¤ := $SEEN$-$UNKNOWN$]⟩ $value$-of-atm-def[$symmetric$])
apply ($rewrite$ at ⟨-[¤ := $SEEN$-$UNKNOWN$]⟩ $index$-of-atm-def[$symmetric$])
by $sepref$

theorem $empty$-$cach$-$code$-$empty$-$cach$-ref[$sepref$-fr-$rules$]:
⟨($empty$-$cach$-$code$, $RETURN$ ∘ $empty$-$cach$-ref)
∈ [$empty$-$cach$-ref-pre]$_a$
$cach$-$refinement$-l-$assn^d$ → $cach$-$refinement$-l-$assn$⟩
(**is** ⟨$?c$ ∈ [$?pre$]$_a$ $?im$ → $?f$⟩)
proof −
have H: ⟨$?c$
∈[$comp$-PRE Id
(λ($cach$, $supp$).
(∀ L∈set $supp$. L < $length$ $cach$) ∧
$length$ $supp$ ≤ Suc ($uint32$-max div 2) ∧
(∀ L<$length$ $cach$. $cach$! L ≠ $SEEN$-$UNKNOWN$ ⟶ L ∈ set $supp$))
(λx y. $True$)
(λx. $nofail$ (($RETURN$ ∘ $empty$-$cach$-ref) x))]$_a$
hrp-$comp$ ($cach$-$refinement$-l-$assn^d$)
Id → hr-$comp$ $cach$-$refinement$-l-$assn$ Id⟩
(**is** ⟨- ∈ [$?pre'$]$_a$ $?im'$ → $?f'$⟩)
using $hfref$-$compI$-PRE[OF $empty$-$cach$-$code.refine$[$unfolded$ PR-$CONST$-def $convert$-$fref$]
$empty$-$cach$-ref-set-$empty$-$cach$-ref[$unfolded$ $convert$-$fref$]] **by** $simp$
have pre: ⟨$?pre'$ h x⟩ **if** ⟨$?pre$ x⟩ **for** x h
using $that$ **by** ($auto$ $simp$: $comp$-PRE-def $trail$-pol-def
ann-$lits$-$split$-$reasons$-def $empty$-$cach$-ref-pre-def)
have im: ⟨$?im'$ = $?im$⟩
by $simp$
have f: ⟨$?f'$ = $?f$⟩

by *auto*
show *?thesis*
apply (*rule hfref-weaken-pre*[*OF*])
defer
using *H* **unfolding** *im f* **apply** *assumption*
using *pre* **..**
qed

sepref-register *fm-add-new-fast*

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* $\Longrightarrow$ *Suc* (*length* (*get-clauses-wl-heur S*)) < *max-snat 64*›
by (*cases S*) (*auto simp*: *isasat-fast-def max-snat-def sint64-max-def*)

sepref-register *update-heuristics*
sepref-def *update-heuristics-impl*
is [*llvm-inline*,*sepref-fr-rules*] ‹*uncurry* (*RETURN oo update-heuristics*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
unfolding *update-heuristics-def heuristic-assn-def*
by *sepref*

sepref-register *cons-trail-Propagated-tr*
sepref-def *propagate-unit-bt-wl-D-fast-code*
is ‹*uncurry propagate-unit-bt-wl-D-int*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit* = *1*]] *vmtf-flush-def*[*simp*] *image-image*[*simp*] *uminus-*$\mathcal{A}_{in}$*-iff*[*simp*]
unfolding *propagate-unit-bt-wl-D-int-def isasat-bounded-assn-def*
PR-CONST-def
unfolding *fold-tuple-optimizations*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *propagate-bt-wl-D-fast-codeXX*
is ‹*uncurry2 propagate-bt-wl-D-heur*›
:: ‹[λ((*L*, *C*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k $*_a$ *clause-ll-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›

supply [[*goals-limit* = *1*]] *append-ll-def*[*simp*] *isasat-fast-length-leD*[*dest*]
propagate-bt-wl-D-fast-code-isasat-fastI2[*intro*] *length-ll-def*[*simp*]
propagate-bt-wl-D-fast-code-isasat-fastI3[*intro*]
unfolding *propagate-bt-wl-D-heur-alt-def*
isasat-bounded-assn-def
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*] *append-ll-def*[*symmetric*]
PR-CONST-def save-phase-def
apply (*rewrite* **in** ‹*add-lbd* (*of-nat* ⌑) -› *annot-unat-unat-upcast*[**where** *'l*=*64*])
apply (*rewrite* **in** ‹(- + ⌑, -)› *unat-const-fold*[**where** *'a*=*64*])
apply (*rewrite at* ‹*RETURN* (-, -, -, -, -, -, ⌑, -)› *unat-const-fold*[**where** *'a*=*32*])
apply (*annot-snat-const TYPE*(*64*))
unfolding *fold-tuple-optimizations*
apply (*rewrite* **in** ‹*isasat-fast* ⌑› *fold-tuple-optimizations*[*symmetric*])+
by *sepref*

lemma *extract-shorter-conflict-list-heur-st-alt-def*:
‹*extract-shorter-conflict-list-heur-st* = (λ(*M*, *N*, (*bD*), *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *ccont*, *vdom*). *do* {

let D = *the-lookup-conflict bD*;
ASSERT(*fst M* $\neq$ []);
let K = *lit-of-last-trail-pol M*;
ASSERT(*0* < *length outl*);
ASSERT(*lookup-conflict-remove1-pre* ($-K$, *D*));
let D = *lookup-conflict-remove1* ($-K$) *D*;
let outl = *outl*[*0* := $-K$];
vm $\leftarrow$ *isa-vmtf-mark-to-rescore-also-reasons M N outl vm*;
(*D*, *cach*, *outl*) $\leftarrow$ *isa-minimize-and-extract-highest-lookup-conflict M N D cach lbd outl*;
ASSERT(*empty-cach-ref-pre cach*);
let cach = *empty-cach-ref cach*;
ASSERT(*outl* $\neq$ [] $\wedge$ *length outl* $\leq$ *uint32-max*);
(*D*, *C*, *n*) $\leftarrow$ *isa-empty-conflict-and-extract-clause-heur M D outl*;
RETURN ((*M*, *N*, *D*, Q', W', *vm*, *clvls*, *cach*, *lbd*, *take 1 outl*, *stats*, *ccont*, *vdom*), *n*, *C*)
})›
unfolding *extract-shorter-conflict-list-heur-st-def*
by (*auto simp*: *the-lookup-conflict-def Let-def intro*!: *ext*)

sepref-register *isa-minimize-and-extract-highest-lookup-conflict*
empty-conflict-and-extract-clause-heur

sepref-def *extract-shorter-conflict-list-heur-st-fast*
is ‹*extract-shorter-conflict-list-heur-st*›
:: ‹[λS. *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$
isasat-bounded-assn^d $\rightarrow$ *isasat-bounded-assn* $\times_a$ *uint32-nat-assn* $\times_a$ *clause-ll-assn*›
supply [[*goals-limit=1*]] *empty-conflict-and-extract-clause-pre-def*[*simp*]
unfolding *extract-shorter-conflict-list-heur-st-alt-def PR-CONST-def isasat-bounded-assn-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
fold-tuple-optimizations
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *find-lit-of-max-level-wl*
extract-shorter-conflict-list-heur-st lit-of-hd-trail-st-heur propagate-bt-wl-D-heur
propagate-unit-bt-wl-D-int
sepref-register *backtrack-wl*

sepref-def *lit-of-hd-trail-st-heur-fast-code*
is ‹*lit-of-hd-trail-st-heur*›
:: ‹[λS. *True*]$_a$ *isasat-bounded-assn*^{k} $\rightarrow$ *unat-lit-assn*›
unfolding *lit-of-hd-trail-st-heur-alt-def isasat-bounded-assn-def*
by *sepref*

sepref-register *save-phase-st*
sepref-def *backtrack-wl-D-fast-code*
is ‹*backtrack-wl-D-nlit-heur*›
:: ‹[*isasat-fast*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
size-conflict-wl-def[*simp*] *isasat-fast-length-leD*[*intro*] *isasat-fast-def*[*simp*]
unfolding *backtrack-wl-D-nlit-heur-def PR-CONST-def*
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*]
size-conflict-wl-def[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemmas [*llvm-inline*] = *add-lbd-def*

experiment
begin

export-llvm
empty-conflict-and-extract-clause-heur-fast-code
empty-cach-code
propagate-bt-wl-D-fast-codeXX
propagate-unit-bt-wl-D-fast-code
extract-shorter-conflict-list-heur-st-fast
lit-of-hd-trail-st-heur-fast-code
backtrack-wl-D-fast-code

end

end
theory *IsaSAT-Initialisation*
imports *Watched-Literals.Watched-Literals-Watch-List-Initialisation IsaSAT-Setup IsaSAT-VMTF Automatic-Refinement.Relators* — for more lemmas
begin

Chapter 15

Initialisation

lemma *bitXOR-1-if-mod-2-int*: ‹*bitOR L 1* = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *int*
  apply (*rule bin-rl-eqI*)
  unfolding *bin-rest-OR bin-last-OR*
   apply (*auto simp*: *bin-rest-def bin-last-def*)
  done

lemma *bitOR-1-if-mod-2-nat*:
  ‹*bitOR L 1* = (*if L mod 2* = *0 then L* + *1 else L*)›
  ‹*bitOR L* (*Suc 0*) = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *nat*
proof −
  have *H*: ‹*bitOR L 1* = *L* + (*if bin-last* (*int L*) *then 0 else 1*)›
    unfolding *bitOR-nat-def*
    apply (*auto simp*: *bitOR-nat-def bin-last-def*
        bitXOR-1-if-mod-2-int)
    done
  show ‹*bitOR L 1* = (*if L mod 2* = *0 then L* + *1 else L*)›
    unfolding *H*
    apply (*auto simp*: *bitOR-nat-def bin-last-def*)
    apply *presburger*+
    done
  then show ‹*bitOR L* (*Suc 0*) = (*if L mod 2* = *0 then L* + *1 else L*)›
    by *simp*
qed

15.1 Code for the initialisation of the Data Structure

The initialisation is done in three different steps:

1. First, we extract all the atoms that appear in the problem and initialise the state with empty values. This part is called *initialisation* below.
2. Then, we go over all clauses and insert them in our memory module. We call this phase *parsing*.
3. Finally, we calculate the watch list.

Splitting the second from the third step makes it easier to add preprocessing and more important to add a bounded mode.

15.1.1 Initialisation of the state

definition (**in** −) *atoms-hash-empty* **where**
[*simp*]: ‹*atoms-hash-empty* - = {}›

definition (**in** −) *atoms-hash-int-empty* **where**
‹*atoms-hash-int-empty* n = *RETURN* (*replicate* n *False*)›

lemma *atoms-hash-int-empty-atoms-hash-empty*:
‹(*atoms-hash-int-empty*, *RETURN o atoms-hash-empty*) ∈
[$\lambda n.$ ($\forall L \in \# \mathcal{L}_{all}$ $\mathcal{A}$. *atm-of* $L < n$)]$_f$ *nat-rel* → ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*use Max-less-iff* **in** ‹*auto simp*: *atoms-hash-rel-def atoms-hash-int-empty-def atoms-hash-empty-def*
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff Ball-def*
dest: *spec*[*of* - *Pos* -]›)

definition (**in** −) *distinct-atms-empty* **where**
‹*distinct-atms-empty* - = {}›

definition (**in** −) *distinct-atms-int-empty* **where**
‹*distinct-atms-int-empty* n = *RETURN* ([], *replicate* n *False*)›

lemma *distinct-atms-int-empty-distinct-atms-empty*:
‹(*distinct-atms-int-empty*, *RETURN o distinct-atms-empty*) ∈
[$\lambda n.$ ($\forall L \in \# \mathcal{L}_{all}$ $\mathcal{A}$. *atm-of* $L < n$)]$_f$ *nat-rel* → ⟨*distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*auto simp*: *distinct-atoms-rel-alt-def distinct-atms-empty-def distinct-atms-int-empty-def*)
by (*metis atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *atms-of-def imageE*)

type-synonym *vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat set*›

type-synonym *isa-vmtf-remove-int-option-fst-As* = ‹*vmtf-option-fst-As* × *nat list* × *bool list*›

definition *vmtf-init*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int-option-fst-As set*›
where
‹*vmtf-init* $\mathcal{A}_{in}$ M = {((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*).
$\mathcal{A}_{in}$ ≠ {#} ⟶ (*fst-As* ≠ *None* ∧ *lst-As* ≠ *None* ∧ ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*),
to-remove) ∈ *vmtf* $\mathcal{A}_{in}$ M)}›

definition *isa-vmtf-init* **where**
‹*isa-vmtf-init* $\mathcal{A}$ M =
((*Id* $\times_r$ *nat-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel* $\times_r$ ⟨*nat-rel*⟩*option-rel*) $\times_f$
distinct-atoms-rel $\mathcal{A}$)$^{-1}$
‘‘ *vmtf-init* $\mathcal{A}$ M›

lemma *isa-vmtf-initI*:
‹(*vm*, *to-remove*′) ∈ *vmtf-init* $\mathcal{A}$ M ⟹ (*to-remove*, *to-remove*′) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹
(*vm*, *to-remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ M›
by (*auto simp*: *isa-vmtf-init-def Image-iff intro*!: *bexI*[*of* - ‹(*vm*, *to-remove*′)›])

lemma *isa-vmtf-init-consD*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ M ⟹
((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *remove*) ∈ *isa-vmtf-init* $\mathcal{A}$ (L # M)›

by (*auto simp*: *isa-vmtf-init-def vmtf-init-def dest*: *vmtf-consD*)

lemma *vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *vmtf-init* $\mathcal{A}$ M $\Longrightarrow$ $L \in$ *vmtf-init* $\mathcal{B}$ M›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-init-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*

lemma *isa-vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ $L \in$ *isa-vmtf-init* $\mathcal{A}$ M $\Longrightarrow$ $L \in$ *isa-vmtf-init* $\mathcal{B}$ M›
using *vmtf-init-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
apply (*subst* (*asm*) *isa-vmtf-init-def*)
by (*cases L*) (*auto intro*!: *isa-vmtf-initI*)

type-synonym (**in** −) *twl-st-wl-heur-init* =
‹*trail-pol* × *arena* × *conflict-option-rel* × *nat* ×
(*nat* × *nat literal* × *bool*) *list list* × *isa-vmtf-remove-int-option-fst-As* × *bool list* ×
nat × *conflict-min-cach-l* × *lbd* × *vdom* × *bool*›

type-synonym (**in** −) *twl-st-wl-heur-init-full* =
‹*trail-pol* × *arena* × *conflict-option-rel* × *nat* ×
(*nat* × *nat literal* × *bool*) *list list* × *isa-vmtf-remove-int-option-fst-As* × *bool list* ×
nat × *conflict-min-cach-l* × *lbd* × *vdom* × *bool*›

The initialisation relation is stricter in the sense that it already includes the relation of atom inclusion.

Remark that we replace $D = None \longrightarrow j \leq length\ M$ by $j \leq length\ M$: this simplifies the proofs and does not make a difference in the generated code, since there are no conflict analysis at that level anyway.

KILL duplicates below, but difference: vmtf vs vmtf_init watch list vs no WL OC vs non-OC

definition *twl-st-heur-parsing-no-WL*
:: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init* × *nat twl-st-wl-init*) *set*›
where
‹*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* =
{((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*), ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena* *N'* *N* (*set vdom*) ∧
set-mset
(*all-lits-of-mm*
({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *UE* + *NS* + *US*)) ⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
mset vdom = *dom-m N*)) ∧
(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ∧
(*D'*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ∧
phase-saving $\mathcal{A}$ φ ∧
no-dup M ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧
(*W'*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
distinct vdom

}›

definition *twl-st-heur-parsing*
:: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init* × (*nat twl-st-wl* × *nat clauses*)) *set*›
where
‹*twl-st-heur-parsing* $\mathcal{A}$ *unbdd* =
{((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*), ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*), *OC*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
((*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$ ∧
valid-arena *N′* *N* (*set vdom*) ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ∧
phase-saving $\mathcal{A}$ *φ* ∧
no-dup M ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧
mset vdom = *dom-m N* ∧
vdom-m $\mathcal{A}$ *W N* = *set-mset* (*dom-m N*) ∧
set-mset
(*all-lits-of-mm*
({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *UE* + *NS* + *US*)) ⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
distinct vdom))
}›

definition *twl-st-heur-parsing-no-WL-wl* :: ‹*nat multiset* ⇒ *bool* ⇒ (- × *nat twl-st-wl-init′*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd* =
{((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*), (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*)).
(*unbdd* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena* *N′* *N* (*set vdom*) ∧ *set-mset* (*dom-m N*) ⊆ *set vdom*)) ∧
(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$ ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ∧
phase-saving $\mathcal{A}$ *φ* ∧
no-dup M ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *UE* + *NS* + *US*))
⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) ∧
(*W′*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
distinct vdom
}›

definition *twl-st-heur-parsing-no-WL-wl-no-watched* :: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init-full* × *nat twl-st-wl-init*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl-no-watched* $\mathcal{A}$ *unbdd* =
{((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*), ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)).

$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$(valid\text{-}arena\ N'\ N\ (set\ vdom) \wedge set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom)) \wedge (M', M) \in trail\text{-}pol\ \mathcal{A} \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in isa\text{-}vmtf\text{-}init\ \mathcal{A}\ M \wedge$
$phase\text{-}saving\ \mathcal{A}\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE + NS + US))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', empty\text{-}watched\ \mathcal{A}) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$distinct\ vdom$
$\}\rangle$

definition *twl-st-heur-post-parsing-wl* :: ⟨*bool* ⇒ (*twl-st-wl-heur-init-full* × *nat twl-st-wl*) *set*⟩ **where**
⟨*twl-st-heur-post-parsing-wl unbdd* =
$\{((M', N', D', j, W', vm, \varphi, clvls, cach, lbd, vdom, failed), (M, N, D, NE, UE, NS, US, Q, W)).$
$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$((M', M) \in trail\text{-}pol\ (all\text{-}atms\ N\ (NE + UE + NS + US)) \wedge$
$set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom \wedge$
$valid\text{-}arena\ N'\ N\ (set\ vdom))) \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ (all\text{-}atms\ N\ (NE + UE + NS + US)) \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in isa\text{-}vmtf\text{-}init\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ M \wedge$
$phase\text{-}saving\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ cach \wedge$
$vdom\text{-}m\ (all\text{-}atms\ N\ (NE + UE + NS + US))\ W\ N \subseteq set\ vdom \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + NE + UE + NS + US))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ (all\text{-}atms\ N\ (NE + UE + NS + US))) \wedge$
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ (all\text{-}atms\ N\ (NE + UE + NS + US))) \wedge$
$isasat\text{-}input\text{-}bounded\ (all\text{-}atms\ N\ (NE + UE + NS + US)) \wedge$
$distinct\ vdom$
$\}\rangle$

VMTF

definition *initialise-VMTF* :: ⟨*nat list* ⇒ *nat* ⇒ *isa-vmtf-remove-int-option-fst-As nres*⟩ **where**
⟨*initialise-VMTF N n* = *do* {
let A = *replicate n* (*VMTF-Node 0 None None*);
to-remove ← *distinct-atms-int-empty n*;
$ASSERT(length\ N \leq uint32\text{-}max)$;
$(n, A, cnext) \leftarrow WHILE_T$
$(\lambda(i, A, cnext).\ i < length\text{-}uint32\text{-}nat\ N)$
$(\lambda(i, A, cnext).\ do\ \{$
$ASSERT(i < length\text{-}uint32\text{-}nat\ N)$;
$let\ L = (N\ !\ i)$;
$ASSERT(L < length\ A)$;
$ASSERT(cnext \neq None \longrightarrow the\ cnext < length\ A)$;
$ASSERT(i + 1 \leq uint32\text{-}max)$;
$RETURN\ (i + 1, vmtf\text{-}cons\ A\ L\ cnext\ (i), Some\ L)$

})
    $(0, A, None)$;
  $RETURN\ ((A,\ n,\ cnext,\ (if\ N = []\ then\ None\ else\ Some\ ((N!0))),\ cnext),\ to\text{-}remove)$
}›

lemma *initialise-VMTF*:
  shows ‹$(uncurry\ initialise\text{-}VMTF,\ uncurry\ (\lambda N\ n.\ RES\ (vmtf\text{-}init\ N\ []))) \in$
    $[\lambda(N,n).\ (\forall L{\in}\#\ N.\ L < n) \wedge (distinct\text{-}mset\ N) \wedge size\ N < uint32\text{-}max \wedge set\text{-}mset\ N = set\text{-}mset\ \mathcal{A}]_f$
    $(\langle nat\text{-}rel\rangle list\text{-}rel\text{-}mset\text{-}rel) \times_f nat\text{-}rel \rightarrow$
    $\langle((\langle Id\rangle list\text{-}rel \times_r nat\text{-}rel \times_r \langle nat\text{-}rel\rangle\ option\text{-}rel \times_r \langle nat\text{-}rel\rangle\ option\text{-}rel \times_r \langle nat\text{-}rel\rangle\ option\text{-}rel)$
      $\times_r distinct\text{-}atoms\text{-}rel\ \mathcal{A}\rangle nres\text{-}rel$›
  (**is** ‹$(?init,\ ?R) \in$ -›)
proof −
  have *vmtf-ns-notin-empty*: ‹$vmtf\text{-}ns\text{-}notin\ []\ 0\ (replicate\ n\ (VMTF\text{-}Node\ 0\ None\ None))$› **for** *n*
    unfolding *vmtf-ns-notin-def*
    by *auto*

  have *K2*: ‹$distinct\ N \Longrightarrow fst\text{-}As < length\ N \Longrightarrow N!fst\text{-}As \in set\ (take\ fst\text{-}As\ N) \Longrightarrow False$›
    for *fst-As x N*
    by (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-take less-not-refl min-less-iff-conj*
      nth-eq-iff-index-eq nth-take)
  have *W-ref*: ‹$WHILE_T\ (\lambda(i,\ A,\ cnext).\ i < length\text{-}uint32\text{-}nat\ N')$
      $(\lambda(i,\ A,\ cnext).\ do\ \{$
          $\text{-} \leftarrow ASSERT\ (i < length\text{-}uint32\text{-}nat\ N')$;
          $let\ L = (N'\ !\ i)$;
          $\text{-} \leftarrow ASSERT\ (L < length\ A)$;
          $\text{-} \leftarrow ASSERT\ (cnext \neq None \longrightarrow the\ cnext < length\ A)$;
          $\text{-} \leftarrow ASSERT\ (i + 1 \leq uint32\text{-}max)$;
          $RETURN$
           $(i + 1,$
            $vmtf\text{-}cons\ A\ L\ cnext\ (i),\ Some\ L)$
        })
      $(0,\ replicate\ n'\ (VMTF\text{-}Node\ 0\ None\ None),$
       $None)$
    $\leq SPEC(\lambda(i,\ A',\ cnext).$
     $vmtf\text{-}ns\ (rev\ ((take\ i\ N')))\ i\ A'$
      $\wedge cnext = (option\text{-}last\ (take\ i\ N')) \wedge\ \ i = length\ N' \wedge$
        $length\ A' = n \wedge vmtf\text{-}ns\text{-}notin\ (rev\ ((take\ i\ N')))\ i\ A'$
)›
    (**is** ‹$\text{-} \leq SPEC\ ?P$›)
    if *H*: ‹$case\ y\ of\ (N,\ n) \Rightarrow (\forall L{\in}\#\ N.\ L < n) \wedge distinct\text{-}mset\ N \wedge size\ N < uint32\text{-}max \wedge$
       $set\text{-}mset\ N = set\text{-}mset\ \mathcal{A}$› **and**
      ref: ‹$(x,\ y) \in \langle Id\rangle list\text{-}rel\text{-}mset\text{-}rel \times_f nat\text{-}rel$› **and**
      st[*simp*]: ‹$x = (N',\ n')$› ‹$y = (N,\ n)$›
     for $N\ N'\ n\ n'\ A\ x\ y$
  proof −
  have [*simp*]: ‹$n = n'$› **and** *NN'*: ‹$(N',\ N) \in \langle Id\rangle list\text{-}rel\text{-}mset\text{-}rel$›
    using *ref* **unfolding** *st* **by** *auto*
  then have *dist*: ‹$distinct\ N'$›
    using *NN' H* **by** (*auto simp*: *list-rel-def br-def list-mset-rel-def list.rel-eq*
      list-all2-op-eq-map-right-iff' distinct-image-mset-inj list-rel-mset-rel-def)

  have *L-N*: ‹$\forall L{\in}set\ N'.\ L < n$›
    using *H ref* **by** (*auto simp*: *list-rel-def br-def list-mset-rel-def*

list-all2-op-eq-map-right-iff′ list-rel-mset-rel-def list.rel-eq)
let *?Q* = ‹λ(*i*, *A*′, *cnext*).
vmtf-ns (*rev* ((*take i N*′))) *i A*′ ∧ *i* ≤ *length N*′ ∧
cnext = (*option-last* (*take i N*′)) ∧
length A′ = *n* ∧ *vmtf-ns-notin* (*rev* ((*take i N*′))) *i A*′›
show *?thesis*
apply (*refine-vcg WHILET-rule*[**where** *R* = ‹*measure* (λ(*i*, -). *length N*′ + *1* − *i*)› **and** *I* = ‹*?Q*›])
subgoal by *auto*
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *S N*′ *x2 A*′
unfolding *assert-bind-spec-conv vmtf-ns-notin-def*
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2*
option-last-def hd-rev last-map intro!: *vmtf-cons dest*: *K2*)
subgoal by *auto*
subgoal
using *L-N dist*
by (*auto simp*: *take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2*
option-last-def hd-rev last-map)
subgoal
using *L-N dist*
by (*auto simp*: *last-take-nth-conv option-last-def*)
subgoal
using *H dist ref*
by (*auto simp*: *last-take-nth-conv option-last-def list-rel-mset-rel-imp-same-length*)
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth option-last-def hd-rev last-map intro*!: *vmtf-cons*
dest: *K2*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth*)
subgoal by *auto*
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth hd-rev last-map option-last-def*
intro!: *vmtf-notin-vmtf-cons dest*: *K2 split*: *if-splits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed
have [*simp*]: ‹*vmtf-*$\mathcal{L}_{all}$ *n*′ [] ((*set N*, {}), {})›
if ‹(*N*, *n*′) ∈ ⟨*Id*⟩*list-rel-mset-rel*› **for** *N N*′ *n*′
using *that* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto simp*: $\mathcal{L}_{all}$*-def atms-of-def image-image image-Un list-rel-def*
br-def list-mset-rel-def list-all2-op-eq-map-right-iff′
list-rel-mset-rel-def list.rel-eq)
have *in-N-in-N1*: ‹*L* ∈ *set N*′ ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ *N*)›
if ‹(*N*′, *N*) ∈ *list-mset-rel*› **for** *L N N*′ *y*
using *that* **by** (*auto simp*: $\mathcal{L}_{all}$*-def atms-of-def image-image image-Un list-rel-def*

list.rel-eq br-def list-mset-rel-def list-all2-op-eq-map-right-iff')

have *length-ba*: ‹$\forall L \in\# N.\ L <$ *length ba* $\Longrightarrow L \in$ *atms-of* $(\mathcal{L}_{all}\ N) \Longrightarrow$
$L <$ *length ba*›
if ‹$(N', y) \in \langle Id \rangle$*list-rel-mset-rel*›
for L *ba* N N' y
using *that*
by (*auto simp*: $\mathcal{L}_{all}$*-def nat-shiftr-div2 list.rel-eq*
atms-of-def image-image image-Un split: *if-splits*)
show *?thesis*
supply *list.rel-eq*[*simp*]
apply (*intro frefI nres-relI*)
unfolding *initialise-VMTF-def uncurry-def conc-Id id-def vmtf-init-def*
distinct-atms-int-empty-def nres-monad1
apply (*refine-rcg*)
subgoal by (*auto dest*: *list-rel-mset-rel-imp-same-length*)
apply (*rule specify-left*)
apply (*rule W-ref*; *assumption?*)
subgoal for N' $N'n'$ n' *Nn* N n *st*
apply (*case-tac st*)
apply *clarify*
apply (*subst RETURN-RES-refine-iff*)
apply (*auto dest*: *list-rel-mset-rel-imp-same-length*)
apply (*rule exI*[*of* - ‹{}›])
apply (*auto simp*: *distinct-atoms-rel-alt-def list-rel-mset-rel-def list-mset-rel-def*
br-def; *fail*)
apply (*rule exI*[*of* - ‹{}›])
unfolding *vmtf-def in-pair-collect-simp prod.case*
apply (*intro conjI impI*)
apply (*rule exI*[*of* - ‹(*rev* (*fst* N'))›])
apply (*rule-tac exI*[*of* - ‹[]›])
apply (*intro conjI impI*)
subgoal
by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-last-def last-map*
hd-rev list-rel-mset-rel-def br-def list-mset-rel-def)

subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map last-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-rev last-map distinct-atms-empty-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last list-rel-mset-rel-def)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last dest: *length-ba*)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last hd-map hd-conv-nth rev-nth last-conv-nth
list-rel-mset-rel-def br-def list-mset-rel-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal by (*auto simp*: *rev-map*[*symmetric*] *vmtf-def option-hd-rev*
map-option-option-last list-rel-mset-rel-def dest: *in-N-in-N1*)
subgoal by (*auto simp*: *distinct-atoms-rel-alt-def list-rel-mset-rel-def list-mset-rel-def*
br-def)
done

done
qed

15.1.2 Parsing

fun (**in** −)*get-conflict-wl-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *conflict-option-rel*⟩ **where**
⟨*get-conflict-wl-heur-init* (-, -, *D*, -) = *D*⟩

fun (**in** −)*get-clauses-wl-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *arena*⟩ **where**
⟨*get-clauses-wl-heur-init* (-, *N*, -) = *N*⟩

fun (**in** −) *get-trail-wl-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *trail-pol*⟩ **where**
⟨*get-trail-wl-heur-init* (*M*, -, -, -, -, -, -) = *M*⟩

fun (**in** −) *get-vdom-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *nat list*⟩ **where**
⟨*get-vdom-heur-init* (-, -, -, -, -, -, -, -, -, *vdom*, -) = *vdom*⟩

fun (**in** −) *is-failed-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *bool*⟩ **where**
⟨*is-failed-heur-init* (-, -, -, -, -, -, -, -, -, -, -, *failed*) = *failed*⟩

definition *propagate-unit-cls*
:: ⟨*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*⟩
where
⟨*propagate-unit-cls* = (*λL* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*).
((*Propagated L 0* # *M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *Q*), *OC*))⟩

definition *propagate-unit-cls-heur*
:: ⟨*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩
where
⟨*propagate-unit-cls-heur* = (*λL* (*M*, *N*, *D*, *Q*). *do* {
M ← *cons-trail-Propagated-tr L 0 M*;
RETURN (*M*, *N*, *D*, *Q*)})⟩

fun *get-unit-clauses-init-wl* :: ⟨*'v twl-st-wl-init* ⇒ *'v clauses*⟩ **where**
⟨*get-unit-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*) = *NE* + *UE*⟩

fun *get-subsumed-clauses-init-wl* :: ⟨*'v twl-st-wl-init* ⇒ *'v clauses*⟩ **where**
⟨*get-subsumed-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*) = *NS* + *US*⟩

fun *get-subsumed-init-clauses-init-wl* :: ⟨*'v twl-st-wl-init* ⇒ *'v clauses*⟩ **where**
⟨*get-subsumed-init-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*) = *NS*⟩

abbreviation *all-lits-st-init* :: ⟨*'v twl-st-wl-init* ⇒ *'v literal multiset*⟩ **where**
⟨*all-lits-st-init S* ≡ *all-lits* (*get-clauses-init-wl S*)
(*get-unit-clauses-init-wl S* + *get-subsumed-init-clauses-init-wl S*)⟩

definition *all-atms-init* :: ⟨- ⇒ - ⇒ *'v multiset*⟩ **where**
⟨*all-atms-init N NUE* = *atm-of* '# *all-lits N NUE*⟩

abbreviation *all-atms-st-init* :: ⟨*'v twl-st-wl-init* ⇒ *'v multiset*⟩ **where**
⟨*all-atms-st-init S* ≡ *atm-of* '# *all-lits-st-init S*⟩

lemma *DECISION-REASON0*[*simp*]: ⟨*DECISION-REASON* ≠ *0*⟩
by (*auto simp*: *DECISION-REASON-def*)

lemma *propagate-unit-cls-heur-propagate-unit-cls*:
‹(*uncurry propagate-unit-cls-heur*, *uncurry* (*propagate-unit-init-wl*)) ∈
[λ(*L*, *S*). *undefined-lit* (*get-trail-init-wl S*) *L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
unfolding *twl-st-heur-parsing-no-WL-def propagate-unit-cls-heur-def propagate-unit-init-wl-def nres-monad3*
apply (*intro frefI nres-relI*)
apply (*clarsimp simp add*: *propagate-unit-init-wl.simps cons-trail-Propagated-tr-def*[*symmetric*] *comp-def curry-def all-atms-def*[*symmetric*] *intro*!: *ASSERT-leI*)
apply (*refine-rcg cons-trail-Propagated-tr2*[**where** $\mathcal{A}$ = $\mathcal{A}$])
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *isa-vmtf-init-consD*
simp: *all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-*$\mathcal{A}_{in}$*-iff*)
done

definition *already-propagated-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*).
((*M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *Q*), *OC*))›

definition *already-propagated-unit-cls-heur*
:: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-heur* = (λ*L* (*M*, *N*, *D*, *Q*, *oth*).
RETURN (*M*, *N*, *D*, *Q*, *oth*))›

lemma *already-propagated-unit-cls-heur-already-propagated-unit-cls*:
‹(*uncurry already-propagated-unit-cls-heur*, *uncurry* (*RETURN oo already-propagated-unit-init-wl*)) ∈
[λ(*C*, *S*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C*]$_f$
list-mset-rel $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-heur-def already-propagated-unit-init-wl-def all-lits-of-mm-add-mset all-lits-of-m-add-mset literals-are-in-*$\mathcal{L}_{in}$*-def*)

definition (**in** −) *set-conflict-unit* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-unit L* - = *Some* {#*L*#}›

definition *set-conflict-unit-heur* **where**
‹*set-conflict-unit-heur* = (λ *L* (*b*, *n*, *xs*). *RETURN* (*False*, *1*, *xs*[*atm-of L* := *Some* (*is-pos L*)]))›

lemma *set-conflict-unit-heur-set-conflict-unit*:
‹(*uncurry set-conflict-unit-heur*, *uncurry* (*RETURN oo set-conflict-unit*)) ∈
[λ(*L*, *D*). *D* = *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def set-conflict-unit-heur-def set-conflict-unit-def option-lookup-clause-rel-def lookup-clause-rel-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff intro*!: *mset-as-position.intros*)

definition *conflict-propagated-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*conflict-propagated-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*).

((*M*, *N*, *set-conflict-unit* *L* *D*, *add-mset* {#*L*#} *NE*, *UE*, *NS*, *US*, {#}), *OC*))›

definition *conflict-propagated-unit-cls-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*conflict-propagated-unit-cls-heur* = (λ*L* (*M*, *N*, *D*, *Q*, *oth*). *do* {
ASSERT(*atm-of* *L* < *length* (*snd* (*snd* *D*)));
D ← *set-conflict-unit-heur* *L* *D*;
ASSERT(*isa-length-trail-pre* *M*);
RETURN (*M*, *N*, *D*, *isa-length-trail* *M*, *oth*)
})›

lemma *conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls*:
‹(*uncurry* *conflict-propagated-unit-cls-heur*, *uncurry* (*RETURN* *oo* *set-conflict-init-wl*)) ∈
[λ(*L*, *S*). $L \in\# \mathcal{L}_{all}\ \mathcal{A}$ ∧ *get-conflict-init-wl* *S* = *None*]$_f$
nat-lit-lit-rel $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
nres-rel›
proof −
have *set-conflict-init-wl-alt-def*:
‹*RETURN* *oo* *set-conflict-init-wl* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*). *do* {
D ← *RETURN* (*set-conflict-unit* *L* *D*);
RETURN ((*M*, *N*, *Some* {#*L*#}, *add-mset* {#*L*#} *NE*, *UE*, *NS*, *US*, {#}), *OC*)
})›
by (*auto* *intro*!: *ext* *simp*: *set-conflict-init-wl-def*)
have [*refine0*]: ‹*D* = *None* ∧ $L \in\# \mathcal{L}_{all}\ \mathcal{A}$ ⟹ (*y*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ *L* = *L*′
⟹
set-conflict-unit-heur *L*′ *y* ≤ ⇓ {(*D*, *D*′). (*D*, *D*′) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧ *D*′ = *Some* {#*L*#}}
(*RETURN* (*set-conflict-unit* *L* *D*))›
for *L* *D* *y* *L*′
apply (*rule* *order-trans*)
apply (*rule* *set-conflict-unit-heur-set-conflict-unit*[*THEN* *fref-to-Down-curry*,
unfolded *comp-def*, *of* $\mathcal{A}$ *L* *D* *L*′ *y*])
subgoal
by *auto*
subgoal
by *auto*
subgoal
unfolding *conc-fun-RETURN*
by (*auto* *simp*: *set-conflict-unit-def*)
done

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2* *del*]
unfolding *set-conflict-init-wl-alt-def* *conflict-propagated-unit-cls-heur-def* *uncurry-def*
apply (*intro* *frefI* *nres-relI*)
apply (*refine-rcg*)
subgoal
by (*auto* *simp*: *twl-st-heur-parsing-no-WL-def* *option-lookup-clause-rel-def*
lookup-clause-rel-def *atms-of-def*)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto* *simp*: *twl-st-heur-parsing-no-WL-def* *conflict-propagated-unit-cls-heur-def* *conflict-propagated-unit-cls-def*
image-image *set-conflict-unit-def*

intro!: *set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*])
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
intro!: *isa-length-trail-pre*)
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
image-image set-conflict-unit-def all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$*-iff*
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*]
intro!: *set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*]
isa-length-trail-pre)
done
qed

definition *add-init-cls-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur unbdd* = (λ*C* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*). *do* {
let C = *C*;
ASSERT(*length C* ≤ *uint32-max* + *2*);
ASSERT(*length C* ≥ *2*);
if unbdd ∨ (*length N* ≤ *sint64-max* − *length C* − *5* ∧ ¬*failed*)
then do {
ASSERT(*length vdom* ≤ *length N*);
(*N*, *i*) ← *fm-add-new True C N*;
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom* @ [*i*], *failed*)
} *else RETURN* (*M*, *N*, *D*, *Q*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *True*)})›

definition *add-init-cls-heur-unb* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-unb* = *add-init-cls-heur True*›

definition *add-init-cls-heur-b* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b* = *add-init-cls-heur False*›

definition *add-init-cls-heur-b'* :: ‹*nat literal list list* ⇒ *nat* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b' C i* = *add-init-cls-heur False* (*C*!*i*)›

lemma *length-C-nempty-iff*: ‹*length C* ≥ *2* ⟷ *C* ≠ [] ∧ *tl C* ≠ []›
by (*cases C*; *cases* ‹*tl C*›) *auto*

context
fixes *unbdd* :: *bool* **and** $\mathcal{A}$ :: ‹*nat multiset*› **and**
CT :: ‹*nat clause-l* × *twl-st-wl-heur-init*› **and**
CSOC :: ‹*nat clause-l* × *nat twl-st-wl-init*› **and**
SOC :: ‹*nat twl-st-wl-init*› **and**
C C' :: ‹*nat clause-l*› **and**
S :: ‹*nat twl-st-wl-init'*› **and** *x1a* **and** *N* :: ‹*nat clauses-l*› **and**
D :: ‹*nat cconflict*› **and** *x2b* **and** *NE UE NS US* :: ‹*nat clauses*› **and**
M :: ‹(*nat*,*nat*) *ann-lits*› **and**
a b c d e f m p q r s t u v w x y **and**
Q **and**
x2e :: ‹*nat lit-queue-wl*› **and** *OC* :: ‹*nat clauses*› **and**

T :: *twl-st-wl-heur-init* **and**
M' :: ‹*trail-pol*› **and** N' :: *arena* **and**
D' :: *conflict-option-rel* **and**
j' :: *nat* **and**
W' :: ‹-› **and**
vm :: ‹*isa-vmtf-remove-int-option-fst-As*› **and**
clvls :: *nat* **and**
cach :: *conflict-min-cach-l* **and**
lbd :: *lbd* **and**
vdom :: *vdom* **and**
failed :: *bool* **and**
φ :: *phase-saver*
assumes
pre: ‹*case CSOC of*
$(C, S) \Rightarrow 2 \leq$ *length* $C \wedge$ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* C) $\wedge$ *distinct* C› **and**
xy: ‹$(CT, CSOC) \in Id \times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*› **and**
st:
‹$CSOC = (C, SOC)$›
‹$SOC = (S, OC)$›
‹$S = (M, a)$›
‹$a = (N, b)$›
‹$b = (D, c)$›
‹$c = (NE, d)$›
‹$d = (UE, e)$›
‹$e = (NS, f)$›
‹$f = (US, Q)$›
‹$CT = (C', T)$›
‹$T = (M', m)$›
‹$m = (N', p)$›
‹$p = (D', q)$›
‹$q = (j', r)$›
‹$r = (W', s)$›
‹$s = (vm, t)$›
‹$t = (\varphi, u)$›
‹$u = (clvls, v)$›
‹$v = (cach, w)$›
‹$w = (lbd, x)$›
‹$x = (vdom, failed)$›
begin

lemma *add-init-pre1*: ‹*length* $C' \leq$ *uint32-max* + *2*›
using *pre clss-size-uint32-max*[*of* $\mathcal{A}$ ‹*mset* C›] *xy st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)

lemma *add-init-pre2*: ‹*2* $\leq$ *length* C'›
using *pre xy st* **by** (*auto simp*:)

private lemma
x1g-x1: ‹$C' = C$› **and**
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and**
valid: ‹*valid-arena* N' N (*set vdom*)› **and**
‹$(D', D) \in$ *option-lookup-clause-rel* $\mathcal{A}$› **and**
‹$j' \leq$ *length* M› **and**
Q: ‹$Q = \{\#-$ *lit-of* $x.$ $x \in\#$ *mset* (*drop* j' (*rev* M))$\#\}$› **and**
‹$vm \in$ *isa-vmtf-init* $\mathcal{A}$ M› **and**
‹*phase-saving* $\mathcal{A}$ φ› **and**

⟨*no-dup M*⟩ **and**
⟨*cach-refinement-empty* $\mathcal{A}$ *cach*⟩ **and**
vdom: ⟨*mset vdom* = *dom-m N*⟩ **and**
var-incl:
⟨*set-mset* (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + *NE* + *NS* + *UE* + *US*))
⊆ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)⟩ **and**
watched: ⟨(W', *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩ **and**
bounded: ⟨*isasat-input-bounded* $\mathcal{A}$⟩
if ⟨¬*failed* ∨ *unbdd*⟩
using *that xy* **unfolding** *st twl-st-heur-parsing-no-WL-def*
by (*auto simp*: *ac-simps*)

lemma *init-fm-add-new*:
⟨¬*failed* ∨ *unbdd* ⟹ *fm-add-new True* C' N'
≤ ⇓ {((*arena*, *i*), (N'', i')). *valid-arena arena* N'' (*insert i* (*set vdom*)) ∧ *i* = i' ∧
i ∉# *dom-m N* ∧ *i* = *length* N' + *header-size C* ∧
i ∉ *set vdom*}
(*SPEC*
(λ(N', *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ N' = *fmupd ia* (*C*, *True*) *N*))⟩
(**is** ⟨- ⟹ - ≤ ⇓ *?qq* -⟩)
unfolding *x1g-x1*
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *valid vdom pre xy valid-arena-in-vdom-le-arena*[*OF valid*] *arena-lifting*(*2*)[*OF valid*]
valid **unfolding** *st*
by (*fastforce simp*: *x1g-x1 vdom-m-def*
intro!: *RETURN-RES-refine valid-arena-append-clause*)

lemma *add-init-cls-final-rel*:
fixes *nN'j'* :: ⟨*arena-el list* × *nat*⟩ **and**
nNj :: ⟨(*nat*, *nat literal list* × *bool*) *fmap* × *nat*⟩ **and**
nN :: ⟨-⟩ **and**
k :: ⟨*nat*⟩ **and** nN' :: ⟨*arena-el list*⟩ **and**
k' :: ⟨*nat*⟩
assumes
⟨(*nN'j'*, *nNj*) ∈ {((*arena*, *i*), (N'', i')). *valid-arena arena* N'' (*insert i* (*set vdom*)) ∧ *i* = i' ∧
i ∉# *dom-m N* ∧ *i* = *length* N' + *header-size C* ∧
i ∉ *set vdom*}⟩ **and**
⟨*nNj* ∈ *Collect* (λ(N', *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ N' = *fmupd ia* (*C*, *True*) *N*)⟩
⟨*nN'j'* = (nN', k')⟩ **and**
⟨*nNj* = (*nN*, *k*)⟩
shows ⟨((M', nN', D', j', W', *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom* @ [k'], *failed*),
(*M*, *nN*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)
∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
proof −
show *?thesis*
using *assms xy pre* **unfolding** *st*
apply (*auto simp*: *twl-st-heur-parsing-no-WL-def ac-simps*
intro!:)
apply (*auto simp*: *vdom-m-simps5 ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*
literals-are-in-$\mathcal{L}_{in}$*-def*)
done
qed
end

lemma *add-init-cls-heur-add-init-cls*:
‹(*uncurry* (*add-init-cls-heur unbdd*), *uncurry* (*add-to-clauses-init-wl*)) ∈
$[\lambda(C, S).$ *length* $C \geq 2 \wedge$ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* C) $\wedge$ *distinct* $C]_f$
Id $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
proof −
have ‹*42* + *Max-mset* (*add-mset 0* (*x1c*)) ∉# *x1c*› **and** ‹*42* + *Max-mset* (*add-mset* (*0* :: *nat*) (*x1c*)) ≠ *0*› **for** *x1c*
apply (*cases* ‹*x1c*›) **apply** (*auto simp*: *max-def*)
apply (*metis Max-ge add.commute add.right-neutral add-le-cancel-left finite-set-mset le-zero-eq set-mset-add-mset-insert union-single-eq-member zero-neq-numeral*)
by (*smt Max-ge Set.set-insert add.commute add.right-neutral add-mset-commute antisym diff-add-inverse diff-le-self finite-insert finite-set-mset insert-DiffM insert-commute set-mset-add-mset-insert union-single-eq-member zero-neq-numeral*)
then have [*iff*]: ‹(∀ *b*. *b* = (*0*::*nat*) ∨ *b* ∈# *x1c*) ⟷ *False*› ‹∃ *b*>*0*. *b* ∉# *x1c*›**for** *x1c*
by *blast*+
have *add-to-clauses-init-wl-alt-def*:
‹*add-to-clauses-init-wl* = (λ*i* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*). *do* {
let b = (*length i* = *2*);
(*N'*, *ia*) ← *SPEC* (λ(*N'*, *ia*). *ia* > *0* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*i*, *True*) *N*);
RETURN ((*M*, *N'*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)
})›
by (*auto simp*: *add-to-clauses-init-wl-def get-fresh-index-def Let-def*
RES-RES2-RETURN-RES RES-RES-RETURN-RES2 RES-RETURN-RES uncurry-def image-iff
intro!: *ext*)
show *?thesis*
unfolding *add-init-cls-heur-def add-to-clauses-init-wl-alt-def uncurry-def Let-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg init-fm-add-new*)
subgoal
by (*rule add-init-pre1*)
subgoal
by (*rule add-init-pre2*)
apply (*rule lhs-step-If*)
apply (*refine-rcg*)
subgoal unfolding *twl-st-heur-parsing-no-WL-def*
by (*force dest*!: *valid-arena-vdom-le*(*2*) *simp*: *distinct-card*)
apply (*rule init-fm-add-new*)
apply *assumption*+
subgoal by *auto*
subgoal by (*rule add-init-cls-final-rel*)
subgoal **unfolding** *RES-RES2-RETURN-RES RETURN-def*
apply *simp*
unfolding *RETURN-def* **apply** (*rule RES-refine*)
by (*auto simp*: *twl-st-heur-parsing-no-WL-def RETURN-def intro*!: *RES-refine*)
done
qed

definition *already-propagated-unit-cls-conflict*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls-conflict* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*).
((*M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *NS*, *US*, {#}), *OC*))›

definition *already-propagated-unit-cls-conflict-heur*

:: ‹*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-conflict-heur* = (λL (*M*, *N*, *D*, *Q*, *oth*). *do* {
ASSERT (*isa-length-trail-pre M*);
RETURN (*M*, *N*, *D*, *isa-length-trail M*, *oth*)
})›

lemma *already-propagated-unit-cls-conflict-heur-already-propagated-unit-cls-conflict*:
‹(*uncurry already-propagated-unit-cls-conflict-heur*,
uncurry (*RETURN oo already-propagated-unit-cls-conflict*)) ∈
$[\lambda(L, S).\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f$ *Id* $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* →
⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-conflict-heur-def*
already-propagated-unit-cls-conflict-def all-lits-of-mm-add-mset
all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$*-iff isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *vmtf-consD*
intro!: *ASSERT-leI isa-length-trail-pre*)

definition (**in** −) *set-conflict-empty* :: ‹*nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-empty* - = *Some* {#}›

definition (**in** −) *lookup-set-conflict-empty* :: ‹*conflict-option-rel* ⇒ *conflict-option-rel*› **where**
‹*lookup-set-conflict-empty* = (λ(*b*, *s*) . (*False*, *s*))›

lemma *lookup-set-conflict-empty-set-conflict-empty*:
‹(*RETURN o lookup-set-conflict-empty*, *RETURN o set-conflict-empty*) ∈
$[\lambda D.\ D = None]_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *set-conflict-empty-def*
lookup-set-conflict-empty-def option-lookup-clause-rel-def
lookup-clause-rel-def)

definition *set-empty-clause-as-conflict-heur*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*set-empty-clause-as-conflict-heur* = (λ (*M*, *N*, (-, (*n*, *xs*)), *Q*, *WS*). *do* {
ASSERT(*isa-length-trail-pre M*);
RETURN (*M*, *N*, (*False*, (*n*, *xs*)), *isa-length-trail M*, *WS*)})›

lemma *set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict*:
‹(*set-empty-clause-as-conflict-heur*, *RETURN o add-empty-conflict-init-wl*) ∈
$[\lambda S.\ get\text{-}conflict\text{-}init\text{-}wl\ S = None]_f$
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *set-empty-clause-as-conflict-heur-def add-empty-conflict-init-wl-def*
twl-st-heur-parsing-no-WL-def set-conflict-empty-def option-lookup-clause-rel-def
lookup-clause-rel-def isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*]
intro!: *isa-length-trail-pre ASSERT-leI*)

definition (**in** −) *add-clause-to-others-heur*
:: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-clause-to-others-heur* = (λ - (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*).
RETURN (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*))›

lemma *add-clause-to-others-heur-add-clause-to-others*:

⟨(*uncurry add-clause-to-others-heur*, *uncurry* (*RETURN oo add-to-other-init*)) ∈ ⟨*Id*⟩*list-rel* $\times_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*⟩
by (*intro frefI nres-relI*)
(*auto simp*: *add-clause-to-others-heur-def add-to-other-init.simps*
twl-st-heur-parsing-no-WL-def)

definition (**in** −)*list-length-1* **where**
[*simp*]: ⟨*list-length-1 C* ⟷ *length C* = *1*⟩

definition (**in** −)*list-length-1-code* **where**
⟨*list-length-1-code C* ⟷ (*case C of* [-] ⇒ *True* | - ⇒ *False*)⟩

definition (**in** −) *get-conflict-wl-is-None-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ *bool*⟩ **where**
⟨*get-conflict-wl-is-None-heur-init* = (λ(*M*, *N*, (*b*, -), *Q*, -). *b*)⟩

definition *init-dt-step-wl-heur*
:: ⟨*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*twl-st-wl-heur-init*) *nres*⟩
where
⟨*init-dt-step-wl-heur unbdd C S* = *do* {
  if get-conflict-wl-is-None-heur-init S
  then do {
    if is-Nil C
    then set-empty-clause-as-conflict-heur S
    else if list-length-1 C
    then do {
      ASSERT (*C* ≠ []);
      let L = *C* ! *0*;
      ASSERT(*polarity-pol-pre* (*get-trail-wl-heur-init S*) *L*);
      let val-L = *polarity-pol* (*get-trail-wl-heur-init S*) *L*;
      if val-L = *None*
      then propagate-unit-cls-heur L S
      else
        if val-L = *Some True*
        then already-propagated-unit-cls-heur C S
        else conflict-propagated-unit-cls-heur L S
    }
    else do {
      ASSERT(*length C* ≥ *2*);
      add-init-cls-heur unbdd C S
    }
  }
  else add-clause-to-others-heur C S
}⟩

named-theorems *twl-st-heur-parsing-no-WL*
lemma [*twl-st-heur-parsing-no-WL*]:
assumes ⟨(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
shows ⟨(*get-trail-wl-heur-init S*, *get-trail-init-wl T*) ∈ *trail-pol* $\mathcal{A}$⟩
using *assms*
by (*cases S*; *auto simp*: *twl-st-heur-parsing-no-WL-def*; *fail*)+

definition *get-conflict-wl-is-None-init* :: ⟨*nat twl-st-wl-init* ⇒ *bool*⟩ **where**

⟨*get-conflict-wl-is-None-init* = (λ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*). *is-None D*)⟩

lemma *get-conflict-wl-is-None-init-alt-def*:
⟨*get-conflict-wl-is-None-init S* ⟷ *get-conflict-init-wl S* = *None*⟩
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*:
⟨(*RETURN o get-conflict-wl-is-None-heur-init*, *RETURN o get-conflict-wl-is-None-init*) ∈
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*⟩
apply (*intro frefI nres-relI*)
apply (*rename-tac x y*, *case-tac x*, *case-tac y*)
by (*auto simp*: *twl-st-heur-parsing-no-WL-def get-conflict-wl-is-None-heur-init-def option-lookup-clause-rel-def get-conflict-wl-is-None-init-def split*: *option.splits*)

definition (**in** −) *get-conflict-wl-is-None-init′* **where**
⟨*get-conflict-wl-is-None-init′* = *get-conflict-wl-is-None*⟩

lemma *init-dt-step-wl-heur-init-dt-step-wl*:
⟨(*uncurry* (*init-dt-step-wl-heur unbdd*), *uncurry init-dt-step-wl*) ∈
[λ(*C*, *S*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*]$_f$
Id $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*⟩
supply [[*goals-limit=1*]]
unfolding *init-dt-step-wl-heur-def init-dt-step-wl-def uncurry-def*
option.case-eq-if get-conflict-wl-is-None-init-alt-def[*symmetric*]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict[*THEN fref-to-Down*,
unfolded comp-def]
propagate-unit-cls-heur-propagate-unit-cls[*THEN fref-to-Down-curry*, *unfolded comp-def*]
already-propagated-unit-cls-heur-already-propagated-unit-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
add-init-cls-heur-add-init-cls[*THEN fref-to-Down-curry*,
unfolded comp-def]
add-clause-to-others-heur-add-clause-to-others[*THEN fref-to-Down-curry*,
unfolded comp-def])
subgoal by (*auto simp*: *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*[*THEN fref-to-Down-unRET-Id*])
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def is-Nil-def split*: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by *auto*
subgoal by *simp*
subgoal by *simp*
subgoal by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
twl-st-heur-parsing-no-WL-def intro!: *polarity-pol-pre split*: *list.splits*)
subgoal for *C′S CT C T C′ S*
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id,
of ⟨*get-trail-init-wl T*⟩ ⟨*hd C*⟩])
(*auto simp*: *polarity-def twl-st-heur-parsing-no-WL-def*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
literals-are-in-$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)

subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def* *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def hd-conv-nth*)
subgoal for $C'S$ CT C T C' S
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id,
of ⟨*get-trail-init-wl T*⟩ ⟨*hd C*⟩])
(*auto simp*: *polarity-def twl-st-heur-parsing-no-WL-def*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
literals-are-in-$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal by (*auto simp*: *list-mset-rel-def br-def*)
subgoal by (*simp add*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by (*simp add*: *hd-conv-nth*)
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal for *x y x1 x2 C x2a*
by (*cases C*; *cases* ⟨*tl C*⟩)
(*auto simp*: *twl-st-heur-parsing-no-WL-def map-fun-rel-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*
split: *list.splits*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
done

lemma (**in** −) *get-conflict-wl-is-None-heur-init-alt-def*:
⟨*RETURN o get-conflict-wl-is-None-heur-init* = (λ(*M*, *N*, (*b*, -), *Q*, *W*, -). *RETURN b*)⟩
by (*auto simp*: *get-conflict-wl-is-None-heur-init-def intro*!: *ext*)

definition *polarity-st-heur-init* :: ⟨*twl-st-wl-heur-init* ⇒ - ⇒ *bool option*⟩ **where**
⟨*polarity-st-heur-init* = (λ(*M*, -) *L*. *polarity-pol M L*)⟩

lemma *polarity-st-heur-init-alt-def*:
⟨*polarity-st-heur-init S L* = *polarity-pol* (*get-trail-wl-heur-init S*) *L*⟩
by (*cases S*) (*auto simp*: *polarity-st-heur-init-def*)

definition *polarity-st-init* :: ⟨*'v twl-st-wl-init* ⇒ *'v literal* ⇒ *bool option*⟩ **where**
⟨*polarity-st-init S* = *polarity* (*get-trail-init-wl S*)⟩

lemma *get-conflict-wl-is-None-init*:
⟨*get-conflict-init-wl S* = *None* ⟷ *get-conflict-wl-is-None-init S*⟩
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

definition *init-dt-wl-heur*
:: ⟨*bool* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩
where
⟨*init-dt-wl-heur unbdd CS S* = *nfoldli CS* (λ-. *True*)

$(\lambda C\ S.\ do\ \{$
$init\text{-}dt\text{-}step\text{-}wl\text{-}heur\ unbdd\ C\ S\})\ S\rangle$

definition $init\text{-}dt\text{-}step\text{-}wl\text{-}heur\text{-}unb :: \langle nat\ clause\text{-}l \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur\text{-}init)\ nres\rangle$
where
$\langle init\text{-}dt\text{-}step\text{-}wl\text{-}heur\text{-}unb = init\text{-}dt\text{-}step\text{-}wl\text{-}heur\ True\rangle$

definition $init\text{-}dt\text{-}wl\text{-}heur\text{-}unb :: \langle nat\ clause\text{-}l\ list \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init\ nres\rangle$
where
$\langle init\text{-}dt\text{-}wl\text{-}heur\text{-}unb = init\text{-}dt\text{-}wl\text{-}heur\ True\rangle$

definition $init\text{-}dt\text{-}step\text{-}wl\text{-}heur\text{-}b :: \langle nat\ clause\text{-}l \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur\text{-}init)\ nres\rangle$
where
$\langle init\text{-}dt\text{-}step\text{-}wl\text{-}heur\text{-}b = init\text{-}dt\text{-}step\text{-}wl\text{-}heur\ False\rangle$

definition $init\text{-}dt\text{-}wl\text{-}heur\text{-}b :: \langle nat\ clause\text{-}l\ list \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\text{-}init\ nres\rangle$ **where**
$\langle init\text{-}dt\text{-}wl\text{-}heur\text{-}b = init\text{-}dt\text{-}wl\text{-}heur\ False\rangle$

15.1.3 Extractions of the atoms in the state

definition $init\text{-}valid\text{-}rep :: nat\ list \Rightarrow nat\ set \Rightarrow bool$ **where**
$\langle init\text{-}valid\text{-}rep\ xs\ l \longleftrightarrow$
$(\forall L\in l.\ L < length\ xs) \wedge$
$(\forall L \in l.\ (xs\ !\ L)\ mod\ 2 = 1) \wedge$
$(\forall L.\ L < length\ xs \longrightarrow (xs\ !\ L)\ mod\ 2 = 1 \longrightarrow L \in l)\rangle$

definition $isasat\text{-}atms\text{-}ext\text{-}rel :: \langle((nat\ list \times nat \times nat\ list) \times nat\ set)\ set\rangle$ **where**
$\langle isasat\text{-}atms\text{-}ext\text{-}rel = \{((xs,\ n,\ atms),\ l).$
$init\text{-}valid\text{-}rep\ xs\ l \wedge$
$n = Max\ (insert\ 0\ l) \wedge$
$length\ xs < uint32\text{-}max \wedge$
$(\forall s\in set\ xs.\ s \le uint64\text{-}max) \wedge$
$finite\ l \wedge$
$distinct\ atms \wedge$
$set\ atms = l \wedge$
$length\ xs \ne 0$
$\}\rangle$

lemma $distinct\text{-}length\text{-}le\text{-}Suc\text{-}Max$:
assumes $\langle distinct\ (b :: nat\ list)\rangle$
shows $\langle length\ b \le Suc\ (Max\ (insert\ 0\ (set\ b)))\rangle$
proof $-$
have $\langle set\ b \subseteq \{0\ ..< Suc\ (Max\ (insert\ 0\ (set\ b)))\}\rangle$
by $(cases\ \langle set\ b = \{\}\rangle)$
$(auto\ simp\ add:\ le\text{-}imp\text{-}less\text{-}Suc)$
from $card\text{-}mono[OF\ \text{-}\ this]$ **show** $?thesis$
using $distinct\text{-}card[OF\ assms(1)]$ **by** $auto$
qed

lemma $isasat\text{-}atms\text{-}ext\text{-}rel\text{-}alt\text{-}def$:
$\langle isasat\text{-}atms\text{-}ext\text{-}rel = \{((xs,\ n,\ atms),\ l).$
$init\text{-}valid\text{-}rep\ xs\ l \wedge$
$n = Max\ (insert\ 0\ l) \wedge$
$length\ xs < uint32\text{-}max \wedge$
$(\forall s\in set\ xs.\ s \le uint64\text{-}max) \wedge$

finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0* ∧
length atms ≤ *Suc n*
}›
by (*auto simp*: *isasat-atms-ext-rel-def distinct-length-le-Suc-Max*)

definition *in-map-atm-of* :: ‹'*a* ⇒ '*a list* ⇒ *bool*› **where**
‹*in-map-atm-of L N* ⟷ *L* ∈ *set N*›

definition (**in** −) *init-next-size* **where**
‹*init-next-size L* = *2* ∗ *L*›

lemma *init-next-size*: ‹*L* ≠ *0* ⟹ *L* + *1* ≤ *uint32-max* ⟹ *L* < *init-next-size L*›
by (*auto simp*: *init-next-size-def uint32-max-def*)

definition *add-to-atms-ext* **where**
‹*add-to-atms-ext* = (λ*i* (*xs*, *n*, *atms*). *do* {
ASSERT(*i* ≤ *uint32-max div 2*);
ASSERT(*length xs* ≤ *uint32-max*);
ASSERT(*length atms* ≤ *Suc n*);
let n = *max i n*;
(*if i* < *length-uint32-nat xs then do* {
ASSERT(*xs*!*i* ≤ *uint64-max*);
let atms = (*if xs*!*i AND 1* = *1 then atms else atms* @ [*i*]);
RETURN (*xs*[*i* := *1*], *n*, *atms*)
}
else do {
ASSERT(*i* + *1* ≤ *uint32-max*);
ASSERT(*length-uint32-nat xs* ≠ *0*);
ASSERT(*i* < *init-next-size i*);
RETURN ((*list-grow xs* (*init-next-size i*) *0*)[*i* := *1*], *n*,
atms @ [*i*])
})
})›

lemma *init-valid-rep-upd-OR*:
‹*init-valid-rep* (*x1b*[*x1a* := *a OR 1*]) *x2* ⟷
init-valid-rep (*x1b*[*x1a* := *1*]) *x2* › (**is** ‹*?A* ⟷ *?B*›)
proof
assume *?A*
then have
1: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *a OR 1*])› **and**
2: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *a OR 1*] ! *L mod 2* = *1*› **and**
3: ‹∀ *L*<*length* (*x1b*[*x1a* := *a OR 1*]).
x1b[*x1a* := *a OR 1*] ! *L mod 2* = *1* ⟶
L ∈ *x2*›
unfolding *init-valid-rep-def* **by** *fast*+
have *1*: ‹∀ *L*∈*x2*. *L* < *length* (*x1b*[*x1a* := *1*])›
using *1* **by** *simp*
then have *2*: ‹∀ *L*∈*x2*. *x1b*[*x1a* := *1*] ! *L mod 2* = *1*›
using *2* **by** (*auto simp*: *nth-list-update'*)
then have *3*: ‹∀ *L*<*length* (*x1b*[*x1a* := *1*]).
x1b[*x1a* := *1*] ! *L mod 2* = *1* ⟶

        $L \in x2$›
    using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
  show *?B*
    using *1 2 3*
    unfolding *init-valid-rep-def* **by** *fast+*
next
  assume *?B*
  then have
    1: ‹$\forall L{\in}x2.\ L < length\ (x1b[x1a := 1])$› **and**
    2: ‹$\forall L{\in}x2.\ x1b[x1a := 1]\ !\ L\ mod\ 2 = 1$› **and**
    3: ‹$\forall L{<}length\ (x1b[x1a := 1]).$
        $x1b[x1a := 1]\ !\ L\ mod\ 2 = 1 \longrightarrow$
        $L \in x2$›
    unfolding *init-valid-rep-def* **by** *fast+*
  have *1*: ‹$\forall L{\in}x2.\ L < length\ (x1b[x1a :=\ \ a\ OR\ 1])$›
    using *1* **by** *simp*
  then have *2*: ‹$\forall L{\in}x2.\ x1b[x1a := a\ OR\ 1]\ !\ L\ mod\ 2 = 1$›
    using *2* **by** (*auto simp*: *nth-list-update′ bitOR-1-if-mod-2-nat*)
  then have *3*: ‹$\forall L{<}length\ (x1b[x1a :=\ \ a\ OR\ 1]).$
        $x1b[x1a :=\ \ a\ OR\ 1]\ !\ L\ mod\ 2 = 1 \longrightarrow$
        $L \in x2$›
    using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
  show *?A*
    using *1 2 3*
    unfolding *init-valid-rep-def* **by** *fast+*
qed

lemma *init-valid-rep-insert*:
  assumes *val*: ‹*init-valid-rep x1b x2*› **and** *le*: ‹$x1a < length\ x1b$›
  shows ‹*init-valid-rep* ($x1b[x1a := Suc\ 0]$) (*insert x1a x2*)›
proof −
  have
    1: ‹$\forall L{\in}x2.\ L < length\ x1b$› **and**
    2: ‹$\forall L{\in}x2.\ x1b\ !\ L\ mod\ 2 = 1$› **and**
    3: ‹$\bigwedge L.\ L{<}length\ x1b \Longrightarrow x1b\ !\ L\ mod\ 2 = 1 \longrightarrow L \in x2$›
    using *val* **unfolding** *init-valid-rep-def* **by** *fast+*
  have *1*: ‹$\forall L{\in}insert\ x1a\ x2.\ L < length\ (x1b[x1a := 1])$›
    using *1 le* **by** *simp*
  then have *2*: ‹$\forall L{\in}insert\ x1a\ x2.\ x1b[x1a := 1]\ !\ L\ mod\ 2 = 1$›
    using *2* **by** (*auto simp*: *nth-list-update′*)
  then have *3*: ‹$\forall L{<}length\ (x1b[x1a := 1]).$
        $x1b[x1a := 1]\ !\ L\ mod\ 2 = 1 \longrightarrow$
        $L \in insert\ x1a\ x2$›
    using *3 le* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
  show *?thesis*
    using *1 2 3*
    unfolding *init-valid-rep-def* **by** *auto*
qed

lemma *init-valid-rep-extend*:
  ‹*init-valid-rep* (*x1b* @ *replicate n 0*) *x2* $\longleftrightarrow$ *init-valid-rep* (*x1b*) *x2*›
   (**is** ‹*?A* $\longleftrightarrow$ *?B*› **is** ‹*init-valid-rep ?x1b* - $\longleftrightarrow$ -›)
proof
  assume *?A*
  then have
    1: ‹$\bigwedge L.\ L{\in}x2 \Longrightarrow L < length\ ?x1b$› **and**

    2: ‹⋀L. L∈x2 ⟹ ?x1b ! L mod 2 = 1› **and**
    3: ‹⋀L. L<length ?x1b ⟹ ?x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    unfolding *init-valid-rep-def* **by** *fast+*
  have *1*: ‹L∈x2 ⟹ L < length x1b› **for** *L*
    using *3[of L] 2[of L] 1[of L]*
    by (*auto simp*: *nth-append split*: *if-splits*)
  then have *2*: ‹∀ L∈x2. x1b ! L mod 2 = 1›
    using *2* **by** (*auto simp*: *nth-list-update'*)
  then have *3*: ‹∀ L<length x1b. x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    using *3* **by** (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat*)
  show *?B*
    using *1 2 3*
    unfolding *init-valid-rep-def* **by** *fast*
next
  assume *?B*
  then have
    1: ‹⋀L. L∈x2 ⟹ L < length x1b› **and**
    2: ‹⋀L. L∈x2 ⟹ x1b ! L mod 2 = 1› **and**
    3: ‹⋀L. L<length x1b ⟶ x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    unfolding *init-valid-rep-def* **by** *fast+*
  have *10*: ‹∀ L∈x2. L < length ?x1b›
    using *1* **by** *fastforce*
  then have *20*: ‹L∈x2 ⟹ ?x1b ! L mod 2 = 1› **for** *L*
    using *1[of L] 2[of L] 3[of L]* **by** (*auto simp*: *nth-list-update' bitOR-1-if-mod-2-nat nth-append*)
  then have *30*: ‹L<length ?x1b ⟹ ?x1b ! L mod 2 = 1 ⟶ L ∈ x2›**for** *L*
    using *1[of L] 2[of L] 3[of L]*
    by (*auto split*: *if-splits simp*: *bitOR-1-if-mod-2-nat nth-append*)
  show *?A*
    using *10 20 30*
    unfolding *init-valid-rep-def* **by** *fast+*
qed

lemma *init-valid-rep-in-set-iff*:
  ‹init-valid-rep x1b x2 ⟹ x ∈ x2 ⟷ (x < length x1b ∧ (x1b!x) mod 2 = 1)›
  unfolding *init-valid-rep-def*
  by *auto*

lemma *add-to-atms-ext-op-set-insert*:
  ‹(uncurry add-to-atms-ext, uncurry (RETURN oo Set.insert))
   ∈ [λ(n, l). n ≤ uint32-max div 2]$_f$ nat-rel $\times_f$ isasat-atms-ext-rel → ⟨isasat-atms-ext-rel⟩nres-rel›
proof −
  have *H*: ‹finite x2 ⟹ Max (insert x1 (insert 0 x2)) = Max (insert x1 x2)›
    ‹finite x2 ⟹ Max (insert 0 (insert x1 x2)) = Max (insert x1 x2)›
    for *x1* **and** *x2* :: ‹nat set›
    by (*subst insert-commute*) *auto*
  have [*simp*]: ‹(a OR Suc 0) mod 2 = Suc 0› **for** *a*
    by (*auto simp add*: *bitOR-1-if-mod-2-nat*)
  show *?thesis*
    apply (*intro frefI nres-relI*)
    unfolding *isasat-atms-ext-rel-def add-to-atms-ext-def uncurry-def*
    apply (*refine-vcg lhs-step-If*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal unfolding *isasat-atms-ext-rel-def[symmetric] isasat-atms-ext-rel-alt-def* **by** *auto*
    subgoal by *auto*
    subgoal for *x y x1 x2 x1a x2a x1b x2b*

unfolding *comp-def*
apply (*rule RETURN-refine*)
apply (*subst in-pair-collect-simp*)
apply (*subst prod.case*)+
apply (*intro conjI impI allI*)
subgoal by (*simp add*: *init-valid-rep-upd-OR init-valid-rep-insert*
del:)
subgoal by (*auto simp*: *H Max-insert*[*symmetric*] *simp del*: *Max-insert*)
subgoal by *auto*
subgoal
unfolding *bitOR-1-if-mod-2-nat*
by (*auto simp del*: *simp*: *uint64-max-def*
elim!: *in-set-upd-cases*)
subgoal
unfolding *bitAND-1-mod-2*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
unfolding *bitAND-1-mod-2*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
unfolding *bitAND-1-mod-2*
by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal
by (*auto simp add*: *init-valid-rep-in-set-iff*)
done
subgoal by (*auto simp*: *uint32-max-def*)
subgoal by (*auto simp*: *uint32-max-def*)
subgoal by (*auto simp*: *uint32-max-def init-next-size-def elim*: *neq-NilE*)
subgoal
unfolding *comp-def list-grow-def*
apply (*rule RETURN-refine*)
apply (*subst in-pair-collect-simp*)
apply (*subst prod.case*)+
apply (*intro conjI impI allI*)
subgoal
unfolding *init-next-size-def*
apply (*simp del*:)
apply (*subst init-valid-rep-insert*)
apply (*auto elim*: *neq-NilE*)
apply (*subst init-valid-rep-extend*)
apply (*auto elim*: *neq-NilE*)
done
subgoal by (*auto simp*: *H Max-insert*[*symmetric*] *simp del*: *Max-insert*)
subgoal by (*auto simp*: *init-next-size-def uint32-max-def*)
subgoal
unfolding *bitOR-1-if-mod-2-nat*
by (*auto simp*: *uint64-max-def*
elim!: *in-set-upd-cases*)
subgoal by (*auto simp*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
subgoal by (*auto simp add*: *init-valid-rep-in-set-iff*)
done
done
qed

definition *extract-atms-cls* :: ⟨*'a clause-l* ⇒ *'a set* ⇒ *'a set*⟩ **where**
⟨*extract-atms-cls* C $\mathcal{A}_{in}$ = *fold* (λL $\mathcal{A}_{in}$. *insert* (*atm-of* L) $\mathcal{A}_{in}$) C $\mathcal{A}_{in}$⟩

definition *extract-atms-cls-i* :: ⟨*nat clause-l* ⇒ *nat set* ⇒ *nat set nres*⟩ **where**
⟨*extract-atms-cls-i* C $\mathcal{A}_{in}$ = *nfoldli* C (λ-. *True*)
(λL $\mathcal{A}_{in}$. *do* {
ASSERT(*atm-of* $L \leq$ *uint32-max div 2*);
RETURN(*insert* (*atm-of* L) $\mathcal{A}_{in}$)})
$\mathcal{A}_{in}$⟩

lemma *fild-insert-insert-swap*:
⟨*fold* (λL. *insert* (f L)) C (*insert* a $\mathcal{A}_{in}$) = *insert* a (*fold* (λL. *insert* (f L)) C $\mathcal{A}_{in}$)⟩
by (*induction* C *arbitrary*: a $\mathcal{A}_{in}$) (*auto simp*: *extract-atms-cls-def*)

lemma *extract-atms-cls-alt-def*: ⟨*extract-atms-cls* C $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ *atm-of* ‘ *set* C⟩
by (*induction* C) (*auto simp*: *extract-atms-cls-def fild-insert-insert-swap*)

lemma *extract-atms-cls-i-extract-atms-cls*:
⟨(*uncurry extract-atms-cls-i*, *uncurry* (*RETURN oo extract-atms-cls*))
∈ [λ(C, $\mathcal{A}_{in}$). ∀ L∈*set* C. *nat-of-lit* $L \leq$ *uint32-max*]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*⟩
proof −
have *H1*: ⟨(*x1a*, *x1*) ∈ ⟨{(L, L'). $L = L'$ ∧ *nat-of-lit* $L \leq$ *uint32-max*}⟩*list-rel*⟩
if
⟨*case* y *of* (C, $\mathcal{A}_{in}$) ⇒ ∀ L∈*set* C. *nat-of-lit* $L \leq$ *uint32-max*⟩ **and**
⟨(x, y) ∈ ⟨*nat-lit-lit-rel*⟩*list-rel* $\times_f$ *Id*⟩ **and**
⟨y = (*x1*, *x2*)⟩ **and**
⟨x = (*x1a*, *x2a*)⟩
for x :: ⟨*nat literal list* × *nat set*⟩ **and** y :: ⟨*nat literal list* × *nat set*⟩ **and**
x1 :: ⟨*nat literal list*⟩ **and** *x2* :: ⟨*nat set*⟩ **and** *x1a* :: ⟨*nat literal list*⟩ **and** *x2a* :: ⟨*nat set*⟩
using *that* **by** (*auto simp*: *list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth*)

have *atm-le*: ⟨*nat-of-lit xa* ≤ *uint32-max* ⟹ *atm-of xa* ≤ *uint32-max div 2*⟩ **for** *xa*
by (*cases xa*) (*auto simp*: *uint32-max-def*)

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *extract-atms-cls-i-def extract-atms-cls-def uncurry-def comp-def*
fold-eq-nfoldli
apply (*intro frefI nres-relI*)
apply (*refine-rcg H1*)
apply *assumption*+
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *atm-le*)
subgoal by *auto*
done
qed

definition *extract-atms-clss*:: ⟨*'a clause-l list* ⇒ *'a set* ⇒ *'a set*⟩ **where**
⟨*extract-atms-clss* N $\mathcal{A}_{in}$ = *fold extract-atms-cls* N $\mathcal{A}_{in}$⟩

definition *extract-atms-clss-i* :: ⟨*nat clause-l list* ⇒ *nat set* ⇒ *nat set nres*⟩ **where**
⟨*extract-atms-clss-i* N $\mathcal{A}_{in}$ = *nfoldli* N (λ-. *True*) *extract-atms-cls-i* $\mathcal{A}_{in}$⟩

lemma *extract-atms-clss-i-extract-atms-clss*:
‹(*uncurry extract-atms-clss-i*, *uncurry* (*RETURN oo extract-atms-clss*))
$\in [\lambda(N, \mathcal{A}_{in}).\ \forall C{\in}set\ N.\ \forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint32\text{-}max]_f$
$\langle Id\rangle list\text{-}rel \times_f Id \rightarrow \langle Id\rangle nres\text{-}rel$›
proof −
have *H1*: ‹$(x1a, x1) \in \langle\{(C, C').\ C = C' \wedge (\forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint32\text{-}max)\}\rangle list\text{-}rel$›
if
‹*case y of* $(N, \mathcal{A}_{in}) \Rightarrow \forall C{\in}set\ N.\ \forall L{\in}set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint32\text{-}max$› **and**
‹$(x, y) \in \langle Id\rangle list\text{-}rel \times_f Id$› **and**
‹$y = (x1, x2)$› **and**
‹$x = (x1a, x2a)$›
for *x* :: ‹*nat literal list list* × *nat set*› **and** *y* :: ‹*nat literal list list* × *nat set*› **and**
x1 :: ‹*nat literal list list*› **and** *x2* :: ‹*nat set*› **and** *x1a* :: ‹*nat literal list list*›
and *x2a* :: ‹*nat set*›
using *that* **by** (*auto simp*: *list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth*)

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *extract-atms-clss-i-def extract-atms-clss-def comp-def fold-eq-nfoldli uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg H1 extract-atms-cls-i-extract-atms-cls*[*THEN fref-to-Down-curry*,
unfolded comp-def])
apply *assumption+*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *fold-extract-atms-cls-union-swap*:
‹*fold extract-atms-cls N* $(\mathcal{A}_{in} \cup a)$ = *fold extract-atms-cls N* $\mathcal{A}_{in} \cup a$›
by (*induction N arbitrary*: *a* $\mathcal{A}_{in}$) (*auto simp*: *extract-atms-cls-alt-def*)

lemma *extract-atms-clss-alt-def*:
‹*extract-atms-clss N* $\mathcal{A}_{in} = \mathcal{A}_{in} \cup ((\bigcup C{\in}set\ N.\ atm\text{-}of\ `\ set\ C))$›
by (*induction N*)
(*auto simp*: *extract-atms-clss-def extract-atms-cls-alt-def*
fold-extract-atms-cls-union-swap)

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss CS'* {})› **for** *CS'*
by (*auto simp*: *extract-atms-clss-alt-def*)

definition *op-extract-list-empty* **where**
‹*op-extract-list-empty* = {}›

definition *extract-atms-clss-imp-empty-rel* **where**
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*replicate 1024 0*, *0*, []))›

lemma *extract-atms-clss-imp-empty-rel*:
‹(λ-. *extract-atms-clss-imp-empty-rel*, λ-. (*RETURN op-extract-list-empty*)) ∈
unit-rel $\rightarrow_f$ ⟨*isasat-atms-ext-rel*⟩ *nres-rel*›
by (*intro frefI nres-relI*)

(*simp add*: *op-extract-list-empty-def uint32-max-def*
isasat-atms-ext-rel-def init-valid-rep-def extract-atms-clss-imp-empty-rel-def
del: *replicate-numeral*)

lemma *extract-atms-cls-Nil*[*simp*]:
⟨*extract-atms-cls* [] $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$⟩
unfolding *extract-atms-cls-def fold.simps* **by** *simp*

lemma *extract-atms-clss-Cons*[*simp*]:
⟨*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)⟩
by (*simp add*: *extract-atms-clss-def*)

definition (**in** −) *all-lits-of-atms-m* :: ⟨′*a multiset* ⇒ ′*a clause*⟩ **where**
⟨*all-lits-of-atms-m N* = *poss N* + *negs N*⟩

lemma (**in** −) *all-lits-of-atms-m-nil*[*simp*]: ⟨*all-lits-of-atms-m* {#} = {#}⟩
unfolding *all-lits-of-atms-m-def* **by** *auto*

definition (**in** −) *all-lits-of-atms-mm* :: ⟨′*a multiset multiset* ⇒ ′*a clause*⟩ **where**
⟨*all-lits-of-atms-mm N* = *poss* ($\bigcup$# *N*) + *negs* ($\bigcup$# *N*)⟩

lemma *all-lits-of-atms-m-all-lits-of-m*:
⟨*all-lits-of-atms-m N* = *all-lits-of-m* (*poss N*)⟩
unfolding *all-lits-of-atms-m-def all-lits-of-m-def*
by (*induction N*) *auto*

Creation of an initial state

definition *init-dt-wl-heur-spec*
:: ⟨*bool* ⇒ *nat multiset* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init* ⇒ *bool*⟩
where
⟨*init-dt-wl-heur-spec unbdd* $\mathcal{A}$ *CS T TOC* ⟷
(∃ *T′ TOC′*. (*TOC*, *TOC′*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧ (*T*, *T′*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧
init-dt-wl-spec CS T′ TOC′)⟩

definition *init-state-wl* :: ⟨*nat twl-st-wl-init′*⟩ **where**
⟨*init-state-wl* = ([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#})⟩

definition *init-state-wl-heur* :: ⟨*nat multiset* ⇒ *twl-st-wl-heur-init nres*⟩ **where**
⟨*init-state-wl-heur* $\mathcal{A}$ = *do* {
M ← *SPEC*(λ*M*. (*M*, []) ∈ *trail-pol* $\mathcal{A}$);
D ← *SPEC*(λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$);
W ← *SPEC* (λ*W*. (*W*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$));
vm ← *RES* (*isa-vmtf-init* $\mathcal{A}$ []);
φ ← *SPEC* (*phase-saving* $\mathcal{A}$);
cach ← *SPEC* (*cach-refinement-empty* $\mathcal{A}$);
let lbd = *empty-lbd*;
let vdom = [];
RETURN (*M*, [], *D*, *0*, *W*, *vm*, φ, *0*, *cach*, *lbd*, *vdom*, *False*)}⟩

definition *init-state-wl-heur-fast* **where**
⟨*init-state-wl-heur-fast* = *init-state-wl-heur*⟩

lemma *init-state-wl-heur-init-state-wl*:
‹(λ-. (*init-state-wl-heur* $\mathcal{A}$), λ-. (*RETURN init-state-wl*)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$ *unit-rel* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *init-state-wl-heur-def init-state-wl-def*
RES-RETURN-RES bind-RES-RETURN-eq RES-RES-RETURN-RES RETURN-def
twl-st-heur-parsing-no-WL-wl-def vdom-m-def empty-watched-def valid-arena-empty
intro!: *RES-refine*)

definition (**in** −)*to-init-state* :: ‹*nat twl-st-wl-init'* ⇒ *nat twl-st-wl-init*› **where**
‹*to-init-state S* = (*S*, {#})›

definition (**in** −) *from-init-state* :: ‹*nat twl-st-wl-init-full* ⇒ *nat twl-st-wl*› **where**
‹*from-init-state* = *fst*›

definition (**in** −) *to-init-state-code* **where**
‹*to-init-state-code* = *id*›

definition *from-init-state-code* **where**
‹*from-init-state-code* = *id*›

definition (**in** −) *conflict-is-None-heur-wl* **where**
‹*conflict-is-None-heur-wl* = (λ(*M*, *N*, *U*, *D*, -). *is-None D*)›

definition (**in** −) *finalise-init* **where**
‹*finalise-init* = *id*›

15.1.4 Parsing

lemma *init-dt-wl-heur-init-dt-wl*:
‹(*uncurry* (*init-dt-wl-heur unbdd*), *uncurry init-dt-wl*) ∈
[λ(*CS*, *S*). (∀ *C* ∈ *set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)) ∧ *distinct-mset-set* (*mset* ‘ *set CS*)]$_f$
⟨*Id*⟩*list-rel* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
proof −
have *H*: ‹⋀*x y x1 x2 x1a x2a*.
(∀ *C*∈*set x1*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)) ∧ *distinct-mset-set* (*mset* ‘ *set x1*) ⟹
(*x1a*, *x1*) ∈ ⟨*Id*⟩*list-rel* ⟹
(*x1a*, *x1*) ∈ ⟨{(*C*, *C'*). *C* = *C'* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧
distinct C}⟩*list-rel*›
apply (*auto simp*: *list-rel-def list-all2-conj*)
apply (*auto simp*: *list-all2-conv-all-nth distinct-mset-set-def*)
done

show *?thesis*
unfolding *init-dt-wl-heur-def init-dt-wl-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*case-tac y rule*: *prod.exhaust*)
apply (*case-tac x rule*: *prod.exhaust*)
apply (*simp only*: *prod.case prod-rel-iff*)
apply (*refine-vcg init-dt-step-wl-heur-init-dt-step-wl*[*THEN fref-to-Down-curry*] *H*)
apply *normalize-goal*+
subgoal by *fast*
subgoal by *fast*

subgoal by *simp*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
done
qed

definition *rewatch-heur-st*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*rewatch-heur-st* = (λ(*M'*, *N'*, *D'*, *j*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*). *do* {
ASSERT(*length vdom* ≤ *length N'*);
W ← *rewatch-heur vdom N' W*;
RETURN (*M'*, *N'*, *D'*, *j*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*)
})›

lemma *rewatch-heur-st-correct-watching*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*› **and** *failed*: ‹¬*is-failed-heur-init S*›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* (*get-clauses-init-wl T*))› **and**
‹$\bigwedge$*x*. *x* ∈# *dom-m* (*get-clauses-init-wl T*) ⟹ *distinct* (*get-clauses-init-wl T* ∝ *x*) ∧
2 ≤ *length* (*get-clauses-init-wl T* ∝ *x*)›
shows ‹*rewatch-heur-st S* ≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *unbdd*)
(*SPEC* (λ((*M*,*N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*), *OC*). *T* = ((*M*,*N*,*D*,*NE*,*UE*,*NS*, *US*, *Q*), *OC*)∧
correct-watching (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)))›
proof −
obtain *M N D NE UE NS US Q OC* **where**
T: ‹*T* = ((*M*,*N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)›
by (*cases T*) *auto*

obtain *M' N' D' j W vm* φ *clvls cach lbd vdom* **where**
S: ‹*S* = (*M'*, *N'*, *D'*, *j*, *W*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *False*)›
using *failed* **by** (*cases S*) *auto*

have *valid*: ‹*valid-arena N' N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
dom-m-vdom: ‹*set-mset* (*dom-m N*) ⊆ *set vdom*› **and**
W: ‹(*W*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)›
using *assms distinct-mset-dom*[*of N*] **apply** (*auto simp*: *twl-st-heur-parsing-no-WL-def S T*
simp flip: *distinct-mset-mset-distinct*)
by (*metis distinct-mset-set-mset-ident set-mset-mset subset-mset.eq-iff*)+
have *H*: ‹*RES* ({(*W*, *W'*).
(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W' N* ⊆ *set-mset* (*dom-m N*)}$^{-1}$ ‘‘
{*W*. *Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)})
≤ *RES* ({(*W*, *W'*).
(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W' N* ⊆ *set-mset* (*dom-m N*)}$^{-1}$ ‘‘
{*W*. *Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)})›
for *W'*
by (*rule order.refl*)
have *eq*: ‹*Watched-Literals-Watch-List-Initialisation.correct-watching-init*
(*M*, *N*, *None*, *NE*, *UE*, *NS*, *US*, {#}, *xa*) ⟹

vdom-m $\mathcal{A}$ *xa N* = *set-mset* (*dom-m N*)› **for** *xa*
by (*auto 5 5 simp*: *Watched-Literals-Watch-List-Initialisation.correct-watching-init.simps vdom-m-def*)
show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *rewatch-heur-st-def T S*
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal by (*auto dest*: *valid-arena-vdom-subset simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist dom-m-vdom W lits*])
apply (*solves simp*)
apply (*solves simp*)
apply (*rule order-trans*[*OF ref-two-step'*])
apply (*rule rewatch-correctness*)
apply (*rule empty-watched-def*)
subgoal
using *assms*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*subst conc-fun-RES*)
apply (*rule H*) **apply** (*rule RETURN-RES-refine*)
apply (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def all-atms-def*[*symmetric*]
intro!: *exI*[*of - N*] *exI*[*of - D*] *exI*[*of - M*]
intro!:)
apply (*rule-tac x=W'* **in** *exI*)
apply (*auto simp*: *eq correct-watching-init-correct-watching dist*)
apply (*rule-tac x=W'* **in** *exI*)
apply (*auto simp*: *eq correct-watching-init-correct-watching dist*)
done
qed

Full Initialisation

definition *rewatch-heur-st-fast* **where**
‹*rewatch-heur-st-fast* = *rewatch-heur-st*›

definition *rewatch-heur-st-fast-pre* **where**
‹*rewatch-heur-st-fast-pre S* =
(($\forall$ *x* $\in$ *set* (*get-vdom-heur-init S*). *x* $\leq$ *sint64-max*) $\wedge$ *length* (*get-clauses-wl-heur-init S*) $\leq$ *sint64-max*)›

definition *init-dt-wl-heur-full*
:: ‹*bool* $\Rightarrow$ - $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-full unb CS S* = *do* {
S $\leftarrow$ *init-dt-wl-heur unb CS S*;
ASSERT($\neg$*is-failed-heur-init S*);
rewatch-heur-st S
}›

definition *init-dt-wl-heur-full-unb*
:: ‹- $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*›
where

⟨*init-dt-wl-heur-full-unb* = *init-dt-wl-heur-full True*⟩

lemma *init-dt-wl-heur-full-init-dt-wl-full*:
assumes
⟨*init-dt-wl-pre CS T*⟩ **and**
⟨∀ *C*∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)⟩ **and**
⟨*distinct-mset-set* (*mset* ‘ *set CS*)⟩ **and**
⟨(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*⟩
shows ⟨*init-dt-wl-heur-full True CS S*
≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*init-dt-wl-full CS T*)⟩
proof −
have *H*: ⟨*valid-arena x1g x1b* (*set x1p*)⟩ ⟨*set x1p* ⊆ *set x1p*⟩ ⟨*set-mset* (*dom-m x1b*) ⊆ *set x1p*⟩
⟨*distinct x1p*⟩ ⟨(*x1j*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩
if
xx': ⟨(*x*, *x'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*⟩ **and**
st: ⟨*x2c* = (*x1e*, *x2d*)⟩
⟨*x2b* = (*x1d*, *x2c*)⟩
⟨*x2a* = (*x1c*, *x2b*)⟩
⟨*x2* = (*x1b*, *x2a*)⟩
⟨*x1* = (*x1a*, *x2*)⟩
⟨*x'* = (*x1*, *x2e*)⟩
⟨*x2o* = (*x1p*, *x2p*)⟩
⟨*x2n* = (*x1o*, *x2o*)⟩
⟨*x2m* = (*x1n*, *x2n*)⟩
⟨*x2l* = (*x1m*, *x2m*)⟩
⟨*x2k* = (*x1l*, *x2l*)⟩
⟨*x2j* = (*x1k*, *x2k*)⟩
⟨*x2i* = (*x1j*, *x2j*)⟩
⟨*x2h* = (*x1i*, *x2i*)⟩
⟨*x2g* = (*x1h*, *x2h*)⟩
⟨*x2f* = (*x1g*, *x2g*)⟩
⟨*x* = (*x1f*, *x2f*)⟩
for *x x' x1 x1a x2 x1b x2a x1c x2b x1d x2c x1e x2d x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p*
proof −
show ⟨*valid-arena x1g x1b* (*set x1p*)⟩ ⟨*set x1p* ⊆ *set x1p*⟩ ⟨*set-mset* (*dom-m x1b*) ⊆ *set x1p*⟩
⟨*distinct x1p*⟩ ⟨(*x1j*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩
using *xx' distinct-mset-dom*[*of x1b*] **unfolding** *st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def empty-watched-def*
simp flip: *set-mset-mset distinct-mset-mset-distinct*)
qed

show *?thesis*
unfolding *init-dt-wl-heur-full-def init-dt-wl-full-def rewatch-heur-st-def*
apply (*refine-rcg rewatch-heur-rewatch*[*of - - - - - -* $\mathcal{A}$]
init-dt-wl-heur-init-dt-wl[*of True* $\mathcal{A}$, *THEN fref-to-Down-curry*])
subgoal using *assms* **by** *fast*
subgoal using *assms* **by** *fast*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*)
subgoal by (*auto dest*: *valid-arena-vdom-subset simp*: *twl-st-heur-parsing-no-WL-def*)
apply ((*rule H*; *assumption*)+)[*5*]
subgoal
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
literals-are-in-$\mathcal{L}_{in}$*-mm-def all-lits-of-mm-union*)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*

empty-watched-def[*symmetric*] *map-fun-rel-def vdom-m-def*)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
empty-watched-def[*symmetric*])
done
qed

lemma *init-dt-wl-heur-full-init-dt-wl-spec-full*:
assumes
‹*init-dt-wl-pre CS T*› **and**
‹$\forall$ *C*$\in$*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)› **and**
‹*distinct-mset-set* (*mset ‘ set CS*)› **and**
‹(*S*, *T*) $\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
shows ‹*init-dt-wl-heur-full True CS S*
$\le$ $\Downarrow$ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*SPEC* (*init-dt-wl-spec-full CS T*))›
apply (*rule order.trans*)
apply (*rule init-dt-wl-heur-full-init-dt-wl-full*[*OF assms*])
apply (*rule ref-two-step'*)
apply (*rule init-dt-wl-full-init-dt-wl-spec-full*[*OF assms*(*1*)])
done

15.1.5 Conversion to normal state

definition *extract-lits-sorted* **where**
‹*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {
vars $\leftarrow$ — insert_sort_nth2 xs vars*RETURN vars*;
RETURN (*vars*, *n*)
})›

definition *lits-with-max-rel* **where**
‹*lits-with-max-rel* = {((*xs*, *n*), $\mathcal{A}_{in}$). *mset xs* = $\mathcal{A}_{in}$ $\wedge$ *n* = *Max* (*insert 0* (*set xs*)) $\wedge$
length xs < *uint32-max*}›

lemma *extract-lits-sorted-mset-set*:
‹(*extract-lits-sorted*, *RETURN o mset-set*)
$\in$ *isasat-atms-ext-rel* $\rightarrow_f$ ⟨*lits-with-max-rel*⟩*nres-rel*›
proof −
have *K*: ‹*RETURN o mset-set* = (λ*v*. *do* {*v'* $\leftarrow$ *SPEC*(λ*v'*. *v'* = *mset-set v*); *RETURN v'*})›
by *auto*
have *K'*: ‹*length x2a* < *uint32-max*› **if** ‹*distinct b*› ‹*init-valid-rep x1* (*set b*)›
‹*length x1* < *uint32-max*› ‹*mset x2a* = *mset b*›**for** *x1 x2a b*
proof −
have ‹*distinct x2a*›
by (*simp add*: *same-mset-distinct-iff that*(*1*) *that*(*4*))
have ‹*length x2a* = *length b*› ‹*set x2a* = *set b*›
using ‹*mset x2a* = *mset b*› **apply** (*metis size-mset*)
using ‹*mset x2a* = *mset b*› **by** (*rule mset-eq-setD*)
then have ‹*set x2a* $\subseteq$ {*0*..<*uint32-max* − *1*}›
using *that* **by** (*auto simp*: *init-valid-rep-def*)
from *card-mono*[*OF - this*] **show** *?thesis*
using ‹*distinct x2a*› **by** (*auto simp*: *uint32-max-def distinct-card*)
qed
have *H-simple*: ‹*RETURN x2a*
$\le$ $\Downarrow$ (*list-mset-rel* $\cap$ {(*v*, *v'*). *length v* < *uint32-max*})
(*SPEC* (λ*v'*. *v'* = *mset-set y*))›

if
⟨$(x, y) \in$ *isasat-atms-ext-rel*⟩ **and**
⟨$x2 = (x1a, x2a)$⟩ **and**
⟨$x = (x1, x2)$⟩
for x :: ⟨*nat list* × *nat* × *nat list*⟩ **and** y :: ⟨*nat set*⟩ **and** $x1$ :: ⟨*nat list*⟩ **and** $x2$ :: ⟨*nat* × *nat list*⟩ **and** $x1a$ :: ⟨*nat*⟩ **and** $x2a$ :: ⟨*nat list*⟩
using *that mset-eq-length* **by** (*auto simp*: *isasat-atms-ext-rel-def list-mset-rel-def br-def mset-set-set RETURN-def intro*: K' *intro*!: *RES-refine dest*: *mset-eq-length*)

show *?thesis*
unfolding *extract-lits-sorted-def reorder-list-def K*
apply (*intro frefI nres-relI*)
apply (*refine-vcg H-simple*)
apply *assumption+*
by (*auto simp*: *lits-with-max-rel-def isasat-atms-ext-rel-def mset-set-set list-mset-rel-def br-def dest*!: *mset-eq-setD*)
qed

TODO Move

The value 160 is random (but larger than the default 16 for array lists).

definition *finalise-init-code* :: ⟨*opts* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*finalise-init-code opts* =
($\lambda(M', N', D', Q', W'$, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*, *cach*, *lbd*, *vdom*, -). *do* {
ASSERT(*lst-As* ≠ *None* ∧ *fst-As* ≠ *None*);
let init-stats = (*0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*, *0*::*64 word*);
let fema = *ema-fast-init*;
let sema = *ema-slow-init*;
let ccount = *restart-info-init*;
let lcount = *0*;
RETURN (M', N', D', Q', W', ((*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*), *to-remove*), *clvls*, *cach*, *lbd*, *take 1*(*replicate 160* (*Pos 0*)), *init-stats*, (*fema*, *sema*, *ccount*, *0*, φ, *0*, *replicate* (*length* φ) *False*, *0*, *replicate* (*length* φ) *False*, *10000*, *1000*, *1*), *vdom*, [], *lcount*, *opts*, [])
})⟩

lemma *isa-vmtf-init-nemptyD*: ⟨((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*) ∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ $\mathcal{A}$ ≠ {#} ⟹ ∃ *y*. *an* = *Some y*⟩
⟨((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*) ∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ $\mathcal{A}$ ≠ {#} ⟹ ∃ *y*. *am* = *Some y*⟩
by (*auto simp*: *isa-vmtf-init-def vmtf-init-def*)

lemma *isa-vmtf-init-isa-vmtf*: ⟨$\mathcal{A}$ ≠ {#} ⟹ ((*ak*, *al*, *Some am*, *Some an*, *bc*), *ao*, *bd*) ∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ ((*ak*, *al*, *am*, *an*, *bc*), *ao*, *bd*) ∈ *isa-vmtf* $\mathcal{A}$ *au*⟩
by (*auto simp*: *isa-vmtf-init-def vmtf-init-def Image-iff intro*!: *isa-vmtfI*)

lemma *heuristic-rel-initI*:
⟨*phase-saving* $\mathcal{A}$ φ ⟹ *length* φ' = *length* φ ⟹ *length* φ'' = *length* φ ⟹ *heuristic-rel* $\mathcal{A}$ (*fema*, *sema*, *ccount*, *0*, (φ,*a*, φ',*b*,φ'',*c*,*d*))⟩
by (*auto simp*: *heuristic-rel-def phase-save-heur-rel-def phase-saving-def*)

lemma *finalise-init-finalise-init-full*:
⟨*get-conflict-wl S* = *None* ⟹

all-atms-st S ≠ {#} ⟹ *size* (*learned-clss-l* (*get-clauses-wl S*)) = *0* ⟹
((*ops'*, *T*), *ops*, *S*) ∈ *Id* $\times_f$ *twl-st-heur-post-parsing-wl True* ⟹
finalise-init-code ops' T ≤ ⇓ {(*S'*, *T'*). (*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init T = *get-clauses-wl-heur S'*} (*RETURN* (*finalise-init S*))›
apply (*cases S*; *cases T*)
apply (*simp add*: *finalise-init-code-def*)
apply (*auto simp*: *finalise-init-def twl-st-heur-def twl-st-heur-parsing-no-WL-def*
twl-st-heur-parsing-no-WL-wl-def
finalise-init-code-def out-learned-def all-atms-def
twl-st-heur-post-parsing-wl-def
intro!: *ASSERT-leI intro*!: *isa-vmtf-init-isa-vmtf heuristic-rel-initI*
dest: *isa-vmtf-init-nemptyD*)
done

lemma *finalise-init-finalise-init*:
‹(*uncurry finalise-init-code*, *uncurry* (*RETURN oo* (λ-. *finalise-init*))) ∈
[λ(-, *S*::*nat twl-st-wl*). *get-conflict-wl S* = *None* ∧ *all-atms-st S* ≠ {#} ∧
size (*learned-clss-l* (*get-clauses-wl S*)) = *0*]$_f$ *Id* $\times_r$
twl-st-heur-post-parsing-wl True → ⟨*twl-st-heur*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *finalise-init-finalise-init-full*[*of* ‹*snd y*› ‹*fst x*› ‹*snd x*› ‹*fst y*›]
by (*cases x*; *cases y*)
(*auto intro*: *weaken-⇓'*)
done

definition (**in** −) *init-rll* :: ‹*nat* ⇒ (*nat*, *'v clause-l* × *bool*) *fmap*› **where**
‹*init-rll n* = *fmempty*›

definition (**in** −) *init-aa* :: ‹*nat* ⇒ *'v list*› **where**
‹*init-aa n* = []›

definition (**in** −) *init-aa'* :: ‹*nat* ⇒ (*clause-status* × *nat* × *nat*) *list*› **where**
‹*init-aa' n* = []›

definition *init-trail-D* :: ‹*nat list* ⇒ *nat* ⇒ *nat* ⇒ *trail-pol nres*› **where**
‹*init-trail-D* $\mathcal{A}_{in}$ *n m* = *do* {
let M0 = [];
let cs = [];
let M = *replicate m UNSET*;
let M' = *replicate n 0*;
let M'' = *replicate n 1*;
RETURN ((*M0*, *M*, *M'*, *M''*, *0*, *cs*))
}›

definition *init-trail-D-fast* **where**
‹*init-trail-D-fast* = *init-trail-D*›

definition *init-state-wl-D'* :: ‹*nat list* × *nat* ⇒ (*trail-pol* × - × -) *nres*› **where**
‹*init-state-wl-D'* = (λ($\mathcal{A}_{in}$, *n*). *do* {
ASSERT(*Suc* (*2* ∗ (*n*)) ≤ *uint32-max*);
let n = *Suc* (*n*);
let m = *2* ∗ *n*;

$M \leftarrow$ *init-trail-D* $\mathcal{A}_{in}$ *n m*;
let N = [];
let D = (*True*, *0*, *replicate n NOTIN*);
let WS = *replicate m* [];
vm $\leftarrow$ *initialise-VMTF* $\mathcal{A}_{in}$ *n*;
let φ = *replicate n False*;
let cach = (*replicate n SEEN-UNKNOWN*, []);
let lbd = *empty-lbd*;
let vdom = [];
RETURN (*M*, *N*, *D*, *0*, *WS*, *vm*, φ, *0*, *cach*, *lbd*, *vdom*, *False*)
})›

lemma *init-trail-D-ref*:
‹(*uncurry2 init-trail-D*, *uncurry2* (*RETURN ooo* (λ - - -. []))) ∈ [λ((*N*, *n*), *m*). *mset N* = $\mathcal{A}_{in}$ ∧
distinct N ∧ (∀ *L*∈*set N*. *L* < *n*) ∧ *m* = *2* ∗ *n* ∧ *isasat-input-bounded* $\mathcal{A}_{in}]_f$
⟨*Id*⟩*list-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* →
⟨*trail-pol* $\mathcal{A}_{in}$⟩ *nres-rel*›
proof −
have *K*: ‹(∀ *L*∈*set N*. *L* < *n*) ⟷
(∀ *L* ∈# ($\mathcal{L}_{all}$ (*mset N*)). *atm-of L* < *n*)› **for** *N n*
apply (*rule iffI*)
subgoal by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
subgoal by (*metis* (*full-types*) *image-eqI in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literal.sel*(*1*)
set-image-mset set-mset-mset)
done
have *K'*: ‹(∀ *L*∈*set N*. *L* < *n*) ⟹
(∀ *L* ∈# ($\mathcal{L}_{all}$ (*mset N*)). *nat-of-lit L* < *2* ∗ *n*)›
(**is** ‹*?A* ⟹ *?B*›) **for** *N n*

proof −
assume *?A*
then show *?B*
apply (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
apply (*case-tac L*)
apply *auto*
done
qed
show *?thesis*
unfolding *init-trail-D-def*
apply (*intro frefI nres-relI*)
unfolding *uncurry-def Let-def comp-def trail-pol-def*
apply *clarify*
unfolding *RETURN-refine-iff*
apply *clarify*
apply (*intro conjI*)
subgoal
by (*auto simp*: *ann-lits-split-reasons-def*
list-mset-rel-def Collect-eq-comp list-rel-def
list-all2-op-eq-map-right-iff' Id-def
br-def in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
dest: *multi-member-split*)
subgoal
by *auto*
subgoal using *K'* **by** (*auto simp*: *polarity-def*)
subgoal
by (*auto simp*:

nat-shiftr-div2 in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff
polarity-atm-def trail-pol-def K
phase-saving-def list-rel-mset-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$
list-rel-def Id-def br-def list-all2-op-eq-map-right-iff'
ann-lits-split-reasons-def
list-mset-rel-def Collect-eq-comp)
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*auto simp*: *control-stack.empty*)
subgoal by *auto*
done
qed

definition [*to-relAPP*]: *mset-rel A* $\equiv$ *p2rel* (*rel-mset* (*rel2p A*))
lemma *in-mset-rel-eq-f-iff*:
‹(*a*, *b*) $\in$ $\langle\{(c, a).\ a = f\ c\}\rangle$*mset-rel* $\longleftrightarrow$ *b* = *f* ‘# *a*›
using *ex-mset*[*of a*]
by (*auto simp*: *mset-rel-def br-def rel2p-def*[*abs-def*] *p2rel-def rel-mset-def*
list-all2-op-eq-map-right-iff' cong: *ex-cong*)

lemma *in-mset-rel-eq-f-iff-set*:
‹$\langle\{(c, a).\ a = f\ c\}\rangle$*mset-rel* = {(*b*, *a*). *a* = *f* ‘# *b*}›
using *in-mset-rel-eq-f-iff*[*of - - f*] **by** *blast*

lemma *init-state-wl-D0*:
‹(*init-state-wl-D'*, *init-state-wl-heur*) $\in$
[$\lambda N.\ N = \mathcal{A}_{in} \wedge$ *distinct-mset* $\mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}_{in}]_f$
lits-with-max-rel O $\langle Id\rangle$*mset-rel* $\rightarrow$
$\langle Id \times_r Id \times_r$
$Id \times_r$ *nat-rel* $\times_r \langle\langle Id\rangle$*list-rel*$\rangle$*list-rel* $\times_r$
$Id \times_r \langle$*bool-rel*$\rangle$*list-rel* $\times_r Id \times_r Id \times_r Id\rangle$*nres-rel*›
(**is** ‹*?C* $\in$ [*?Pre*]$_f$ *?arg* $\rightarrow$ $\langle$*?im*$\rangle$*nres-rel*›)
proof −
have *init-state-wl-heur-alt-def*: ‹*init-state-wl-heur* $\mathcal{A}_{in}$ = *do* {
$M \leftarrow$ *SPEC* ($\lambda M.$ (*M*, []) $\in$ *trail-pol* $\mathcal{A}_{in}$);
$N \leftarrow$ *RETURN* [];
$D \leftarrow$ *SPEC* ($\lambda D.$ (*D*, *None*) $\in$ *option-lookup-clause-rel* $\mathcal{A}_{in}$);
$W \leftarrow$ *SPEC* ($\lambda W.$ (*W*, *empty-watched* $\mathcal{A}_{in}$) $\in$ $\langle Id\rangle$*map-fun-rel* ($D_0\ \mathcal{A}_{in}$));
vm $\leftarrow$ *RES* (*isa-vmtf-init* $\mathcal{A}_{in}$ []);
$\varphi \leftarrow$ *SPEC* (*phase-saving* $\mathcal{A}_{in}$);
cach $\leftarrow$ *SPEC* (*cach-refinement-empty* $\mathcal{A}_{in}$);
let lbd = *empty-lbd*;
let vdom = [];
RETURN (*M*, *N*, *D*, *0*, *W*, *vm*, φ, *0*, *cach*, *lbd*, *vdom*, *False*)}› **for** $\mathcal{A}_{in}$
unfolding *init-state-wl-heur-def Let-def* **by** *auto*

have *tr*: ‹*distinct-mset* $\mathcal{A}_{in} \wedge (\forall L \in \# \mathcal{A}_{in}.\ L < b) \Longrightarrow$
$(\mathcal{A}_{in}', \mathcal{A}_{in}) \in \langle Id\rangle$*list-rel-mset-rel* $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}_{in} \Longrightarrow$
$b' = 2 * b \Longrightarrow$
init-trail-D $\mathcal{A}_{in}'$ *b* (*2* $*$ *b*) $\leq \Downarrow$ (*trail-pol* $\mathcal{A}_{in}$) (*RETURN* [])› **for** *b' b* $\mathcal{A}_{in}$ $\mathcal{A}_{in}'$ *x*
by (*rule init-trail-D-ref*[*unfolded fref-def nres-rel-def*, *simplified*, *rule-format*])

(*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def*)

have [*simp*]: ‹*comp-fun-idem* (*max* :: *'a* :: {*zero,linorder*} ⇒ -)›
unfolding *comp-fun-idem-def comp-fun-commute-def comp-fun-idem-axioms-def*
by (*auto simp*: *max-def*[*abs-def*] *intro*!: *ext*)
have [*simp*]: ‹*fold max x a* = *Max* (*insert a* (*set x*))› **for** *x* **and** *a* :: ‹*'a* :: {*zero,linorder*}›
by (*auto simp*: *Max.eq-fold comp-fun-idem.fold-set-fold*)
have *in-N0*: ‹$L \in set\ \mathcal{A}_{in} \Longrightarrow L < Suc\ ((Max\ (insert\ 0\ (set\ \mathcal{A}_{in}))))$›
for $L\ \mathcal{A}_{in}$
using *Max-ge*[*of* ‹*insert 0* (*set* $\mathcal{A}_{in}$)› *L*]
by (*auto simp del*: *Max-ge simp*: *nat-shiftr-div2*)
define *P* **where** ‹$P\ x = \{(a, b).\ b = [] \wedge (a, b) \in trail\text{-}pol\ x\}$› **for** *x*
have *P*: ‹$(c, []) \in P\ x \longleftrightarrow (c, []) \in trail\text{-}pol\ x$› **for** *c x*
unfolding *P-def* **by** *auto*
have [*simp*]: ‹$\{p.\ \exists x.\ p = (x, x)\} = \{(y, x).\ x = y\}$›
by *auto*
have [*simp*]: ‹$\bigwedge a\ \mathcal{A}_{in}.\ (a, \mathcal{A}_{in}) \in \langle nat\text{-}rel\rangle mset\text{-}rel \longleftrightarrow \mathcal{A}_{in} = a$›
by (*auto simp*: *Id-def br-def in-mset-rel-eq-f-iff list-rel-mset-rel-def*
in-mset-rel-eq-f-iff)

have [*simp*]: ‹$(a, mset\ a) \in \langle Id\rangle list\text{-}rel\text{-}mset\text{-}rel$› **for** *a*
unfolding *list-rel-mset-rel-def*
by (*rule relcompI* [*of* - ‹*a*›])
(*auto simp*: *list-rel-def Id-def br-def list-all2-op-eq-map-right-iff'*
list-mset-rel-def)
have *init*: ‹*init-trail-D x1* (*Suc* (*x2*))
(*2* ∗ *Suc* (*x2*)) ≤
$SPEC\ (\lambda c.\ (c, []) \in trail\text{-}pol\ \mathcal{A}_{in})$›
if ‹*distinct-mset* $\mathcal{A}_{in}$› **and** *x*: ‹$(\mathcal{A}_{in}', \mathcal{A}_{in}) \in ?arg$› **and**
‹$\mathcal{A}_{in}' = (x1, x2)$› **and** ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
for $\mathcal{A}_{in}\ \mathcal{A}_{in}'\ x1\ x2$
unfolding *x P*
by (*rule tr*[*unfolded conc-fun-RETURN*])
(*use that* **in** ‹*auto simp*: *lits-with-max-rel-def dest*: *in-N0*›)

have *H*:
‹(*replicate* (*2* ∗ *Suc* (*b*)) [], *empty-watched* $\mathcal{A}_{in}$)
$\in \langle Id\rangle map\text{-}fun\text{-}rel\ ((\lambda L.\ (nat\text{-}of\text{-}lit\ L, L))$ ‘ *set-mset* ($\mathcal{L}_{all}\ \mathcal{A}_{in}$))›
if ‹$(x, \mathcal{A}_{in}) \in ?arg$› **and**
‹$x = (a, b)$›
for $\mathcal{A}_{in}$ *x a b*
using *that* **unfolding** *map-fun-rel-def*
by (*auto simp*: *empty-watched-def* $\mathcal{L}_{all}$*-def*
lits-with-max-rel-def
intro!: *nth-replicate dest*!: *in-N0*
simp del: *replicate.simps*)
have *initialise-VMTF*: ‹(∀ *L*∈#*aa*. *L* < *b*) ∧ *distinct-mset aa* ∧ (*a*, *aa*) ∈
⟨*Id*⟩*list-rel-mset-rel* ∧ *size aa* < *uint32-max* ⟹
initialise-VMTF a b ≤ *RES* (*isa-vmtf-init aa* [])›
for *aa b a*
using *initialise-VMTF*[*of aa*, *THEN fref-to-Down-curry*, *of aa b a b*]
by (*auto simp*: *isa-vmtf-init-def conc-fun-RES*)
have [*simp*]: ‹$(x, y) \in \langle Id\rangle list\text{-}rel\text{-}mset\text{-}rel \Longrightarrow L \in\# y \Longrightarrow$
$L < Suc\ ((Max\ (insert\ 0\ (set\ x))))$›
for *x y L*
by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def Id-def*

list-all2-op-eq-map-right-iff′ list-mset-rel-def dest: *in-N0*)

have *initialise-VMTF*: ‹*initialise-VMTF a* (*Suc* (*b*)) ≤
⇓ *Id* (*RES* (*isa-vmtf-init y* []))›
if ‹(*x*, *y*) ∈ *?arg*› **and** ‹*distinct-mset y*› **and** ‹*length a* < *uint32-max*› **and** ‹*x* = (*a*, *b*)› **for** *x y a b*
using *that*
by (*auto simp*: *P-def lits-with-max-rel-def intro*!: *initialise-VMTF in-N0*)
have *K*[*simp*]: ‹(*x*, $\mathcal{A}_{in}$) ∈ ⟨*Id*⟩*list-rel-mset-rel* ⟹
L ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹ *L* < *Suc* ((*Max* (*insert 0* (*set x*))))›
for *x L* $\mathcal{A}_{in}$
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def Id-def*
list-all2-op-eq-map-right-iff′ list-mset-rel-def)
have *cach*: ‹*RETURN* (*replicate* (*Suc* (*b*)) *SEEN-UNKNOWN*, [])
≤ ⇓ *Id*
(*SPEC* (*cach-refinement-empty y*))›
if
‹*y* = $\mathcal{A}_{in}$ ∧ *distinct-mset* $\mathcal{A}_{in}$› **and**
‹(*x*, *y*) ∈ *?arg*› **and**
‹*x* = (*a*, *b*)›
for *M W vm vma φ x y a b*
proof −
show *?thesis*
unfolding *cach-refinement-empty-def RETURN-RES-refine-iff*
cach-refinement-alt-def Bex-def
by (*rule exI*[*of* - ‹(*replicate* (*Suc* (*b*)) *SEEN-UNKNOWN*, [])›]) (*use that* **in**
‹*auto simp*: *map-fun-rel-def empty-watched-def* $\mathcal{L}_{all}$*-def*
list-mset-rel-def lits-with-max-rel-def
simp del: *replicate-Suc*
dest!: *in-N0 intro*: *K*›)
qed
have *conflict*: ‹*RETURN* (*True*, *0*, *replicate* (*Suc* (*b*)) *NOTIN*)
≤ *SPEC* (λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}_{in}$)›
if
‹*y* = $\mathcal{A}_{in}$ ∧ *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$› **and**
‹((*a*, *b*), $\mathcal{A}_{in}$) ∈ *lits-with-max-rel O* ⟨*Id*⟩*mset-rel*› **and**
‹*x* = (*a*, *b*)›
for *a b x y*
proof −
have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹
L < *Suc* (*b*)› **for** *L*
using *that in-N0* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
lits-with-max-rel-def)
then show *?thesis*
by (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def simp del: *replicate-Suc*
intro: *mset-as-position.intros*)
qed
have [*simp*]:
‹*NO-MATCH 0 a1* ⟹ *max 0* (*Max* (*insert a1* (*set a2*))) = *max a1* (*Max* (*insert 0* (*set a2*)))›
for *a1* :: *nat* **and** *a2*
by (*metis* (*mono-tags*, *lifting*) *List.finite-set Max-insert all-not-in-conv finite-insert insertI1 insert-commute*)
have *le-uint32*: ‹∀ *L*∈#$\mathcal{L}_{all}$ (*mset a*). *nat-of-lit L* ≤ *uint32-max* ⟹
Suc (*2* ∗ (*Max* (*insert 0* (*set a*)))) ≤ *uint32-max*› **for** *a*
apply (*induction a*)
apply (*auto simp*: *uint32-max-def*)

apply (*auto simp*: *max-def* $\mathcal{L}_{all}$-*add-mset*)
done

show *?thesis*
apply (*intro frefI nres-relI*)
subgoal for *x y*
unfolding *init-state-wl-heur-alt-def init-state-wl-D'-def*
apply (*rewrite* **in** ‹*let - = Suc -in -*› *Let-def*)
apply (*rewrite* **in** ‹*let - = 2 ∗ -in -*› *Let-def*)
apply (*cases x*; *simp only*: *prod.case*)
apply (*refine-rcg init*[*of y x*] *initialise-VMTF cach*)
subgoal for *a b* **by** (*auto simp*: *lits-with-max-rel-def intro*: *le-uint32*)
subgoal by (*auto intro*!: *K*[*of -* $\mathcal{A}_{in}$] *simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
lits-with-max-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict*)
subgoal by (*rule RETURN-rule*) (*rule H*; *simp only*:)
apply *assumption*
subgoal by *fast*
subgoal by (*auto simp*: *lits-with-max-rel-def P-def*)
subgoal by *simp*
subgoal unfolding *phase-saving-def lits-with-max-rel-def* **by** (*auto intro*!: *K*)
subgoal by *fast*
subgoal by *fast*
apply *assumption*
apply (*rule refl*)
subgoal by (*auto simp*: *P-def init-rll-def option-lookup-clause-rel-def*
lookup-clause-rel-def lits-with-max-rel-def
simp del: *replicate.simps*
intro!: *mset-as-position.intros K*)
done
done
qed

lemma *init-state-wl-D'*:
‹(*init-state-wl-D'*, *init-state-wl-heur*) ∈
[$\lambda\mathcal{A}_{in}$. *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$
lits-with-max-rel O ⟨*Id*⟩*mset-rel* →
⟨*Id* $\times_r$ *Id* $\times_r$
Id $\times_r$ *nat-rel* $\times_r$ ⟨⟨*Id*⟩*list-rel*⟩*list-rel* $\times_r$
Id $\times_r$ ⟨*bool-rel*⟩*list-rel* $\times_r$ *Id* $\times_r$ *Id* $\times_r$ *Id* $\times_r$ *Id*⟩*nres-rel*›
apply −
apply (*intro frefI nres-relI*)
by (*rule init-state-wl-D0*[*THEN fref-to-Down*, *THEN order-trans*]) *auto*

lemma *init-state-wl-heur-init-state-wl'*:
‹(*init-state-wl-heur*, *RETURN o* (λ-. *init-state-wl*))
∈ [λN. $N = \mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *Id* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}_{in}$ *True*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *comp-def*
using *init-state-wl-heur-init-state-wl*[*THEN fref-to-Down*, *of* $\mathcal{A}_{in}$ ‹()› ‹()›]
by *auto*

lemma *all-blits-are-in-problem-init-blits-in*: ‹*all-blits-are-in-problem-init S* $\Longrightarrow$ *blits-in-*$\mathcal{L}_{in}$ *S*›
unfolding *blits-in-*$\mathcal{L}_{in}$*-def*
by (*cases S*)
(*auto simp*: *all-blits-are-in-problem-init.simps ac-simps*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-lits-def*)

lemma *correct-watching-init-blits-in-*$\mathcal{L}_{in}$:
assumes ‹*correct-watching-init S*›
shows ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
proof −
show *?thesis*
using *assms*
by (*cases S*)
(*auto simp*: *all-blits-are-in-problem-init-blits-in*
correct-watching-init.simps)
qed

fun *append-empty-watched* **where**
‹*append-empty-watched* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, (λ-. [])), *OC*)›

fun *remove-watched* :: ‹*'v twl-st-wl-init-full* ⇒ *'v twl-st-wl-init*› **where**
‹*remove-watched* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, -), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*), *OC*)›

definition *init-dt-wl'* :: ‹*'v clause-l list* ⇒ *'v twl-st-wl-init* ⇒ *'v twl-st-wl-init-full nres*› **where**
‹*init-dt-wl' CS S* = *do*{
S ← *init-dt-wl CS S*;
RETURN (*append-empty-watched S*)
}›

lemma *init-dt-wl'-spec*: ‹*init-dt-wl-pre CS S* $\Longrightarrow$ *init-dt-wl' CS S* ≤ ⇓
({(*S* :: *'v twl-st-wl-init-full*, *S'* :: *'v twl-st-wl-init*).
remove-watched S = *S'*}) (*SPEC* (*init-dt-wl-spec CS S*))›
unfolding *init-dt-wl'-def*
by (*refine-vcg bind-refine-spec*[*OF - init-dt-wl-init-dt-wl-spec*])
(*auto intro*!: *RETURN-RES-refine*)

lemma *init-dt-wl'-init-dt*:
‹*init-dt-wl-pre CS S* $\Longrightarrow$ (*S*, *S'*) ∈ *state-wl-l-init* $\Longrightarrow$ ∀ *C*∈*set CS*. *distinct C* $\Longrightarrow$
init-dt-wl' CS S ≤ ⇓
({(*S* :: *'v twl-st-wl-init-full*, *S'* :: *'v twl-st-wl-init*).
remove-watched S = *S'*} *O state-wl-l-init*) (*init-dt CS S'*)›
unfolding *init-dt-wl'-def*
apply (*refine-vcg bind-refine*[*of - - - - -* ‹*RETURN*›, *OF init-dt-wl-init-dt*, *simplified*])
subgoal for *S T*
by (*cases S*; *cases T*)
auto
done

definition *isasat-init-fast-slow* :: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*isasat-init-fast-slow* =
(λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, φ, *clvls*, *cach*, *lbd*, *vdom*, *failed*).
RETURN (*trail-pol-slow-of-fast M'*, *N'*, *D'*, *j*, *convert-wlists-to-nat-conv W'*, *vm*, φ,

clvls, *cach*, *lbd*, *vdom*, *failed*))›

lemma *isasat-init-fast-slow-alt-def*:
‹*isasat-init-fast-slow S* = *RETURN S*›
unfolding *isasat-init-fast-slow-def trail-pol-slow-of-fast-alt-def*
convert-wlists-to-nat-conv-def
by *auto*

end
theory *IsaSAT-Initialisation-LLVM*
imports *IsaSAT-Setup-LLVM IsaSAT-VMTF-LLVM Watched-Literals.Watched-Literals-Watch-List-Initialisation*
Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Initialisation
begin

abbreviation *unat-rel32* :: (*32 word* × *nat*) *set* **where** *unat-rel32* ≡ *unat-rel*
abbreviation *unat-rel64* :: (*64 word* × *nat*) *set* **where** *unat-rel64* ≡ *unat-rel*
abbreviation *snat-rel32* :: (*32 word* × *nat*) *set* **where** *snat-rel32* ≡ *snat-rel*
abbreviation *snat-rel64* :: (*64 word* × *nat*) *set* **where** *snat-rel64* ≡ *snat-rel*

type-synonym (**in** −)*vmtf-assn-option-fst-As* =
‹*vmtf-node-assn ptr* × *64 word* × *32 word* × *32 word* × *32 word*›

type-synonym (**in** −)*vmtf-remove-assn-option-fst-As* =
‹*vmtf-assn-option-fst-As* × (*32 word array-list64*) × *1 word ptr*›

abbreviation (**in** −) *vmtf-conc-option-fst-As* :: ‹- ⇒ - ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*vmtf-conc-option-fst-As* ≡ (*array-assn vmtf-node-assn* $\times_a$ *uint64-nat-assn* $\times_a$
atom.option-assn $\times_a$ *atom.option-assn* $\times_a$ *atom.option-assn*)›

abbreviation *vmtf-remove-conc-option-fst-As*
:: ‹*isa-vmtf-remove-int-option-fst-As* ⇒ *vmtf-remove-assn-option-fst-As* ⇒ *assn*›
where
‹*vmtf-remove-conc-option-fst-As* ≡ *vmtf-conc-option-fst-As* $\times_a$ *distinct-atoms-assn*›

sepref-register *atoms-hash-empty*
sepref-def (**in** −) *atoms-hash-empty-code*
is ‹*atoms-hash-int-empty*›
:: ‹*sint32-nat-assn*^{k} $\rightarrow_a$ *atoms-hash-assn*›
unfolding *atoms-hash-int-empty-def array-fold-custom-replicate*
by *sepref*

sepref-def *distinct-atms-empty-code*
is ‹*distinct-atms-int-empty*›
:: ‹*sint64-nat-assn*^{k} $\rightarrow_a$ *distinct-atoms-assn*›
unfolding *distinct-atms-int-empty-def array-fold-custom-replicate*
al-fold-custom-empty[**where** *'l*=*64*]
by *sepref*

lemmas [*sepref-fr-rules*] = *distinct-atms-empty-code.refine atoms-hash-empty-code.refine*

type-synonym (**in** −)*twl-st-wll-trail-init* =
‹*trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × *vmtf-remove-assn-option-fst-As* × *phase-saver-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *vdom-fast-assn* × *1 word*›

definition *isasat-init-assn*
:: ‹*twl-st-wl-heur-init* ⇒ *trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × - × *phase-saver-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *vdom-fast-assn* × *1 word* ⇒ *assn*›
where
‹*isasat-init-assn* =
trail-pol-fast-assn $\times_a$ *arena-fast-assn* $\times_a$
conflict-option-rel-assn $\times_a$
sint64-nat-assn $\times_a$
watchlist-fast-assn $\times_a$
vmtf-remove-conc-option-fst-As $\times_a$ *phase-saver-assn* $\times_a$
uint32-nat-assn $\times_a$
cach-refinement-l-assn $\times_a$
lbd-assn $\times_a$
vdom-fast-assn $\times_a$
bool1-assn›

sepref-def *initialise-VMTF-code*
is ‹*uncurry initialise-VMTF*›
:: ‹$[\lambda(N, n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow vmtf\text{-}remove\text{-}conc\text{-}option\text{-}fst\text{-}As$›
unfolding *initialise-VMTF-def vmtf-cons-def Suc-eq-plus1 atom.fold-option length-uint32-nat-def option.case-eq-if*
apply (*rewrite* **in** ‹*let* - = ⌑ *in* - › *array-fold-custom-replicate op-list-replicate-def*[*symmetric*])
apply (*rewrite at 0* **in** ‹*VMTF-Node* ⌑› *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹*VMTF-Node* (⌑ + *1*)› *annot-snat-unat-conv*)
apply (*rewrite at 1* **in** ‹*VMTF-Node* ⌑› *unat-const-fold*[**where** *'a=64*])
apply (*annot-snat-const TYPE*(*64*))
apply (*rewrite* **in** ‹*list-update* - - -› *annot-index-of-atm*)
apply (*rewrite* **in** ‹*if* - *then* - *else list-update* - - -› *annot-index-of-atm*)
apply (*rewrite at* ‹⌑› **in** ‹- ! *atom.the* -› *annot-index-of-atm*)+
apply (*rewrite at* ‹*RETURN* ((-, ⌑, -), -)› *annot-snat-unat-conv*)
supply [[*goals-limit* = *1*]]
by *sepref*

declare *initialise-VMTF-code.refine*[*sepref-fr-rules*]
sepref-register *cons-trail-Propagated-tr*
sepref-def *propagate-unit-cls-code*
is ‹*uncurry* (*propagate-unit-cls-heur*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]] *DECISION-REASON-def*[*simp*]
unfolding *propagate-unit-cls-heur-def isasat-init-assn-def PR-CONST-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare *propagate-unit-cls-code.refine*[*sepref-fr-rules*]

definition *already-propagated-unit-cls-heur'* **where**
‹*already-propagated-unit-cls-heur'* = (λ(*M*, *N*, *D*, *Q*, *oth*).
RETURN (*M*, *N*, *D*, *Q*, *oth*))›

lemma *already-propagated-unit-cls-heur'-alt*:
‹*already-propagated-unit-cls-heur L* = *already-propagated-unit-cls-heur'*›
unfolding *already-propagated-unit-cls-heur-def already-propagated-unit-cls-heur'-def*
by *auto*

sepref-def *already-propagated-unit-cls-code*
is ‹*already-propagated-unit-cls-heur'*›
:: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-heur'-def isasat-init-assn-def*
PR-CONST-def
by *sepref*

declare *already-propagated-unit-cls-code.refine*[*sepref-fr-rules*]

sepref-def *set-conflict-unit-code*
is ‹*uncurry set-conflict-unit-heur*›
:: ‹$[\lambda(L, (b, n, xs)).$ *atm-of* $L <$ *length* $xs]_a$
unat-lit-assn^k $*_a$ *conflict-option-rel-assn*^{d} $\rightarrow$ *conflict-option-rel-assn*›
supply [[*goals-limit=1*]]
unfolding *set-conflict-unit-heur-def ISIN-def*[*symmetric*] *conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

declare *set-conflict-unit-code.refine*[*sepref-fr-rules*]

sepref-def *conflict-propagated-unit-cls-code*
is ‹*uncurry* (*conflict-propagated-unit-cls-heur*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *conflict-propagated-unit-cls-heur-def isasat-init-assn-def*
PR-CONST-def
by *sepref*

declare *conflict-propagated-unit-cls-code.refine*[*sepref-fr-rules*]

sepref-register *fm-add-new*

lemma *add-init-cls-code-bI*:
assumes
‹*length at'* $\leq$ *Suc* (*Suc uint32-max*)› **and**
‹*2* $\leq$ *length at'*› **and**
‹*length a1'j* $\leq$ *length a1'a*› **and**
‹*length a1'a* $\leq$ *sint64-max* $-$ *length at'* $-$ *5*›
shows ‹*append-and-length-fast-code-pre* ((*True*, *at'*), *a1'a*)› ‹*5* $\leq$ *sint64-max* $-$ *length at'*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *uint64-max-def uint32-max-def sint64-max-def*)

lemma *add-init-cls-code-bI2*:
assumes
‹*length at'* $\leq$ *Suc* (*Suc uint32-max*)›
shows ‹*5* $\leq$ *sint64-max* $-$ *length at'*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *uint64-max-def uint32-max-def sint64-max-def*)

lemma *add-init-clss-codebI*:
  assumes
    ⟨*length at′* ≤ *Suc* (*Suc uint32-max*)⟩ **and**
    ⟨*2* ≤ *length at′*⟩ **and**
    ⟨*length a1′j* ≤ *length a1′a*⟩ **and**
    ⟨*length a1′a* ≤ *uint64-max* − (*length at′* + *5*)⟩
  shows ⟨*length a1′j* < *uint64-max*⟩
  using *assms* **by** (*auto simp*: *uint64-max-def uint32-max-def*)

abbreviation *clauses-ll-assn* **where**
  ⟨*clauses-ll-assn* ≡ *aal-assn′ TYPE*(*64*) *TYPE*(*64*) *unat-lit-assn*⟩

definition *fm-add-new-fast′* **where**
  ⟨*fm-add-new-fast′ b C i* = *fm-add-new-fast b* (*C*!*i*)⟩

lemma *op-list-list-llen-alt-def*: ⟨*op-list-list-llen xss i* = *length* (*xss* ! *i*)⟩
  unfolding *op-list-list-llen-def*
  by *auto*

lemma *op-list-list-idx-alt-def*: ⟨*op-list-list-idx xs i j* = *xs* ! *i* ! *j*⟩
  unfolding *op-list-list-idx-def* **..**

sepref-def *append-and-length-fast-code*
  is ⟨*uncurry3 fm-add-new-fast′*⟩
  :: ⟨$[\lambda(((b,\ C),\ i),\ N).\ i < length\ C \wedge append\text{-}and\text{-}length\text{-}fast\text{-}code\text{-}pre\ ((b,\ C!i),\ N)]_a$
    $bool1\text{-}assn^k *_a clauses\text{-}ll\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a (arena\text{-}fast\text{-}assn)^d \rightarrow$
      $arena\text{-}fast\text{-}assn \times_a sint64\text{-}nat\text{-}assn$⟩
  supply [[*goals-limit*=*1*]]
  supply [*simp*] = *fm-add-new-bounds1*[*simplified*]
  supply [*split*] = *if-splits*
  unfolding *fm-add-new-fast-def fm-add-new-def append-and-length-fast-code-pre-def*
    fm-add-new-fast′-def op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
    is-short-clause-def header-size-def
  apply (*rewrite at AActivity* ⌑ *unat-const-fold*[**where** ′*a*=*32*])+
  apply (*rewrite at APos* ⌑ *unat-const-fold*[**where** ′*a*=*32*])+
  apply (*rewrite at op-list-list-llen - - −* *2 annot-snat-unat-downcast*[**where** ′*l*=*32*])
  apply (*annot-snat-const TYPE*(*64*))
  by *sepref*

sepref-register *fm-add-new-fast′*

sepref-def *add-init-cls-code-b*
  is ⟨*uncurry2 add-init-cls-heur-b′*⟩
  :: ⟨$[\lambda((xs,\ i),\ S).\ i < length\ xs]_a$
    $(clauses\text{-}ll\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow isasat\text{-}init\text{-}assn$⟩
  supply [[*goals-limit*=*1*]] *append-ll-def*[*simp*]*add-init-clss-codebI*[*intro*]
    add-init-cls-code-bI[*intro*] *add-init-cls-code-bI2*[*intro*]
  unfolding *add-init-cls-heur-def add-init-cls-heur-b-def*
  PR-CONST-def
  Let-def length-uint64-nat-def add-init-cls-heur-b′-def
  op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
  unfolding *isasat-init-assn-def*
    nth-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
    delete-index-and-swap-ll-def[*symmetric*]
    append-ll-def[*symmetric*] *fm-add-new-fast-def*[*symmetric*]
  fm-add-new-fast′-def[*symmetric*]

apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare
add-init-cls-code-b.refine[*sepref-fr-rules*]

sepref-def *already-propagated-unit-cls-conflict-code*
is ⟨*uncurry already-propagated-unit-cls-conflict-heur*⟩
:: ⟨$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-conflict-heur-def isasat-init-assn-def*
PR-CONST-def
by *sepref*

declare *already-propagated-unit-cls-conflict-code.refine*[*sepref-fr-rules*]

sepref-def (**in** −) *set-conflict-empty-code*
is ⟨*RETURN o lookup-set-conflict-empty*⟩
:: ⟨$conflict\text{-}option\text{-}rel\text{-}assn^d \rightarrow_a conflict\text{-}option\text{-}rel\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *lookup-set-conflict-empty-def conflict-option-rel-assn-def*
by *sepref*

declare *set-conflict-empty-code.refine*[*sepref-fr-rules*]

sepref-def *set-empty-clause-as-conflict-code*
is ⟨*set-empty-clause-as-conflict-heur*⟩
:: ⟨$isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *set-empty-clause-as-conflict-heur-def isasat-init-assn-def*
conflict-option-rel-assn-def lookup-clause-rel-assn-def
by *sepref*

declare *set-empty-clause-as-conflict-code.refine*[*sepref-fr-rules*]

definition (**in** −) *add-clause-to-others-heur′*
:: ⟨*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*⟩ **where**
⟨*add-clause-to-others-heur′* = (λ (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*).
RETURN (*M*, *N*, *D*, *Q*, *NS*, *US*, *WS*))⟩

lemma *add-clause-to-others-heur′-alt*: ⟨*add-clause-to-others-heur L* = *add-clause-to-others-heur′*⟩
unfolding *add-clause-to-others-heur′-def add-clause-to-others-heur-def*
..
sepref-def *add-clause-to-others-code*
is ⟨*add-clause-to-others-heur′*⟩
:: ⟨$isasat\text{-}init\text{-}assn^d \rightarrow_a isasat\text{-}init\text{-}assn$⟩
supply [[*goals-limit=1*]]
unfolding *add-clause-to-others-heur-def isasat-init-assn-def add-clause-to-others-heur′-def*
by *sepref*

declare *add-clause-to-others-code.refine*[*sepref-fr-rules*]

sepref-def *get-conflict-wl-is-None-init-code*
is ⟨*RETURN o get-conflict-wl-is-None-heur-init*⟩
:: ⟨$isasat\text{-}init\text{-}assn^k \rightarrow_a bool1\text{-}assn$⟩
unfolding *get-conflict-wl-is-None-heur-init-alt-def isasat-init-assn-def length-ll-def*[*symmetric*]

conflict-option-rel-assn-def
supply [[*goals-limit=1*]]
by *sepref*

declare *get-conflict-wl-is-None-init-code.refine*[*sepref-fr-rules*]

sepref-def *polarity-st-heur-init-code*
is ‹*uncurry* (*RETURN oo polarity-st-heur-init*)›
:: ‹$[\lambda(S, L).$ *polarity-pol-pre* (*get-trail-wl-heur-init S*) $L]_a$ *isasat-init-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *tri-bool-assn*›
unfolding *polarity-st-heur-init-def isasat-init-assn-def*
supply [[*goals-limit = 1*]]
by *sepref*

declare *polarity-st-heur-init-code.refine*[*sepref-fr-rules*]

sepref-register *init-dt-step-wl*
get-conflict-wl-is-None-heur-init already-propagated-unit-cls-heur
conflict-propagated-unit-cls-heur add-clause-to-others-heur
add-init-cls-heur set-empty-clause-as-conflict-heur

sepref-register *polarity-st-heur-init propagate-unit-cls-heur*

lemma *is-Nil-length*: ‹*is-Nil xs* $\longleftrightarrow$ *length xs* = *0*›
by (*cases xs*) *auto*

definition *init-dt-step-wl-heur-b'*
:: ‹*nat clause-l list* $\Rightarrow$ *nat* $\Rightarrow$ *twl-st-wl-heur-init* $\Rightarrow$ *twl-st-wl-heur-init nres*› **where**
‹*init-dt-step-wl-heur-b' C i* = *init-dt-step-wl-heur-b* (*C*!*i*)›

sepref-def *init-dt-step-wl-code-b*
is ‹*uncurry2* (*init-dt-step-wl-heur-b'*)›
:: ‹$[\lambda((xs, i), S).\ i < $ *length xs*$]_a$ (*clauses-ll-assn*)^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow$
isasat-init-assn›
supply [[*goals-limit=1*]]
supply *polarity-None-undefined-lit*[*simp*] *polarity-st-init-def*[*simp*]
option.splits[*split*] *get-conflict-wl-is-None-heur-init-alt-def*[*simp*]
tri-bool-eq-def[*simp*]
unfolding *init-dt-step-wl-heur-def PR-CONST-def*
init-dt-step-wl-heur-b-def
init-dt-step-wl-heur-b'-def list-length-1-def is-Nil-length
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
already-propagated-unit-cls-heur'-alt
add-init-cls-heur-b'-def[*symmetric*] *add-clause-to-others-heur'-def*[*symmetric*]
add-clause-to-others-heur'-alt
unfolding *watched-app-def*[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
unfolding *is-Nil-length get-conflict-wl-is-None-init*
polarity-st-heur-init-alt-def[*symmetric*]
get-conflict-wl-is-None-heur-init-alt-def[*symmetric*]
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *UNSET-def*[*symmetric*]
tri-bool-eq-def[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare
init-dt-step-wl-code-b.refine[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-unb*

abbreviation *isasat-atms-ext-rel-assn* **where**
⟨*isasat-atms-ext-rel-assn* ≡ *larray64-assn uint64-nat-assn* $\times_a$ *uint32-nat-assn* $\times_a$
arl64-assn atom-assn⟩

abbreviation *nat-lit-list-hm-assn* **where**
⟨*nat-lit-list-hm-assn* ≡ *hr-comp isasat-atms-ext-rel-assn isasat-atms-ext-rel*⟩

sepref-def *init-next-size-impl*
is ⟨*RETURN o init-next-size*⟩
:: ⟨$[\lambda L.\ L \leq$ *uint32-max div 2*$]_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*⟩
unfolding *init-next-size-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

find-in-thms *op-list-grow-init* **in** *sepref-fr-rules*
sepref-def *nat-lit-lits-init-assn-assn-in*
is ⟨*uncurry add-to-atms-ext*⟩
:: ⟨*atom-assn*^{k} $*_a$ *isasat-atms-ext-rel-assn*^{d} $\rightarrow_a$ *isasat-atms-ext-rel-assn*⟩
supply [[*goals-limit=1*]]
unfolding *add-to-atms-ext-def length-uint32-nat-def*
apply (*rewrite at* ⟨*max* ⌑ -⟩ *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ⟨⌑ < -⟩ *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ⟨*list-grow* - (*init-next-size* ⌑)⟩ *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ⟨*list-grow* - (*init-next-size* ⌑)⟩ *index-of-atm-def*[*symmetric*])
apply (*rewrite at* ⟨⌑ < -⟩ *annot-unat-unat-upcast*[**where** *'l=64*])
unfolding *max-def list-grow-alt*
op-list-grow-init'-alt
apply (*annot-all-atm-idxs*)
apply (*rewrite at* ⟨*op-list-grow-init* ⌑⟩ *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ⟨- < ⌑⟩ *annot-snat-unat-conv*)
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

find-theorems *nfoldli WHILET*
lemma [*sepref-fr-rules*]:
⟨(*uncurry nat-lit-lits-init-assn-assn-in*, *uncurry* (*RETURN* ∘∘ *op-set-insert*))
∈ $[\lambda(a,\ b).\ a \leq$ *uint32-max div 2*$]_a$
atom-assn^k $*_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*⟩
by (*rule nat-lit-lits-init-assn-assn-in.refine*[*FCOMP add-to-atms-ext-op-set-insert*
[*unfolded convert-fref op-set-insert-def*[*symmetric*]]])

lemma *while-nfoldli*:
do {
(-,σ) ← *WHILE*$_T$ (*FOREACH-cond c*) (λx. *do* {*ASSERT* (*FOREACH-cond c x*); *FOREACH-body f x*}) (*l*,σ);
RETURN σ

} ≤ *nfoldli l c f σ*
apply (*induct l arbitrary*: *σ*)
apply (*subst WHILET-unfold*)
apply (*simp add*: *FOREACH-cond-def*)

apply (*subst WHILET-unfold*)
apply (*auto*
simp: *FOREACH-cond-def FOREACH-body-def*
intro: *bind-mono Refine-Basic.bind-mono*(*1*))
done

definition *extract-atms-cls-i′* **where**
⟨*extract-atms-cls-i′ C i* = *extract-atms-cls-i* (*C*!*i*)⟩

lemma *aal-assn-boundsD′*:
assumes *A*: *rdomp* (*aal-assn′ TYPE*(′*l*::*len2*) *TYPE*(′*ll*::*len2*) *A*) *xss* **and** ⟨*i* < *length xss*⟩
shows *length* (*xss* ! *i*) < *max-snat LENGTH*(′*ll*)
using *aal-assn-boundsD-aux1*[*OF A*] *assms*
by *auto*

sepref-def *extract-atms-cls-imp*
is ⟨*uncurry2 extract-atms-cls-i′*⟩
:: ⟨[λ((*N*, *i*), -). *i* < *length N*]$_a$
(*clauses-ll-assn*)^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*⟩
supply [*dest*!] = *aal-assn-boundsD′*
unfolding *extract-atms-cls-i-def extract-atms-cls-i′-def*
apply (*subst nfoldli-by-idx*[*abs-def*])
unfolding *nfoldli-upt-by-while*
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare *extract-atms-cls-imp.refine*[*sepref-fr-rules*]

sepref-def *extract-atms-clss-imp*
is ⟨*uncurry extract-atms-clss-i*⟩
:: ⟨(*clauses-ll-assn*)^k $*_a$ *nat-lit-list-hm-assn*^{d} $\rightarrow_a$ *nat-lit-list-hm-assn*⟩
supply [*dest*] = *aal-assn-boundsD′*
unfolding *extract-atms-clss-i-def*
apply (*subst nfoldli-by-idx*)
unfolding *nfoldli-upt-by-while Let-def extract-atms-cls-i′-def*[*symmetric*]
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
op-list-list-len-def[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
⟨(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* ∘∘ *extract-atms-clss*))
∈ [λ(*a*, *b*). ∀ *C*∈*set a*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*]$_a$
(*clauses-ll-assn*)^k $*_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*⟩
using *extract-atms-clss-imp.refine*[*FCOMP extract-atms-clss-i-extract-atms-clss*[*unfolded convert-fref*]]
by *simp*

sepref-def *extract-atms-clss-imp-empty-assn*

is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *isasat-atms-ext-rel-assn*›
unfolding *extract-atms-clss-imp-empty-rel-def*
larray-fold-custom-replicate
supply [[*goals-limit=1*]]
apply (*rewrite at* ‹(-, -, ⌑)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹(⌑, -, -)› *annotate-assn*[**where** *A*=‹*larray64-assn uint64-nat-assn*›])
apply (*rewrite* **in** ‹(⌑, -, -)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite* **in** ‹(-, ⌑, -)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

lemma *extract-atms-clss-imp-empty-assn*[*sepref-fr-rules*]:
‹(*uncurry0 extract-atms-clss-imp-empty-assn*, *uncurry0* (*RETURN op-extract-list-empty*))
$\in$ $unit\text{-}assn^k \rightarrow_a$ *nat-lit-list-hm-assn*›
using *extract-atms-clss-imp-empty-assn.refine*[*unfolded uncurry0-def*, *FCOMP extract-atms-clss-imp-empty-rel*
[*unfolded convert-fref*]]
unfolding *uncurry0-def*
by *simp*

lemma *extract-atms-clss-imp-empty-rel-alt-def*:
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*op-larray-custom-replicate 1024 0*, *0*, []))›
by (*auto simp*: *extract-atms-clss-imp-empty-rel-def*)

Full Initialisation

sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
$isasat\text{-}init\text{-}assn^d \rightarrow$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-def PR-CONST-def rewatch-heur-st-fast-pre-def*
isasat-init-assn-def rewatch-heur-st-fast-def
by *sepref*

declare
rewatch-heur-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *rewatch-heur-st init-dt-step-wl-heur*

sepref-def *init-dt-wl-heur-code-b*
is ‹*uncurry* (*init-dt-wl-heur-b*)›
:: ‹$(clauses\text{-}ll\text{-}assn)^k *_a isasat\text{-}init\text{-}assn^d \rightarrow_a$
isasat-init-assn›
supply [[*goals-limit=1*]]
unfolding *init-dt-wl-heur-def PR-CONST-def init-dt-step-wl-heur-b-def*[*symmetric*] *if-True*
init-dt-wl-heur-b-def
apply (*subst nfoldli-by-idx*[*abs-def*])
unfolding *nfoldli-upt-by-while op-list-list-len-def*[*symmetric*] *Let-def*
init-dt-step-wl-heur-b'-def[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare
init-dt-wl-heur-code-b.refine[*sepref-fr-rules*]

definition *extract-lits-sorted'* **where**
‹*extract-lits-sorted' xs n vars* = *extract-lits-sorted* (*xs*, *n*, *vars*)›

lemma *extract-lits-sorted-extract-lits-sorted'*:
‹*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {*res* ← *extract-lits-sorted' xs n vars*; *mop-free xs*; *RETURN res*})›
by (*auto simp*: *extract-lits-sorted'-def mop-free-def intro*!: *ext*)

sepref-def (**in** −) *extract-lits-sorted'-impl*
is ‹*uncurry2 extract-lits-sorted'*›
:: ‹[λ((*xs*, *n*), *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
(*larray64-assn uint64-nat-assn*)^k $*_a$ *uint32-nat-assn*^{k} $*_a$
(*arl64-assn atom-assn*)^d →
arl64-assn atom-assn $\times_a$ *uint32-nat-assn*›
unfolding *extract-lits-sorted'-def extract-lits-sorted-def nres-monad1*
prod.case
by *sepref*

lemmas [*sepref-fr-rules*] = *extract-lits-sorted'-impl.refine*

sepref-def (**in** −) *extract-lits-sorted-code*
is ‹*extract-lits-sorted*›
:: ‹[λ(*xs*, *n*, *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
isasat-atms-ext-rel-assn^d →
arl64-assn atom-assn $\times_a$ *uint32-nat-assn*›
apply (*subst extract-lits-sorted-extract-lits-sorted'*)
unfolding *extract-lits-sorted'-def extract-lits-sorted-def nres-monad1*
prod.case
supply [[*goals-limit* = *1*]]
supply *mset-eq-setD*[*dest*] *mset-eq-length*[*dest*]
by *sepref*

declare *extract-lits-sorted-code.refine*[*sepref-fr-rules*]

abbreviation *lits-with-max-assn* **where**
‹*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* $\times_a$ *uint32-nat-assn*) *lits-with-max-rel*›

lemma *extract-lits-sorted-hnr*[*sepref-fr-rules*]:
‹(*extract-lits-sorted-code*, *RETURN* ∘ *mset-set*) ∈ *nat-lit-list-hm-assn*^{d} $\rightarrow_a$ *lits-with-max-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*hrr-comp isasat-atms-ext-rel*
(λ- -. *al-assn atom-assn* $\times_a$ *unat-assn*) (λ-. *lits-with-max-rel*) =
(λ- -. *lits-with-max-assn*)›
by (*auto simp*: *hrr-comp-def intro*!: *ext*)

have *H*: ‹*?c*
∈ [*comp-PRE isasat-atms-ext-rel* (λ-. *True*)
(λ- (*xs*, *n*, *vars*). ∀ *x*∈#*mset vars*. *x* < *length xs*) (λ-. *True*)]$_a$
hrp-comp (*isasat-atms-ext-rel-assn*^{d}) *isasat-atms-ext-rel* → *lits-with-max-assn*›
(**is** ‹- ∈ [*?pre'*]$_a$ *?im'* → *?f'*›)
using *hfref-compI-PRE-aux*[*OF extract-lits-sorted-code.refine*

extract-lits-sorted-mset-set[*unfolded convert-fref*]]
unfolding *H*
by *auto*

have *pre*: ‹*?pre' x*› **if** ‹*?pre x*› **for** *x*
using *that* **by** (*auto simp*: *comp-PRE-def isasat-atms-ext-rel-def init-valid-rep-def*)
have *im*: ‹*?im' = ?im*›
unfolding *prod-hrp-comp hrp-comp-dest hrp-comp-keep* **by** *simp*
show *?thesis*
apply (*rule hfref-weaken-pre*[*OF*])
defer
using *H* **unfolding** *im PR-CONST-def* **apply** *assumption*
using *pre* **..**
qed

definition *INITIAL-OUTL-SIZE* :: ‹*nat*› **where**
[*simp*]: ‹*INITIAL-OUTL-SIZE = 160*›

sepref-def *INITIAL-OUTL-SIZE-impl*
is ‹*uncurry0* (*RETURN INITIAL-OUTL-SIZE*)›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *sint64-nat-assn*›
unfolding *INITIAL-OUTL-SIZE-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

definition *atom-of-value* :: ‹*nat* ⇒ *nat*› **where** [*simp*]: ‹*atom-of-value x = x*›

lemma *atom-of-value-simp-hnr*:
‹(∃ *x*. (↑(*x = unat xi* ∧ *P x*) ∧* ↑(*x = unat xi*)) *s*) =
(∃ *x*. (↑(*x = unat xi* ∧ *P x*)) *s*)›
‹(∃ *x*. (↑(*x = unat xi* ∧ *P x*)) *s*) = (↑(*P* (*unat xi*))) *s*›
unfolding *import-param-3*[*symmetric*]
by (*auto simp*: *pred-lift-extract-simps*)

lemma *atom-of-value-hnr*[*sepref-fr-rules*]:
‹(*return o* (λ*x*. *x*), *RETURN o atom-of-value*) ∈ $[\lambda n.\ n < 2\ \hat{}31]_a\ (uint32\text{-}nat\text{-}assn)^d \rightarrow atom\text{-}assn$›
apply *sepref-to-hoare*
apply *vcg'*
apply (*auto simp*: *unat-rel-def atom-rel-def unat.rel-def br-def ENTAILS-def*
atom-of-value-simp-hnr pure-true-conv Defer-Slot.remove-slot)
apply (*rule Defer-Slot.remove-slot*)
done

sepref-register *atom-of-value*

lemma [*sepref-gen-algo-rules*]: ‹*GEN-ALGO* (*Pos 0*) (*is-init unat-lit-assn*)›
by (*auto simp*: *unat-lit-rel-def is-init-def unat-rel-def unat.rel-def*
br-def nat-lit-rel-def GEN-ALGO-def)

sepref-def *finalise-init-code'*
is ‹*uncurry finalise-init-code*›
:: ‹$[\lambda(\text{-}, S).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\text{-}init\ S) \leq sint64\text{-}max]_a$
$opts\text{-}assn^d *_a isasat\text{-}init\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
supply [[*goals-limit=1*]]

unfolding *finalise-init-code-def isasat-init-assn-def isasat-bounded-assn-def INITIAL-OUTL-SIZE-def*[*symmetric*] *atom.fold-the vmtf-remove-assn-def heuristic-assn-def*
apply (*rewrite at* ‹*Pos* ¤› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹*Pos* ¤› *atom-of-value-def*[*symmetric*])
apply (*rewrite at* ‹*take* ¤› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(-, -,-,¤, -,-,-,-,-)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(-, -,-,¤, -,-,-)› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const TYPE*(*64*))
apply (*rewrite at* ‹(-, ¤, -)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite at* ‹(-, ¤)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*take* - ¤› *al-fold-custom-replicate*)
apply (*rewrite at* ‹*replicate* - *False*› *annotate-assn*[**where** *A=phase-saver'-assn*])
apply (*rewrite* **in** ‹*replicate* - *False*› *array-fold-custom-replicate*)
apply (*rewrite at* ‹*replicate* - *False*› *annotate-assn*[**where** *A=phase-saver'-assn*])
apply (*rewrite* **in** ‹*replicate* - *False*› *array-fold-custom-replicate*)
by *sepref*

declare *finalise-init-code'.refine*[*sepref-fr-rules*]

sepref-register *initialise-VMTF*
abbreviation *snat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*snat64-assn* ≡ *snat-assn*›
abbreviation *snat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*snat32-assn* ≡ *snat-assn*›
abbreviation *unat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*unat64-assn* ≡ *unat-assn*›
abbreviation *unat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*unat32-assn* ≡ *unat-assn*›

sepref-def *init-trail-D-fast-code*
is ‹*uncurry2 init-trail-D-fast*›
:: ‹$(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k \rightarrow_a trail\text{-}pol\text{-}fast\text{-}assn$›
unfolding *init-trail-D-def PR-CONST-def init-trail-D-fast-def trail-pol-fast-assn-def*
apply (*rewrite* **in** ‹*let* - = ¤ *in* -› *annotate-assn*[**where** *A*=‹*arl64-assn unat-lit-assn*›])
apply (*rewrite* **in** ‹*let* - = ¤ *in* -› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let* - = -; - = ¤ *in* -› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let* - = -; - = ¤ *in* -› *annotate-assn*[**where** *A*=‹*arl64-assn uint32-nat-assn*›])

apply (*rewrite* **in** ‹*let* - = -;- = ¤ *in* -› *annotate-assn*[**where** *A*=‹*larray64-assn* (*tri-bool-assn*)›])
apply (*rewrite* **in** ‹*let* - = -;- = -;- = ¤ *in* -› *annotate-assn*[**where** *A*=‹*larray64-assn uint32-nat-assn*›])
apply (*rewrite* **in** ‹*let* - = - *in* -› *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let* - = - *in* -› *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let* - = - *in* -› *larray-fold-custom-replicate*)
apply (*rewrite at* ‹(-, ¤, -)› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹(*op-larray-custom-replicate* - ¤)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const TYPE*(*64*))
supply [[*goals-limit* = *1*]]
by *sepref*

declare *init-trail-D-fast-code.refine*[*sepref-fr-rules*]

sepref-def *init-state-wl-D'-code*
is ‹*init-state-wl-D'*›
:: ‹$(arl64\text{-}assn\ atom\text{-}assn \times_a uint32\text{-}nat\text{-}assn)^k \rightarrow_a isasat\text{-}init\text{-}assn$›

supply[[*goals-limit=1*]]
unfolding *init-state-wl-D′-def PR-CONST-def init-trail-D-fast-def*[*symmetric*] *isasat-init-assn-def cach-refinement-l-assn-def Suc-eq-plus1-left conflict-option-rel-assn-def lookup-clause-rel-assn-def*
apply (*rewrite at* ‹*let - = 1 + ⌑ in -*› *annot-unat-snat-upcast*[**where** ′*l=64*])
apply (*rewrite at* ‹*let - =* (-, ⌑) *in -*› *al-fold-custom-empty*[**where** ′*l=64*])
apply (*rewrite at* ‹*let - =* (⌑,-) *in -*› *annotate-assn*[**where** *A=* ‹*array-assn minimize-status-assn*›])
apply (*rewrite at* ‹*let - =* (-, ⌑) *in -*› *annotate-assn*[**where** *A=* ‹*arl64-assn atom-assn*›])
apply (*rewrite* **in** ‹*replicate -* []› *aal-fold-custom-empty*(*1*)[**where** ′*l=64* **and** ′*ll=64*])
apply (*rewrite at* ‹*let -= -; -= ⌑ in -*› *annotate-assn*[**where** *A=*‹*watchlist-fast-assn*›])
apply (*rewrite at* ‹*let -= ⌑; -=-;-=-;- = - in RETURN -*› *annotate-assn*[**where** *A=*‹*phase-saver-assn*›])
apply (*rewrite* **in** ‹*let -= ⌑; -=-;-=-;- = - in RETURN -*› *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let -=* (*True*, -, ⌑) *in -*› *array-fold-custom-replicate*)
unfolding *array-fold-custom-replicate*
apply (*rewrite at* ‹*let - = ⌑ in let - =* (*True*, -, -) *in -*› *al-fold-custom-empty*[**where** ′*l=64*])
apply (*rewrite* **in** ‹*let -=* (*True*, ⌑, -) *in -*› *unat-const-fold*[**where** ′*a=32*])
apply (*rewrite at* ‹*let - = ⌑ in -*› *annotate-assn*[**where** *A=*‹*arena-fast-assn*›])
apply (*rewrite at* ‹*let -= ⌑ in RETURN -*› *annotate-assn*[**where** *A =* ‹*vdom-fast-assn*›])
apply (*rewrite* **in** ‹*let -= ⌑ in RETURN -*› *al-fold-custom-empty*[**where** ′*l=64*])
apply (*rewrite at* ‹(-,⌑, - ,-, -, *False*)› *unat-const-fold*[**where** ′*a=32*])
apply (*annot-snat-const TYPE*(*64*))
apply (*rewrite at* ‹*RETURN ⌑*› *annotate-assn*[**where** *A=*‹*isasat-init-assn*›, *unfolded isasat-init-assn-def conflict-option-rel-assn-def cach-refinement-l-assn-def lookup-clause-rel-assn-def*])
by *sepref*

declare *init-state-wl-D′-code.refine*[*sepref-fr-rules*]

lemma *to-init-state-code-hnr*:
‹(*return o to-init-state-code*, *RETURN o id*) ∈ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
unfolding *to-init-state-code-def*
by *sepref-to-hoare vcg′*

abbreviation (**in** −)*lits-with-max-assn-clss* **where**
‹*lits-with-max-assn-clss* ≡ *hr-comp lits-with-max-assn* (⟨*nat-rel*⟩*mset-rel*)›

experiment
begin
export-llvm *init-state-wl-D′-code*
rewatch-heur-st-fast-code
init-dt-wl-heur-code-b

end

end
theory *IsaSAT-Conflict-Analysis*
imports *IsaSAT-Setup IsaSAT-VMTF*
begin

Skip and resolve **definition** *maximum-level-removed-eq-count-dec* **where**
‹*maximum-level-removed-eq-count-dec L S* ⟷
get-maximum-level-remove (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L* =
count-decided (*get-trail-wl S*)›

definition *maximum-level-removed-eq-count-dec-pre* **where**
‹*maximum-level-removed-eq-count-dec-pre* =

(λ(L, S). L = −lit-of (hd (get-trail-wl S)) ∧ L ∈# the (get-conflict-wl S) ∧
get-conflict-wl S ≠ None ∧ get-trail-wl S ≠ [] ∧ count-decided (get-trail-wl S) ≥ 1)›

definition *maximum-level-removed-eq-count-dec-heur* **where**
‹*maximum-level-removed-eq-count-dec-heur L S* =
RETURN (*get-count-max-lvls-heur S* > *1*)›

lemma *get-maximum-level-eq-count-decided-iff*:
‹*ya* ≠ {#} ⟹ *get-maximum-level xa ya* = *count-decided xa* ⟷ (∃ *L* ∈# *ya*. *get-level xa L* = *count-decided xa*)›
apply (*rule iffI*)
defer
subgoal
using *count-decided-ge-get-maximum-level*[*of xa*]
apply (*auto dest*!: *multi-member-split dest*: *le-antisym simp*: *get-maximum-level-add-mset max-def*)
using *le-antisym* **by** *blast*
subgoal
using *get-maximum-level-exists-lit-of-max-level*[*of ya xa*]
by *auto*
done

lemma *get-maximum-level-card-max-lvl-ge1*:
‹*count-decided xa* > *0* ⟹ *get-maximum-level xa ya* = *count-decided xa* ⟷ *card-max-lvl xa ya* > *0*›
apply (*cases* ‹*ya* = {#}›)
subgoal by *auto*
subgoal
by (*auto simp*: *card-max-lvl-def get-maximum-level-eq-count-decided-iff dest*: *multi-member-split*
dest!: *multi-nonempty-split*[*of* ‹*filter-mset - -*›] *filter-mset-eq-add-msetD*
simp flip: *nonempty-has-size*)
done

lemma *card-max-lvl-remove-hd-trail-iff*:
‹*xa* ≠ [] ⟹ − *lit-of* (*hd xa*) ∈# *ya* ⟹ *0* < *card-max-lvl xa* (*remove1-mset* (− *lit-of* (*hd xa*)) *ya*) ⟷ *Suc 0* < *card-max-lvl xa ya*›
by (*cases xa*)
(*auto dest*!: *multi-member-split simp*: *card-max-lvl-add-mset*)

lemma *maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec*:
‹(*uncurry maximum-level-removed-eq-count-dec-heur*,
uncurry mop-maximum-level-removed-wl) ∈
[λ-. *True*]$_f$
Id $\times_r$ *twl-st-heur-conflict-ana* → ⟨*bool-rel*⟩*nres-rel*›
unfolding *maximum-level-removed-eq-count-dec-heur-def mop-maximum-level-removed-wl-def*
uncurry-def
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply *refine-rcg*
apply (*cases x*)
apply (*auto simp*: *count-decided-st-def counts-maximum-level-def twl-st-heur-conflict-ana-def*
maximum-level-removed-eq-count-dec-heur-def maximum-level-removed-eq-count-dec-def
maximum-level-removed-eq-count-dec-pre-def mop-maximum-level-removed-wl-pre-def
mop-maximum-level-removed-l-pre-def mop-maximum-level-removed-pre-def state-wl-l-def
twl-st-l-def get-maximum-level-card-max-lvl-ge1 card-max-lvl-remove-hd-trail-iff)

done
done

lemma *get-trail-wl-heur-def*: ‹*get-trail-wl-heur* = (λ(*M*, *S*). *M*)›
by (*intro ext*, *rename-tac S*, *case-tac S*) *auto*

definition *lit-and-ann-of-propagated-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* × *nat*› **where**
‹*lit-and-ann-of-propagated-st S* = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*))›

definition *lit-and-ann-of-propagated-st-heur*
:: ‹*twl-st-wl-heur* ⇒ (*nat literal* × *nat*) *nres*›
where
‹*lit-and-ann-of-propagated-st-heur* = (λ((*M*, -, -, *reasons*, -), -). *do* {
ASSERT(*M* ≠ [] ∧ *atm-of* (*last M*) < *length reasons*);
RETURN (*last M*, *reasons* ! (*atm-of* (*last M*)))})›

lemma *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*:
‹(*lit-and-ann-of-propagated-st-heur*, *mop-hd-trail-wl*) ∈
[λ*S*. *True*]$_f$ *twl-st-heur-conflict-ana* → ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *lit-and-ann-of-propagated-st-heur-def mop-hd-trail-wl-def*
apply *refine-rcg*
apply (*auto simp*: *twl-st-heur-conflict-ana-def mop-hd-trail-wl-def mop-hd-trail-wl-pre-def*
mop-hd-trail-l-pre-def twl-st-l-def state-wl-l-def mop-hd-trail-pre-def last-rev hd-map
lit-and-ann-of-propagated-st-def trail-pol-alt-def ann-lits-split-reasons-def
intro!: *ASSERT-leI ASSERT-refine-right simp flip*: *rev-map elim*: *is-propedE*)
apply (*auto elim*!: *is-propedE*)
done

definition *tl-state-wl-heur-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*tl-state-wl-heur-pre* =
(λ(*M*, *N*, *D*, *WS*, *Q*, ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), -). *fst M* ≠ [] ∧
tl-trailt-tr-pre M ∧
vmtf-unset-pre (*atm-of* (*last* (*fst M*))) ((*A*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∧
atm-of (*last* (*fst M*)) < *length A* ∧
(*next-search* ≠ *None* ⟶ *the next-search* < *length A*))›

definition *tl-state-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*› **where**
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*));
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of* (*lit-of-last-trail-pol M*))
vmtf, *clvls*))
})›

lemma *tl-state-wl-heur-alt-def*:
‹*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, *clvls*));
let L = *lit-of-last-trail-pol M*;
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset* (*atm-of L*) *vmtf*, *clvls*))
})›
by (*auto simp*: *tl-state-wl-heur-def Let-def intro*!: *ext*)

lemma *card-max-lvl-Cons*:

assumes ‹*no-dup* (*L* # *a*)› ‹*distinct-mset y*›‹¬*tautology y*› ‹¬*is-decided L*›
shows ‹*card-max-lvl* (*L* # *a*) *y* =
(*if* (*lit-of L* ∈# *y* ∨ −*lit-of L* ∈# *y*) ∧ *count-decided a* ≠ *0 then card-max-lvl a y* + *1*
else card-max-lvl a y)›
proof −
have [*simp*]: ‹*count-decided a* = *0* ⟹ *get-level a L* = *0*› **for** *L*
by (*simp add*: *count-decided-0-iff*)
have [*simp*]: ‹*lit-of L* ∉# *A* ⟹
− *lit-of L* ∉# *A* ⟹
{#*La* ∈# *A*. *La* ≠ *lit-of L* ∧ *La* ≠ − *lit-of L* ⟶ *get-level a La* = *b*#} =
{#*La* ∈# *A*. *get-level a La* = *b*#}› **for** *A b*
apply (*rule filter-mset-cong*)
apply (*rule refl*)
by *auto*
show *?thesis*
using *assms* **by** (*auto simp*: *card-max-lvl-def get-level-cons-if tautology-add-mset*
atm-of-eq-atm-of
dest!: *multi-member-split*)
qed

lemma *card-max-lvl-tl*:
assumes ‹*a* ≠ []› ‹*distinct-mset y*›‹¬*tautology y*› ‹¬*is-decided* (*hd a*)› ‹*no-dup a*›
‹*count-decided a* ≠ *0*›
shows ‹*card-max-lvl* (*tl a*) *y* =
(*if* (*lit-of*(*hd a*) ∈# *y* ∨ −*lit-of*(*hd a*) ∈# *y*)
then card-max-lvl a y − *1 else card-max-lvl a y*)›
using *assms* **by** (*cases a*) (*auto simp*: *card-max-lvl-Cons*)

definition *tl-state-wl-pre* **where**
‹*tl-state-wl-pre S* ⟷ *get-trail-wl S* ≠ [] ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
−(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
¬*is-decided* (*hd* (*get-trail-wl S*)) ∧
count-decided (*get-trail-wl S*) > *0*›

lemma *tl-state-out-learned*:
‹*lit-of* (*hd a*) ∉# *the at* ⟹
− *lit-of* (*hd a*) ∉# *the at* ⟹
¬ *is-decided* (*hd a*) ⟹
out-learned (*tl a*) *at an* ⟷ *out-learned a at an*›
by (*cases a*) (*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of*
intro!: *filter-mset-cong*)

lemma *mop-tl-state-wl-pre-tl-state-wl-heur-pre*:
‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹ *mop-tl-state-wl-pre y* ⟹ *tl-state-wl-heur-pre x*›
using *tl-trailt-tr-pre*[*of* ‹*get-trail-wl y*› ‹*get-trail-wl-heur x*› ‹*all-atms-st y*›]
unfolding *mop-tl-state-wl-pre-def tl-state-wl-heur-pre-def mop-tl-state-l-pre-def*
mop-tl-state-pre-def tl-state-wl-heur-pre-def
apply (*auto simp*: *twl-st-heur-conflict-ana-def state-wl-l-def twl-st-l-def trail-pol-alt-def*
rev-map[*symmetric*] *last-rev hd-map*
intro!: *vmtf-unset-pre′*[**where** *M* = ‹*get-trail-wl y*›])
apply (*auto simp*: *neq-Nil-conv literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons phase-saving-def isa-vmtf-def*
vmtf-def

dest!: *multi-member-split*)
done

lemma *mop-tl-state-wl-pre-simps*:
‹*mop-tl-state-wl-pre* ([], *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\longleftrightarrow$ *False*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$
lit-of (*hd xa*) $\in\#$ $\mathcal{L}_{all}$ (*all-atms ax* (*az* + *bga* + *NS* + *US*))›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ *lit-of* (*hd xa*) $\notin\#$ *the ay*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ $-$*lit-of* (*hd xa*) $\notin\#$ *the ay*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ *lit-of* (*hd xa*) $\notin\#$ *ay'*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ $-$*lit-of* (*hd xa*) $\notin\#$ *ay'*›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ *is-proped* (*hd xa*)›
‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NS*, *US*, *bh*, *bi*) $\Longrightarrow$ *count-decided xa* > *0*›
unfolding *mop-tl-state-wl-pre-def tl-state-wl-heur-pre-def mop-tl-state-l-pre-def*
mop-tl-state-pre-def tl-state-wl-heur-pre-def
apply (*auto simp*: *twl-st-heur-conflict-ana-def state-wl-l-def twl-st-l-def trail-pol-alt-def*
rev-map[*symmetric*] *last-rev hd-map mset-take-mset-drop-mset'* $\mathcal{L}_{all}$*-all-atms-all-lits*
simp flip: *image-mset-union all-lits-def all-lits-alt-def2*)
done

abbreviation *twl-st-heur-conflict-ana'* :: ‹*nat* $\Rightarrow$ (*twl-st-wl-heur* $\times$ *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-conflict-ana'* *r* $\equiv$ {(*S*, *T*). (*S*, *T*) $\in$ *twl-st-heur-conflict-ana* $\wedge$
length (*get-clauses-wl-heur S*) = *r*}›

lemma *tl-state-wl-heur-tl-state-wl*:
‹(*tl-state-wl-heur*, *mop-tl-state-wl*) $\in$
[λ-. *True*]$_f$ *twl-st-heur-conflict-ana'* *r* $\rightarrow$ ⟨*bool-rel* $\times_f$ *twl-st-heur-conflict-ana'* *r*⟩*nres-rel*›
supply [[*goals-limit=1*]]
unfolding *tl-state-wl-heur-def mop-tl-state-wl-def*
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal for *x y a b aa ba ab bb ac bc ad bd ae be*
using *mop-tl-state-wl-pre-tl-state-wl-heur-pre*[*of x y*] **by** *simp*
subgoal
apply (*auto simp*: *twl-st-heur-conflict-ana-def tl-state-wl-heur-def tl-state-wl-def vmtf-unset-vmtf-tl*
mop-tl-state-wl-pre-simps lit-of-last-trail-pol-lit-of-last-trail[*THEN fref-to-Down-unRET-Id*]
card-max-lvl-tl tl-state-out-learned
dest: *no-dup-tlD*
intro!: *tl-trail-tr*[*THEN fref-to-Down-unRET*] *isa-vmtf-tl-isa-vmtf*)
apply (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
apply (*auto simp*: *lit-of-hd-trail-def mop-tl-state-wl-pre-simps counts-maximum-level-def*)
apply (*subst card-max-lvl-tl*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
apply (*subst tl-state-out-learned*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
apply (*subst tl-state-out-learned*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
done
done

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* $\Longrightarrow$
arena-act-pre (*mark-used arena C*) *C*›

unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
apply *clarify*
apply (*rule-tac x=N* **in** *exI*)
apply (*rule-tac x=vdom* **in** *exI*)
by (*auto simp*: *arena-act-pre-def*
simp: *valid-arena-mark-used*)

definition (**in** −) *get-max-lvl-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*get-max-lvl-st S L* = *get-maximum-level-remove* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L*›

definition *update-confl-tl-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*update-confl-tl-wl-heur* = (λ*L C* (*M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*). *do* {
ASSERT (*clvls* ≥ *1*);
let L′ = *atm-of L*;
ASSERT(*arena-is-valid-clause-idx N C*);
((*b*, (*n*, *xs*)), *clvls*, *lbd*, *outl*) ←
if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C* (*b*, (*n*, *xs*)) *clvls lbd outl*
else isa-resolve-merge-conflict-gt2 M N C (*b*, (*n*, *xs*)) *clvls lbd outl*;
ASSERT(*curry lookup-conflict-remove1-pre L* (*n*, *xs*) ∧ *clvls* ≥ *1*);
let (*n*, *xs*) = *lookup-conflict-remove1 L* (*n*, *xs*);
ASSERT(*arena-act-pre N C*);
let N = *mark-used N C*;
ASSERT(*arena-act-pre N C*);
let N = *arena-incr-act N C*;
ASSERT(*vmtf-unset-pre L′ vm*);
ASSERT(*tl-trailt-tr-pre M*);
RETURN (*False*, (*tl-trailt-tr M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *isa-vmtf-unset L′ vm*,
clvls − *1*, *cach*, *lbd*, *outl*, *stats*))
})›

lemma *card-max-lvl-remove1-mset-hd*:
‹−*lit-of* (*hd M*) ∈# *y* ⟹ *is-proped* (*hd M*) ⟹
card-max-lvl M (*remove1-mset* (−*lit-of* (*hd M*)) *y*) = *card-max-lvl M y* − *1*›
by (*cases M*) (*auto dest*!: *multi-member-split simp*: *card-max-lvl-add-mset*)

lemma *update-confl-tl-wl-heur-state-helper*:
‹(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ⟹ *get-trail-wl S* ≠ [] ⟹
is-proped (*hd* (*get-trail-wl S*)) ⟹ *L* = *lit-of* (*hd* (*get-trail-wl S*))›
by (*cases S*; *cases* ‹*hd* (*get-trail-wl S*)›) *auto*

lemma (**in** −) *not-ge-Suc0*: ‹¬*Suc 0* ≤ *n* ⟷ *n* = *0*›
by *auto*

definition *update-confl-tl-wl-pre′* :: ‹((*nat literal* × *nat*) × *nat twl-st-wl*) ⇒ *bool*› **where**
‹*update-confl-tl-wl-pre′* = (λ((*L*, *C*), *S*).
C ∈# *dom-m* (*get-clauses-wl S*) ∧
get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧
− *L* ∈# *the* (*get-conflict-wl S*) ∧
L ∉# *the* (*get-conflict-wl S*) ∧
(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ∧
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
is-proped (*hd* (*get-trail-wl S*)) ∧
C > *0* ∧

card-max-lvl (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1* ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
− *L* ∉ *set* (*get-clauses-wl S* ∝ *C*) ∧
(*length* (*get-clauses-wl S* ∝ *C*) ≠ *2* ⟶
L ∉ *set* (*tl* (*get-clauses-wl S* ∝ *C*)) ∧
get-clauses-wl S ∝ *C* ! *0* = *L* ∧
mset (*tl* (*get-clauses-wl S* ∝ *C*)) = *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
(∀ *L*∈*set* (*tl*(*get-clauses-wl S* ∝ *C*)). − *L* ∉# *the* (*get-conflict-wl S*)) ∧
card-max-lvl (*get-trail-wl S*) (*mset* (*tl* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*))) ∧
L ∈ *set* (*watched-l* (*get-clauses-wl S* ∝ *C*)) ∧
distinct (*get-clauses-wl S* ∝ *C*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
¬*tautology* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
¬*tautology* (*remove1-mset L* (*remove1-mset* (− *L*)
((*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*))))) ∧
count-decided (*get-trail-wl S*) > *0* ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*)) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(∀ *K*. *K* ∈# *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ⟶ − *K* ∉# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) > *0* ∧
Suc 0 ≤ *card-max-lvl* (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
size (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
lit-of (*hd* (*get-trail-wl S*)) = *L* ∧
card-max-lvl (*get-trail-wl S*) ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*tl* (*get-trail-wl S*)) (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
out-learned (*tl* (*get-trail-wl S*)) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) =
out-learned (*get-trail-wl S*) (*Some* ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)))
)›

lemma *remove1-mset-union-distrib1*:
‹*L* ∉# *B* ⟹ *remove1-mset L* (*A* ∪# *B*) = *remove1-mset L A* ∪# *B*› **and**
remove1-mset-union-distrib2:
‹*L* ∉# *A* ⟹ *remove1-mset L* (*A* ∪# *B*) = *A* ∪# *remove1-mset L B*›
by (*auto simp*: *remove1-mset-union-distrib*)

lemma *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*:
assumes ‹*update-confl-tl-wl-pre L C S*›
shows ‹*update-confl-tl-wl-pre′* ((*L*, *C*), *S*)›
proof −
obtain *x xa* **where**
Sx: ‹(*S*, *x*) ∈ *state-wl-l None*› **and**
‹*correct-watching S*› **and**
x-xa: ‹(*x*, *xa*) ∈ *twl-st-l None*› **and**
st-invs: ‹*twl-struct-invs xa*› **and**
list-invs: ‹*twl-list-invs x*› **and**
‹*twl-stgy-invs xa*› **and**

C: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)› **and**
nempty: ‹*get-trail-wl S* ≠ []› **and**
‹*literals-to-update-wl S* = {#}› **and**
hd: ‹*hd* (*get-trail-wl S*) = *Propagated L C*› **and**
C-0: ‹*0* < *C*› **and**
confl: ‹*get-conflict-wl S* ≠ *None*› **and**
‹*0* < *count-decided* (*get-trail-wl S*)› **and**
‹*get-conflict-wl S* ≠ *Some* {#}› **and**
‹*L* ∈ *set* (*get-clauses-wl S* ∝ *C*)› **and**
uL-D: ‹− *L* ∈# *the* (*get-conflict-wl S*)› **and**
xa: ‹*hd* (*get-trail xa*) = *Propagated L* (*mset* (*get-clauses-wl S* ∝ *C*))› **and**
L: ‹*L* ∈# *all-lits-st S*› **and**
blits: ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
using *assms*
unfolding *update-confl-tl-wl-pre-def*
update-confl-tl-l-pre-def update-confl-tl-pre-def
prod.case **apply** −
by *normalize-goal+*
(*simp flip*: *all-lits-def all-lits-alt-def2*
add: *mset-take-mset-drop-mset′* $\mathcal{L}_{all}$*-all-atms-all-lits*)

have
dist: ‹*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of xa*)› **and**
M-lev: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of xa*)› **and**
confl′: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of xa*)› **and**
st-inv: ‹*twl-st-inv xa*›
using *st-invs* **unfolding** *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast+*

have *eq*: ‹*lits-of-l* (*tl* (*get-trail-wl S*)) = *lits-of-l* (*tl* (*get-trail xa*))›
using *Sx x-xa* **unfolding** *list.set-map*[*symmetric*] *lits-of-def*
by (*cases S*; *cases x*; *cases xa*;
auto simp: *state-wl-l-def twl-st-l-def map-tl list-of-l-convert-map-lit-of simp del*: *list.set-map*)

have *card-max*: ‹*card-max-lvl* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1*›
using *hd uL-D nempty* **by** (*cases* ‹*get-trail-wl S*›; *auto dest*!: *multi-member-split simp*: *card-max-lvl-def*)

have *dist-C*: ‹*distinct-mset* (*the* (*get-conflict-wl S*))›
using *dist Sx x-xa confl C* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def*
by (*auto simp*: *twl-st*)
have *dist*: ‹*distinct* (*get-clauses-wl S* ∝ *C*)›
using *dist Sx x-xa confl C* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-alt-def*
by (*auto simp*: *image-image ran-m-def dest*!: *multi-member-split*)

have *n-d*: ‹*no-dup* (*get-trail-wl S*)›
using *Sx x-xa M-lev* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *twl-st*)
have *CNot-D*: ‹*get-trail-wl S* ⊨*as CNot* (*the* (*get-conflict-wl S*))›
using *confl′ confl Sx x-xa* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
by (*auto simp*: *twl-st*)
then have ‹*tl* (*get-trail-wl S*) ⊨*as CNot* (*the* (*get-conflict-wl S*) − {#−*L*#})›
using *dist-C uL-D n-d hd nempty* **by** (*cases* ‹*get-trail-wl S*›)
(*auto dest*!: *multi-member-split simp*: *true-annots-true-cls-def-iff-negation-in-model*)
moreover have *CNot-C′*: ‹*tl* (*get-trail-wl S*) ⊨*as CNot* (*mset* (*get-clauses-wl S* ∝ *C*) − {#*L*#})›
and
L-C: ‹*L* ∈# *mset* (*get-clauses-wl S* ∝ *C*)›

using *confl′ nempty x-xa xa hd Sx nempty eq*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
by (*cases* ‹*get-trail xa*›; *fastforce simp*: *twl-st twl-st-l true-annots-true-cls-def-iff-negation-in-model dest*: *spec*[*of* - ‹[]›])+

ultimately have *tl*: ‹tl ($get\text{-}trail\text{-}wl$ S) $\models as$ $CNot$ ((the ($get\text{-}conflict\text{-}wl$ S) $-$ {#$-L$#}) $\cup$# ($mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $-$ {#L#}))›
by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)
then have ‹(the ($get\text{-}conflict\text{-}wl$ S) $-$ {#$-L$#}) $\cup$# ($mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $-$ {#L#}) = (the ($get\text{-}conflict\text{-}wl$ S) $\cup$# $mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $-$ {#L, $-$ L#})›
using *multi-member-split*[*OF L-C*] *uL-D dist dist-C n-d hd nempty*
apply (*cases* ‹*get-trail-wl S*›; *auto dest*!: *multi-member-split simp*: *true-annots-true-cls-def-iff-negation-in-model*)
apply (*subst sup-union-left1*)
apply (*auto dest*!: *multi-member-split dest*: *in-lits-of-l-defined-litD*)
done
with *tl* **have** ‹tl ($get\text{-}trail\text{-}wl$ S) $\models as$ $CNot$ (the ($get\text{-}conflict\text{-}wl$ S) $\cup$# $mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $-$ {#L, $-$ L#})› **by** *simp*
with *distinct-consistent-interp*[*OF no-dup-tlD*[*OF n-d*]] **have** *1*: ‹$\neg tautology$ (the ($get\text{-}conflict\text{-}wl$ S) $\cup$# $mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $-$ {#L, $-$ L#})›
unfolding *true-annots-true-cls*
by (*rule consistent-CNot-not-tautology*)
have *False* **if** ‹$-$ L $\in$# $mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$)›
using *multi-member-split*[*OF L-C*] *hd nempty n-d CNot-C′ multi-member-split*[*OF that*]
by (*cases* ‹*get-trail-wl S*›; *auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset true-annots-true-cls-def-iff-negation-in-model dest*!: *in-lits-of-l-defined-litD*)
then have *2*: ‹$-L \notin set$ ($get\text{-}clauses\text{-}wl$ $S \propto C$)›
by *auto*

have ‹$length$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) ≥ 2›
using *st-inv C Sx x-xa* **by** (*cases xa*; *cases x*; *cases S*; *cases* ‹*irred* (*get-clauses-wl S*) *C*›; *auto simp*: *twl-st-inv.simps state-wl-l-def twl-st-l-def conj-disj-distribR Collect-disj-eq image-Un ran-m-def Collect-conv-if dest*!: *multi-member-split*)
then have [*simp*]: ‹$length$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) $\neq 2 \longleftrightarrow length$ ($get\text{-}clauses\text{-}wl$ $S \propto C$) > 2›
by (*cases* ‹$get\text{-}clauses\text{-}wl$ $S \propto C$›; *cases* ‹tl ($get\text{-}clauses\text{-}wl$ $S \propto C$)›; *auto simp*: *twl-list-invs-def all-conj-distrib dest*: *in-set-takeD*)

have *CNot-C*: ‹$\neg tautology$ ($mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$))›
using *CNot-C′ Sx hd nempty C-0 dist multi-member-split*[*OF L-C*] *dist consistent-CNot-not-tautology*[*OF distinct-consistent-interp*[*OF no-dup-tlD*[*OF n-d*]] *CNot-C′*[*unfolded true-annots-true-cls*]] *2*
unfolding *true-annots-true-cls-def-iff-negation-in-model*
apply (*auto simp*: *tautology-add-mset dest*: *arg-cong*[*of* ‹*mset* -› - *set-mset*])
by (*metis member-add-mset set-mset-mset*)

have *stupid*: $K \in\# removeAll\text{-}mset\ L\ D \longleftrightarrow K \neq L \land K \in\# D$ **for** *K L D*
by *auto*
have *K*: ‹$K \in\# remove1\text{-}mset\ L$ ($mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$)) $\Longrightarrow$ $-$ $K \notin\# the$ ($get\text{-}conflict\text{-}wl$ S)›
and
uL-C: ‹$-L \notin\#$ ($mset$ ($get\text{-}clauses\text{-}wl$ $S \propto C$))› **for** *K*
apply (*subst* (*asm*) *distinct-mset-remove1-All*)

subgoal using *dist* **by** *auto*
apply (*subst* (*asm*)*stupid*)
apply (*rule conjE*, *assumption*)
apply (*drule multi-member-split*)
using *1 uL-D CNot-C dist 2*[*unfolded in-multiset-in-set*[*symmetric*]] *dist-C*
consistent-CNot-not-tautology[*OF distinct-consistent-interp*[*OF n-d*]
CNot-D[*unfolded true-annots-true-cls*]] ‹L ∈# *mset* (*get-clauses-wl* $S \propto C$)›
by (*auto dest*!: *multi-member-split*[*of* ‹$-L$›] *multi-member-split in-set-takeD*
simp: *tautology-add-mset add-mset-eq-add-mset uminus-lit-swap diff-union-swap2*
simp del: *set-mset-mset in-multiset-in-set*
distinct-mset-mset-distinct simp flip: *distinct-mset-mset-distinct*)
have *size*: ‹*size* (*remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) ∪# *the* (*get-conflict-wl S*)) > *0*›
using *uL-D uL-C* **by** (*auto dest*!: *multi-member-split*)

have *max-lvl*: ‹*Suc 0* ≤ *card-max-lvl* (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) ∪# *the* (*get-conflict-wl S*))›
using *uL-D hd nempty uL-C* **by** (*cases* ‹*get-trail-wl S*›; *auto simp*: *card-max-lvl-def dest*!: *multi-member-split*)

have *s*: ‹*size* (*remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) ∪# *the* (*get-conflict-wl S*)) =
size (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl* $S \propto C$) − {#*L*, − *L*#}) + *1*›
apply (*subst* (*2*) *subset-mset.sup.commute*)
using *uL-D hd nempty uL-C dist-C* **apply** (*cases* ‹*get-trail-wl S*›; *auto simp*: *dest*!: *multi-member-split*)
by (*metis* (*no-types*, *hide-lams*) ‹*remove1-mset* (− *L*) (*the* (*get-conflict-wl S*)) ∪# *remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) = *the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl* $S \propto C$) − {#*L*, − *L*#}› *add-mset-commute add-mset-diff-bothsides add-mset-remove-trivial set-mset-mset subset-mset.sup-commute sup-union-left1*)

have *SC-0*: ‹*length* (*get-clauses-wl* $S \propto C$) > *2* ⟹ *L* ∉ *set* (*tl* (*get-clauses-wl* $S \propto C$)) ∧ *get-clauses-wl* $S \propto C$! *0* = *L* ∧
mset (*tl* (*get-clauses-wl* $S \propto C$)) = *remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) ∧
(∀ *L*∈*set* (*tl*(*get-clauses-wl* $S \propto C$)). − *L* ∉# *the* (*get-conflict-wl S*)) ∧
card-max-lvl (*get-trail-wl S*) (*mset* (*tl* (*get-clauses-wl* $S \propto C$)) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl* $S \propto C$)) ∪# *the* (*get-conflict-wl S*))›
‹*L* ∈ *set* (*watched-l* (*get-clauses-wl* $S \propto C$))›
‹*L* ∈# *mset* (*get-clauses-wl* $S \propto C$)›
using *list-invs Sx hd nempty C-0 dist L-C K*
by (*cases* ‹*get-trail-wl S*›; *cases* ‹*get-clauses-wl* $S \propto C$›;
auto simp: *twl-list-invs-def all-conj-distrib dest*: *in-set-takeD*; *fail*)+

have *max*: ‹*card-max-lvl* (*get-trail-wl S*) ((*mset* (*get-clauses-wl* $S \propto C$) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*tl* (*get-trail-wl S*)) (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl* $S \propto C$) − {#*L*, − *L*#}) + *Suc 0*›
using *multi-member-split*[*OF uL-D*] *L-C hd nempty n-d dist dist-C 1* ‹*0* < *count-decided* (*get-trail-wl S*)› *uL-C n-d*
consistent-CNot-not-tautology[*OF distinct-consistent-interp*[*OF n-d*]
CNot-D[*unfolded true-annots-true-cls*]] *CNot-C* **apply** (*cases* ‹*get-trail-wl S*›;
auto dest!: *simp*: *card-max-lvl-Cons card-max-lvl-add-mset subset-mset.sup-commute*[*of* - ‹*remove1-mset* - -›]
subset-mset.sup-commute[*of* - ‹*mset* -›] *tautology-add-mset remove1-mset-union-distrib1 remove1-mset-union-distrib2*)
by (*simp add*: *distinct-mset-remove1-All*[*of* ‹*mset* (*get-clauses-wl* $S \propto C$)›])

have *xx*: ‹*out-learned* (*tl* (*get-trail-wl S*)) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) *outl* ⟷
out-learned (*get-trail-wl S*) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) *outl*› **for** *outl*
apply (*subst tl-state-out-learned*)
apply (*cases* ‹*get-trail-wl S*›; *use L-C hd nempty dist dist-C* **in** *auto*)
apply (*meson distinct-mem-diff-mset distinct-mset-mset-distinct distinct-mset-union-mset union-single-eq-member*)
apply (*cases* ‹*get-trail-wl S*›; *use L-C hd nempty dist dist-C* **in** *auto*)
apply (*metis add-mset-commute distinct-mem-diff-mset distinct-mset-mset-distinct distinct-mset-union-mset union-single-eq-member*)
apply (*cases* ‹*get-trail-wl S*›; *use L-C hd nempty dist dist-C* **in** *auto*)
..

have [*simp*]: ‹*get-level* (*get-trail-wl S*) *L* = *count-decided* (*get-trail-wl S*)›
by (*cases* ‹*get-trail-wl S*›; *use L-C hd nempty dist dist-C* **in** *auto*)
have *yy*: ‹*out-learned* (*get-trail-wl S*) (*Some* ((*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*)) − {#*L*, − *L*#})) *outl* ⟷
out-learned (*get-trail-wl S*) (*Some* ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*))) *outl*› **for** *outl*
by (*use L-C hd nempty dist dist-C* **in** ‹*auto simp add*: *out-learned-def ac-simps*›)

have *xx*: ‹*out-learned* (*tl* (*get-trail-wl S*)) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) =
out-learned (*get-trail-wl S*) (*Some* ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)))›
using *xx yy* **by** (*auto intro*!: *ext*)
show *?thesis*
using *Sx x-xa C C-0 confl nempty uL-D L blits 1 2 card-max dist-C dist SC-0 max*
consistent-CNot-not-tautology[*OF distinct-consistent-interp*[*OF n-d*]
CNot-D[*unfolded true-annots-true-cls*]] *CNot-C K xx*
‹*0* < *count-decided* (*get-trail-wl S*)› *size max-lvl s*
literals-are-$\mathcal{L}_{in}$*-literals-are-in-*$\mathcal{L}_{in}$*-conflict*[*OF Sx st-invs x-xa* - , *of* ‹*all-atms-st S*›]
literals-are-$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*[*OF Sx st-invs x-xa* - , *of* ‹*all-atms-st S*›]
unfolding *update-confl-tl-wl-pre′-def*
by (*clarsimp simp flip*: *all-lits-def simp add*: *hd mset-take-mset-drop-mset′* $\mathcal{L}_{all}$*-all-atms-all-lits*)

qed

lemma (**in** −)*out-learned-add-mset-highest-level*:
‹*L* = *lit-of* (*hd M*) ⟹ *out-learned M* (*Some* (*add-mset* (− *L*) *A*)) *outl* ⟷
out-learned M (*Some A*) *outl*›
by (*cases M*)
(*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of count-decided-ge-get-level*
uminus-lit-swap cong: *if-cong*
intro!: *filter-mset-cong2*)

lemma (**in** −)*out-learned-tl-Some-notin*:
‹*is-proped* (*hd M*) ⟹ *lit-of* (*hd M*) ∉# *C* ⟹ −*lit-of* (*hd M*) ∉# *C* ⟹
out-learned M (*Some C*) *outl* ⟷ *out-learned* (*tl M*) (*Some C*) *outl*›
by (*cases M*) (*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of*
intro!: *filter-mset-cong2*)

lemma *literals-are-in-*$\mathcal{L}_{in}$*-mm-all-atms-self*[*simp*]:
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms ca NUE*) {#*mset* (*fst x*). *x* ∈# *ran-m ca*#}›

by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-mm-def in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*
all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff)

lemma *mset-as-position-remove3*:
‹*mset-as-position xs* $(D - \{\#L\#\}) \Longrightarrow$ *atm-of* $L <$ *length xs* $\Longrightarrow$ *distinct-mset* $D \Longrightarrow$
mset-as-position (*xs*[*atm-of L* := *None*]) $(D - \{\#L, -L\#\})$›
using *mset-as-position-remove2*[*of xs* ‹$D - \{\#L\#\}$› ‹*L*›] *mset-as-position-tautology*[*of xs* ‹*remove1-mset*
L D›]
mset-as-position-distinct-mset[*of xs* ‹*remove1-mset L D*›]
by (*cases* ‹$L \in\# D$›; *cases* ‹$-L \in\# D$›)
(*auto dest*!: *multi-member-split simp*: *minus-notin-trivial ac-simps add-mset-eq-add-mset tautology-add-mset*)

lemma *update-confl-tl-wl-heur-update-confl-tl-wl*:
‹(*uncurry2* (*update-confl-tl-wl-heur*), *uncurry2 mop-update-confl-tl-wl*) $\in$
$[\lambda$-. *True*$]_f$
Id $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-conflict-ana'* *r* $\rightarrow$ $\langle$*bool-rel* $\times_f$ *twl-st-heur-conflict-ana'* *r*$\rangle$*nres-rel*›
proof −
have *mop-update-confl-tl-wl-alt-def*: ‹*mop-update-confl-tl-wl* = (λ*L C* (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*).
do {
ASSERT(*update-confl-tl-wl-pre L C* (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*));
D ← *RETURN* (*resolve-cls-wl'* (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*) *C L*);
N ← *RETURN N*;
N ← *RETURN N*;
RETURN (*False*, (*tl M*, *N*, *Some D*, *NE*, *UE*, *WS*, *Q*))
})›
by (*auto simp*: *mop-update-confl-tl-wl-def update-confl-tl-wl-def intro*!: *ext*)
define *rr* **where**
‹*rr L M N C b n xs clvls lbd outl* = *do* {
((*b*, (*n*, *xs*)), *clvls*, *lbd*, *outl*) ←
if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C* (*b*, (*n*, *xs*)) *clvls lbd outl*
else isa-resolve-merge-conflict-gt2 M N C (*b*, (*n*, *xs*)) *clvls lbd outl*;
ASSERT(*curry lookup-conflict-remove1-pre L* (*n*, *xs*) $\wedge$ *clvls* $\geq$ *1*);
let (*nxs*) = *lookup-conflict-remove1 L* (*n*, *xs*);
RETURN ((*b*, (*nxs*)), *clvls*, *lbd*, *outl*) }›
for *L M N C b n xs clvls lbd outl*
have *update-confl-tl-wl-heur-alt-def*:
‹*update-confl-tl-wl-heur* = (λ*L C* (*M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*). *do* {
ASSERT (*clvls* $\geq$ *1*);
let L' = *atm-of L*;
ASSERT(*arena-is-valid-clause-idx N C*);
((*b*, (*n*, *xs*)), *clvls*, *lbd*, *outl*) ← *rr L M N C b n xs clvls lbd outl*;
ASSERT(*arena-act-pre N C*);
let N = *mark-used N C*;
ASSERT(*arena-act-pre N C*);
let N = *arena-incr-act N C*;
ASSERT(*vmtf-unset-pre* L' *vm*);
ASSERT(*tl-trailt-tr-pre M*);
RETURN (*False*, (*tl-trailt-tr M*, *N*, (*b*, (*n*, *xs*)), *Q*, *W*, *isa-vmtf-unset* L' *vm*,
clvls − *1*, *cach*, *lbd*, *outl*, *stats*))
})›
by (*auto simp*: *update-confl-tl-wl-heur-def Let-def rr-def intro*!: *ext bind-cong*[*OF refl*])

note [[*goals-limit*=*1*]]
have *rr*: ‹(((*x1g*, *x2g*), *x1h*, *x1i*, (*x1k*, *x1l*, *x2k*), *x1m*, *x1n*, *x1p*, *x1q*, *x1r*,
x1s, *x1t*, *m*, *n*, *p*, *q*, *ra*, *s*, *t*),
(*x1*, *x2*), *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)

$\in$ *nat-lit-lit-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-conflict-ana' r* $\Longrightarrow$
CLS = ((*x1*, *x2*), *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) $\Longrightarrow$
CLS' =
((*x1g*, *x2g*), *x1h*, *x1i*, (*x1k*, *x1l*, *x2k*), *x1m*, *x1n*, *x1p*, *x1q*, *x1r*,
x1s, *x1t*, *m*, *n*, *p*, *q*, *ra*, *s*, *t*) $\Longrightarrow$
update-confl-tl-wl-pre x1 x2 (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) $\Longrightarrow$
1 $\leq$ *x1q* $\Longrightarrow$
arena-is-valid-clause-idx x1i x2g $\Longrightarrow$
rr x1g x1h x1i x2g x1k x1l x2k x1q x1s x1t
$\leq \Downarrow$ {((*C*, *clvls*, *lbd*, *outl*), *D*). (*C*, *Some D*) $\in$ *option-lookup-clause-rel* (*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)) $\wedge$
clvls = *card-max-lvl x1a* (*remove1-mset x1* (*mset* (*x1b* $\propto$ *x2*)) $\cup$# *the x1c*) $\wedge$
out-learned x1a (*Some* (*remove1-mset x1* (*mset* (*x1b* $\propto$ *x2*)) $\cup$# *the x1c*)) (*outl*) $\wedge$
size (*remove1-mset x1* (*mset* (*x1b* $\propto$ *x2*)) $\cup$# *the x1c*) =
size ((*mset* (*x1b* $\propto$ *x2*)) $\cup$# *the x1c* − {#*x1*, −*x1*#}) + *Suc 0* $\wedge$
D = *resolve-cls-wl'* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) *x2 x1*}
(*RETURN* (*resolve-cls-wl'* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) *x2 x1*))›
for *m n p q ra s t x1 x2 x1a x1b x1c x1d x1e x1f x2f x1g x2g x1h x1i x1k*
x1l x2k x1m x1n x1p x1q x1r x1s x1t CLS CLS' NS US
unfolding *rr-def*
apply (*refine-vcg lhs-step-If*)
apply (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[**where**
vdom = ‹*set* (*get-vdom* (*x1h*, *x1i*, (*x1k*, *x1l*, *x2k*), *x1m*, *x1n*, *x1p*, *x1q*, *x1r*,
x1s, *x1t*, *m*, *n*, *p*, *q*, *ra*, *s*, *t*))› **and** *M* = *x1a* **and** *N* = *x1b* **and** *C* = *x1c* **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) ›, *THEN order-trans*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre' simp*: *update-confl-tl-wl-pre'-def*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal unfolding *Down-id-eq*
apply (*rule lookup-merge-eq2-spec*[**where** *M* = *x1a* **and** *C* = ‹*the x1c*› **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)›, *THEN order-trans*])
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def intro*!: *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def twl-st-heur-conflict-ana-def*)

subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def merge-conflict-m-eq2-def conc-fun-SPEC lookup-conflict-remove1-pre-def*
atms-of-def
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def lookup-conflict-remove1-def
remove1-mset-union-distrib1 remove1-mset-union-distrib2 subset-mset.sup.commute[*of* - ‹*remove1-mset*
- -›]
subset-mset.sup.commute[*of* - ‹*mset* (- ∝ -)›]
intro!: *mset-as-position-remove3*
intro!: *ASSERT-leI*)
done
subgoal
apply (*subst* (*asm*) *arena-lifting*(*4*)[**where** *vdom* = ‹*set p*› **and** *N* = *x1b*, *symmetric*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def*)
apply (*rule isa-resolve-merge-conflict-gt2*[**where**
𝒜 = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)› **and** *vdom* = ‹*set p*›,
THEN fref-to-Down-curry6, *of x1a x1b x2g x1c x1q x1s x1t*,
THEN order-trans])
subgoal unfolding *merge-conflict-m-pre-def*
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def conc-fun-SPEC lookup-conflict-remove1-pre-def atms-of-def*
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def
merge-conflict-m-def lookup-conflict-remove1-def subset-mset.sup.commute[*of* - ‹*mset* (- ∝ -)›]
remove1-mset-union-distrib1 remove1-mset-union-distrib2
intro!: *mset-as-position-remove3*
intro!: *ASSERT-leI*)
done
done

show *?thesis*
supply [[*goals-limit* = *1*]]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
update-confl-tl-wl-pre-update-confl-tl-wl-pre′[*dest*!]
apply (*intro frefI nres-relI*)
subgoal for *CLS′ CLS*
apply (*cases CLS′*; *cases CLS*; *hypsubst+*)
unfolding *uncurry-def update-confl-tl-wl-heur-alt-def comp-def Let-def*
update-confl-tl-wl-def mop-update-confl-tl-wl-alt-def prod.case
apply (*refine-rcg*; *remove-dummy-vars*)
subgoal
by (*auto simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
RES-RETURN-RES RETURN-def counts-maximum-level-def)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def*)
apply (*rule rr*; *assumption*)
subgoal by (*simp add*: *arena-act-pre-def*)
subgoal using *arena-act-pre-mark-used* **by** *blast*

subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def*
intro!: *vmtf-unset-pre'*)
subgoal for *m n p q ra s t ha ia ja x1 x2 x1a x1b x1c x1d x1e x1f x1g x2g x1h x1i x1k x1l x2k x1m x1n x1o x1p x1q x1r x1s x1t D x1v x1w x2v x1x x1y*
by (*rule tl-trailt-tr-pre*[*of x1a* - ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)›])
(*clarsimp-all dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*
simp: *update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def*
intro!: *tl-trailt-tr-pre*)
subgoal by (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre'-def*
valid-arena-arena-incr-act valid-arena-mark-used subset-mset.sup.commute[*of* - ‹*remove1-mset* - -›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl'-def isa-vmtf-tl-isa-vmtf no-dup-tlD*
counts-maximum-level-def)
done
done
qed

lemma *phase-saving-le*: ‹*phase-saving* $\mathcal{A}$ φ $\Longrightarrow$ *A* $\in\#$ $\mathcal{A}$ $\Longrightarrow$ *A* < *length* φ›
‹*phase-saving* $\mathcal{A}$ φ $\Longrightarrow$ *B* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\Longrightarrow$ *atm-of B* < *length* φ›
by (*auto simp*: *phase-saving-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *isa-vmtf-le*:
‹((*a*, *b*), *M*) ∈ *isa-vmtf* $\mathcal{A}$ *M'* $\Longrightarrow$ *A* $\in\#$ $\mathcal{A}$ $\Longrightarrow$ *A* < *length a*›
‹((*a*, *b*), *M*) ∈ *isa-vmtf* $\mathcal{A}$ *M'* $\Longrightarrow$ *B* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\Longrightarrow$ *atm-of B* < *length a*›
by (*auto simp*: *isa-vmtf-def vmtf-def vmtf-*$\mathcal{L}_{all}$*-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *isa-vmtf-next-search-le*:
‹((*a*, *b*, *c*, *c'*, *Some d*), *M*) ∈ *isa-vmtf* $\mathcal{A}$ *M'* $\Longrightarrow$ *d* < *length a*›
by (*auto simp*: *isa-vmtf-def vmtf-def vmtf-*$\mathcal{L}_{all}$*-def atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)

lemma *trail-pol-nempty*: ‹¬(([], *aa*, *ab*, *ac*, *ad*, *b*), *L* # *ys*) ∈ *trail-pol* $\mathcal{A}$›
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

definition *is-decided-hd-trail-wl-heur* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*is-decided-hd-trail-wl-heur* = (λ*S*. *is-None* (*snd* (*last-trail-pol* (*get-trail-wl-heur S*))))›

lemma *is-decided-hd-trail-wl-heur-hd-get-trail*:
‹(*RETURN o is-decided-hd-trail-wl-heur*, *RETURN o* (λ*M*. *is-decided* (*hd* (*get-trail-wl M*))))
∈ [λ*M*. *get-trail-wl M* ≠ []]$_f$ *twl-st-heur-conflict-ana' r* → ⟨*bool-rel*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *is-decided-hd-trail-wl-heur-def twl-st-heur-conflict-ana-def neq-Nil-conv*
trail-pol-def ann-lits-split-reasons-def is-decided-no-proped-iff last-trail-pol-def
split: *option.splits*)

definition *is-decided-hd-trail-wl-heur-pre* **where**
‹*is-decided-hd-trail-wl-heur-pre* =
(λ*S*. *fst* (*get-trail-wl-heur S*) ≠ [] ∧ *last-trail-pol-pre* (*get-trail-wl-heur S*))›

definition *skip-and-resolve-loop-wl-D-heur-inv* **where**
‹*skip-and-resolve-loop-wl-D-heur-inv* S_0' =
(λ(*brk*, *S'*). ∃ *S* S_0. (*S'*, *S*) ∈ *twl-st-heur-conflict-ana* ∧ (S_0', S_0) ∈ *twl-st-heur-conflict-ana* ∧
skip-and-resolve-loop-wl-inv S_0 *brk S* ∧
length (*get-clauses-wl-heur S'*) = *length* (*get-clauses-wl-heur* S_0'))›

definition *update-confl-tl-wl-heur-pre*
:: ‹$(nat \times nat\ literal) \times twl\text{-}st\text{-}wl\text{-}heur \Rightarrow bool$›
where
‹*update-confl-tl-wl-heur-pre* =
$(\lambda((i, L), (M, N, D, W, Q, ((A, m, fst\text{-}As, lst\text{-}As, next\text{-}search), \text{-}), clvls, cach, lbd,$
$outl, \text{-})).$
$i > 0 \wedge$
$(fst\ M) \neq [] \wedge$
$atm\text{-}of\ ((last\ (fst\ M))) < length\ A \wedge (next\text{-}search \neq None \longrightarrow\ the\ next\text{-}search < length\ A) \wedge$
$L = (last\ (fst\ M))$
)›

definition *lit-and-ann-of-propagated-st-heur-pre* **where**
‹*lit-and-ann-of-propagated-st-heur-pre* = $(\lambda((M, \text{-}, \text{-}, reasons, \text{-}), \text{-}).\ atm\text{-}of\ (last\ M) < length\ reasons$
$\wedge\ M \neq [])$›

definition *atm-is-in-conflict-st-heur-pre*
:: ‹$nat\ literal \times twl\text{-}st\text{-}wl\text{-}heur \Rightarrow bool$›
where
‹*atm-is-in-conflict-st-heur-pre* $= (\lambda(L, (M,N,(\text{-}, (\text{-}, D)), \text{-})).\ atm\text{-}of\ L < length\ D)$›

definition *skip-and-resolve-loop-wl-D-heur*
:: ‹$twl\text{-}st\text{-}wl\text{-}heur \Rightarrow twl\text{-}st\text{-}wl\text{-}heur\ nres$›
where
‹*skip-and-resolve-loop-wl-D-heur* S_0 =
do {
(-, S) ←
$WHILE_T{}^{skip\text{-}and\text{-}resolve\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
$(\lambda(brk, S).\ \neg brk \wedge \neg is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur\ S)$
$(\lambda(brk, S).$
do {
$ASSERT(\neg brk \wedge \neg is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur\ S)$;
$(L, C) \leftarrow lit\text{-}and\text{-}ann\text{-}of\text{-}propagated\text{-}st\text{-}heur\ S$;
$b \leftarrow atm\text{-}is\text{-}in\text{-}conflict\text{-}st\text{-}heur\ (-L)\ S$;
if b then
tl-state-wl-heur S
else do {
$b \leftarrow maximum\text{-}level\text{-}removed\text{-}eq\text{-}count\text{-}dec\text{-}heur\ L\ S$;
if b
then do {
update-confl-tl-wl-heur L C S}
else
RETURN (*True*, S)
}
}
)
(*False*, S_0);
RETURN S
}
›

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*atm-is-in-conflict-st-heur*), *uncurry* (*mop-lit-notin-conflict-wl*)) ∈
$[\lambda(L, S).\ True]_f$

Id $\times_r$ *twl-st-heur-conflict-ana* $\rightarrow$ ⟨*Id*⟩ *nres-rel*›
proof −
have *1*: ‹*aaa* $\in\#$ $\mathcal{L}_{all}$ *A* $\Longrightarrow$ *atm-of aaa* $\in$ *atms-of* ($\mathcal{L}_{all}$ *A*)› **for** *aaa A*
by (*auto simp*: *atms-of-def*)
show *?thesis*
unfolding *atm-is-in-conflict-st-heur-def twl-st-heur-def option-lookup-clause-rel-def uncurry-def comp-def mop-lit-notin-conflict-wl-def twl-st-heur-conflict-ana-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
apply *clarsimp*
subgoal
apply (*rule atm-in-conflict-lookup-pre*)
unfolding $\mathcal{L}_{all}$*-all-atms-all-lits*[*symmetric*]
apply *assumption*+
done
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x1e x2d x2e*
apply (*subst atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*, *of* ‹*all-atms-st x2*› ‹*atm-of x1*› ‹*the* (*get-conflict-wl* (*snd y*))›])
apply (*simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def*)[]
apply (*auto simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def option-lookup-clause-rel-def*)[]
apply (*simp add*: *atm-in-conflict-def atm-of-in-atms-of-iff*)
done
done
qed

lemma *skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D*:
‹(*skip-and-resolve-loop-wl-D-heur*, *skip-and-resolve-loop-wl*)
$\in$ *twl-st-heur-conflict-ana*′ *r* $\rightarrow_f$ ⟨*twl-st-heur-conflict-ana*′ *r*⟩*nres-rel*›
proof −
have *H*[*refine0*]: ‹(*x*, *y*) $\in$ *twl-st-heur-conflict-ana* $\Longrightarrow$
((*False*, *x*), *False*, *y*)
$\in$ *bool-rel* $\times_f$
twl-st-heur-conflict-ana′ (*length* (*get-clauses-wl-heur x*))› **for** *x y*
by *auto*

show *?thesis*
supply [[*goals-limit=1*]]
unfolding *skip-and-resolve-loop-wl-D-heur-def skip-and-resolve-loop-wl-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
update-confl-tl-wl-heur-update-confl-tl-wl[*THEN fref-to-Down-curry2*, *unfolded comp-def*]
maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec
[*THEN fref-to-Down-curry*] *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*[*THEN fref-to-Down*]
tl-state-wl-heur-tl-state-wl[*THEN fref-to-Down*]
atm-is-in-conflict-st-heur-is-in-conflict-st[*THEN fref-to-Down-curry*])
subgoal by *fast*
subgoal for *S T brkS brkT*
unfolding *skip-and-resolve-loop-wl-D-heur-inv-def*
apply (*subst case-prod-beta*)
apply (*rule exI*[*of* - ‹*snd brkT*›])
apply (*rule exI*[*of* - ‹*T*›])
apply (*subst* (*asm*) (*1*) *surjective-pairing*[*of brkS*])
apply (*subst* (*asm*) *surjective-pairing*[*of brkT*])
unfolding *prod-rel-iff*
by *auto*

subgoal for *x y xa x′ x1 x2 x1a x2a*
apply (*subst is-decided-hd-trail-wl-heur-hd-get-trail*[*of r*, *THEN fref-to-Down-unRET-Id*, *of x2a*])
subgoal
unfolding *skip-and-resolve-loop-wl-inv-def skip-and-resolve-loop-inv-l-def skip-and-resolve-loop-inv-def*
apply (*subst* (*asm*) *case-prod-beta*)+
unfolding *prod.case*
apply *normalize-goal*+
by (*auto simp*:)
subgoal by *auto*
subgoal by *auto*
done
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition (**in** −) *get-count-max-lvls-code* **where**
‹*get-count-max-lvls-code* = (λ(-, -, -, -, -, -, -, *clvls*, -). *clvls*)›

lemma *is-decided-hd-trail-wl-heur-alt-def*:
‹*is-decided-hd-trail-wl-heur* = (λ(*M*, -). *is-None* (*snd* (*last-trail-pol M*)))›
by (*auto intro*!: *ext simp*: *is-decided-hd-trail-wl-heur-def*)

lemma *atm-of-in-atms-of*: ‹*atm-of x* ∈ *atms-of C* ⟷ *x* ∈# *C* ∨ −*x* ∈# *C*›
using *atm-of-notin-atms-of-iff* **by** *blast*

definition *atm-is-in-conflict* **where**
‹*atm-is-in-conflict L D* ⟷ *atm-of L* ∈ *atms-of* (*the D*)›

fun *is-in-option-lookup-conflict* **where**
is-in-option-lookup-conflict-def[*simp del*]:
‹*is-in-option-lookup-conflict L* (*a*, *n*, *xs*) ⟷ *is-in-lookup-conflict* (*n*, *xs*) *L*›

lemma *is-in-option-lookup-conflict-atm-is-in-conflict-iff*:
assumes
‹*ba* ≠ *None*› **and** *aa*: ‹*aa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *uaa*: ‹− *aa* ∉# *the ba*› **and**
‹((*b*, *c*, *d*), *ba*) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
shows ‹*is-in-option-lookup-conflict aa* (*b*, *c*, *d*) =
atm-is-in-conflict aa ba›
proof −
obtain *yb* **where** *ba*[*simp*]: ‹*ba* = *Some yb*›
using *assms* **by** *auto*

have *map*: ‹*mset-as-position d yb*› **and** *le*: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length d*› **and** [*simp*]: ‹¬*b*›
using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)

have *aa-d*: ‹*atm-of aa* < *length d*› **and** *uaa-d*: ‹*atm-of* (−*aa*) < *length d*›
using *le aa* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*)
from *mset-as-position-in-iff-nth*[*OF map aa-d*]
have *1*: ‹(*aa* ∈# *yb*) = (*d* ! *atm-of aa* = *Some* (*is-pos aa*))›
.

from *mset-as-position-in-iff-nth*[*OF map uaa-d*] **have** *2*: ‹(*d* ! *atm-of aa* ≠ *Some* (*is-pos* (−*aa*)))›
using *uaa* **by** *simp*

then show *?thesis*
using *uaa 1 2*
by (*auto simp*: *is-in-lookup-conflict-def is-in-option-lookup-conflict-def atm-is-in-conflict-def atm-of-in-atms-of is-neg-neg-not-is-neg*
split: *option.splits*)
qed

lemma *is-in-option-lookup-conflict-atm-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-option-lookup-conflict*), *uncurry* (*RETURN oo atm-is-in-conflict*))
∈ [λ(*L*, *D*). *D* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ −*L* ∉# *the D*]$_f$
Id $\times_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*case-tac x*, *case-tac y*)
by (*simp add*: *is-in-option-lookup-conflict-atm-is-in-conflict-iff*[*of - -* $\mathcal{A}$])

lemma *is-in-option-lookup-conflict-alt-def*:
‹*RETURN oo is-in-option-lookup-conflict* =
RETURN oo (λ*L* (-, *n*, *xs*). *is-in-lookup-conflict* (*n*, *xs*) *L*)›
by (*auto intro*!: *ext simp*: *is-in-option-lookup-conflict-def*)

lemma *skip-and-resolve-loop-wl-DI*:
assumes
‹*skip-and-resolve-loop-wl-D-heur-inv S* (*b*, *T*)›
shows ‹*is-decided-hd-trail-wl-heur-pre T*›
using *assms* **apply** −
unfolding *skip-and-resolve-loop-wl-inv-def skip-and-resolve-loop-inv-l-def skip-and-resolve-loop-inv-def skip-and-resolve-loop-wl-D-heur-inv-def is-decided-hd-trail-wl-heur-pre-def*
apply (*subst* (*asm*) *case-prod-beta*)+
unfolding *prod.case*
apply *normalize-goal*+
apply (*clarsimp simp*: *twl-st-heur-def state-wl-l-def twl-st-l-def twl-st-heur-conflict-ana-def trail-pol-alt-def last-trail-pol-pre-def last-rev hd-map literals-are-in-*$\mathcal{L}_{in}$*-trail-def simp flip*: *rev-map dest*: *multi-member-split*)
apply (*case-tac x*)
apply (*clarsimp-all dest*!: *multi-member-split simp*: *ann-lits-split-reasons-def*)
done

lemma *isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv*:
‹*isasat-fast x* ⟹
skip-and-resolve-loop-wl-D-heur-inv x
(*False*, *a2'*) ⟹ *isasat-fast a2'*›
unfolding *skip-and-resolve-loop-wl-D-heur-inv-def isasat-fast-def*
by *auto*

end
theory *IsaSAT-Conflict-Analysis-LLVM*

imports *IsaSAT-Conflict-Analysis IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM*
begin
thm *fold-tuple-optimizations*

lemma *get-count-max-lvls-heur-def*:
‹*get-count-max-lvls-heur* = (λ(-, -, -, -, -, -, *clvls*, -). *clvls*)›
by (*auto intro*!: *ext*)

sepref-def *get-count-max-lvls-heur-impl*
is ‹*RETURN o get-count-max-lvls-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
unfolding *get-count-max-lvls-heur-def isasat-bounded-assn-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *get-count-max-lvls-heur-impl.refine*

sepref-def *maximum-level-removed-eq-count-dec-fast-code*
is ‹*uncurry* (*maximum-level-removed-eq-count-dec-heur*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
unfolding *maximum-level-removed-eq-count-dec-heur-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

declare
maximum-level-removed-eq-count-dec-fast-code.refine[*sepref-fr-rules*]

lemma *is-decided-hd-trail-wl-heur-alt-def*:
‹*is-decided-hd-trail-wl-heur* = (λ((*M*, *xs*, *lvls*, *reasons*, *k*), -).
let r = *reasons* ! (*atm-of* (*last M*)) *in*
r = *DECISION-REASON*)›
unfolding *is-decided-hd-trail-wl-heur-def last-trail-pol-def*
by (*auto simp*: *is-decided-hd-trail-wl-heur-pre-def last-trail-pol-def*
Let-def intro!: *ext split*: *if-splits*)

sepref-def *is-decided-hd-trail-wl-fast-code*
is ‹*RETURN o is-decided-hd-trail-wl-heur*›
:: ‹$[is\text{-}decided\text{-}hd\text{-}trail\text{-}wl\text{-}heur\text{-}pre]_a\ isasat\text{-}bounded\text{-}assn^k \rightarrow bool1\text{-}assn$›
supply [[*goals-limit*=*1*]]
unfolding *is-decided-hd-trail-wl-heur-alt-def isasat-bounded-assn-def*
is-decided-hd-trail-wl-heur-pre-def last-trail-pol-def trail-pol-fast-assn-def
last-trail-pol-pre-def
by *sepref*

declare
is-decided-hd-trail-wl-fast-code.refine[*sepref-fr-rules*]

sepref-def *lit-and-ann-of-propagated-st-heur-fast-code*
is ‹*lit-and-ann-of-propagated-st-heur*›
:: ‹$[\lambda\text{-}.\ True]_a$
$isasat\text{-}bounded\text{-}assn^k \rightarrow (unat\text{-}lit\text{-}assn \times_a sint64\text{-}nat\text{-}assn)$›
supply [[*goals-limit*=*1*]]
supply *get-trail-wl-heur-def*[*simp*]
unfolding *lit-and-ann-of-propagated-st-heur-def isasat-bounded-assn-def*
lit-and-ann-of-propagated-st-heur-pre-def trail-pol-fast-assn-def
unfolding *fold-tuple-optimizations*

by *sepref*

declare
lit-and-ann-of-propagated-st-heur-fast-code.refine[*sepref-fr-rules*]

definition *is-UNSET* **where** [*simp*]: ‹*is-UNSET* $x \longleftrightarrow x = \mathit{UNSET}$›
lemma *tri-bool-is-UNSET-refine-aux*:
‹($\lambda x.\ x = 0$, *is-UNSET*) $\in$ *tri-bool-rel-aux* $\rightarrow$ *bool-rel* ›
by (*auto simp*: *tri-bool-rel-aux-def*)

sepref-definition *is-UNSET-impl*
is ‹*RETURN o* ($\lambda x.\ x= 0$)›
:: ‹$(\mathit{unat\text{-}assn}'\ \mathit{TYPE}(8))^k \rightarrow_a \mathit{bool1\text{-}assn}$›
apply (*annot-unat-const* $\mathit{TYPE}(8)$)
by *sepref*

sepref-def *is-in-option-lookup-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-option-lookup-conflict*)›
:: ‹$[\lambda(L, (c, n, xs)).\ \mathit{atm\text{-}of}\ L < \mathit{length}\ xs]_a$
$\mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{conflict\text{-}option\text{-}rel\text{-}assn}^k \rightarrow \mathit{bool1\text{-}assn}$›
unfolding *is-in-option-lookup-conflict-alt-def is-in-lookup-conflict-def PROTECT-def*
is-NOTIN-alt-def[*symmetric*] *conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*

sepref-def *atm-is-in-conflict-st-heur-fast-code*
is ‹*uncurry* (*atm-is-in-conflict-st-heur*)›
:: ‹$[\lambda\text{-}.\ \mathit{True}]_a\ \mathit{unat\text{-}lit\text{-}assn}^k *_a \mathit{isasat\text{-}bounded\text{-}assn}^k \rightarrow \mathit{bool1\text{-}assn}$›
supply [[*goals-limit=1*]]
unfolding *atm-is-in-conflict-st-heur-def atm-is-in-conflict-st-heur-pre-def isasat-bounded-assn-def*
atm-in-conflict-lookup-def trail-pol-fast-assn-def NOTIN-def[*symmetric*]
is-NOTIN-def[*symmetric*] *conflict-option-rel-assn-def lookup-clause-rel-assn-def*
unfolding *fold-tuple-optimizations atm-in-conflict-lookup-pre-def*
by *sepref*

declare *atm-is-in-conflict-st-heur-fast-code.refine*[*sepref-fr-rules*]

sepref-def (**in** −) *lit-of-last-trail-fast-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹$[\lambda(M).\ \mathit{fst}\ M \neq []]_a\ \mathit{trail\text{-}pol\text{-}fast\text{-}assn}^k \rightarrow \mathit{unat\text{-}lit\text{-}assn}$›
unfolding *lit-of-last-trail-pol-def trail-pol-fast-assn-def*
by *sepref*

declare *lit-of-last-trail-fast-code.refine*[*sepref-fr-rules*]

lemma *tl-state-wl-heurI*: ‹*tl-state-wl-heur-pre* $(a, b) \Longrightarrow \mathit{fst}\ a \neq []$›
‹*tl-state-wl-heur-pre* $(a, b) \Longrightarrow$ *tl-trailt-tr-pre* a›
‹*tl-state-wl-heur-pre* $(a1', a1'a, a1'b, a1'c, a1'd, a1'e, a1'f, a2'f) \Longrightarrow$
vmtf-unset-pre (*atm-of* (*lit-of-last-trail-pol* $a1'$)) $a1'e$›
by (*auto simp*: *tl-state-wl-heur-pre-def tl-trailt-tr-pre-def*

vmtf-unset-pre-def lit-of-last-trail-pol-def)

lemma *tl-state-wl-heur-alt-def*:
⟨*tl-state-wl-heur* = (λ(*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*). *do* {
ASSERT(*tl-state-wl-heur-pre* (*M*, *N*, *D*, *WS*, *Q*, *vmtf*, φ, *clvls*));
let L = (*atm-of* (*lit-of-last-trail-pol M*));
RETURN (*False*, (*tl-trailt-tr M*, *N*, *D*, *WS*, *Q*, *isa-vmtf-unset L vmtf*, φ, *clvls*))
})⟩
by (*auto simp*: *tl-state-wl-heur-def*)

sepref-def *tl-state-wl-heur-fast-code*
is ⟨*tl-state-wl-heur*⟩
:: ⟨$[\lambda\text{-}.\ True]_a$ *isasat-bounded-assn*^{d} → *bool1-assn* $\times_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *if-splits*[*split*] *tl-state-wl-heurI*[*simp*]
unfolding *tl-state-wl-heur-alt-def*[*abs-def*] *isasat-bounded-assn-def get-trail-wl-heur-def vmtf-unset-def bind-ref-tag-def short-circuit-conv*
unfolding *fold-tuple-optimizations*
apply (*rewrite* **in** ⟨*ASSERT* ¤⟩ *fold-tuple-optimizations*[*symmetric*])+
by *sepref*

declare
tl-state-wl-heur-fast-code.refine[*sepref-fr-rules*]

definition *None-lookup-conflict* :: ⟨- ⇒ - ⇒ *conflict-option-rel*⟩ **where**
⟨*None-lookup-conflict b xs* = (*b*, *xs*)⟩

sepref-def *None-lookup-conflict-impl*
is ⟨*uncurry* (*RETURN oo None-lookup-conflict*)⟩
:: ⟨*bool1-assn*^{k} $*_a$ *lookup-clause-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*⟩
unfolding *None-lookup-conflict-def conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*

sepref-register *None-lookup-conflict*
declare *None-lookup-conflict-impl.refine*[*sepref-fr-rules*]

definition *extract-valuse-of-lookup-conflict* :: ⟨*conflict-option-rel* ⇒ *bool*⟩ **where**
⟨*extract-valuse-of-lookup-conflict* = (λ(*b*, (-, *xs*)). *b*)⟩

sepref-def *extract-valuse-of-lookup-conflict-impl*
is ⟨*RETURN o extract-valuse-of-lookup-conflict*⟩
:: ⟨*conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *extract-valuse-of-lookup-conflict-def conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*

sepref-register *extract-valuse-of-lookup-conflict*
declare *extract-valuse-of-lookup-conflict-impl.refine*[*sepref-fr-rules*]

sepref-register *isasat-lookup-merge-eq2 update-confl-tl-wl-heur*

lemma *update-confl-tl-wl-heur-alt-def*:
⟨*update-confl-tl-wl-heur* = (λ*L C* (*M*, *N*, *bnxs*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*). *do* {

ASSERT (*clvls* ≥ *1*);
let L′ = atm-of L;
ASSERT(*arena-is-valid-clause-idx N C*);
(*bnxs*, *clvls*, *lbd*, *outl*) ←
if arena-length N C = 2 then isasat-lookup-merge-eq2 L M N C bnxs clvls lbd outl
else isa-resolve-merge-conflict-gt2 M N C bnxs clvls lbd outl;
let b = extract-valuse-of-lookup-conflict bnxs;
let nxs = the-lookup-conflict bnxs;
ASSERT(*curry lookup-conflict-remove1-pre L nxs* ∧ *clvls* ≥ *1*);
let nxs = lookup-conflict-remove1 L nxs;
ASSERT(*arena-act-pre N C*);
let N = mark-used N C;
ASSERT(*arena-act-pre N C*);
let N = arena-incr-act N C;
ASSERT(*vmtf-unset-pre L′ vm*);
ASSERT(*tl-trailt-tr-pre M*);
RETURN (*False*, (*tl-trailt-tr M*, *N*, (*None-lookup-conflict b nxs*), *Q*, *W*, *isa-vmtf-unset L′ vm*,
clvls − *1*, *cach*, *lbd*, *outl*, *stats*))
})›
unfolding *update-confl-tl-wl-heur-def*
by (*auto intro*!: *ext bind-cong simp*: *None-lookup-conflict-def the-lookup-conflict-def*
extract-valuse-of-lookup-conflict-def Let-def)

sepref-def *update-confl-tl-wl-fast-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹[λ((*i*, *L*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$*isasat-bounded-assn*^{d} → *bool1-assn* $\times_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *isasat-fast-length-leD*[*intro*]
unfolding *update-confl-tl-wl-heur-alt-def isasat-bounded-assn-def*
PR-CONST-def
apply (*rewrite at* ‹*If* (- = ¤)› *snat-const-fold*[**where** ′*a=64*])
apply (*annot-unat-const TYPE* (*32*))
unfolding *fold-tuple-optimizations*
by *sepref*

declare *update-confl-tl-wl-fast-code.refine*[*sepref-fr-rules*]

sepref-register *is-in-conflict-st atm-is-in-conflict-st-heur*
sepref-def *skip-and-resolve-loop-wl-D-fast*
is ‹*skip-and-resolve-loop-wl-D-heur*›
:: ‹[λ*S*. *isasat-fast S*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
skip-and-resolve-loop-wl-DI[*intro*]
isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv[*intro*]
unfolding *skip-and-resolve-loop-wl-D-heur-def*
unfolding *fold-tuple-optimizations*
apply (*rewrite at* ‹¬- ∧ ¬ -› *short-circuit-conv*)
by *sepref*

declare *skip-and-resolve-loop-wl-D-fast.refine*[*sepref-fr-rules*]

experiment
begin
export-llvm
get-count-max-lvls-heur-impl
maximum-level-removed-eq-count-dec-fast-code

is-decided-hd-trail-wl-fast-code
lit-and-ann-of-propagated-st-heur-fast-code
is-in-option-lookup-conflict-code
atm-is-in-conflict-st-heur-fast-code
lit-of-last-trail-fast-code
tl-state-wl-heur-fast-code
None-lookup-conflict-impl
extract-valuse-of-lookup-conflict-impl
update-confl-tl-wl-fast-code
skip-and-resolve-loop-wl-D-fast

end

end
theory *IsaSAT-Propagate-Conflict*
imports *IsaSAT-Setup IsaSAT-Inner-Propagation*
begin

Chapter 16

Propagation Loop And Conflict

16.1 Unit Propagation, Inner Loop

definition (**in** −) *length-ll-fs* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs* = (λ(-, -, -, -, -, -, -, -, *W*) *L*. *length* (*W L*))›

definition (**in** −) *length-ll-fs-heur* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs-heur S L* = *length* (*watched-by-int S L*)›

lemma *length-ll-fs-heur-alt-def*:
‹*length-ll-fs-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -) *L*. *length* (*W* ! *nat-of-lit L*))›
unfolding *length-ll-fs-heur-def*
apply (*intro ext*)
apply (*case-tac S*)
by *auto*

lemma (**in** −) *get-watched-wl-heur-def*: ‹*get-watched-wl-heur* = (λ(*M*, *N*, *D*, *Q*, *W*, -). *W*)›
by (*intro ext*, *rename-tac x*, *case-tac x*) *auto*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur b*) ≤ *uint64-max* ⟹
unit-propagation-inner-loop-wl-loop-D-heur-inv b a (*a1'*, *a1'a*, *a2'a*) ⟹
length (*get-clauses-wl-heur a2'a*) ≤ *uint64-max*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv-def*
by *auto*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-alt-def*:
‹*unit-propagation-inner-loop-wl-loop-D-heur L* S_0 = *do* {
ASSERT (*length* (*watched-by-int* S_0 *L*) ≤ *length* (*get-clauses-wl-heur* S_0));
n ← *mop-length-watched-by-int* S_0 *L*;
let b = (*0*, *0*, S_0);
$WHILE_T{}^{unit\text{-}propagation\text{-}inner\text{-}loop\text{-}wl\text{-}loop\text{-}D\text{-}heur\text{-}inv\ S_0\ L}$
(λ(*j*, *w*, *S*). *w* < *n* ∧ *get-conflict-wl-is-None-heur S*)
(λ(*j*, *w*, *S*). *do* {
unit-propagation-inner-loop-body-wl-heur L j w S
})
b
}›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def Let-def* **..**

16.2 Unit propagation, Outer Loop

lemma *select-and-remove-from-literals-to-update-wl-heur-alt-def*:
⟨*select-and-remove-from-literals-to-update-wl-heur* =
(λ(M', N', D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*). *do* {
ASSERT(j < *length* (*fst* M'));
ASSERT(j + *1* ≤ *uint32-max*);
L ← *isa-trail-nth* M' j;
RETURN ((M', N', D', j+*1*, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*), −L)
})
⟩
unfolding *select-and-remove-from-literals-to-update-wl-heur-def*
apply (*intro ext*)
apply (*rename-tac S*; *case-tac S*)
by (*auto intro*!: *ext simp*: *rev-trail-nth-def Let-def*)

definition *literals-to-update-wl-literals-to-update-wl-empty* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*literals-to-update-wl-literals-to-update-wl-empty S* ⟷
literals-to-update-wl-heur S < *isa-length-trail* (*get-trail-wl-heur S*)⟩

lemma *literals-to-update-wl-literals-to-update-wl-empty-alt-def*:
⟨*literals-to-update-wl-literals-to-update-wl-empty* =
(λ(M', N', D', j, W', *vm*, φ, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fast-ema*, *slow-ema*, *ccount*,
vdom, *lcount*). j < *isa-length-trail* M')⟩
unfolding *literals-to-update-wl-literals-to-update-wl-empty-def isa-length-trail-def*
by (*auto intro*!: *ext split*: *prod.splits*)

lemma *unit-propagation-outer-loop-wl-D-invI*:
⟨*unit-propagation-outer-loop-wl-D-heur-inv* S_0 S ⟹
isa-length-trail-pre (*get-trail-wl-heur S*)⟩
unfolding *unit-propagation-outer-loop-wl-D-heur-inv-def*
by *blast*

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
⟨*length* (*get-clauses-wl-heur x*) ≤ *uint64-max* ⟹
unit-propagation-outer-loop-wl-D-heur-inv x s' ⟹
length (*get-clauses-wl-heur a1'*) =
length (*get-clauses-wl-heur s'*) ⟹
length (*get-clauses-wl-heur s'*) ≤ *uint64-max*⟩
by (*auto simp*: *unit-propagation-outer-loop-wl-D-heur-inv-def*)

end
theory *IsaSAT-Propagate-Conflict-LLVM*
imports *IsaSAT-Propagate-Conflict IsaSAT-Inner-Propagation-LLVM*
begin

lemma *length-ll*[*def-pat-rules*]: ⟨*length-ll*\$*xs*\$*i* ≡ *op-list-list-llen*\$*xs*\$*i*⟩
by (*auto simp*: *op-list-list-llen-def length-ll-def*)

sepref-def *length-ll-fs-heur-fast-code*
is ⟨*uncurry* (*RETURN oo length-ll-fs-heur*)⟩

:: ‹[λ(S, L). *nat-of-lit L* < *length* (*get-watched-wl-heur S*)]$_a$
isasat-bounded-assn^k $*_a$ *unat-lit-assn*^{k} → *sint64-nat-assn*›
unfolding *length-ll-fs-heur-alt-def get-watched-wl-heur-def isasat-bounded-assn-def*
length-ll-def[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

sepref-def *mop-length-watched-by-int-impl* [*llvm-inline*]
is ‹*uncurry mop-length-watched-by-int*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
unfolding *mop-length-watched-by-int-alt-def isasat-bounded-assn-def*
length-ll-def[*symmetric*]
supply [[*goals-limit=1*]]
by *sepref*

sepref-register *unit-propagation-inner-loop-body-wl-heur*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur b*) ≤ *sint64-max* ⟹
unit-propagation-inner-loop-wl-loop-D-heur-inv b a (*a1'*, *a1'a*, *a2'a*) ⟹
length (*get-clauses-wl-heur a2'a*) ≤ *sint64-max*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv-def*
by *auto*

sepref-def *unit-propagation-inner-loop-wl-loop-D-fast*
is ‹*uncurry unit-propagation-inner-loop-wl-loop-D-heur*›
:: ‹[λ(L, S). *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$
unat-lit-assn^k $*_a$ *isasat-bounded-assn*^{d} → *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def PR-CONST-def*
unfolding *watched-by-nth-watched-app watched-app-def*[*symmetric*]
length-ll-fs-heur-def[*symmetric*]
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
is-None-def[*symmetric*] *get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
length-ll-fs-def[*symmetric*]
unfolding *fold-tuple-optimizations*
supply [[*goals-limit=1*]] *unit-propagation-inner-loop-wl-loop-D-heur-fast*[*intro*] *length-ll-fs-heur-def*[*simp*]
apply (*annot-snat-const TYPE* (*64*))
by *sepref*

lemma *le-uint64-max-minus-4-uint64-max*: ‹*a* ≤ *sint64-max* − *4* ⟹ *Suc a* < *max-snat 64*›
by (*auto simp*: *sint64-max-def max-snat-def*)

definition *cut-watch-list-heur2-inv* **where**
‹*cut-watch-list-heur2-inv L n* = (λ(*j*, *w*, *W*). *j* ≤ *w* ∧ *w* ≤ *n* ∧ *nat-of-lit L* < *length W*)›

lemma *cut-watch-list-heur2-alt-def*:
‹*cut-watch-list-heur2* = (λ*j w L* (*M*, *N*, *D*, *Q*, *W*, *oth*). *do* {
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let n = *length* (*W*!(*nat-of-lit L*));
(*j*, *w*, *W*) ← $\mathit{WHILE}_T{}^{\mathit{cut\text{-}watch\text{-}list\text{-}heur2\text{-}inv}\ L\ n}$
(λ(*j*, *w*, *W*). *w* < *n*)
(λ(*j*, *w*, *W*). *do* {
ASSERT(*w* < *length* (*W*!(*nat-of-lit L*)));
RETURN (*j+1*, *w+1*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*j* := *W*!(*nat-of-lit L*)!*w*]])

})
(*j*, *w*, *W*);
ASSERT($j \leq$ *length* (*W* ! *nat-of-lit L*) $\wedge$ *nat-of-lit L* $<$ *length W*);
let W = *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*)];
RETURN (*M*, *N*, *D*, *Q*, *W*, *oth*)
})›
unfolding *cut-watch-list-heur2-inv-def cut-watch-list-heur2-def*
by *auto*

lemma *cut-watch-list-heur2I*:
‹*length* (*a1'd* ! *nat-of-lit baa*) $\leq$ *sint64-max* $-$ *4* $\Longrightarrow$
cut-watch-list-heur2-inv baa (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) $\Longrightarrow$
a1'f $<$ *length-ll a2'f* (*nat-of-lit baa*) $\Longrightarrow$
ez $\leq$ *bba* $\Longrightarrow$
Suc a1'e $<$ *max-snat 64*›
‹*length* (*a1'd* ! *nat-of-lit baa*) $\leq$ *sint64-max* $-$ *4* $\Longrightarrow$
cut-watch-list-heur2-inv baa (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) $\Longrightarrow$
a1'f $<$ *length-ll a2'f* (*nat-of-lit baa*) $\Longrightarrow$
ez $\leq$ *bba* $\Longrightarrow$
Suc a1'f $<$ *max-snat 64*›
‹*cut-watch-list-heur2-inv baa* (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) $\Longrightarrow$ *nat-of-lit baa* $<$ *length a2'f*›
‹*cut-watch-list-heur2-inv baa* (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) $\Longrightarrow$ *a1'f* $<$ *length-ll a2'f* (*nat-of-lit baa*) $\Longrightarrow$
a1'e $<$ *length* (*a2'f* ! *nat-of-lit baa*)›
by (*auto simp*: *max-snat-def sint64-max-def cut-watch-list-heur2-inv-def length-ll-def*)

sepref-def *cut-watch-list-heur2-fast-code*
is ‹*uncurry3 cut-watch-list-heur2*›
:: ‹[λ(((*j*, *w*), *L*), *S*). *length* (*watched-by-int S L*) $\leq$ *sint64-max*$-$*4*]$_a$
sint64-nat-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *unat-lit-assn*^{k} $*_a$
isasat-bounded-assn^d $\rightarrow$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *cut-watch-list-heur2I*[*intro*] *length-ll-def*[*simp*]
unfolding *cut-watch-list-heur2-alt-def isasat-bounded-assn-def length-ll-def*[*symmetric*]
nth-rll-def[*symmetric*]
op-list-list-take-alt-def[*symmetric*]
op-list-list-upd-alt-def[*symmetric*]
unfolding *fold-tuple-optimizations*
apply (*annot-snat-const TYPE* (*64*))
by *sepref*

sepref-def *unit-propagation-inner-loop-wl-D-fast-code*
is ‹*uncurry unit-propagation-inner-loop-wl-D-heur*›
:: ‹[λ(*L*, *S*). *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$
unat-lit-assn^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *PR-CONST-def unit-propagation-inner-loop-wl-D-heur-def*
by *sepref*

sepref-def *select-and-remove-from-literals-to-update-wlfast-code*
is ‹*select-and-remove-from-literals-to-update-wl-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn* $\times_a$ *unat-lit-assn*›

supply [[*goals-limit=1*]]
unfolding *select-and-remove-from-literals-to-update-wl-heur-alt-def isasat-bounded-assn-def*
unfolding *fold-tuple-optimizations*
supply [[*goals-limit* = *1*]]
apply (*annot-snat-const TYPE* (*64*))
by *sepref*

sepref-def *literals-to-update-wl-literals-to-update-wl-empty-fast-code*
is ⟨*RETURN o literals-to-update-wl-literals-to-update-wl-empty*⟩
:: ⟨[λS. *isa-length-trail-pre* (*get-trail-wl-heur S*)]$_a$ *isasat-bounded-assn*^{k} $\rightarrow$ *bool1-assn*⟩
unfolding *literals-to-update-wl-literals-to-update-wl-empty-alt-def*
isasat-bounded-assn-def
by *sepref*

sepref-register *literals-to-update-wl-literals-to-update-wl-empty*
select-and-remove-from-literals-to-update-wl-heur

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
⟨*length* (*get-clauses-wl-heur x*) $\leq$ *sint64-max* $\Longrightarrow$
unit-propagation-outer-loop-wl-D-heur-inv x s' $\Longrightarrow$
length (*get-clauses-wl-heur a1'*) =
length (*get-clauses-wl-heur s'*) $\Longrightarrow$
length (*get-clauses-wl-heur s'*) $\leq$ *sint64-max*⟩
by (*auto simp*: *unit-propagation-outer-loop-wl-D-heur-inv-def*)

sepref-def *unit-propagation-outer-loop-wl-D-fast-code*
is ⟨*unit-propagation-outer-loop-wl-D-heur*⟩
:: ⟨[λS. *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *unit-propagation-outer-loop-wl-D-heur-fast*[*intro*]
unit-propagation-outer-loop-wl-D-invI[*intro*]
unfolding *unit-propagation-outer-loop-wl-D-heur-def*
literals-to-update-wl-literals-to-update-wl-empty-def[*symmetric*]
by *sepref*

experiment begin

export-llvm
length-ll-fs-heur-fast-code
unit-propagation-inner-loop-wl-loop-D-fast
cut-watch-list-heur2-fast-code
unit-propagation-inner-loop-wl-D-fast-code
isa-trail-nth-fast-code
select-and-remove-from-literals-to-update-wlfast-code
literals-to-update-wl-literals-to-update-wl-empty-fast-code
unit-propagation-outer-loop-wl-D-fast-code

end

end
theory *IsaSAT-Decide*
imports *IsaSAT-Setup IsaSAT-VMTF*
begin

Chapter 17

Decide

lemma (**in** $-$)*not-is-None-not-None*: ‹$\neg$*is-None s* $\Longrightarrow$ *s* $\neq$ *None*›
  by (*auto split*: *option.splits*)

definition *vmtf-find-next-undef-upd*
  :: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*)*ann-lits* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$
      (((*nat*,*nat*)*ann-lits* $\times$ *vmtf-remove-int*) $\times$ *nat option*)*nres*›
where
  ‹*vmtf-find-next-undef-upd* $\mathcal{A}$ = (λ*M vm*. *do*{
    L $\leftarrow$ *vmtf-find-next-undef* $\mathcal{A}$ *vm M*;
    RETURN ((*M*, *update-next-search L vm*), *L*)
  })›

definition *isa-vmtf-find-next-undef-upd*
  :: ‹*trail-pol* $\Rightarrow$ *isa-vmtf-remove-int* $\Rightarrow$
      ((*trail-pol* $\times$ *isa-vmtf-remove-int*) $\times$ *nat option*)*nres*›
where
  ‹*isa-vmtf-find-next-undef-upd* = (λ*M vm*. *do*{
    L $\leftarrow$ *isa-vmtf-find-next-undef vm M*;
    RETURN ((*M*, *update-next-search L vm*), *L*)
  })›

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
  ‹(*uncurry isa-vmtf-find-next-undef-upd*, *uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$)) $\in$
     trail-pol $\mathcal{A}$ $\times_r$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\rightarrow_f$
       ⟨*trail-pol* $\mathcal{A}$ $\times_f$ (*Id* $\times_r$ *distinct-atoms-rel* $\mathcal{A}$) $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
  unfolding *isa-vmtf-find-next-undef-upd-def vmtf-find-next-undef-upd-def uncurry-def*
    defined-atm-def[*symmetric*]
  apply (*intro frefI nres-relI*)
  apply (*refine-rcg isa-vmtf-find-next-undef-vmtf-find-next-undef*[*THEN fref-to-Down-curry*])
  subgoal by *auto*
  subgoal by (*auto simp*: *update-next-search-def split*: *prod.splits*)
  done

definition *lit-of-found-atm* **where**
‹*lit-of-found-atm* φ *L* = *SPEC* (λ*K*. (*L* = *None* $\longrightarrow$ *K* = *None*) $\wedge$
  (*L* $\neq$ *None* $\longrightarrow$ *K* $\neq$ *None* $\wedge$ *atm-of* (*the K*) = *the L*))›

definition *find-undefined-atm*
  :: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*) *ann-lits* $\Rightarrow$ *vmtf-remove-int* $\Rightarrow$
     (((*nat*,*nat*) *ann-lits* $\times$ *vmtf-remove-int*) $\times$ *nat option*) *nres*›
where

‹*find-undefined-atm* $\mathcal{A}$ *M* - = *SPEC*(λ((*M'*, *vm*), *L*).
  (*L* ≠ *None* ⟶ *Pos* (*the L*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *undefined-atm M* (*the L*)) ∧
  (*L* = *None* ⟶ (∀ *K*∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M K*)) ∧ *M* = *M'* ∧ *vm* ∈ *vmtf* $\mathcal{A}$ *M*)›

definition *lit-of-found-atm-D-pre* **where**
‹*lit-of-found-atm-D-pre* = (λ(φ, *L*). *L* ≠ *None* ⟶ (*the L* < *length* φ ∧ *the L* ≤ *uint32-max div 2*))›

definition *find-unassigned-lit-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ (*twl-st-wl-heur* × *nat literal option*) *nres*›
where
‹*find-unassigned-lit-wl-D-heur* = (λ(*M*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
    vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
  ((*M*, *vm*), *L*) ← *isa-vmtf-find-next-undef-upd M vm*;
  ASSERT(*L* ≠ *None* ⟶ *get-saved-phase-heur-pre* (*the L*) *heur*);
  L ← *lit-of-found-atm heur L*;
  RETURN ((*M*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
    vdom, *avdom*, *lcount*, *opts*, *old-arena*), *L*)
})›

lemma *lit-of-found-atm-D-pre*:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹ (*L* ≠ *None* ⟹ *the L* ∈# $\mathcal{A}$) ⟹
  L ≠ *None* ⟹ *get-saved-phase-heur-pre* (*the L*) *heur*›
by (*auto simp*: *lit-of-found-atm-D-pre-def phase-saving-def heuristic-rel-def phase-save-heur-rel-def*
  get-saved-phase-heur-pre-def
  atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff dest*: *bspec*[*of* - - ‹*Pos* (*the L*)›])

definition *find-unassigned-lit-wl-D-heur-pre* **where**
‹*find-unassigned-lit-wl-D-heur-pre S* ⟷
  (
    ∃ *T U*.
      (*S*, *T*) ∈ *state-wl-l None* ∧
      (*T*, *U*) ∈ *twl-st-l None* ∧
      twl-struct-invs U ∧
      literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ∧
      get-conflict-wl S = *None*
)›

lemma *vmtf-find-next-undef-upd*:
‹(*uncurry* (*vmtf-find-next-undef-upd* $\mathcal{A}$), *uncurry* (*find-undefined-atm* $\mathcal{A}$)) ∈
  [λ(*M*, *vm*). *vm* ∈ *vmtf* $\mathcal{A}$ *M*]$_f$ *Id* $\times_f$ *Id* → ⟨*Id* $\times_f$ *Id* $\times_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel*›
unfolding *vmtf-find-next-undef-upd-def find-undefined-atm-def*
  update-next-search-def uncurry-def
apply (*intro frefI nres-relI*)
apply (*clarify*)
apply (*rule bind-refine-spec*)
 prefer *2*
 apply (*rule vmtf-find-next-undef-ref*[*simplified*])
by (*auto intro*!: *RETURN-SPEC-refine simp*: *image-image defined-atm-def*[*symmetric*])

lemma *find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D*:
‹(*find-unassigned-lit-wl-D-heur*, *find-unassigned-lit-wl*) ∈
  [*find-unassigned-lit-wl-D-heur-pre*]$_f$
  twl-st-heur''' *r* → ⟨{((*T*, *L*), (*T'*, *L'*)). (*T*, *T'*) ∈ *twl-st-heur'''* *r* ∧ *L* = *L'* ∧
    (*L* ≠ *None* ⟶ *undefined-lit* (*get-trail-wl T'*) (*the L*) ∧ *the L* ∈# $\mathcal{L}_{all}$ (*all-atms-st T'*)) ∧
    get-conflict-wl T' = *None*}⟩*nres-rel*›
proof −

have [*simp*]: ‹*undefined-lit M* (*Pos* (*atm-of y*)) = *undefined-lit M y*› **for** *M y*
by (*auto simp*: *defined-lit-map*)
have [*simp*]: ‹*defined-atm M* (*atm-of y*) = *defined-lit M y*› **for** *M y*
by (*auto simp*: *defined-lit-map defined-atm-def*)

have *ID-R*: ‹*Id* $\times_r$ ⟨*Id*⟩*option-rel* = *Id*›
by *auto*
have *atms*: ‹*atms-of* ($\mathcal{L}_{all}$ (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*))) =
atms-of-mm (*mset* ‘# *init-clss-lf N*) ∪
atms-of-mm NE ∪ *atms-of-mm NS* ∧ *D* = *None*› (**is** *?A*) **and**
atms-2: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms N* (*NE* + *UE* + *NS* + *US*))) = *set-mset* ($\mathcal{L}_{all}$ (*all-atms N* (*NE*+*NS*)))› (**is** *?B*) **and**
atms-3: ‹*y* ∈ *atms-of-ms* ((λ*x*. *mset* (*fst x*)) ‘ *set-mset* (*ran-m N*)) ⟹
y ∉ *atms-of-mm NE* ⟹ *y* ∉ *atms-of-mm NS* ⟹
y ∈ *atms-of-ms* ((λ*x*. *mset* (*fst x*)) ‘ {*a*. *a* ∈# *ran-m N* ∧ *snd a*})› (**is** ‹*?C1* ⟹ *?C2* ⟹*?C3* ⟹ *?C*›)
if *inv*: ‹*find-unassigned-lit-wl-D-heur-pre* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*)›
for *M N D NE UE WS Q y NS US*
proof −
obtain *T U* **where**
S-T: ‹((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*), *T*) ∈ *state-wl-l None*› **and**
T-U: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
inv: ‹*twl-struct-invs U*› **and**
$\mathcal{A}_{in}$: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*)) (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*)› **and**
confl: ‹*get-conflict-wl* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*) = *None*›
using *inv* **unfolding** *find-unassigned-lit-wl-D-heur-pre-def*
apply − **apply** *normalize-goal*+
by *blast*

have ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)› **and**
unit: ‹*entailed-clss-inv U*›
using *inv* **unfolding** *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast*+
then show *?A*
using $\mathcal{A}_{in}$ *confl S-T T-U* **unfolding** *is-*$\mathcal{L}_{all}$*-alt-def state-wl-l-def twl-st-l-def*
literals-are-$\mathcal{L}_{in}$*-def all-atms-def all-lits-def*
apply −
apply (*subst* (*asm*) *all-clss-l-ran-m*[*symmetric*], *subst* (*asm*) *image-mset-union*)+
apply (*subst all-clss-l-ran-m*[*symmetric*], *subst image-mset-union*)
by (*auto simp*: *cdcl*$_W$*-restart-mset.no-strange-atm-def entailed-clss-inv.simps*
mset-take-mset-drop-mset mset-take-mset-drop-mset′ atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *all-lits-def*
clauses-def all-lits-of-mm-union atm-of-all-lits-of-mm
simp del: *entailed-clss-inv.simps*)

then show *?B* **and** ‹*?C1* ⟹ *?C2* ⟹ *?C3* ⟹ *?C*›
apply −
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *all-atms-def all-lits-def*
apply (*subst* (*asm*) *all-clss-l-ran-m*[*symmetric*], *subst* (*asm*) *image-mset-union*)+
apply (*subst all-clss-l-ran-m*[*symmetric*], *subst image-mset-union*)+
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff*
all-lits-of-mm-union atms-of-def $\mathcal{L}_{all}$*-union image-Un atm-of-eq-atm-of*
atm-of-all-lits-of-mm atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
qed

define *unassigned-atm* **where**
‹$unassigned\text{-}atm\ S\ L \equiv \exists\ M\ N\ D\ NE\ UE\ NS\ US\ WS\ Q.$
$S = (M, N, D, NE, UE, NS, US, WS, Q) \wedge$
$(L \neq None \longrightarrow$
$undefined\text{-}lit\ M\ (the\ L) \wedge the\ L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge$
$atm\text{-}of\ (the\ L) \in atms\text{-}of\text{-}mm\ (mset\ \text{‘}\#\ ran\text{-}mf\ N + (NE{+}UE) + (NS{+}US))) \wedge$
$(L = None \longrightarrow (\nexists L'.\ undefined\text{-}lit\ M\ L' \wedge$
$atm\text{-}of\ L' \in atms\text{-}of\text{-}mm\ (mset\ \text{‘}\#\ ran\text{-}mf\ N + (NE{+}UE) + (NS{+}US))))$›
for *L* :: ‹*nat literal option*› **and** *S* :: ‹*nat twl-st-wl*›
have *unassigned-atm-alt-def*:
‹$unassigned\text{-}atm\ S\ L \longleftrightarrow (\exists\ M\ N\ D\ NE\ UE\ NS\ US\ WS\ Q.$
$S = (M, N, D, NE, UE, NS, US, WS, Q) \wedge$
$(L \neq None \longrightarrow$
$undefined\text{-}lit\ M\ (the\ L) \wedge the\ L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S) \wedge$
$atm\text{-}of\ (the\ L) \in\#\ all\text{-}atms\text{-}st\ S) \wedge$
$(L = None \longrightarrow (\nexists L'.\ undefined\text{-}lit\ M\ L' \wedge$
$atm\text{-}of\ L' \in\#\ all\text{-}atms\text{-}st\ S)))$›
for *L* :: ‹*nat literal option*› **and** *S* :: ‹*nat twl-st-wl*›
unfolding *find-unassigned-lit-wl-def RES-RES-RETURN-RES unassigned-atm-def RES-RES-RETURN-RES all-lits-def in-all-lits-of-mm-ain-atms-of-iff* $in\text{-}\mathcal{L}_{all}\text{-}atm\text{-}of\text{-}\mathcal{A}_{in}$ *in-set-all-atms-iff*
by (*cases S*) *auto*

have *find-unassigned-lit-wl-D-alt-def*:
‹$find\text{-}unassigned\text{-}lit\text{-}wl\ S = do\ \{$
$L \leftarrow SPEC(unassigned\text{-}atm\ S);$
$L \leftarrow RES\ \{L, map\text{-}option\ uminus\ L\};$
$SPEC(\lambda((M, N, D, NE, UE, WS, Q), L').$
$S = (M, N, D, NE, UE, WS, Q) \wedge L = L')$
$\}$› **for** *S*
unfolding *find-unassigned-lit-wl-def RES-RES-RETURN-RES unassigned-atm-def RES-RES-RETURN-RES all-lits-def in-all-lits-of-mm-ain-atms-of-iff* $in\text{-}\mathcal{L}_{all}\text{-}atm\text{-}of\text{-}\mathcal{A}_{in}$ *in-set-all-atms-iff*
by (*cases S*) *auto*

have *isa-vmtf-find-next-undef-upd*:
‹$isa\text{-}vmtf\text{-}find\text{-}next\text{-}undef\text{-}upd\ (a, aa, ab, ac, ad, b)$
$((aj, ak, al, am, bb), an, bc)$
$\leq \Downarrow \{(((M, vm), A), L).\ A = map\text{-}option\ atm\text{-}of\ L \wedge$
$unassigned\text{-}atm\ (bt, bu, bv, bw, bx, by, bz, baa, bab)\ L \wedge$
$vm \in isa\text{-}vmtf\ (all\text{-}atms\text{-}st\ (bt, bu, bv, bw, bx, by, bz, baa, bab))\ bt \wedge$
$(L \neq None \longrightarrow the\ A \in\#\ all\text{-}atms\text{-}st\ (bt, bu, bv, bw, bx, by, bz, baa, bab)) \wedge$
$(M, bt) \in trail\text{-}pol\ (all\text{-}atms\text{-}st\ (bt, bu, bv, bw, bx, by, bz, baa, bab))\}$
$(SPEC\ (unassigned\text{-}atm\ (bt, bu, bv, bw, bx, by, bz, baa, bab)))$›
(**is** ‹$- \leq \Downarrow\ ?find\ -$›)
if
pre: ‹$find\text{-}unassigned\text{-}lit\text{-}wl\text{-}D\text{-}heur\text{-}pre\ (bt, bu, bv, bw, bx, by, bz, baa, bab)$› **and**
T: ‹$(((a, aa, ab, ac, ad, b), ae, (af, ag, ba), ah, ai,$
$((aj, ak, al, am, bb), an, bc), ao, (aq, bd), ar, as,$
$(at, au, av, aw, be), heur, bo, bp, bq, br, bs),$
$bt, bu, bv, bw, bx, by, bz, baa, bab)$
$\in twl\text{-}st\text{-}heur$› **and**
‹$r =$
$length$
$(get\text{-}clauses\text{-}wl\text{-}heur$

((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, (*af*, *ag*, *ba*), *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, (*aq*, *bd*), *ar*, *as*,
(*at*, *au*, *av*, *aw*, *be*), *heur*, *bo*, *bp*, *bq*, *br*, *bs*))⟩
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx by bz heur baa bab*
proof −
let *?𝒜* = ⟨*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)⟩
have *pol*:
⟨((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *bt*) ∈ *trail-pol* (*all-atms bu* (*bw* + *bx* + *by* + *bz*))⟩
using *that* **by** (*cases bz*; *auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*])
obtain *vm0* **where**
vm0: ⟨((*an*, *bc*), *vm0*) ∈ *distinct-atoms-rel* (*all-atms bu* (*bw* + *bx* + *by* + *bz*))⟩ **and**
vm: ⟨((*aj*, *ak*, *al*, *am*, *bb*), *vm0*) ∈ *vmtf* (*all-atms bu* (*bw* + *bx* + *by* + *bz*)) *bt*⟩
using *T* **by** (*cases bz*; *auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*] *isa-vmtf-def*)
have [*intro*]:
⟨*Multiset.Ball* ($\mathcal{L}_{all}$ (*all-atms bu* (*bw* + *bx* + *by* + *bz*))) (*defined-lit bt*) ⟹ *atm-of L′* ∈# *all-atms bu* (*bw* + *bx* + *by* + *bz*) ⟹ *undefined-lit bt L′*⟹ *False*⟩ **for** *L′*
by (*auto simp*: *atms-of-ms-def*
all-lits-of-mm-union ran-m-def all-lits-of-mm-add-mset $\mathcal{L}_{all}$-*union*
eq-commute[*of* - ⟨*the* (*fmlookup* - -)⟩] $\mathcal{L}_{all}$-*atm-of-all-lits-of-m*
atms-of-def $\mathcal{L}_{all}$-*add-mset*
dest!: *multi-member-split*
)

show *?thesis*
apply (*rule order.trans*)
apply (*rule isa-vmtf-find-next-undef-vmtf-find-next-undef*[*of ?𝒜*, *THEN fref-to-Down-curry*, *of* - - *bt* ⟨((*aj*, *ak*, *al*, *am*, *bb*), *vm0*)⟩])
subgoal by *fast*
subgoal
using *pol vm0* **by** (*auto simp*: *twl-st-heur-def all-atms-def*[*symmetric*])
apply (*rule order.trans*)
apply (*rule ref-two-step′*)
apply (*rule vmtf-find-next-undef-upd*[*THEN fref-to-Down-curry*, *of ?𝒜 bt* ⟨((*aj*, *ak*, *al*, *am*, *bb*), *vm0*)⟩])
subgoal using *vm* **by** (*auto simp*: *all-atms-def*)
subgoal by *auto*
subgoal using *vm vm0 pre*
apply (*auto 5 0 simp*: *find-undefined-atm-def unassigned-atm-alt-def conc-fun-RES all-atms-def*[*symmetric*]
mset-take-mset-drop-mset′ atms-2 defined-atm-def
intro!: *RES-refine intro*: *isa-vmtfI*)
apply (*auto intro*: *isa-vmtfI simp*: *defined-atm-def atms-2*)
apply (*rule-tac x* = ⟨*Some* (*Pos y*)⟩ **in** *exI*)
apply (*auto intro*: *isa-vmtfI simp*: *defined-atm-def atms-2* *in*-$\mathcal{L}_{all}$-*atm-of*-$\mathcal{A}_{in}$
in-set-all-atms-iff atms-3)
done
done
qed

have *lit-of-found-atm*: ⟨*lit-of-found-atm ao′ x2a*
≤ ⇓ {(*L*, *L′*). *L* = *L′* ∧ *map-option atm-of L* = *x2a*}
(*RES* {*L*, *map-option uminus L*})⟩
if
⟨*find-unassigned-lit-wl-D-heur-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)⟩ **and**

⟨(((a, aa, ab, ac, ad, b), ae, (af, ag, ba), ah, ai,
((aj, ak, al, am, bb), an, bc), ao, (aq, bd), ar, as,
(at, au, av, aw, be), heur, bo, bp, bq, br, bs),
bt, bu, bv, bw, bx, by, bz, baa, bab)
∈ twl-st-heur⟩ **and**
⟨r =
length
(get-clauses-wl-heur
((a, aa, ab, ac, ad, b), ae, (af, ag, ba), ah, ai,
((aj, ak, al, am, bb), an, bc), ao, (aq, bd), ar, as,
(at, au, av, aw, be), heur, bo, bp, bq, br, bs))⟩ **and**
⟨(x, L) ∈ ?find bt bu bv bw bx by bz baa bab⟩ **and**
⟨x1 = (x1a, x2)⟩ **and**
⟨x = (x1, x2a)⟩
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx by bz x L x1 x1a x2 x2a heur baa bab ao′*
proof −
show *?thesis*
using *that* **unfolding** *lit-of-found-atm-def*
by (*auto simp*: *atm-of-eq-atm-of twl-st-heur-def intro*!: *RES-refine*)
qed
show *?thesis*
unfolding *find-unassigned-lit-wl-D-heur-def find-unassigned-lit-wl-D-alt-def find-undefined-atm-def ID-R*
apply (*intro frefI nres-relI*)
apply *clarify*
apply *refine-vcg*
apply (*rule isa-vmtf-find-next-undef-upd*; *assumption*)
subgoal
by (*rule lit-of-found-atm-D-pre*)
(*auto simp add*: *twl-st-heur-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff Ball-def image-image mset-take-mset-drop-mset′ atms all-atms-def*[*symmetric*] *unassigned-atm-def simp del*: *twl-st-of-wl.simps dest*!: *atms intro*!: *RETURN-RES-refine*)
apply (*rule lit-of-found-atm*; *assumption*)
subgoal for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at au av aw ax ay az be bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx - - - - - - - by bz ca cb cc cd ce cf cg ch ci - - x L x1 x1a x2 x2a La Lb*
by (*cases L*)
(*clarsimp-all simp*: *twl-st-heur-def unassigned-atm-def atm-of-eq-atm-of uminus-$\mathcal{A}_{in}$-iff simp del*: *twl-st-of-wl.simps dest*!: *atms intro*!: *RETURN-RES-refine*;
auto simp: *atm-of-eq-atm-of uminus-$\mathcal{A}_{in}$-iff*)+
done
qed

definition *lit-of-found-atm-D*
:: ⟨*bool list* ⇒ *nat option* ⇒ (*nat literal option*)*nres*⟩ **where**
⟨*lit-of-found-atm-D* = (λ(φ::*bool list*) *L*. *do*{
case L of
None ⇒ *RETURN None*
| *Some L* ⇒ *do* {
ASSERT (*L*<*length* φ);
if φ!*L then RETURN* (*Some* (*Pos L*)) *else RETURN* (*Some* (*Neg L*))
}

})›

lemma *lit-of-found-atm-D-lit-of-found-atm*:
‹(*uncurry lit-of-found-atm-D*, *uncurry lit-of-found-atm*) ∈
[*lit-of-found-atm-D-pre*]$_f$ *Id* $\times_f$ *Id* → ⟨*Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *lit-of-found-atm-D-def lit-of-found-atm-def*
by (*auto split*: *option.splits if-splits simp*: *pw-le-iff refine-pw-simps lit-of-found-atm-D-pre-def*)

definition *decide-lit-wl-heur* :: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*decide-lit-wl-heur* = (λ*L'* (*M*, *N*, *D*, *Q*, *W*, *vmtf*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *fema*, *sema*). *do* {
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
ASSERT(*cons-trail-Decided-tr-pre* (*L'*, *M*));
RETURN (*cons-trail-Decided-tr L' M*, *N*, *D*, *j*, *W*, *vmtf*, *clvls*, *cach*, *lbd*, *outl*, *incr-decision stats*, *fema*, *sema*)})›

definition *mop-get-saved-phase-heur-st* :: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *bool nres*› **where**
‹*mop-get-saved-phase-heur-st* =
(λ*L* (*M'*, *N'*, *D'*, *Q'*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*).
mop-get-saved-phase-heur L heur)›

definition *decide-wl-or-skip-D-heur*
:: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*decide-wl-or-skip-D-heur S* = (*do* {
(*S*, *L*) ← *find-unassigned-lit-wl-D-heur S*;
case L of
None ⇒ *RETURN* (*True*, *S*)
| *Some L* ⇒ *do* {
T ← *decide-lit-wl-heur L S*;
RETURN (*False*, *T*)}
})
›

lemma *decide-wl-or-skip-D-heur-decide-wl-or-skip-D*:
‹(*decide-wl-or-skip-D-heur*, *decide-wl-or-skip*) ∈ *twl-st-heur''' r* $\to_f$ ⟨*bool-rel* $\times_f$ *twl-st-heur''' r*⟩ *nres-rel*›
proof −
have [*simp*]:
‹*rev* (*cons-trail-Decided L M*) = *rev M* @ [*Decided L*]›
‹*no-dup* (*cons-trail-Decided L M*) = *no-dup* (*Decided L* # *M*)›
‹*isa-vmtf* $\mathcal{A}$ (*cons-trail-Decided L M*) = *isa-vmtf* $\mathcal{A}$ (*Decided L* # *M*)›
for *M L* $\mathcal{A}$
by (*auto simp*: *cons-trail-Decided-def*)

have *final*: ‹*decide-lit-wl-heur xb x1a*
≤ *SPEC*
(λ*T*. *do* {
RETURN (*False*, *T*)}
≤ *SPEC*
(λ*c*. (*c*, *False*, *decide-lit-wl x'a x1*)
∈ *bool-rel* $\times_f$ *twl-st-heur''' r*))›
if
‹(*x*, *y*) ∈ *twl-st-heur''' r*› **and**

⟨(*xa*, *x'*)
∈ {((*T*, *L*), *T'*, *L'*).
(*T*, *T'*) ∈ *twl-st-heur'''* *r* ∧
L = *L'* ∧
(*L* ≠ *None* ⟶
undefined-lit (*get-trail-wl* *T'*) (*the* *L*) ∧
the *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st* *T'*)) ∧
get-conflict-wl *T'* = *None*}⟩ **and**
st:
⟨*x'* = (*x1*, *x2*)⟩
⟨*xa* = (*x1a*, *x2a*)⟩
⟨*x2a* = *Some* *xb*⟩
⟨*x2* = *Some* *x'a*⟩ **and**
⟨(*xb*, *x'a*) ∈ *nat-lit-lit-rel*⟩
for *x y xa x' x1 x2 x1a x2a xb x'a*
proof −
show *?thesis*
unfolding *decide-lit-wl-heur-def*
decide-lit-wl-def
apply (*cases x1a*)
apply *refine-vcg*
subgoal
by (*rule isa-length-trail-pre*[*of* - ⟨*get-trail-wl x1*⟩ ⟨*all-atms-st x1*⟩])
(*use that*(*2*) **in** ⟨*auto simp*: *twl-st-heur-def st all-atms-def*[*symmetric*]⟩)
subgoal
by (*rule cons-trail-Decided-tr-pre*[*of* - ⟨*get-trail-wl x1*⟩ ⟨*all-atms-st x1*⟩])
(*use that*(*2*) **in** ⟨*auto simp*: *twl-st-heur-def st all-atms-def*[*symmetric*]⟩)
subgoal
using *that*(*2*) **unfolding** *cons-trail-Decided-def*[*symmetric*] *st*
apply (*auto simp*: *twl-st-heur-def*)[]
apply (*clarsimp simp add*: *twl-st-heur-def all-atms-def*[*symmetric*]
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*] *out-learned-def*
intro!: *cons-trail-Decided-tr*[*THEN fref-to-Down-unRET-uncurry*]
isa-vmtf-consD2)
by (*auto simp add*: *twl-st-heur-def all-atms-def*[*symmetric*]
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*] *out-learned-def*
intro!: *cons-trail-Decided-tr*[*THEN fref-to-Down-unRET-uncurry*]
isa-vmtf-consD2)
done
qed

have *decide-wl-or-skip-alt-def*: ⟨*decide-wl-or-skip S* = (*do* {
ASSERT(*decide-wl-or-skip-pre S*);
(*S*, *L*) ← *find-unassigned-lit-wl S*;
case L of
None ⇒ *RETURN* (*True*, *S*)
| *Some L* ⇒ *RETURN* (*False*, *decide-lit-wl L S*)
})⟩ **for** *S*
unfolding *decide-wl-or-skip-def* **by** *auto*
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *decide-wl-or-skip-D-heur-def decide-wl-or-skip-alt-def decide-wl-or-skip-pre-def*
decide-l-or-skip-pre-def twl-st-of-wl.simps[*symmetric*]
apply (*intro nres-relI frefI same-in-Id-option-rel*)
apply (*refine-vcg find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D*[*of r*, *THEN fref-to-Down*])
subgoal for *x y*

unfolding *decide-wl-or-skip-pre-def find-unassigned-lit-wl-D-heur-pre-def decide-wl-or-skip-pre-def decide-l-or-skip-pre-def decide-or-skip-pre-def*
apply *normalize-goal+*
apply (*rule-tac x = xa* **in** *exI*)
apply (*rule-tac x = xb* **in** *exI*)
apply *auto*
done
apply (*rule same-in-Id-option-rel*)
subgoal by (*auto simp del*: *simp*: *twl-st-heur-def*)
subgoal by (*auto simp del*: *simp*: *twl-st-heur-def*)
apply (*rule final*; *assumption?*)
done
qed

lemma *bind-triple-unfold*:
⟨*do* {
((*M*, *vm*), *L*) ← (*P* :: - *nres*);
f ((*M*, *vm*), *L*)
} =
do {
x ← *P*;
f x
}⟩
by (*intro bind-cong*) *auto*

definition *decide-wl-or-skip-D-heur′* **where**
⟨*decide-wl-or-skip-D-heur′* = (λ(*M*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
((*M*, *vm*), *L*) ← *isa-vmtf-find-next-undef-upd M vm*;
ASSERT(*L* ≠ *None* ⟶ *get-saved-phase-heur-pre* (*the L*) *heur*);
case L of
None ⇒ *RETURN* (*True*, (*M*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*))
| *Some L* ⇒ *do* {
b ← *mop-get-saved-phase-heur L heur*;
let L = (*if b then Pos L else Neg L*);
T ← *decide-lit-wl-heur L* (*M*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*);
RETURN (*False*, *T*)
}
})
⟩

lemma *decide-wl-or-skip-D-heur′-decide-wl-or-skip-D-heur*:
⟨*decide-wl-or-skip-D-heur′ S* ≤ ⇓*Id* (*decide-wl-or-skip-D-heur S*)⟩
proof −
have [*iff*]:
⟨{*K*. (∃ *y*. *K* = *Some y*) ∧ *atm-of* (*the K*) = *x2d*} = {*Some* (*Pos x2d*), *Some* (*Neg x2d*)}⟩ **for** *x2d*
apply (*auto simp*: *atm-of-eq-atm-of*)
apply (*case-tac y*)
apply *auto*
done

show *?thesis*
apply (*cases S*, *simp only*:)
unfolding *decide-wl-or-skip-D-heur-def find-unassigned-lit-wl-D-heur-def*

nres-monad3 prod.case decide-wl-or-skip-D-heur′-def
apply (*subst* (*3*) *bind-triple-unfold*[*symmetric*])
unfolding *decide-wl-or-skip-D-heur-def find-unassigned-lit-wl-D-heur-def*
nres-monad3 prod.case lit-of-found-atm-def mop-get-saved-phase-heur-def
apply *refine-vcg*
subgoal by *fast*
subgoal
by (*auto split*: *option.splits simp*: *bind-RES*)
done
qed

lemma *decide-wl-or-skip-D-heur′-decide-wl-or-skip-D-heur2*:
‹(*decide-wl-or-skip-D-heur′*, *decide-wl-or-skip-D-heur*) $\in$ *Id* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*use decide-wl-or-skip-D-heur′-decide-wl-or-skip-D-heur* **in** *auto*)

end
theory *IsaSAT-Decide-LLVM*
imports *IsaSAT-Decide IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM IsaSAT-Rephase-LLVM*
begin

sepref-def *decide-lit-wl-fast-code*
is ‹*uncurry decide-lit-wl-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *decide-lit-wl-heur-def isasat-bounded-assn-def*

unfolding *fold-tuple-optimizations*
apply *sepref-dbg-preproc*
apply *sepref-dbg-cons-init*
apply *sepref-dbg-id*
apply *sepref-dbg-monadify*
apply *sepref-dbg-opt-init*
apply *sepref-dbg-trans*
apply *sepref-dbg-opt*
apply *sepref-dbg-cons-solve*
apply *sepref-dbg-cons-solve*
apply *sepref-dbg-constraints*
done

sepref-register *find-unassigned-lit-wl-D-heur decide-lit-wl-heur*

sepref-register *isa-vmtf-find-next-undef*

sepref-def *isa-vmtf-find-next-undef-code* **is**
uncurry isa-vmtf-find-next-undef :: *vmtf-remove-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *atom.option-assn*
unfolding *isa-vmtf-find-next-undef-def vmtf-remove-assn-def*
unfolding *atom.fold-option*
apply (*rewrite* **in** *WHILEIT* - ¤ *short-circuit-conv*)
supply [[*goals-limit = 1*]]
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *update-next-search*

sepref-def *update-next-search-code* **is**
uncurry (*RETURN oo update-next-search*) :: *atom.option-assn*^{k} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$ *vmtf-remove-assn*
unfolding *update-next-search-def vmtf-remove-assn-def*
by *sepref*

sepref-register *isa-vmtf-find-next-undef-upd mop-get-saved-phase-heur*
sepref-def *isa-vmtf-find-next-undef-upd-code* **is**
uncurry isa-vmtf-find-next-undef-upd
:: *trail-pol-fast-assn*^{d} $*_a$ *vmtf-remove-assn*^{d} $\rightarrow_a$ (*trail-pol-fast-assn* $\times_a$ *vmtf-remove-assn*) $\times_a$ *atom.option-assn*
unfolding *isa-vmtf-find-next-undef-upd-def*
by *sepref*

lemma *mop-get-saved-phase-heur-alt-def*:
‹*mop-get-saved-phase-heur* = (λ*L* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *target*, *best*). *do* {
ASSERT (*L* < *length* φ);
RETURN (φ ! *L*)
})›
unfolding *mop-get-saved-phase-heur-def*
get-saved-phase-heur-pre-def get-saved-phase-heur-def
by *auto*

sepref-def *mop-get-saved-phase-heur-impl*
is ‹*uncurry mop-get-saved-phase-heur*›
:: ‹*atom-assn*^{k} $*_a$ *heuristic-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *mop-get-saved-phase-heur-alt-def*[*abs-def*] *heuristic-assn-def*
apply *annot-all-atm-idxs*
by *sepref*

sepref-def *decide-wl-or-skip-D-fast-code*
is ‹*decide-wl-or-skip-D-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *isasat-bounded-assn*›
supply[[*goals-limit=1*]]
decide-lit-wl-fast-code.refine[*unfolded isasat-bounded-assn-def*, *sepref-fr-rules*]
save-phase-heur-st.refine[*unfolded isasat-bounded-assn-def*, *sepref-fr-rules*]
apply (*rule hfref-refine-with-pre*[*OF decide-wl-or-skip-D-heur′-decide-wl-or-skip-D-heur*, *unfolded Down-id-eq*])
unfolding *decide-wl-or-skip-D-heur′-def isasat-bounded-assn-def*
unfolding *fold-tuple-optimizations option.case-eq-if atom.fold-option*
by *sepref*

experiment begin

export-llvm
decide-lit-wl-fast-code
isa-vmtf-find-next-undef-code
update-next-search-code
isa-vmtf-find-next-undef-upd-code
decide-wl-or-skip-D-fast-code

end

end
theory *IsaSAT-CDCL*
imports *IsaSAT-Propagate-Conflict IsaSAT-Conflict-Analysis IsaSAT-Backtrack*
IsaSAT-Decide IsaSAT-Show

begin

Chapter 18

Combining Together: the Other Rules

definition *cdcl-twl-o-prog-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
‹*cdcl-twl-o-prog-wl-D-heur S* =
do {
if get-conflict-wl-is-None-heur S
then decide-wl-or-skip-D-heur S
else do {
if count-decided-st-heur S > *0*
then do {
T ← *skip-and-resolve-loop-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur T*));
U ← *backtrack-wl-D-nlit-heur T*;
U ← *isasat-current-status U*; — Print some information every once in a while
RETURN (*False*, *U*)
}
else RETURN (*True*, *S*)
}
}
›

lemma *twl-st-heur''D-twl-st-heurD*:
assumes *H*: ‹(⋀𝒟 *r*. *f* ∈ *twl-st-heur''* 𝒟 *r* →$_f$ ⟨*twl-st-heur''* 𝒟 *r*⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* →$_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
proof −
obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
unfolding *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
using *assms*[*of* ‹*dom-m* (*get-clauses-wl y*)› ‹*length* (*get-clauses-wl-heur x*)›,
unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
rule-format] **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*drule spec*[*of* - *x*])
apply (*drule spec*[*of* - *y*])

apply *simp*
apply (*rule weaken-*$\Downarrow$*'*[*of* - ‹*twl-st-heur''* (*dom-m* (*get-clauses-wl y*))
(*length* (*get-clauses-wl-heur x*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *twl-st-heur'''D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge$*r*. *f* $\in$ *twl-st-heur''' r* $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩ *nres-rel*)›
shows ‹*f* $\in$ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- $\in$ *?A B*›)
proof −
obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
unfolding *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
using *assms*[*of* ‹*length* (*get-clauses-wl-heur x*)›,
unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
rule-format] **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*drule spec*[*of* - *x*])
apply (*drule spec*[*of* - *y*])
apply *simp*
apply (*rule weaken-*$\Downarrow$*'*[*of* - ‹*twl-st-heur'''* (*length* (*get-clauses-wl-heur x*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *twl-st-heur'''D-twl-st-heurD-prod*:
assumes *H*: ‹($\bigwedge$*r*. *f* $\in$ *twl-st-heur''' r* $\rightarrow_f$ ⟨*A* $\times_r$ *twl-st-heur''' r*⟩ *nres-rel*)›
shows ‹*f* $\in$ *twl-st-heur* $\rightarrow_f$ ⟨*A* $\times_r$ *twl-st-heur*⟩ *nres-rel*› (**is** ‹- $\in$ *?A B*›)
proof −
obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
unfolding *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
using *assms*[*of* ‹*length* (*get-clauses-wl-heur x*)›,
unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
rule-format] **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*drule spec*[*of* - *x*])
apply (*drule spec*[*of* - *y*])
apply *simp*
apply (*rule weaken-*$\Downarrow$*'*[*of* - ‹*A* $\times_r$ *twl-st-heur'''* (*length* (*get-clauses-wl-heur x*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done

qed

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*:
‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) = *r*} $\rightarrow_f$
⟨*bool-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) ≤ *r* + *6* + *uint32-max div 2*}⟩*nres-rel*›
proof −
have *H*: ‹(*x*, *y*) ∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) =
length (*get-clauses-wl-heur x*)} ⟹
(*x*, *y*)
∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧
length (*get-clauses-wl-heur S*) =
length (*get-clauses-wl-heur x*)}› **for** *x y*
by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
show *?thesis*
unfolding *cdcl-twl-o-prog-wl-D-heur-def cdcl-twl-o-prog-wl-def*
get-conflict-wl-is-None
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
decide-wl-or-skip-D-heur-decide-wl-or-skip-D[**where** *r*=*r*, *THEN fref-to-Down*, *THEN order-trans*]
skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
backtrack-wl-D-nlit-backtrack-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
isasat-current-status-id[*THEN fref-to-Down*, *THEN order-trans*])
subgoal
by (*auto simp*: *twl-st-heur-state-simp*
get-conflict-wl-is-None-heur-get-conflict-wl-is-None[*THEN fref-to-Down-unRET-Id*])
apply (*assumption*)
subgoal by (*rule conc-fun-R-mono*) *auto*
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-count-decided-st-alt-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
apply *assumption*
subgoal by (*auto simp*: *conc-fun-RES RETURN-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
done
qed

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2*:
‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*} $\rightarrow_f$
⟨*bool-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*}⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rule cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*[*THEN fref-to-Down*, *THEN order-trans*])
apply (*auto intro*!: *conc-fun-R-mono*)
done

Combining Together: Full Strategy **definition** *cdcl-twl-stgy-prog-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cdcl-twl-stgy-prog-wl-D-heur* S_0 =
do {
do {

    $(brk, T) \leftarrow WHILE_T$
    $(\lambda(brk, -). \neg brk)$
    $(\lambda(brk, S).$
    do {
      $T \leftarrow$ *unit-propagation-outer-loop-wl-D-heur S*;
      cdcl-twl-o-prog-wl-D-heur T
    })
    $(False, S_0)$;
   RETURN T
  }
}
›

theorem *unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D*:
‹(*unit-propagation-outer-loop-wl-D-heur*, *unit-propagation-outer-loop-wl*) $\in$
  twl-st-heur $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*›
using *twl-st-heur''D-twl-st-heurD*[*OF*
  unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D']
.

lemma *cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D*:
‹(*cdcl-twl-stgy-prog-wl-D-heur*, *cdcl-twl-stgy-prog-wl*) $\in$ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
proof –
  have *H*: ‹$(x, y) \in \{(S, T)$.
        $(S, T) \in$ *twl-st-heur* $\wedge$
        length (*get-clauses-wl-heur S*) =
        length (*get-clauses-wl-heur x*)} $\Longrightarrow$
      (x, y)
      $\in \{(S, T)$.
        $(S, T) \in$ *twl-st-heur-conflict-ana* $\wedge$
        length (*get-clauses-wl-heur S*) =
        length (*get-clauses-wl-heur x*)}› **for** *x y*
    by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
  show *?thesis*
    unfolding *cdcl-twl-stgy-prog-wl-D-heur-def cdcl-twl-stgy-prog-wl-def*
    apply (*intro frefI nres-relI*)
    subgoal for *x y*
    apply (*refine-vcg*
      unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN twl-st-heur''D-twl-st-heurD*,
THEN fref-to-Down]
      cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[*THEN fref-to-Down*])
    subgoal by (*auto simp*: *twl-st-heur-state-simp*)
    subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur'-def*)
    subgoal by (*auto simp*: *twl-st-heur'-def*)
    subgoal by (*auto simp*: *twl-st-heur-state-simp*)
    subgoal by (*auto simp*: *twl-st-heur-state-simp*)
    done
    done
qed

definition *cdcl-twl-stgy-prog-break-wl-D-heur* :: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
  ‹*cdcl-twl-stgy-prog-break-wl-D-heur* S_0 =
  do {
    $b \leftarrow$ *RETURN* (*isasat-fast* S_0);

$(b,\ brk,\ T) \leftarrow \mathit{WHILE}_T{}^{\lambda(b,\ brk,\ T).\ True}$
    $(\lambda(b,\ brk,\ \text{-}).\ b \wedge \neg brk)$
    $(\lambda(b,\ brk,\ S).$
    do {
      $\mathit{ASSERT}(\mathit{isasat\text{-}fast}\ S)$;
      $T \leftarrow \mathit{unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur}\ S$;
      $\mathit{ASSERT}(\mathit{isasat\text{-}fast}\ T)$;
      $(brk,\ T) \leftarrow \mathit{cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\text{-}heur}\ T$;
      $b \leftarrow \mathit{RETURN}\ (\mathit{isasat\text{-}fast}\ T)$;
      $\mathit{RETURN}(b,\ brk,\ T)$
    })
    $(b,\ False,\ S_0)$;
  if brk then RETURN T
  else cdcl-twl-stgy-prog-wl-D-heur T
}›

definition *cdcl-twl-stgy-prog-bounded-wl-heur* :: ‹*twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*›
where
  ‹*cdcl-twl-stgy-prog-bounded-wl-heur* S_0 =
  do {
    $b \leftarrow \mathit{RETURN}\ (\mathit{isasat\text{-}fast}\ S_0)$;
    $(b,\ brk,\ T) \leftarrow \mathit{WHILE}_T{}^{\lambda(b,\ brk,\ T).\ True}$
      $(\lambda(b,\ brk,\ \text{-}).\ b \wedge \neg brk)$
      $(\lambda(b,\ brk,\ S).$
      do {
        $\mathit{ASSERT}(\mathit{isasat\text{-}fast}\ S)$;
        $T \leftarrow \mathit{unit\text{-}propagation\text{-}outer\text{-}loop\text{-}wl\text{-}D\text{-}heur}\ S$;
        $\mathit{ASSERT}(\mathit{isasat\text{-}fast}\ T)$;
        $(brk,\ T) \leftarrow \mathit{cdcl\text{-}twl\text{-}o\text{-}prog\text{-}wl\text{-}D\text{-}heur}\ T$;
        $b \leftarrow \mathit{RETURN}\ (\mathit{isasat\text{-}fast}\ T)$;
        $\mathit{RETURN}(b,\ brk,\ T)$
      })
      $(b,\ False,\ S_0)$;
    RETURN (*brk*, *T*)
  }›

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
  assumes *r*: ‹$r \leq \mathit{sint64\text{-}max}$›
  shows ‹(*cdcl-twl-stgy-prog-bounded-wl-heur*, *cdcl-twl-stgy-prog-early-wl*) ∈
   $\mathit{twl\text{-}st\text{-}heur}'''\ r \rightarrow_f \langle \mathit{bool\text{-}rel} \times_r \mathit{twl\text{-}st\text{-}heur}\rangle \mathit{nres\text{-}rel}$›
proof −
  have *A*[*refine0*]: ‹$\mathit{RETURN}\ (\mathit{isasat\text{-}fast}\ x) \leq \Downarrow$
     $\{(b,\ b').\ b = b' \wedge (b = (\mathit{isasat\text{-}fast}\ x))\}\ (\mathit{RES}\ \mathit{UNIV})$›
    for *x*
    by (*auto intro*: *RETURN-RES-refine*)
  have *twl-st-heur''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur* ⟹
    (*x1e*, *x1b*)
    ∈ *twl-st-heur''*
      (*dom-m* (*get-clauses-wl x1b*))
      (*length* (*get-clauses-wl-heur x1e*))›
    for *x1e x1b*
    by (*auto simp*: *twl-st-heur'-def*)
  have *twl-st-heur'''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur''* $\mathcal{D}$ *r* ⟹
    (*x1e*, *x1b*)

$\in$ *twl-st-heur'''* *r*⟩
for *x1e x1b r* $\mathcal{D}$
by (*auto simp*: *twl-st-heur'-def*)
have *H*: ⟨*SPEC* (λ-::*bool*. *True*) = *RES UNIV*⟩ **by** *auto*
show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*]
unfolding *cdcl-twl-stgy-prog-bounded-wl-heur-def*
cdcl-twl-stgy-prog-early-wl-def H
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
WHILEIT-refine[**where** *R* = ⟨{((*ebrk*, *brk*, *T*), (*ebrk'*, *brk'*, *T'*)).
(*ebrk* = *ebrk'*) ∧ (*brk* = *brk'*) ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧
(*ebrk* ⟶ *isasat-fast T*) ∧ *length* (*get-clauses-wl-heur T*) ≤ *sint64-max*}⟩])
subgoal using *r* **by** *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
subgoal by (*auto simp*: *isasat-fast-def*)
apply (*rule twl-st-heur'''*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def sint64-max-def uint32-max-def*)
subgoal by *auto*
done
qed

end
theory *IsaSAT-CDCL-LLVM*
imports *IsaSAT-CDCL IsaSAT-Propagate-Conflict-LLVM IsaSAT-Conflict-Analysis-LLVM*
IsaSAT-Backtrack-LLVM
IsaSAT-Decide-LLVM IsaSAT-Show-LLVM
begin

sepref-register *get-conflict-wl-is-None decide-wl-or-skip-D-heur skip-and-resolve-loop-wl-D-heur*
backtrack-wl-D-nlit-heur isasat-current-status count-decided-st-heur get-conflict-wl-is-None-heur

sepref-def *cdcl-twl-o-prog-wl-D-fast-code*
is ⟨*cdcl-twl-o-prog-wl-D-heur*⟩
:: ⟨[*isasat-fast*]$_a$
isasat-bounded-assn^d → *bool1-assn* $\times_a$ *isasat-bounded-assn*⟩
unfolding *cdcl-twl-o-prog-wl-D-heur-def PR-CONST-def*
unfolding *get-conflict-wl-is-None get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

declare
cdcl-twl-o-prog-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-register *unit-propagation-outer-loop-wl-D-heur*
cdcl-twl-o-prog-wl-D-heur

definition *length-clauses-heur* **where**
⟨*length-clauses-heur S* = *length* (*get-clauses-wl-heur S*)⟩

lemma *length-clauses-heur-alt-def*: ⟨*length-clauses-heur* = (λ(*M*, *N*, -). *length N*)⟩
by (*auto intro*!: *ext simp*: *length-clauses-heur-def*)

sepref-def *length-clauses-heur-impl*
is ⟨*RETURN o length-clauses-heur*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*⟩
unfolding *length-clauses-heur-alt-def isasat-bounded-assn-def*
by *sepref*

declare *length-clauses-heur-impl.refine* [*sepref-fr-rules*]

lemma *isasat-fast-alt-def*: ⟨*isasat-fast S* = (*length-clauses-heur S* $\leq$ *9223372034707292154*)⟩
by (*auto simp*: *isasat-fast-def sint64-max-def uint32-max-def length-clauses-heur-def*)

sepref-def *isasat-fast-impl*
is ⟨*RETURN o isasat-fast*⟩
:: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
unfolding *isasat-fast-alt-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

declare *isasat-fast-impl.refine*[*sepref-fr-rules*]

sepref-def *cdcl-twl-stgy-prog-wl-D-code*
is ⟨*cdcl-twl-stgy-prog-bounded-wl-heur*⟩
:: ⟨*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *isasat-bounded-assn*⟩
unfolding *cdcl-twl-stgy-prog-bounded-wl-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]] *isasat-fast-length-leD*[*dest*]
by *sepref*

declare *cdcl-twl-stgy-prog-wl-D-code.refine*[*sepref-fr-rules*]

export-llvm *cdcl-twl-stgy-prog-wl-D-code* **file** *code/isasat.ll*

end
theory *IsaSAT-Restart-Heuristics*
imports
Watched-Literals.WB-Sort Watched-Literals.Watched-Literals-Watch-List-Restart IsaSAT-Rephase IsaSAT-Setup IsaSAT-VMTF IsaSAT-Sorting
begin

Chapter 19

Restarts

lemma *all-init-atms-alt-def*:
‹*set-mset* (*all-init-atms N NE*) = *atms-of-mm* (*mset* ‘# *init-clss-lf N*) ∪ *atms-of-mm NE*›
unfolding *all-init-atms-def all-init-lits-def*
by (*auto simp*: *in-all-lits-of-mm-ain-atms-of-iff*
all-lits-of-mm-def atms-of-ms-def image-UN
atms-of-def
dest!: *multi-member-split*[*of* ‹(-, -)› ‹*ran-m N*›]
dest: *multi-member-split atm-of-lit-in-atms-of*
simp del: *set-image-mset*)

lemma *in-set-all-init-atms-iff*:
‹*y* ∈# *all-init-atms bu bw* ⟷
y ∈ *atms-of-mm* (*mset* ‘# *init-clss-lf bu*) ∨ *y* ∈ *atms-of-mm bw*›
by (*auto simp*: *all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff*
atm-of-all-lits-of-mm all-init-atms-alt-def
simp: *in-all-lits-of-mm-ain-atms-of-iff*
all-lits-of-mm-def atms-of-ms-def image-UN
atms-of-def
dest!: *multi-member-split*[*of* ‹(-, -)› ‹*ran-m N*›]
dest: *multi-member-split atm-of-lit-in-atms-of*
simp del: *set-image-mset*)

lemma *twl-st-heur-change-subsumed-clauses*:
assumes ‹((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*),
(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)) ∈ *twl-st-heur*›
‹*set-mset* (*all-atms N* ((*NE*+*UE*)+(*NS*+*US*))) = *set-mset* (*all-atms N* ((*NE*+*UE*)+(*NS'*+*US'*)))›
shows ‹((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*),
(*M*, *N*, *D*, *NE*, *UE*, *NS'*, *US'*, *Q*, *W*)) ∈ *twl-st-heur*›
proof −
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong phase-saving-cong*
cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong heuristic-rel-cong
show *?thesis*
using *cong*[*OF assms*(*2*)] *assms*(*1*)
apply (*auto simp add*: *twl-st-heur-def*)
apply *fastforce*
apply *force*
done

qed

This is a list of comments (how does it work for glucose and cadical) to prepare the future refinement:

1. Reduction

   - every 2000+300*n (rougly since inprocessing changes the real number, cadical) (split over initialisation file); don't restart if level < 2 or if the level is less than the fast average
   - curRestart * nbclausesbeforereduce; curRestart = (conflicts / nbclausesbeforereduce) + 1 (glucose)

2. Killed

   - half of the clauses that **can** be deleted (i.e., not used since last restart), not strictly LBD, but a probability of being useful.
   - half of the clauses

3. Restarts:

   - EMA-14, aka restart if enough clauses and slow_glue_avg * opts.restartmargin > fast_glue (file ema.cpp)
   - (lbdQueue.getavg() * K) > (sumLBD / conflictsRestarts), *conflictsRestarts* > *LOWER-BOUND-FO* && *lbdQueue.isvalid*() && *trail.size*() > *R* ∗ *trailQueue.getavg*()

declare *all-atms-def*[*symmetric*,*simp*]

definition *twl-st-heur-restart* :: ‹(*twl-st-wl-heur* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-restart* =
{((*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*),
(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)).
(*M'*, *M*) ∈ *trail-pol* (*all-init-atms N* (*NE*+*NS*)) ∧
valid-arena N' N (*set vdom*) ∧
(*D'*, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N* (*NE*+*NS*)) ∧
(*D* = *None* ⟶ *j* ≤ *length M*) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms N* (*NE*+*NS*))) ∧
vm ∈ *isa-vmtf* (*all-init-atms N* (*NE*+*NS*)) *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M D* ∧
cach-refinement-empty (*all-init-atms N* (*NE*+*NS*)) *cach* ∧
out-learned M D outl ∧
lcount = *size* (*learned-clss-lf N*) ∧
vdom-m (*all-init-atms N* (*NE*+*NS*)) *W N* ⊆ *set vdom* ∧
mset avdom ⊆# *mset vdom* ∧
isasat-input-bounded (*all-init-atms N* (*NE*+*NS*)) ∧
isasat-input-nempty (*all-init-atms N* (*NE*+*NS*)) ∧
distinct vdom ∧ *old-arena* = [] ∧
heuristic-rel (*all-init-atms N* (*NE*+*NS*)) *heur*
}›

abbreviation *twl-st-heur''''* **where**
‹*twl-st-heur''''* $r \equiv \{(S,\ T).\ (S,\ T) \in$ *twl-st-heur* $\wedge$ *length* (*get-clauses-wl-heur* S) $\leq r\}$›

abbreviation *twl-st-heur-restart'''* **where**
‹*twl-st-heur-restart'''* $r \equiv$
$\{(S,\ T).\ (S,\ T) \in$ *twl-st-heur-restart* $\wedge$ *length* (*get-clauses-wl-heur* S) $= r\}$›

abbreviation *twl-st-heur-restart''''* **where**
‹*twl-st-heur-restart''''* $r \equiv$
$\{(S,\ T).\ (S,\ T) \in$ *twl-st-heur-restart* $\wedge$ *length* (*get-clauses-wl-heur* S) $\leq r\}$›

definition *twl-st-heur-restart-ana* :: ‹*nat* $\Rightarrow$ (*twl-st-wl-heur* $\times$ *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-restart-ana* r =
$\{(S,\ T).\ (S,\ T) \in$ *twl-st-heur-restart* $\wedge$ *length* (*get-clauses-wl-heur* S) $= r\}$›

lemma *twl-st-heur-restart-anaD*: ‹$x \in$ *twl-st-heur-restart-ana* $r \Longrightarrow x \in$ *twl-st-heur-restart*›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma *twl-st-heur-restartD*:
‹$x \in$ *twl-st-heur-restart* $\Longrightarrow x \in$ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur* (*fst* x)))›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

definition *clause-score-ordering2* **where**
‹*clause-score-ordering2* = (λ(*lbd*, *act*) (*lbd'*, *act'*). *lbd* < *lbd'* $\vee$ (*lbd* = *lbd'* $\wedge$ *act* $\leq$ *act'*))›

lemma *unbounded-id*: ‹*unbounded* (*id* :: *nat* $\Rightarrow$ *nat*)›
by (*auto simp*: *bounded-def*) *presburger*

global-interpretation *twl-restart-ops id*
by *unfold-locales*

global-interpretation *twl-restart id*
by *standard* (*rule unbounded-id*)

We first fix the function that proves termination. We don't take the "smallest" function possible (other possibilites that are growing slower include $\lambda n.\ n >> 50$). Remark that this scheme is not compatible with Luby (TODO: use Luby restart scheme every once in a while like CryptoMinisat?)

lemma *get-slow-ema-heur-alt-def*:
‹*RETURN o get-slow-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, (*ccount*, -)), *lcount*). *RETURN sema*)›
by *auto*

lemma *get-fast-ema-heur-alt-def*:
‹*RETURN o get-fast-ema-heur* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, (*fema*, *sema*, *ccount*), *lcount*). *RETURN fema*)›
by *auto*

lemma *get-learned-count-alt-def*:
‹*RETURN o get-learned-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, -, *vdom*, *avdom*, *lcount*, *opts*). *RETURN lcount*)›
by *auto*

definition (**in** −) *find-local-restart-target-level-int-inv* **where**
⟨*find-local-restart-target-level-int-inv ns cs* =
  (λ(*brk*, *i*). *i* ≤ *length cs* ∧ *length cs* < *uint32-max*)⟩

definition *find-local-restart-target-level-int*
 :: ⟨*trail-pol* ⇒ *isa-vmtf-remove-int* ⇒ *nat nres*⟩
where
⟨*find-local-restart-target-level-int* =
  (λ(*M*, *xs*, *lvls*, *reasons*, *k*, *cs*) ((*ns* :: *nat-vmtf-node list*, *m* :: *nat*, *fst-As*::*nat*, *lst-As*::*nat*,
    next-search::*nat option*), -). *do* {
  (*brk*, *i*) ← $WHILE_T{}^{\textit{find-local-restart-target-level-int-inv ns cs}}$
    (λ(*brk*, *i*). ¬*brk* ∧ *i* < *length-uint32-nat cs*)
    (λ(*brk*, *i*). *do* {
      ASSERT(*i* < *length cs*);
      let t = (*cs* ! *i*);
  ASSERT(*t* < *length M*);
  let L = *atm-of* (*M* ! *t*);
      ASSERT(*L* < *length ns*);
      let brk = *stamp* (*ns* ! *L*) < *m*;
      RETURN (*brk*, *if brk then i else i+1*)
    })
    (*False*, *0*);
  RETURN i
 })⟩

definition *find-local-restart-target-level* **where**
⟨*find-local-restart-target-level M* - = *SPEC*(λ*i*. *i* ≤ *count-decided M*)⟩

lemma *find-local-restart-target-level-alt-def*:
⟨*find-local-restart-target-level M vm* = *do* {
   (*b*, *i*) ← *SPEC*(λ(*b*::*bool*, *i*). *i* ≤ *count-decided M*);
   RETURN i
 }⟩
unfolding *find-local-restart-target-level-def* **by** (*auto simp*: *RES-RETURN-RES2 uncurry-def*)

lemma *find-local-restart-target-level-int-find-local-restart-target-level*:
 ⟨(*uncurry find-local-restart-target-level-int*, *uncurry find-local-restart-target-level*) ∈
   $[\lambda(M,\ vm).\ vm \in \textit{isa-vmtf}\ \mathcal{A}\ M]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* → ⟨*nat-rel*⟩*nres-rel*⟩
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-alt-def*
 uncurry-def Let-def
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b ae af ag ah ba bb ai aj ak al am bc bd*
 apply (*refine-rcg WHILEIT-rule*[**where** *R* = ⟨*measure* (λ(*brk*, *i*). (*If brk 0 1*) + *length b* − *i*)⟩]
   assert.ASSERT-leI)
 subgoal by *auto*
 subgoal
  unfolding *find-local-restart-target-level-int-inv-def*
  by (*auto simp*: *trail-pol-alt-def control-stack-length-count-dec*)
 subgoal by *auto*
 subgoal by (*auto simp*: *trail-pol-alt-def intro*: *control-stack-le-length-M*)
 subgoal for *s x1 x2*
  by (*subgoal-tac* ⟨*a* ! (*b* ! *x2*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$⟩)
   (*auto simp*: *trail-pol-alt-def rev-map lits-of-def rev-nth*

vmtf-def atms-of-def isa-vmtf-def
intro!: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*)
subgoal by (*auto simp*: *find-local-restart-target-level-int-inv-def*)
subgoal by (*auto simp*: *trail-pol-alt-def control-stack-length-count-dec*
find-local-restart-target-level-int-inv-def)
subgoal by *auto*
done
done

definition *empty-Q* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*empty-Q* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fema*, *sema*, *ccount*, *wasted*), *vdom*, *lcount*). *do*{
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
RETURN (*M*, *N*, *D*, *j*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fema*, *sema*,
restart-info-restart-done ccount, *wasted*), *vdom*, *lcount*)
})›

definition *restart-abs-wl-heur-pre* :: ‹*twl-st-wl-heur* ⇒ *bool* ⇒ *bool*› **where**
‹*restart-abs-wl-heur-pre S brk* ⟷ (∃ *T*. (*S*, *T*) ∈ *twl-st-heur* ∧ *restart-abs-wl-pre T brk*)›

find-decomp-wl-st-int is the wrong function here, because unlike in the backtrack case, we also have to update the queue of literals to update. This is done in the function *empty-Q*.

definition *find-local-restart-target-level-st* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
‹*find-local-restart-target-level-st S* = *do* {
find-local-restart-target-level-int (*get-trail-wl-heur S*) (*get-vmtf-heur S*)
}›

lemma *find-local-restart-target-level-st-alt-def*:
‹*find-local-restart-target-level-st* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *stats*). *do* {
find-local-restart-target-level-int M vm})›
apply (*intro ext*)
apply (*case-tac x*)
by (*auto simp*: *find-local-restart-target-level-st-def*)

definition *cdcl-twl-local-restart-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*cdcl-twl-local-restart-wl-D-heur* = (λ*S*. *do* {
ASSERT(*restart-abs-wl-heur-pre S False*);
lvl ← *find-local-restart-target-level-st S*;
if lvl = *count-decided-st-heur S*
then RETURN S
else do {
S ← *find-decomp-wl-st-int lvl S*;
S ← *empty-Q S*;
incr-lrestart-stat S
}
})›

named-theorems *twl-st-heur-restart*

lemma [*twl-st-heur-restart*]:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹(*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)›

using *assms* **by** (*cases S*; *cases T*)
(*simp only*: *twl-st-heur-restart-def get-trail-wl-heur.simps get-trail-wl.simps mem-Collect-eq prod.case get-clauses-wl.simps get-unit-init-clss-wl.simps get-subsumed-init-clauses-wl.simps*)

lemma *trail-pol-literals-are-in-$\mathcal{L}_{in}$-trail*:
‹$(M', M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow$ *literals-are-in-$\mathcal{L}_{in}$-trail* $\mathcal{A}$ *M*›
unfolding *literals-are-in-$\mathcal{L}_{in}$-trail-def trail-pol-def*
by *auto*

lemma *refine-generalise1*: $A \le B \Longrightarrow$ *do* $\{x \leftarrow B;\ C\ x\} \le D \Longrightarrow$ *do* $\{x \leftarrow A;\ C\ x\} \le (D{::}\ 'a$ *nres*)
using *Refine-Basic.bind-mono*(*1*) *dual-order.trans* **by** *blast*

lemma *refine-generalise2*: $A \le B \Longrightarrow$ *do* $\{x \leftarrow$ *do* $\{x \leftarrow B;\ A'\ x\};\ C\ x\} \le D \Longrightarrow$
do $\{x \leftarrow$ *do* $\{x \leftarrow A;\ A'\ x\};\ C\ x\} \le (D{::}\ 'a$ *nres*)
by (*simp add*: *refine-generalise1*)

lemma *cdcl-twl-local-restart-wl-D-spec-int*:
‹*cdcl-twl-local-restart-wl-spec* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) ≥ (*do* {
ASSERT(*restart-abs-wl-pre* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) *False*);
i ← *SPEC*(λ-. *True*);
if i
then RETURN (*M*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *Q*, *W*)
else do {
(*M*, *Q'*) ← *SPEC*(λ(*M'*, *Q'*). (∃ *K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
Q' = {#}) ∨ (*M'* = *M* ∧ *Q'* = *Q*));
RETURN (*M*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *Q'*, *W*)
}
})›
proof −
have *If-Res*: ‹(*if i then* (*RETURN f*) *else* (*RES g*)) = (*RES* (*if i then* {*f*} *else g*))› **for** *i f g*
by *auto*
show *?thesis*
unfolding *cdcl-twl-local-restart-wl-spec-def prod.case RES-RETURN-RES2 If-Res*
by *refine-vcg*
(*auto simp*: *If-Res RES-RETURN-RES2 RES-RES-RETURN-RES uncurry-def image-iff split*:*if-splits*)
qed

lemma *trail-pol-no-dup*: ‹(*M*, *M'*) ∈ *trail-pol* $\mathcal{A} \Longrightarrow$ *no-dup M'*›
by (*auto simp*: *trail-pol-def*)

lemma *heuristic-rel-restart-info-done*[*intro*!, *simp*]:
‹*heuristic-rel* $\mathcal{A}$ (*fema*, *sema*, *ccount*, *wasted*) $\Longrightarrow$
heuristic-rel $\mathcal{A}$ ((*fema*, *sema*, *restart-info-restart-done ccount*, *wasted*))›
by (*auto simp*: *heuristic-rel-def*)

lemma *cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec*:
‹(*cdcl-twl-local-restart-wl-D-heur*, *cdcl-twl-local-restart-wl-spec*) ∈
twl-st-heur''' *r* $\to_f$ ⟨*twl-st-heur'''* *r*⟩*nres-rel*›
proof −
have *K*: ‹((*case S of*
(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fema*, *sema*,
ccount), *vdom*, *lcount*) ⇒
ASSERT (*isa-length-trail-pre M*) ≫=

(λ-. *RES* {(*M*, *N*, *D*, *isa-length-trail M*, *W*, *vm*, *clvls*, *cach*,
lbd, *outl*, *stats*, (*fema*, *sema*,
restart-info-restart-done ccount), *vdom*, *lcount*)}))) =
((*ASSERT* (*case S of* (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fema*, *sema*,
ccount), *vdom*, *lcount*) ⇒ *isa-length-trail-pre M*)) ≫=
(λ -. (*case S of*
(*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fema*, *sema*,
ccount), *vdom*, *lcount*) ⇒ *RES* {(*M*, *N*, *D*, *isa-length-trail M*, *W*, *vm*, *clvls*, *cach*,
lbd, *outl*, *stats*, (*fema*, *sema*,
restart-info-restart-done ccount), *vdom*, *lcount*)})))› **for** *S*
by (*cases S*) *auto*

have *K2*: ‹(*case S of*
(*a*, *b*) ⇒ *RES* (Φ *a b*)) =
(*RES* (*case S of* (*a*, *b*) ⇒ Φ *a b*))› **for** *S*
by (*cases S*) *auto*

have [*dest*]: ‹*av* = *None*› ‹*out-learned a av am* ⟹ *out-learned x1 av am*›
if ‹*restart-abs-wl-pre* (*a*, *au*, *av*, *aw*, *ax*, *NS*, *US*, *ay*, *bd*) *False*›
for *a au av aw ax ay bd x1 am NS US*
using *that*
unfolding *restart-abs-wl-pre-def restart-abs-l-pre-def*
restart-prog-pre-def
by (*auto simp*: *twl-st-l-def state-wl-l-def out-learned-def*)
have [*refine0*]:
‹*find-local-restart-target-level-int* (*get-trail-wl-heur S*) (*get-vmtf-heur S*) ≤
⇓ {(*i*, *b*). *b* = (*i* = *count-decided* (*get-trail-wl T*)) ∧
i ≤ *count-decided* (*get-trail-wl T*)} (*SPEC* (λ-. *True*))›
if ‹(*S*, *T*) ∈ *twl-st-heur*› **for** *S T*
apply (*rule find-local-restart-target-level-int-find-local-restart-target-level*[*THEN*
fref-to-Down-curry, *THEN order-trans*, *of* ‹*all-atms-st T*› ‹*get-trail-wl T*› ‹*get-vmtf-heur S*›])
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal by (*auto simp*: *find-local-restart-target-level-def conc-fun-RES*)
done
have *H*:
‹*set-mset* (*all-atms-st S*) =
set-mset (*all-init-atms-st S*)› (**is** *?A*)
‹*set-mset* (*all-atms-st S*) =
set-mset (*all-atms* (*get-clauses-wl S*) (*get-unit-clauses-wl S* + *get-subsumed-init-clauses-wl S*))›
(**is** *?B*)
‹*get-conflict-wl S* = *None*› (**is** *?C*)
if *pre*: ‹*restart-abs-wl-pre S False*›
for *S*
proof −
obtain *T U* **where**
ST: ‹(*S*, *T*) ∈ *state-wl-l None*› **and**
‹*correct-watching S*› **and**
‹*blits-in-*$\mathcal{L}_{in}$ *S*› **and**
TU: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
struct: ‹*twl-struct-invs U*› **and**
‹*twl-list-invs T*› **and**
‹*clauses-to-update-l T* = {#}› **and**
‹*twl-stgy-invs U*› **and**
confl: ‹*get-conflict U* = *None*›
using *pre* **unfolding** *restart-abs-wl-pre-def restart-abs-l-pre-def restart-prog-pre-def* **apply** −

by *blast*

show *?C*
using *ST TU confl* **by** *auto*

have *alien*: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)›
using *struct* **unfolding** *twl-struct-invs-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast+*
then show *?A* **and** *?B*
subgoal
using *ST TU* **unfolding** *set-eq-iff in-set-all-atms-iff*
in-set-all-atms-iff in-set-all-init-atms-iff get-unit-clauses-wl-alt-def
apply (*subst all-clss-lf-ran-m*[*symmetric*])
unfolding *image-mset-union*
apply (*auto simp*: $cdcl_W$*-restart-mset.no-strange-atm-def twl-st twl-st-l in-set-all-atms-iff*
in-set-all-init-atms-iff)
done
subgoal
using *ST TU alien* **unfolding** *set-eq-iff in-set-all-atms-iff*
in-set-all-atms-iff in-set-all-init-atms-iff get-unit-clauses-wl-alt-def
apply (*subst all-clss-lf-ran-m*[*symmetric*])
apply (*subst all-clss-lf-ran-m*[*symmetric*])
unfolding *image-mset-union*
by (*auto simp*: $cdcl_W$*-restart-mset.no-strange-atm-def twl-st twl-st-l in-set-all-atms-iff*
in-set-all-init-atms-iff)
done
qed
have *P*: ‹*P*›
if
ST: ‹(((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, *heur*, *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, (*aq*, *bd*), *ar*, *as*,
(*at′*, *au*, *av*, *aw*, *be*), ((*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*)), *bo*, *bp*, *bq*, *br*, *bs*),
bt, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)
∈ *twl-st-heur*› **and**
‹*restart-abs-wl-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*) *False*› **and**
‹*restart-abs-wl-heur-pre*
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, *heur*, *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, (*aq*, *bd*), *ar*, *as*,
(*at′*, *au*, *av*, *aw*, *be*), ((*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*)), *bo*, *bp*, *bq*, *br*, *bs*)
False› **and**
lvl: ‹(*lvl*, *i*)
∈ {(*i*, *b*).
b = (*i* = *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*))) ∧
i ≤ *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*))}› **and**
‹*i* ∈ {-. *True*}› **and**
‹*lvl* ≠
count-decided-st-heur
((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *ae*, *heur*, *ah*, *ai*,
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*), *ao*, (*aq*, *bd*), *ar*, *as*,
(*at′*, *au*, *av*, *aw*, *be*), ((*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*)), *bo*, *bp*, *bq*, *br*, *bs*)› **and**
i: ‹¬ *i*› **and**
H: ‹($\bigwedge$*vm0*. ((*an*, *bc*), *vm0*) ∈ *distinct-atoms-rel* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*))
$\Longrightarrow$

((*aj*, *ak*, *al*, *am*, *bb*), *vm0*) ∈ *vmtf* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)) *bt* ⟹
isa-find-decomp-wl-imp (*a*, *aa*, *ab*, *ac*, *ad*, *b*) *lvl*
((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*)
≤ ⇓ {(*a*, *b*). (*a*,*b*) ∈ *trail-pol* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)) $\times_f$
(*Id* $\times_f$ *distinct-atoms-rel* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)))}
(*find-decomp-w-ns* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)) *bt lvl vm0*) ⟹ *P*)›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao aq bd ar as at′*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
bw bx by bz lvl i x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f
x1g x2g x1h x2h x1i x2i P NS US heur
proof −
let *?𝒜* = ‹*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NS*, *US*, *by*, *bz*)›
have
tr: ‹((*a*, *aa*, *ab*, *ac*, *ad*, *b*), *bt*) ∈ *trail-pol ?𝒜*› **and**
‹*valid-arena ae bu* (*set bo*)› **and**
‹(*heur*, *bv*)
∈ *option-lookup-clause-rel ?𝒜*› **and**
‹*by* = {#− *lit-of x*. *x* ∈# *mset* (*drop ah* (*rev bt*))#}› **and**
‹(*ai*, *bz*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 *?𝒜*)› **and**
vm: ‹((*aj*, *ak*, *al*, *am*, *bb*), *an*, *bc*) ∈ *isa-vmtf ?𝒜 bt*› **and**
‹*no-dup bt*› **and**
‹*ao* ∈ *counts-maximum-level bt bv*› **and**
‹*cach-refinement-empty ?𝒜* (*aq*, *bd*)› **and**
‹*out-learned bt bv as*› **and**
‹*bq* = *size* (*learned-clss-l bu*)› **and**
‹*vdom-m ?𝒜 bz bu* ⊆ *set bo*› **and**
‹*set bp* ⊆ *set bo*› **and**
‹∀ *L*∈#$\mathcal{L}_{all}$ *?𝒜*. *nat-of-lit L* ≤ *uint32-max*› **and**
‹*?𝒜* ≠ {#}› **and**
bounded: ‹*isasat-input-bounded ?𝒜*› **and**
heur: ‹*heuristic-rel ?𝒜* ((*ax*, *ay*, *az*, *bf*, *bg*), (*bh*, *bi*, *bj*, *bk*, *bl*),
(*bm*, *bn*))›
using *ST* **unfolding** *twl-st-heur-def all-atms-def*[*symmetric*]
by (*auto*)

obtain *vm0* **where**
vm: ‹((*aj*, *ak*, *al*, *am*, *bb*), *vm0*) ∈ *vmtf ?𝒜 bt*› **and**
vm0: ‹((*an*, *bc*), *vm0*) ∈ *distinct-atoms-rel ?𝒜*›
using *vm*
by (*auto simp*: *isa-vmtf-def*)
have *n-d*: ‹*no-dup bt*›
using *tr* **by** (*auto simp*: *trail-pol-def*)
show *?thesis*
apply (*rule H*)
apply (*rule vm0*)
apply (*rule vm*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*, *THEN order-trans*,
of bt lvl ‹((*aj*, *ak*, *al*, *am*, *bb*), *vm0*)› - - - ‹*?𝒜*›])
subgoal using *lvl i* **by** *auto*
subgoal using *vm0 tr* **by** *auto*
apply (*subst* (*3*) *Down-id-eq*[*symmetric*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule find-decomp-wl-imp-find-decomp-wl′*[*THEN fref-to-Down-curry2*, *of - bt lvl*
‹((*aj*, *ak*, *al*, *am*, *bb*), *vm0*)›])
subgoal

using *that*(*1−8*) *vm vm0 bounded n-d tr*
by (*auto simp*: *find-decomp-w-ns-pre-def dest*: *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*)
subgoal by *auto*
using *ST*
by (*auto simp*: *find-decomp-w-ns-def conc-fun-RES twl-st-heur-def*)
qed
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong*
cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong heuristic-rel-cong

show *?thesis*
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-local-restart-wl-D-heur-def*
unfolding
find-decomp-wl-st-int-def find-local-restart-target-level-def incr-lrestart-stat-def
empty-Q-def find-local-restart-target-level-st-def nres-monad-laws
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule ref-two-step*)
prefer *2*
apply (*rule cdcl-twl-local-restart-wl-D-spec-int*)
unfolding *bind-to-let-conv Let-def RES-RETURN-RES2 nres-monad-laws*
apply (*refine-vcg*)
subgoal unfolding *restart-abs-wl-heur-pre-def* **by** *blast*
apply *assumption*
subgoal by (*auto simp*: *twl-st-heur-def count-decided-st-heur-def trail-pol-def*)
subgoal
by (*drule H*(*2*)) (*simp add*: *twl-st-heur-change-subsumed-clauses*)

apply (*rule P*)
apply *assumption+*
apply (*rule refine-generalise1*)
apply *assumption*
subgoal for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap bd aq ar*
as at au av aw ax ay be az bf bg bh bi bj bk bl bm bn bo bp bq br bs
bt bu bv bw bx - - by bz ca cb cc cd ce cf cg ch ci cj ck cl cm cn co cp
lvl i vm0
unfolding *RETURN-def RES-RES2-RETURN-RES RES-RES13-RETURN-RES find-decomp-w-ns-def*
conc-fun-RES
RES-RES13-RETURN-RES K K2
apply (*auto simp*: *intro-spec-iff intro*!: *ASSERT-leI isa-length-trail-pre*)
apply (*auto simp*: *isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *isa-vmtfI trail-pol-no-dup*)
apply (*frule twl-st-heur-change-subsumed-clauses*[**where** $US' = \langle\{\#\}\rangle$ **and** $NS' = cm$])
apply (*solves* ‹*auto dest*: *H*(*2*)›)[]
apply (*frule H*(*2*))
apply (*frule H*(*3*))
apply (*clarsimp simp*: *twl-st-heur-def*)
apply (*rule-tac x=aja* **in** *exI*)
apply (*auto simp*: *isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *isa-vmtfI trail-pol-no-dup*)
apply (*rule trail-pol-cong*)
apply *assumption*
apply *fast*
apply (*rule isa-vmtf-cong*)

apply *assumption*
apply (*fast intro*: *isa-vmtfI*)
done
done
qed

definition *remove-all-annot-true-clause-imp-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* ⇒ *nat watcher list* ⇒ *nat* × *twl-st-wl-heur* ⇒ *bool*›
where
‹*remove-all-annot-true-clause-imp-wl-D-heur-inv S xs* = (λ(*i*, *T*).
∃ *S′ T′*. (*S*, *S′*) ∈ *twl-st-heur-restart* ∧ (*T*, *T′*) ∈ *twl-st-heur-restart* ∧
remove-all-annot-true-clause-imp-wl-inv S′ (*map fst xs*) (*i*, *T′*))
›

definition *remove-all-annot-true-clause-one-imp-heur*
:: ‹*nat* × *nat* × *arena* ⇒ (*nat* × *arena*) *nres*›
where
‹*remove-all-annot-true-clause-one-imp-heur* = (λ(*C*, *j*, *N*). *do* {
case arena-status N C of
DELETED ⇒ *RETURN* (*j*, *N*)
| *IRRED* ⇒ *RETURN* (*j*, *extra-information-mark-to-delete N C*)
| *LEARNED* ⇒ *RETURN* (*j*−*1*, *extra-information-mark-to-delete N C*)
})›

definition *remove-all-annot-true-clause-imp-wl-D-pre*
:: ‹*nat multiset* ⇒ *nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*›
where
‹*remove-all-annot-true-clause-imp-wl-D-pre* $\mathcal{A}$ *L S* ⟷ (*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)›

definition *remove-all-annot-true-clause-imp-wl-D-heur-pre* **where**
‹*remove-all-annot-true-clause-imp-wl-D-heur-pre L S* ⟷
(∃ *S′*. (*S*, *S′*) ∈ *twl-st-heur-restart*
∧ *remove-all-annot-true-clause-imp-wl-D-pre* (*all-init-atms-st S′*) *L S′*)›

definition *remove-all-annot-true-clause-imp-wl-D-heur*
:: ‹*nat literal* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*remove-all-annot-true-clause-imp-wl-D-heur* = (λ*L* (*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
stats, *heur*, *vdom*, *avdom*, *lcount*, *opts*). *do* {
ASSERT(*remove-all-annot-true-clause-imp-wl-D-heur-pre L* (*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*,
cach, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*));
let xs = *W*!(*nat-of-lit L*);
(-, *lcount′*, *N*) ← $WHILE_T$$^{\lambda(i,\ j,\ N).}$ *remove-all-annot-true-clause-imp-wl-D-heur-inv* (*M*, *N0*, *D*, *Q*, *W*, *vm*,
(λ(*i*, *j*, *N*). *i* < *length xs*)
(λ(*i*, *j*, *N*). *do* {
ASSERT(*i* < *length xs*);
if clause-not-marked-to-delete-heur (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom*, *lcount*, *opts*) *i*
then do {
(*j*, *N*) ← *remove-all-annot-true-clause-one-imp-heur* (*fst* (*xs*!*i*), *j*, *N*);
ASSERT(*remove-all-annot-true-clause-imp-wl-D-heur-inv*
(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,

heur, *vdom*, *avdom*, *lcount*, *opts*) *xs*
(*i*, *M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom*, *j*, *opts*));
RETURN (*i+1*, *j*, *N*)
}
else
RETURN (*i+1*, *j*, *N*)
})
(*0*, *lcount*, *N0*);
RETURN (*M*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom*, *lcount'*, *opts*)
})›

definition *minimum-number-between-restarts* :: ‹*64 word*› **where**
‹*minimum-number-between-restarts = 50*›

definition *five-uint64* :: ‹*64 word*› **where**
‹*five-uint64 = 5*›

definition *upper-restart-bound-not-reached* :: ‹*twl-st-wl-heur ⇒ bool*› **where**
‹*upper-restart-bound-not-reached* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *heur*, *vdom*, *avdom*, *lcount*, *opts*).
of-nat lcount < 3000 + 1000 ∗ restarts)›

definition (**in** −) *lower-restart-bound-not-reached* :: ‹*twl-st-wl-heur ⇒ bool*› **where**
‹*lower-restart-bound-not-reached* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *heur*,
vdom, *avdom*, *lcount*, *opts*, *old*).
(¬*opts-reduce opts* ∨ (*opts-restart opts* ∧ (*of-nat lcount < 2000 + 1000 ∗ restarts*))))›

definition *reorder-vdom-wl* :: ‹*'v twl-st-wl ⇒ 'v twl-st-wl nres*› **where**
‹*reorder-vdom-wl S = RETURN S*›

definition *sort-clauses-by-score* :: ‹*arena ⇒ nat list ⇒ nat list nres*› **where**
‹*sort-clauses-by-score arena vdom = do* {
ASSERT(∀ *i*∈*set vdom*. *valid-sort-clause-score-pre-at arena i*);
SPEC(λ*vdom'*. *mset vdom = mset vdom'*)
}›

definition (**in** −) *quicksort-clauses-by-score* :: ‹*arena ⇒ nat list ⇒ nat list nres*› **where**
‹*quicksort-clauses-by-score arena* =
full-quicksort-ref clause-score-ordering2 (*clause-score-extract arena*)›

lemma *quicksort-clauses-by-score-sort*:
‹(*quicksort-clauses-by-score*, *sort-clauses-by-score*) ∈
Id → *Id* → ⟨*Id*⟩*nres-rel*›
by (*intro fun-relI nres-relI*)
(*auto simp*: *quicksort-clauses-by-score-def sort-clauses-by-score-def*
reorder-list-def clause-score-extract-def clause-score-ordering2-def
le-ASSERT-iff
intro!: *insert-sort-reorder-list*[*THEN fref-to-Down*, *THEN order-trans*])

definition *remove-deleted-clauses-from-avdom* :: ‹-› **where**
‹*remove-deleted-clauses-from-avdom N avdom0 = do* {

```
  let n = length avdom0;
  (i, j, avdom) ← WHILE_T λ(i, j, avdom). i ≤ j ∧ j ≤ n ∧ length avdom = length avdom0 ∧       mset (take i avdom @ drop
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < length avdom);
      if (avdom ! j) ∈# dom-m N then RETURN (i+1, j+1, swap avdom i j)
      else RETURN (i, j+1, avdom)
    })
    (0, 0, avdom0);
  ASSERT(i ≤ length avdom);
  RETURN (take i avdom)
}›

lemma remove-deleted-clauses-from-avdom:
  ‹remove-deleted-clauses-from-avdom N avdom0 ≤ SPEC(λavdom. mset avdom ⊆# mset avdom0)›
  unfolding remove-deleted-clauses-from-avdom-def Let-def
  apply (refine-vcg WHILEIT-rule[where R = ‹measure (λ(i, j, avdom). length avdom − j)›])
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal for s a b aa ba x1 x2 x1a x2a
    by (cases ‹Suc a ≤ aa›)
     (auto simp: drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last
       mset-append[symmetric] Cons-nth-drop-Suc simp del: mset-append
     simp flip: take-Suc-conv-app-nth)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal for s a b aa ba x1 x2 x1a x2a
    by (cases ‹Suc aa ≤ length x2a›)
      (auto simp: drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last
        Cons-nth-drop-Suc[symmetric] intro: subset-mset.dual-order.trans
     simp flip: take-Suc-conv-app-nth)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  done

definition isa-remove-deleted-clauses-from-avdom :: ‹-› where
‹isa-remove-deleted-clauses-from-avdom arena avdom0 = do {
  ASSERT(length avdom0 ≤ length arena);
  let n = length avdom0;
  (i, j, avdom) ← WHILE_T λ(i, j, -). i ≤ j ∧ j ≤ n
    (λ(i, j, avdom). j < n)
    (λ(i, j, avdom). do {
      ASSERT(j < n);
      ASSERT(arena-is-valid-clause-vdom arena (avdom!j) ∧ j < length avdom ∧ i < length avdom);
      if arena-status arena (avdom ! j) ≠ DELETED then RETURN (i+1, j+1, swap avdom i j)
```

else RETURN (*i*, *j*+*1*, *avdom*)
}) (*0*, *0*, *avdom0*);
ASSERT(*i* ≤ *length avdom*);
RETURN (*take i avdom*)
}›

lemma *isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*:
‹*valid-arena arena N* (*set vdom*) ⟹ *mset avdom0* ⊆# *mset vdom* ⟹ *distinct vdom* ⟹
isa-remove-deleted-clauses-from-avdom arena avdom0 ≤ ⇓*Id* (*remove-deleted-clauses-from-avdom N avdom0*)›
unfolding *isa-remove-deleted-clauses-from-avdom-def remove-deleted-clauses-from-avdom-def Let-def*
apply (*refine-vcg WHILEIT-refine*[**where** *R*= ‹*Id* $\times_r$ *Id* $\times_r$ ⟨*Id*⟩*list-rel*›])
subgoal by (*auto dest*!: *valid-arena-vdom-le*(*2*) *size-mset-mono simp*: *distinct-card*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c* **unfolding** *arena-is-valid-clause-vdom-def*
by (*force intro*!: *exI*[*of - N*] *exI*[*of - vdom*] *dest*!: *mset-eq-setD dest*: *mset-le-add-mset simp*: *Cons-nth-drop-Suc*[*symmetric*])
subgoal by *auto*
subgoal by *auto*
subgoal
by (*force simp*: *arena-lifting arena-dom-status-iff*(*1*) *Cons-nth-drop-Suc*[*symmetric*]
dest!: *mset-eq-setD dest*: *mset-le-add-mset*)
subgoal by *auto*
subgoal
by (*force simp*: *arena-lifting arena-dom-status-iff*(*1*) *Cons-nth-drop-Suc*[*symmetric*]
dest!: *mset-eq-setD dest*: *mset-le-add-mset*)
subgoal by *auto*
subgoal by *auto*
done

definition (**in** −) *sort-vdom-heur* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*sort-vdom-heur* = (λ(*M′*, *arena*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*). *do* {
ASSERT(*length avdom* ≤ *length arena*);
avdom ← *isa-remove-deleted-clauses-from-avdom arena avdom*;
ASSERT(*valid-sort-clause-score-pre arena avdom*);
ASSERT(*length avdom* ≤ *length arena*);
avdom ← *sort-clauses-by-score arena avdom*;
RETURN (*M′*, *arena*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*)
})›

lemma *sort-clauses-by-score-reorder*:
‹*valid-arena arena N* (*set vdom′*) ⟹ *set vdom* ⊆ *set vdom′* ⟹
sort-clauses-by-score arena vdom ≤ *SPEC*(λ*vdom′*. *mset vdom* = *mset vdom′*)›
unfolding *sort-clauses-by-score-def*
apply *refine-vcg*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
valid-sort-clause-score-pre-at-def
apply (*auto simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*2*−)

arena-dom-status-iff (*1*)[*symmetric*] *in-set-conv-nth*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of* - ‹*get-clauses-wl y*›] *dest*!: *set-mset-mono mset-subset-eqD*)
using *arena-dom-status-iff* (*1*) *nth-mem* **by** *blast*

lemma *sort-vdom-heur-reorder-vdom-wl*:
‹(*sort-vdom-heur*, *reorder-vdom-wl*) ∈ *twl-st-heur-restart-ana r* $\to_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
proof −
show *?thesis*
unfolding *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
apply (*rule ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left*)
apply (*rule-tac N1* = ‹*get-clauses-wl y*› **and** *vdom1* = ‹*get-vdom x*› **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF* - - - *remove-deleted-clauses-from-avdom*])
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2l x1m x2m x1n x2n x1o x2o
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff* (*2*−)
arena-dom-status-iff (*1*)[*symmetric*]
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of* - ‹*get-clauses-wl y*›] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal for *x y*
apply (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of* - ‹*get-clauses-wl y*› ‹*get-vdom x*›])
by (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*)
done
qed

lemma (**in** −) *insort-inner-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a ba* ⟹
mset ba = *mset a2′* ⟹
a1′ < *length a2′* ⟹
valid-sort-clause-score-pre-at a (*a2′* ! *a1′*)›
unfolding *valid-sort-clause-score-pre-def all-set-conv-nth valid-sort-clause-score-pre-at-def*
by (*metis in-mset-conv-nth*)+

lemma *sort-clauses-by-score-invI*:
⟨*valid-sort-clause-score-pre a b* ⟹
mset b = mset a2' ⟹ *valid-sort-clause-score-pre a a2'*⟩
using *mset-eq-setD* **unfolding** *valid-sort-clause-score-pre-def* **by** *blast*

definition *partition-main-clause* **where**
⟨*partition-main-clause arena = partition-main clause-score-ordering* (*clause-score-extract arena*)⟩

definition *partition-clause* **where**
⟨*partition-clause arena = partition-between-ref clause-score-ordering* (*clause-score-extract arena*)⟩

lemma *valid-sort-clause-score-pre-swap*:
⟨*valid-sort-clause-score-pre a b* ⟹ *x < length b* ⟹
ba < length b ⟹ *valid-sort-clause-score-pre a* (*swap b x ba*)⟩
by (*auto simp*: *valid-sort-clause-score-pre-def*)

definition *div2* **where** [*simp*]: ⟨*div2 n = n div 2*⟩

definition *safe-minus* **where** ⟨*safe-minus a b =* (*if b ≥ a then 0 else a − b*)⟩

definition *max-restart-decision-lvl* :: *nat* **where**
⟨*max-restart-decision-lvl = 300*⟩

definition *max-restart-decision-lvl-code* :: ⟨*32 word*⟩ **where**
⟨*max-restart-decision-lvl-code = 300*⟩

definition *GC-EVERY* :: ⟨*64 word*⟩ **where**
⟨*GC-EVERY = 15*⟩ — hard-coded limit

fun (**in** −) *get-reductions-count* :: ⟨*twl-st-wl-heur* ⇒ *64 word*⟩ **where**
⟨*get-reductions-count* (-, -, -, -, -, -, -,-,-,-,
(-, -, -, *lres*, -, -), -)
= *lres*⟩

definition *get-restart-phase* :: ⟨*twl-st-wl-heur* ⇒ *64 word*⟩ **where**
⟨*get-restart-phase* = (λ(-, -, -, -, -, -, -, -, -, -, *heur*, -).
current-restart-phase heur)⟩

definition *GC-required-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool nres* **where**
⟨*GC-required-heur S n = do* {
n ← *RETURN* (*full-arena-length-st S*);
wasted ← *RETURN* (*wasted-bytes-st S*);
RETURN (*3∗wasted* > ((*of-nat n*)>>*2*))
}⟩

definition *FLAG-no-restart* :: ⟨*8 word*⟩ **where**
⟨*FLAG-no-restart = 0*⟩

definition *FLAG-restart* :: ⟨*8 word*⟩ **where**
⟨*FLAG-restart = 1*⟩

definition *FLAG-GC-restart* :: ⟨*8 word*⟩ **where**
⟨*FLAG-GC-restart = 2*⟩

definition *restart-flag-rel* :: ⟨(*8 word* × *restart-type*) *set*⟩ **where**
⟨*restart-flag-rel* = {(*FLAG-no-restart*, *NO-RESTART*), (*FLAG-restart*, *RESTART*), (*FLAG-GC-restart*, *GC*)}⟩

definition *restart-required-heur* :: *twl-st-wl-heur* ⇒ *nat* ⇒ *8 word nres* **where**
⟨*restart-required-heur S n* = *do* {
let opt-red = *opts-reduction-st S*;
let opt-res = *opts-restart-st S*;
let curr-phase = *get-restart-phase S*;
let lcount = *get-learned-count S*;
let can-res = (*lcount* > *n*);

if ¬*can-res* ∨ ¬*opt-res* ∨ ¬*opt-red then RETURN FLAG-no-restart*
else if curr-phase = *QUIET-PHASE*
then do {
GC-required ← *GC-required-heur S n*;
let upper = *upper-restart-bound-not-reached S*;
if (*opt-res* ∨ *opt-red*) ∧ ¬*upper*
then RETURN FLAG-GC-restart
else RETURN FLAG-no-restart
}
else do {
let sema = *ema-get-value* (*get-slow-ema-heur S*);
let limit = (*shiftr* (*11* ∗ *sema*) (*4*::*nat*));
let fema = *ema-get-value* (*get-fast-ema-heur S*);
let ccount = *get-conflict-count-since-last-restart-heur S*;
let min-reached = (*ccount* > *minimum-number-between-restarts*);
let level = *count-decided-st-heur S*;
let should-not-reduce = (¬*opt-red* ∨ *upper-restart-bound-not-reached S*);
let should-reduce = ((*opt-res* ∨ *opt-red*) ∧
(*should-not-reduce* ⟶ *limit* > *fema*) ∧ *min-reached* ∧ *can-res* ∧
level > *2* ∧ ~~This comment from Marijn Heule seems not to help: **term** ⟨level ≤ max-restart-decision-lvl⟩~~
of-nat level > (*shiftr fema 32*));
GC-required ← *GC-required-heur S n*;
if should-reduce
then if GC-required
then RETURN FLAG-GC-restart
else RETURN FLAG-restart
else RETURN FLAG-no-restart
}
}⟩

lemma (**in** −) *get-reduction-count-alt-def*:
⟨*RETURN o get-reductions-count* = (λ(*M*, *N0*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
(-, -, -, *lres*, -, -), *heur*, *lcount*). *RETURN lres*)⟩
by *auto*

definition *mark-to-delete-clauses-wl-D-heur-pre* :: ⟨*twl-st-wl-heur* ⇒ *bool*⟩ **where**
⟨*mark-to-delete-clauses-wl-D-heur-pre S* ⟷
(∃ *S′*. (*S*, *S′*) ∈ *twl-st-heur-restart* ∧ *mark-to-delete-clauses-wl-pre S′*)⟩

lemma *mark-to-delete-clauses-wl-post-alt-def*:
⟨*mark-to-delete-clauses-wl-post S0 S* ⟷

($\exists$ *T0 T*.
  (*S0*, *T0*) $\in$ *state-wl-l None* $\wedge$
  (*S*, *T*) $\in$ *state-wl-l None* $\wedge$
  blits-in-$\mathcal{L}_{in}$ *S0* $\wedge$
  blits-in-$\mathcal{L}_{in}$ *S* $\wedge$
  ($\exists$ *U0 U*. (*T0*, *U0*) $\in$ *twl-st-l None* $\wedge$
    (*T*, *U*) $\in$ *twl-st-l None* $\wedge$
    remove-one-annot-true-clause** *T0 T* $\wedge$
    twl-list-invs T0 $\wedge$
    twl-struct-invs U0 $\wedge$
    twl-list-invs T $\wedge$
    twl-struct-invs U $\wedge$
    get-conflict-l T0 = *None* $\wedge$
  clauses-to-update-l T0 = {#}) $\wedge$
  correct-watching S0 $\wedge$ *correct-watching S*)›
unfolding *mark-to-delete-clauses-wl-post-def mark-to-delete-clauses-l-post-def*
  mark-to-delete-clauses-l-pre-def
apply (*rule iffI*; *normalize-goal*+)
subgoal for *T0 T U0*
  apply (*rule exI*[*of - T0*])
  apply (*rule exI*[*of - T*])
  apply (*intro conjI*)
  apply *auto*[*4*]
  apply (*rule exI*[*of - U0*])
  apply *auto*
  using *rtranclp-remove-one-annot-true-clause-cdcl-twl-restart-l2*[*of T0 T U0*]
    rtranclp-cdcl-twl-restart-l-list-invs[*of T0*]
  apply (*auto dest*:)
  using *rtranclp-cdcl-twl-restart-l-list-invs* **by** *blast*
subgoal for *T0 T U0 U*
  apply (*rule exI*[*of - T0*])
  apply (*rule exI*[*of - T*])
  apply (*intro conjI*)
  apply *auto*[*3*]
  apply (*rule exI*[*of - U0*])
  apply *auto*
  done
done

lemma *mark-to-delete-clauses-wl-D-heur-pre-alt-def*:
  ‹*mark-to-delete-clauses-wl-D-heur-pre S* $\longleftrightarrow$
    ($\exists$ *S'*. (*S*, *S'*) $\in$ *twl-st-heur* $\wedge$ *mark-to-delete-clauses-wl-pre S'*)› (**is** *?A*) **and**
  mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur:
    ‹*mark-to-delete-clauses-wl-pre T* $\Longrightarrow$
      (*S*, *T*) $\in$ *twl-st-heur* $\longleftrightarrow$ (*S*, *T*) $\in$ *twl-st-heur-restart*› (**is** ‹- $\Longrightarrow$ - *?B*›) **and**
  mark-to-delete-clauses-wl-post-twl-st-heur:
    ‹*mark-to-delete-clauses-wl-post T0 T* $\Longrightarrow$
      (*S*, *T*) $\in$ *twl-st-heur* $\longleftrightarrow$ (*S*, *T*) $\in$ *twl-st-heur-restart*› (**is** ‹- $\Longrightarrow$ - *?C*›)
proof –
  note *cong* = *trail-pol-cong heuristic-rel-cong*
    option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong phase-saving-cong*
    cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
    isasat-input-bounded-cong

  show *?A*
    supply [[*goals-limit=1*]]

unfolding *mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def mark-to-delete-clauses-l-pre-def*
apply (*rule iffI*)
apply *normalize-goal*+
subgoal for *T U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*rule exI*[*of - T*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - U*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - V*])
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
apply *normalize-goal*+
subgoal for *T U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*rule exI*[*of - T*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - U*])
apply (*intro conjI*) **defer**
apply (*rule exI*[*of - V*])
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
show ‹*mark-to-delete-clauses-wl-pre T* ⟹ *?B*›
supply [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def mark-to-delete-clauses-l-pre-def mark-to-delete-clauses-wl-pre-def*
apply *normalize-goal*+
apply (*rule iffI*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
show ‹*mark-to-delete-clauses-wl-post T0 T* ⟹ *?C*›
supply [[*goals-limit=1*]]

unfolding *mark-to-delete-clauses-wl-post-alt-def*
apply *normalize-goal*+
apply (*rule iffI*)
subgoal for *U0 U V0 V*
using *literals-are-$\mathcal{L}_{in}$'-literals-are-$\mathcal{L}_{in}$-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done
subgoal for *U0 U V0 V*
using *literals-are-$\mathcal{L}_{in}$'-literals-are-$\mathcal{L}_{in}$-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
done

qed

lemma *mark-garbage-heur-wl*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
‹¬ *irred* (*get-clauses-wl T*) *C*› **and** ‹*i* < *length* (*get-avdom S*)›
shows ‹(*mark-garbage-heur C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def mset-butlast-remove1-mset*
simp del: *all-init-atms-def*[*symmetric*]
intro: *valid-arena-extra-information-mark-to-delete'*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *mark-garbage-heur-wl-ana*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
‹¬ *irred* (*get-clauses-wl T*) *C*› **and** ‹*i* < *length* (*get-avdom S*)›
shows ‹(*mark-garbage-heur C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart-ana r*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-garbage-heur-def mark-garbage-wl-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If init-clss-l-fmdrop-irrelev
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro: *valid-arena-extra-information-mark-to-delete'*

dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *mark-unused-st-heur-ana*:
assumes
⟨(*S*, *T*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
⟨*C* ∈# *dom-m* (*get-clauses-wl T*)⟩
shows ⟨(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart-ana r*⟩
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-unused-st-heur-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused valid-arena-arena-decr-act*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *twl-st-heur-restart-valid-arena*[*twl-st-heur-restart*]:
assumes
⟨(*S*, *T*) ∈ *twl-st-heur-restart*⟩
shows ⟨*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))⟩
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def*)

lemma *twl-st-heur-restart-get-avdom-nth-get-vdom*[*twl-st-heur-restart*]:
assumes
⟨(*S*, *T*) ∈ *twl-st-heur-restart*⟩ ⟨*i* < *length* (*get-avdom S*)⟩
shows ⟨*get-avdom S* ! *i* ∈ *set* (*get-vdom S*)⟩
using *assms* **by** (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *set-mset-mono*)

lemma [*twl-st-heur-restart*]:
assumes
⟨(*S*, *T*) ∈ *twl-st-heur-restart*⟩ **and**
⟨*C* ∈ *set* (*get-avdom S*)⟩
shows ⟨*clause-not-marked-to-delete-heur S C* ⟷
(*C* ∈# *dom-m* (*get-clauses-wl T*))⟩ **and**
⟨*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*⟩**and**
⟨*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*⟩
⟨*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)⟩
proof −
show ⟨*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))⟩
using *assms*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-dom-status-iff(*1*)
split: *prod.splits*)
assume *C*: ⟨*C* ∈# *dom-m* (*get-clauses-wl T*)⟩
show ⟨*arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*⟩
using *assms C*
by (*cases S*; *cases T*)

(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹*arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹*arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
qed

definition *number-clss-to-keep* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
‹*number-clss-to-keep* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *heur*,
vdom, *avdom*, *lcount*).
RES UNIV)›

definition *number-clss-to-keep-impl* :: ‹*twl-st-wl-heur* ⇒ *nat nres*› **where**
‹*number-clss-to-keep-impl* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*,
(*props*, *decs*, *confl*, *restarts*, -), *heur*,
vdom, *avdom*, *lcount*).
let n = *unat* (*1000* + *150* ∗ *restarts*) *in RETURN* (*if n* ≥ *sint64-max then sint64-max else n*))›

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹(*number-clss-to-keep-impl*, *number-clss-to-keep*) ∈ *Id* $\rightarrow_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*auto simp*: *number-clss-to-keep-impl-def number-clss-to-keep-def Let-def intro*!: *frefI nres-relI*)

definition (**in** −) *MINIMUM-DELETION-LBD* :: *nat* **where**
‹*MINIMUM-DELETION-LBD* = *3*›

lemma *in-set-delete-index-and-swapD*:
‹*x* ∈ *set* (*delete-index-and-swap xs i*) ⟹ *x* ∈ *set xs*›
apply (*cases* ‹*i* < *length xs*›)
apply (*auto dest*!: *in-set-butlastD*)
by (*metis List.last-in-set in-set-upd-cases list.size*(*3*) *not-less-zero*)

lemma *delete-index-vdom-heur-twl-st-heur-restart*:
‹(*S*, *T*) ∈ *twl-st-heur-restart* ⟹ *i* < *length* (*get-avdom S*) ⟹
(*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart*›
by (*auto simp*: *twl-st-heur-restart-def delete-index-vdom-heur-def*
dest: *in-set-delete-index-and-swapD*)

lemma *delete-index-vdom-heur-twl-st-heur-restart-ana*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *i* < *length* (*get-avdom S*) ⟹
(*delete-index-vdom-heur i S*, *T*) ∈ *twl-st-heur-restart-ana r*›
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def delete-index-vdom-heur-def*
dest: *in-set-delete-index-and-swapD*)

definition *mark-clauses-as-unused-wl-D-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›
where
‹*mark-clauses-as-unused-wl-D-heur* = (λ*i S*. *do* {
(-, *T*) ← $WHILE_T$
(λ(*i*, *S*). *i* < *length* (*get-avdom S*))
(λ(*i*, *T*). *do* {
ASSERT(*i* < *length* (*get-avdom T*));
ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-avdom S*));
ASSERT(*access-vdom-at-pre T i*);
let C = *get-avdom T* ! *i*;
ASSERT(*clause-not-marked-to-delete-heur-pre* (*T*, *C*));
if ¬*clause-not-marked-to-delete-heur T C then RETURN* (*i*, *delete-index-vdom-heur i T*)
else do {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*i+1*, (*mark-unused-st-heur C T*))
}
})
(*i*, *S*);
RETURN T
})›

lemma *avdom-delete-index-vdom-heur*[*simp*]:
‹*get-avdom* (*delete-index-vdom-heur i S*) =
delete-index-and-swap (*get-avdom S*) *i*›
by (*cases S*) (*auto simp*: *delete-index-vdom-heur-def*)

lemma *incr-wasted-st*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*›
shows ‹(*incr-wasted-st C S*, *T*) ∈ *twl-st-heur-restart-ana r*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def incr-wasted-st-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def heuristic-rel-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused valid-arena-arena-decr-act*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *incr-wasted-st-twl-st*[*simp*]:
‹*get-avdom* (*incr-wasted-st w T*) = *get-avdom T*›
‹*get-vdom* (*incr-wasted-st w T*) = *get-vdom T*›
‹*get-trail-wl-heur* (*incr-wasted-st w T*) = *get-trail-wl-heur T*›
‹*get-clauses-wl-heur* (*incr-wasted-st C T*) = *get-clauses-wl-heur T*›
‹*get-conflict-wl-heur* (*incr-wasted-st C T*) = *get-conflict-wl-heur T*›
‹*get-learned-count* (*incr-wasted-st C T*) = *get-learned-count T*›
‹*get-conflict-count-heur* (*incr-wasted-st C T*) = *get-conflict-count-heur T*›
by (*cases T*; *auto simp*: *incr-wasted-st-def*)+

lemma *mark-clauses-as-unused-wl-D-heur*:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*›

shows ‹*mark-clauses-as-unused-wl-D-heur i S* $\leq \Downarrow$ (*twl-st-heur-restart-ana r*) (*SPEC* ((=) *T*))›
proof −
have *1*: ‹ $\Downarrow$ (*twl-st-heur-restart-ana r*) (*SPEC* ((=) *T*)) = *do* {
(*i*, *T*) ← *SPEC* (λ(*i*::*nat*, *T'*). (*T'*, *T*) ∈ *twl-st-heur-restart-ana r*);
RETURN T
}›
by (*auto simp*: *RES-RETURN-RES2 uncurry-def conc-fun-RES*)
show *?thesis*
unfolding *mark-clauses-as-unused-wl-D-heur-def 1 mop-arena-length-st-def*
apply (*rule Refine-Basic.bind-mono*)
subgoal
apply (*refine-vcg*
WHILET-rule[**where** *R* = ‹*measure* (λ(*i*, *T*). *length* (*get-avdom T*) − *i*)› **and**
I = ‹λ(-, *S'*). (*S'*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *length* (*get-avdom S'*) ≤ *length*(*get-avdom S*)›])
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *access-vdom-at-pre-def* **by** *auto*
subgoal for *st a S'*
unfolding *clause-not-marked-to-delete-heur-pre-def*
arena-is-valid-clause-vdom-def
by (*auto 7 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *set-mset-mono*
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *exI*[*of* - ‹*set* (*get-vdom S'*)›])
subgoal
by (*auto intro*: *delete-index-vdom-heur-twl-st-heur-restart-ana*)
subgoal by *auto*
subgoal by *auto*
subgoal
unfolding *arena-is-valid-clause-idx-def*
arena-is-valid-clause-vdom-def arena-act-pre-def
by (*fastforce simp*: *twl-st-heur-restart-def twl-st-heur-restart*
dest!: *twl-st-heur-restart-anaD*)
subgoal for *s a b*
apply (*auto intro*!: *mark-unused-st-heur-ana*)
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*
arena-is-valid-clause-idx-def
arena-is-valid-clause-vdom-def arena-act-pre-def
by (*fastforce simp*: *twl-st-heur-restart-def twl-st-heur-restart*
intro!: *mark-unused-st-heur-ana*
dest!: *twl-st-heur-restart-anaD*)
subgoal
unfolding *twl-st-heur-restart-ana-def*
by (*auto simp*: *twl-st-heur-restart-def*)
subgoal
by (*auto intro*!: *mark-unused-st-heur-ana incr-wasted-st simp*: *twl-st-heur-restart*
dest: *twl-st-heur-restart-anaD*)
subgoal by *auto*
done
subgoal by *auto*
done
qed

definition *mark-to-delete-clauses-wl-D-heur*
:: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*›

where
‹*mark-to-delete-clauses-wl-D-heur* = (λ*S0*. *do* {
  ASSERT(*mark-to-delete-clauses-wl-D-heur-pre S0*);
  S ← *sort-vdom-heur S0*;
  l ← *number-clss-to-keep S*;
  ASSERT(*length* (*get-avdom S*) ≤ *length* (*get-clauses-wl-heur S0*));
  (*i*, *T*) ← *WHILE*$_T$$^{\lambda\text{-}.\ True}$
    (λ(*i*, *S*). *i* < *length* (*get-avdom S*))
    (λ(*i*, *T*). *do* {
      ASSERT(*i* < *length* (*get-avdom T*));
      ASSERT(*access-vdom-at-pre T i*);
      let C = *get-avdom T* ! *i*;
      ASSERT(*clause-not-marked-to-delete-heur-pre* (*T*, *C*));
      b ← *mop-clause-not-marked-to-delete-heur T C*;
      if ¬*b then RETURN* (*i*, *delete-index-vdom-heur i T*)
      else do {
        ASSERT(*access-lit-in-clauses-heur-pre* ((*T*, *C*), *0*));
        ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *length* (*get-clauses-wl-heur S0*));
        ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-clauses-wl-heur T*));
        L ← *mop-access-lit-in-clauses-heur T C 0*;
        D ← *get-the-propagation-reason-pol* (*get-trail-wl-heur T*) *L*;
        lbd ← *mop-arena-lbd* (*get-clauses-wl-heur T*) *C*;
        length ← *mop-arena-length* (*get-clauses-wl-heur T*) *C*;
        status ← *mop-arena-status* (*get-clauses-wl-heur T*) *C*;
        used ← *mop-marked-as-used* (*get-clauses-wl-heur T*) *C*;
        let can-del = (*D* ≠ *Some C*) ∧
    lbd > *MINIMUM-DELETION-LBD* ∧
          status = *LEARNED* ∧
          length ≠ *2* ∧
    ¬*used*;
        if can-del
        then
          do {
            wasted ← *mop-arena-length-st T C*;
            T ← *mop-mark-garbage-heur C i* (*incr-wasted-st* (*of-nat wasted*) *T*);
            RETURN (*i*, *T*)
          }
        else do {
    T ← *mop-mark-unused-st-heur C T*;
          RETURN (*i+1*, *T*)
  }
      }
    })
    (*l*, *S*);
  ASSERT(*length* (*get-avdom T*) ≤ *length* (*get-clauses-wl-heur S0*));
  T ← *mark-clauses-as-unused-wl-D-heur i T*;
  incr-restart-stat T
})›

lemma *twl-st-heur-restart-same-annotD*:
‹(*S*, *T*) ∈ *twl-st-heur-restart* ⟹ *Propagated L C* ∈ *set* (*get-trail-wl T*) ⟹
  Propagated L C' ∈ *set* (*get-trail-wl T*) ⟹ *C* = *C'*›
‹(*S*, *T*) ∈ *twl-st-heur-restart* ⟹ *Propagated L C* ∈ *set* (*get-trail-wl T*) ⟹
  Decided L ∈ *set* (*get-trail-wl T*) ⟹ *False*›
by (*auto simp*: *twl-st-heur-restart-def dest*: *no-dup-no-propa-and-dec*
  no-dup-same-annotD)

lemma $\mathcal{L}_{all}$*-mono*:
⟨*set-mset* $\mathcal{A} \subseteq$ *set-mset* $\mathcal{B} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow L \in\# \mathcal{L}_{all}\ \mathcal{B}$⟩
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *all-lits-of-mm-mono2*:
⟨$x \in\#$ (*all-lits-of-mm A*) $\Longrightarrow$ *set-mset A* $\subseteq$ *set-mset B* $\Longrightarrow x \in\#$ (*all-lits-of-mm B*)⟩
by (*auto simp*: *all-lits-of-mm-def*)

lemma $\mathcal{L}_{all}$*-init-all*:
⟨$L \in\# \mathcal{L}_{all}$ (*all-init-atms-st x1a*) $\Longrightarrow L \in\# \mathcal{L}_{all}$ (*all-atms-st x1a*)⟩
apply (*rule* $\mathcal{L}_{all}$*-mono*)
defer
apply *assumption*
by (*cases x1a*)
(*auto simp*: *all-init-atms-def all-lits-def all-init-lits-def*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-atms-def intro*: *all-lits-of-mm-mono2 intro*!: *imageI*
simp del: *all-init-atms-def*[*symmetric*]
simp flip: *image-mset-union*)

lemma *get-vdom-mark-garbage*[*simp*]:
⟨*get-vdom* (*mark-garbage-heur C i S*) = *get-vdom S*⟩
⟨*get-avdom* (*mark-garbage-heur C i S*) = *delete-index-and-swap* (*get-avdom S*) *i*⟩
by (*cases S*; *auto simp*: *mark-garbage-heur-def*; *fail*)+

lemma *mark-to-delete-clauses-wl-D-heur-alt-def*:
⟨*mark-to-delete-clauses-wl-D-heur* = (λ*S0*. *do* {
ASSERT (*mark-to-delete-clauses-wl-D-heur-pre S0*);
S ← *sort-vdom-heur S0*;
\- ← *RETURN* (*get-avdom S*);
l ← *number-clss-to-keep S*;
ASSERT
(*length* (*get-avdom S*) ≤ *length* (*get-clauses-wl-heur S0*));
(*i*, *T*) ←
$WHILE_T{}^{\lambda\text{-}.\ True}$ (λ(*i*, *S*). *i* < *length* (*get-avdom S*))
(λ(*i*, *T*). *do* {
ASSERT (*i* < *length* (*get-avdom T*));
ASSERT (*access-vdom-at-pre T i*);
ASSERT
(*clause-not-marked-to-delete-heur-pre*
(*T*, *get-avdom T* ! *i*));
b ← *mop-clause-not-marked-to-delete-heur T*
(*get-avdom T* ! *i*);
if ¬*b then RETURN* (*i*, *delete-index-vdom-heur i T*)
else do {
ASSERT
(*access-lit-in-clauses-heur-pre*
((*T*, *get-avdom T* ! *i*), *0*));
ASSERT
(*length* (*get-clauses-wl-heur T*)
≤ *length* (*get-clauses-wl-heur S0*));
ASSERT
(*length* (*get-avdom T*)
≤ *length* (*get-clauses-wl-heur T*));
L ← *mop-access-lit-in-clauses-heur T*

(*get-avdom* T ! i) *0*;
$D \leftarrow$ *get-the-propagation-reason-pol*
(*get-trail-wl-heur* T) L;
ASSERT
(*get-clause-LBD-pre* (*get-clauses-wl-heur* T)
(*get-avdom* T ! i));
ASSERT
(*arena-is-valid-clause-idx*
(*get-clauses-wl-heur* T) (*get-avdom* T ! i));
ASSERT
(*arena-is-valid-clause-vdom*
(*get-clauses-wl-heur* T) (*get-avdom* T ! i));
ASSERT
(*marked-as-used-pre*
(*get-clauses-wl-heur* T) (*get-avdom* T ! i));
let can-del = ($D \neq$ *Some* (*get-avdom* T ! i) $\wedge$
MINIMUM-DELETION-LBD
< *arena-lbd* (*get-clauses-wl-heur* T)
(*get-avdom* T ! i) $\wedge$
arena-status (*get-clauses-wl-heur* T)
(*get-avdom* T ! i) =
LEARNED $\wedge$
arena-length (*get-clauses-wl-heur* T)
(*get-avdom* T ! i) $\neq$
2 $\wedge$
$\neg$ *marked-as-used* (*get-clauses-wl-heur* T)
(*get-avdom* T ! i));
if can-del
then do {
wasted $\leftarrow$ *mop-arena-length-st* T (*get-avdom* T ! i);
ASSERT(*mark-garbage-pre*
(*get-clauses-wl-heur* T, *get-avdom* T ! i) $\wedge$
1 $\leq$ *get-learned-count* T $\wedge$ i < *length* (*get-avdom* T));
RETURN
(i, *mark-garbage-heur* (*get-avdom* T ! i) i (*incr-wasted-st* (*of-nat wasted*) T))
}
else do {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur* T) (*get-avdom* T ! i));
RETURN
(i + *1*,
mark-unused-st-heur (*get-avdom* T ! i) T)
}
}
})
(l, S);
ASSERT
(*length* (*get-avdom* T) $\leq$ *length* (*get-clauses-wl-heur* *S0*));
mark-clauses-as-unused-wl-D-heur i T $\gg=$ *incr-restart-stat*
})›
unfolding *mark-to-delete-clauses-wl-D-heur-def*
mop-arena-lbd-def mop-arena-status-def mop-arena-length-def
mop-marked-as-used-def bind-to-let-conv Let-def
nres-monad3 mop-mark-garbage-heur-def mop-mark-unused-st-heur-def
incr-wasted-st-twl-st
by (*auto intro*!: *ext simp*: *get-clauses-wl-heur.simps*)

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D*:
‹(*mark-to-delete-clauses-wl-D-heur*, *mark-to-delete-clauses-wl*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
proof −
have *mark-to-delete-clauses-wl-D-alt-def*:
‹*mark-to-delete-clauses-wl* = (λ*S0*. *do* {
ASSERT(*mark-to-delete-clauses-wl-pre S0*);
S ← *reorder-vdom-wl S0*;
xs ← *collect-valid-indices-wl S*;
l ← *SPEC*(λ-::*nat*. *True*);
(-, *S*, -) ← $WHILE_T{}^{mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}inv\ S\ xs}$
(λ(*i*, *T*, *xs*). *i* < *length xs*)
(λ(*i*, *T*, *xs*). *do* {
b ← *RETURN* (*xs*!*i* ∈# *dom-m* (*get-clauses-wl T*));
if ¬*b then RETURN* (*i*, *T*, *delete-index-and-swap xs i*)
else do {
ASSERT(*0* < *length* (*get-clauses-wl T*∝(*xs*!*i*)));
ASSERT (*get-clauses-wl T* ∝ (*xs* ! *i*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st T*));
K ← *RETURN* (*get-clauses-wl T* ∝ (*xs* ! *i*) ! *0*);
b ← *RETURN* (); — propagation reason
can-del ← *SPEC*(λ*b*. *b* ⟶
(*Propagated* (*get-clauses-wl T*∝(*xs*!*i*)!*0*) (*xs*!*i*) ∉ *set* (*get-trail-wl T*)) ∧
¬*irred* (*get-clauses-wl T*) (*xs*!*i*) ∧ *length* (*get-clauses-wl T* ∝ (*xs*!*i*)) ≠ *2*);
ASSERT(*i* < *length xs*);
if can-del
then
RETURN (*i*, *mark-garbage-wl* (*xs*!*i*) *T*, *delete-index-and-swap xs i*)
else
RETURN (*i*+*1*, *T*, *xs*)
}
})
(*l*, *S*, *xs*);
remove-all-learned-subsumed-clauses-wl S
})›
unfolding *mark-to-delete-clauses-wl-def reorder-vdom-wl-def bind-to-let-conv Let-def*
by (*force intro*!: *ext*)
have *mono*: ‹*g* = *g′* ⟹ *do* {*f*; *g*} = *do* {*f*; *g′*}›
‹(⋀*x*. *h x* = *h′ x*) ⟹ *do* {*x* ← *f*; *h x*} = *do* {*x* ← *f*; *h′ x*}› **for** *f f′* :: ‹- *nres*› **and** *g g′* **and** *h h′*
by *auto*

have [*refine0*]: ‹*RETURN* (*get-avdom x*) ≤ ⇓ {(*xs*, *xs′*). *xs* = *xs′* ∧ *xs* = *get-avdom x*} (*collect-valid-indices-wl y*)›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*›
for *x y*
proof −
show *?thesis* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed
have *init-rel*[*refine0*]: ‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r* ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
((*l*, *x*), *la*, *y*) ∈ *nat-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *get-avdom S* = *get-avdom x*}›
for *x y l la*
by *auto*

define *reason-rel* **where**
‹*reason-rel K x1a* ≡ {(*C*, - :: *unit*).
(*C* ≠ *None*) = (*Propagated K* (*the C*) ∈ *set* (*get-trail-wl x1a*)) ∧
(*C* = *None*) = (*Decided K* ∈ *set* (*get-trail-wl x1a*) ∨
K ∉ *lits-of-l* (*get-trail-wl x1a*)) ∧
(∀ *C1*. (*Propagated K C1* ∈ *set* (*get-trail-wl x1a*) ⟶ *C1* = *the C*))}› **for** *K* :: ‹*nat literal*› **and** *x1a*
have *get-the-propagation-reason*:
‹*get-the-propagation-reason-pol* (*get-trail-wl-heur x2b*) *L*
≤ *SPEC* (λ*D*. (*D*, ()) ∈ *reason-rel K x1a*)›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧
V = *y* ∧
(*mark-to-delete-clauses-wl-pre y* ⟶
mark-to-delete-clauses-wl-pre V) ∧
(*mark-to-delete-clauses-wl-D-heur-pre x* ⟶
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*ys*, *xs*) ∈ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-avdom S*}› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {-. *True*}› **and**
xa-x': ‹(*xa*, *x'*)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)› **and**
‹*case x' of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*x1b* < *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
dom: ‹(*b*, *ba*)
∈ {(*b*, *b'*).
(*b*, *b'*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*› **and**
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))› **and**
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-avdom x2b* ! *x1b*), *0*)› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
L: ‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)› **and**
L': ‹(*L*, *K*)
∈ {(*L*, *L'*).
(*L*, *L'*) ∈ *nat-lit-lit-rel* ∧
L' = *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*}›
for *x y S Sa xs' xs l la xa x' x1 x2 x1a x2a x1' x2' x3 x1b ys x2b L K b ba*
proof −
have *L*: ‹*arena-lit* (*get-clauses-wl-heur x2b*) (*x2a* ! *x1*) ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
using *L that* **by** (*auto simp*: *twl-st-heur-restart st arena-lifting dest*: $\mathcal{L}_{all}$-*init-all twl-st-heur-restart-anaD*)

show *?thesis*
apply (*rule order.trans*)
apply (*rule get-the-propagation-reason-pol*[*THEN fref-to-Down-curry*,

of ‹*all-init-atms-st x1a*› ‹*get-trail-wl x1a*›
‹*arena-lit* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b* + *0*)›])
subgoal
using *xa-x′ L L′* **by** (*auto simp add*: *twl-st-heur-restart-def st*)
subgoal
using *xa-x′ L′ dom* **by** (*auto simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def st arena-lifting*)
using *that* **unfolding** *get-the-propagation-reason-def reason-rel-def* **apply** −
by (*auto simp*: *twl-st-heur-restart lits-of-def get-the-propagation-reason-def conc-fun-RES*
dest!: *twl-st-heur-restart-anaD dest*: *twl-st-heur-restart-same-annotD imageI*[*of* - - *lit-of*])
qed
have ‹((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom*, *lcount*),
S′)
∈ *twl-st-heur-restart* ⟹
((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*, *vdom*, *avdom′*, *lcount*),
S′)
∈ *twl-st-heur-restart*›
if ‹*mset avdom′* ⊆# *mset avdom*›
for *M′ N′ D′ j W′ vm clvls cach lbd outl stats fast-ema slow-ema ccount vdom lcount S′ avdom′ avdom heur*
using *that* **unfolding** *twl-st-heur-restart-def*
by *auto*
then have *mark-to-delete-clauses-wl-D-heur-pre-vdom′*:
‹*mark-to-delete-clauses-wl-D-heur-pre* (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom′*, *lcount*) ⟹
mark-to-delete-clauses-wl-D-heur-pre (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom*, *lcount*)›
if ‹*mset avdom* ⊆# *mset avdom′*›
for *M′ N′ D′ j W′ vm clvls cach lbd outl stats fast-ema slow-ema avdom avdom′ ccount vdom lcount heur*
using *that*
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def*
by *metis*
have [*refine0*]:
‹*sort-vdom-heur S* ≤ ⇓ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧ *V* = *T* ∧
(*mark-to-delete-clauses-wl-pre T* ⟶ *mark-to-delete-clauses-wl-pre V*) ∧
(*mark-to-delete-clauses-wl-D-heur-pre S* ⟶ *mark-to-delete-clauses-wl-D-heur-pre U*)}
(*reorder-vdom-wl T*)›
if ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **for** *S T*
using *that* **unfolding** *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*refine-rcg ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left*)
apply (*rule-tac N1* = ‹*get-clauses-wl T*› **and** *vdom1* = ‹(*get-vdom S*)› **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF* - - - *remove-deleted-clauses-from-avdom*])
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l*
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l*
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l*

by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def*
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def intro*!: *exI*[*of* - ‹*get-clauses-wl T*›] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def valid-arena-vdom-subset twl-st-heur-restart-def dest*!: *size-mset-mono valid-arena-vdom-subset*)
subgoal
apply (*rewrite at* ‹- $\leq$ ¤› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of* - ‹*get-clauses-wl T*› ‹*get-vdom S*›])
by (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def intro*: *mark-to-delete-clauses-wl-D-heur-pre-vdom′ dest*: *mset-eq-setD*)
done
have *already-deleted*:
‹((*x1b*, *delete-index-vdom-heur x1b x2b*), *x1*, *x1a*, *delete-index-and-swap x2a x1*) $\in$ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) $\in$ *twl-st-heur-restart-ana r* $\wedge$ *xs* = *get-avdom Sa*}›
if
‹(*x*, *y*) $\in$ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
$\in$ {(*U*, *V*).
(*U*, *V*) $\in$ *twl-st-heur-restart-ana r* $\wedge$
V = *y* $\wedge$
(*mark-to-delete-clauses-wl-pre y* $\longrightarrow$
mark-to-delete-clauses-wl-pre V) $\wedge$
(*mark-to-delete-clauses-wl-D-heur-pre x* $\longrightarrow$
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*l*, *la*) $\in$ *nat-rel*› **and**
‹*la* $\in$ {-. *True*}› **and**
xx: ‹(*xa*, *x′*)
$\in$ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) $\in$ *twl-st-heur-restart-ana r* $\wedge$ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) $\Rightarrow$ *i* < *length* (*get-avdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) $\Rightarrow$ *i* < *length xs*› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
le: ‹*x1b* < *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
‹(*b*, *ba*) $\in$ {(*b*, *b′*). (*b*, *b′*) $\in$ *bool-rel* $\wedge$ *b* = (*x2a* ! *x1* $\in$# *dom-m* (*get-clauses-wl x1a*))}› **and**
‹$\neg$*ba*›
for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa ba b*
proof −
show *?thesis*
using *xx le* **unfolding** *st*
by (*auto simp*: *twl-st-heur-restart-ana-def delete-index-vdom-heur-def*

twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
learned-clss-l-l-fmdrop size-remove1-mset-If
intro: *valid-arena-extra-information-mark-to-delete′*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)

qed

have *get-learned-count-ge*: ‹*Suc 0* ≤ *get-learned-count x2b*›
if
xy: ‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹(*xa*, *x′*)
∈ *nat-rel* $\times_f$
{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
dom: ‹(*b*, *ba*)
∈ {(*b*, *b′*).
(*b*, *b′*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*› **and**
‹*MINIMUM-DELETION-LBD*
< *arena-lbd* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) ∧
arena-status (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) = *LEARNED* ∧
arena-length (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) ≠ *2* ∧
¬ *marked-as-used* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*)› **and**
‹*can-del*› **for** *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b D can-del b ba*

proof −
have ‹¬*irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)› **and** ‹(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*›
using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting*
dest!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena*)
then show *?thesis*
using *dom* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*)

qed

have *mop-clause-not-marked-to-delete-heur*:
‹*mop-clause-not-marked-to-delete-heur x2b* (*get-avdom x2b* ! *x1b*)
≤ *SPEC*
(λ*c*. (*c*, *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))
∈ {(*b*, *b′*). (*b*,*b′*) ∈ *bool-rel* ∧ (*b* ⟷ *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))})›
if
‹(*xa*, *x′*)
∈ *nat-rel* $\times_f$
{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl-inv Sa xs x′*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)›
for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *mop-clause-not-marked-to-delete-heur-def*
apply *refine-vcg*

subgoal
  using *that* **by** *blast*
subgoal
  using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting dest*!: *twl-st-heur-restart-anaD*)
done

have *init*:
‹$(u, xs) \in \{(xs, xs'). xs = xs' \wedge xs = get\text{-}avdom\ S\} \Longrightarrow$
$(l, la) \in nat\text{-}rel \Longrightarrow$
$(S, Sa) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r \Longrightarrow$
$((l, S), la, Sa, xs) \in\ nat\text{-}rel \times_f$
  $\{(Sa, (T, xs)). (Sa, T) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r \wedge xs = get\text{-}avdom\ Sa\}$›
  for *x y S Sa xs l la u*
by *auto*
have *mop-access-lit-in-clauses-heur*:
‹$mop\text{-}access\text{-}lit\text{-}in\text{-}clauses\text{-}heur\ x2b\ (get\text{-}avdom\ x2b\ !\ x1b)\ 0$
  $\leq SPEC$
   $(\lambda c. (c, get\text{-}clauses\text{-}wl\ x1a \propto (x2a\ !\ x1)\ !\ 0)$
     $\in \{(L, L'). (L, L') \in nat\text{-}lit\text{-}lit\text{-}rel \wedge L' = get\text{-}clauses\text{-}wl\ x1a \propto (x2a\ !\ x1)\ !\ 0\})$›
if
 ‹$(x, y) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r$› **and**
 ‹$mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}pre\ y$› **and**
 ‹$mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}D\text{-}heur\text{-}pre\ x$› **and**
 ‹(S, Sa)
  $\in \{(U, V).$
   $(U, V) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r \wedge$
   $V = y \wedge$
   $(mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}pre\ y \longrightarrow$
    $mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}pre\ V) \wedge$
   $(mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}D\text{-}heur\text{-}pre\ x \longrightarrow$
    $mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}D\text{-}heur\text{-}pre\ U)\}$› **and**
 ‹$(uu, xs) \in \{(xs, xs'). xs = xs' \wedge xs = get\text{-}avdom\ S\}$› **and**
 ‹$(l, la) \in nat\text{-}rel$› **and**
 ‹$la \in \{uu. True\}$› **and**
 ‹$length\ (get\text{-}avdom\ S) \leq length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x)$› **and**
 ‹(xa, x')
  $\in nat\text{-}rel \times_f$
   $\{(Sa, T, xs).$
   $(Sa, T) \in twl\text{-}st\text{-}heur\text{-}restart\text{-}ana\ r \wedge xs = get\text{-}avdom\ Sa\}$› **and**
 ‹$case\ xa\ of\ (i, S) \Rightarrow i < length\ (get\text{-}avdom\ S)$› **and**
 ‹$case\ x'\ of\ (i, T, xs) \Rightarrow i < length\ xs$› **and**
 ‹$mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl\text{-}inv\ Sa\ xs\ x'$› **and**
 ‹$x2 = (x1a, x2a)$› **and**
 ‹$x' = (x1, x2)$› **and**
 ‹$xa = (x1b, x2b)$› **and**
 ‹$x1b < length\ (get\text{-}avdom\ x2b)$› **and**
 ‹$access\text{-}vdom\text{-}at\text{-}pre\ x2b\ x1b$› **and**
 ‹$clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur\text{-}pre\ (x2b, get\text{-}avdom\ x2b\ !\ x1b)$› **and**
 ‹(b, ba)
  $\in \{(b, b').$
   $(b, b') \in bool\text{-}rel \wedge$
   $b = (x2a\ !\ x1 \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ x1a))\}$› **and**
 ‹$\neg \neg b$› **and**
 ‹$\neg \neg ba$› **and**
 ‹$0 < length\ (get\text{-}clauses\text{-}wl\ x1a \propto (x2a\ !\ x1))$› **and**

⟨*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*
∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)⟩ **and**
pre: ⟨*access-lit-in-clauses-heur-pre* ((*x2b*, *get-avdom x2b* ! *x1b*), *0*)⟩
for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba*
unfolding *mop-access-lit-in-clauses-heur-def mop-arena-lit2-def*
apply *refine-vcg*
subgoal using *pre* **unfolding** *access-lit-in-clauses-heur-pre-def* **by** *simp*
subgoal using *that* **by** (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting*)
done

have *incr-restart-stat*: ⟨*incr-restart-stat T*
≤ ⇓ (*twl-st-heur-restart-ana r*) (*remove-all-learned-subsumed-clauses-wl S*)⟩
if ⟨(*T*, *S*) ∈ *twl-st-heur-restart-ana r*⟩ **for** *S T i*
using *that*
by (*cases S*; *cases T*)
(*auto simp*: *conc-fun-RES incr-restart-stat-def*
twl-st-heur-restart-ana-def twl-st-heur-restart-def
remove-all-learned-subsumed-clauses-wl-def
RES-RETURN-RES)

have [*refine0*]: ⟨*mark-clauses-as-unused-wl-D-heur i T* ≫= *incr-restart-stat*
≤ ⇓ (*twl-st-heur-restart-ana r*)
(*remove-all-learned-subsumed-clauses-wl S*)⟩
if ⟨(*T*, *S*) ∈ *twl-st-heur-restart-ana r*⟩ **for** *S T i*
apply (*cases S*)
apply (*rule bind-refine-res*[**where** *R* = *Id*, *simplified*])
defer
apply (*rule mark-clauses-as-unused-wl-D-heur*[*unfolded conc-fun-RES*, *OF that*, *of i*])
apply (*rule incr-restart-stat*[*THEN order-trans*, *of - S*])
by *auto*

show *?thesis*
supply *sort-vdom-heur-def*[*simp*] *twl-st-heur-restart-anaD*[*dest*] [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-D-heur-alt-def mark-to-delete-clauses-wl-D-alt-def*
access-lit-in-clauses-heur-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg sort-vdom-heur-reorder-vdom-wl*[*THEN fref-to-Down*])
subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y S T* **unfolding** *number-clss-to-keep-def* **by** (*cases S*) (*auto*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule init*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-vdom-at-pre-def*)
subgoal for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d*
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def*
prod.simps
by (*rule exI*[*of -* ⟨*get-clauses-wl x2a*⟩], *rule exI*[*of -* ⟨*set* (*get-vdom x2d*)⟩])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-get-avdom-nth-get-vdom*)
apply (*rule mop-clause-not-marked-to-delete-heur*; *assumption*)
subgoal for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b*

by (*auto simp*: *twl-st-heur-restart*)
subgoal
by (*rule already-deleted*)
subgoal for *x y - - - - - xs l la xa x′ x1 x2 x1a x2a*
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of -* ‹*get-avdom x2a* ! *x1a*›], *simp*, *rule exI*[*of -* ‹*get-clauses-wl x1*›])
(*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal premises *p* **using** *p*(*7*−) **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule mop-access-lit-in-clauses-heur*; *assumption*)
apply (*rule get-the-propagation-reason*; *assumption*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps get-clause-LBD-pre-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl x1a*›], *rule exI*[*of -* ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl x1a*›], *rule exI*[*of -* ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl x1a*›], *rule exI*[*of -* ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart arena-dom-status-iff dest*: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal
unfolding *marked-as-used-pre-def*
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal
unfolding *marked-as-used-pre-def*
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart*)
subgoal
by (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting*)
subgoal by *fast*
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *mop-arena-length-st-def*
apply (*rule mop-arena-length*[*THEN fref-to-Down-curry*, *THEN order-trans*, *of* ‹*get-clauses-wl x1a*› ‹*get-avdom x2b* ! *x1b*› *- -* ‹*set* (*get-vdom x2b*)›])
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-restart-valid-arena*)
subgoal
apply (*auto intro*!: *incr-wasted-st-twl-st ASSERT-leI*)
subgoal
unfolding *prod.simps mark-garbage-pre-def arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
by (*rule exI*[*of -* ‹*get-clauses-wl x1a*›], *rule exI*[*of -* ‹*set* (*get-vdom x2b*)›])

(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal
apply (*rule get-learned-count-ge*; *assumption?*; *fast?*)
apply *auto*
done
subgoal
by (*use arena-lifting*(*24*)[*of* ‹*get-clauses-wl-heur x2b*› - - ‹*get-avdom x2b* ! *x1*›] **in**
‹*auto intro*!: *incr-wasted-st mark-garbage-heur-wl-ana*
dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD*›)
done
done
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *dest*!: *set-mset-mono mset-subset-eqD*)
subgoal
by (*auto intro*!: *mark-unused-st-heur-ana*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal
by *auto*
done
qed

definition *cdcl-twl-full-restart-wl-prog-heur* **where**
‹*cdcl-twl-full-restart-wl-prog-heur S* = *do* {
- ← *ASSERT* (*mark-to-delete-clauses-wl-D-heur-pre S*);
T ← *mark-to-delete-clauses-wl-D-heur S*;
RETURN T
}›

lemma *cdcl-twl-full-restart-wl-prog-heur-cdcl-twl-full-restart-wl-prog-D*:
‹(*cdcl-twl-full-restart-wl-prog-heur*, *cdcl-twl-full-restart-wl-prog*) ∈
twl-st-heur''' *r* $\rightarrow_f$ ⟨*twl-st-heur*''' *r*⟩*nres-rel*›
unfolding *cdcl-twl-full-restart-wl-prog-heur-def cdcl-twl-full-restart-wl-prog-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D[*THEN fref-to-Down*])
subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-alt-def*
by *fast*
apply (*rule twl-st-heur-restartD*)
subgoal
by (*subst mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur*[*symmetric*]) *auto*
subgoal
by (*auto simp*: *mark-to-delete-clauses-wl-post-twl-st-heur twl-st-heur-restart-anaD*)
(*auto simp*: *twl-st-heur-restart-ana-def*)
done

definition *cdcl-twl-restart-wl-heur* **where**
‹*cdcl-twl-restart-wl-heur S* = *do* {
let b = *lower-restart-bound-not-reached S*;
if b then cdcl-twl-local-restart-wl-D-heur S
else cdcl-twl-full-restart-wl-prog-heur S

})

lemma *cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog*:
⟨(*cdcl-twl-restart-wl-heur*, *cdcl-twl-restart-wl-prog*) ∈
twl-st-heur''' r $\rightarrow_f$ ⟨*twl-st-heur''' r*⟩*nres-rel*⟩
unfolding *cdcl-twl-restart-wl-prog-def cdcl-twl-restart-wl-heur-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec[*THEN fref-to-Down*]
cdcl-twl-full-restart-wl-prog-heur-cdcl-twl-full-restart-wl-prog-D[*THEN fref-to-Down*])
subgoal by *auto*
subgoal by *auto*
done

definition *isasat-replace-annot-in-trail*
:: ⟨*nat literal* ⇒ *nat* ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩
where
⟨*isasat-replace-annot-in-trail L C* = (λ((*M*, *val*, *lvls*, *reason*, *k*), *oth*). *do* {
ASSERT(*atm-of L* < *length reason*);
RETURN ((*M*, *val*, *lvls*, *reason*[*atm-of L* := *0*], *k*), *oth*)
})⟩

lemma $\mathcal{L}_{all}$*-atm-of-all-init-lits-of-mm*:
⟨*set-mset* ($\mathcal{L}_{all}$ (*atm-of* '# *all-init-lits N NUE*)) = *set-mset* (*all-init-lits N NUE*)⟩
by (*auto simp*: *all-init-lits-def* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*)

lemma *trail-pol-replace-annot-in-trail-spec*:
assumes
⟨*atm-of x2* < *length x1e*⟩ **and**
x2: ⟨*atm-of x2* ∈# *all-init-atms-st* (*ys* @ *Propagated x2 C* # *zs*, *x2n'*)⟩ **and**
⟨(((*x1b*, *x1c*, *x1d*, *x1e*, *x2d*), *x2n*),
(*ys* @ *Propagated x2 C* # *zs*, *x2n'*))
∈ *twl-st-heur-restart-ana r*⟩
shows
⟨(((*x1b*, *x1c*, *x1d*, *x1e*[*atm-of x2* := *0*], *x2d*), *x2n*),
(*ys* @ *Propagated x2 0* # *zs*, *x2n'*))
∈ *twl-st-heur-restart-ana r*⟩
proof −
let *?S* = ⟨(*ys* @ *Propagated x2 C* # *zs*, *x2n'*)⟩
let *?*$\mathcal{A}$ = ⟨*all-init-atms-st ?S*⟩
have *pol*: ⟨((*x1b*, *x1c*, *x1d*, *x1e*, *x2d*), *ys* @ *Propagated x2 C* # *zs*)
∈ *trail-pol* (*all-init-atms-st ?S*)⟩
using *assms*(*3*) **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
by *auto*
obtain *x y* **where**
x2d: ⟨*x2d* = (*count-decided* (*ys* @ *Propagated x2 C* # *zs*), *y*)⟩ **and**
reasons: ⟨((*map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*)), *x1e*),
ys @ *Propagated x2 C* # *zs*)
∈ *ann-lits-split-reasons ?*$\mathcal{A}$⟩ **and**
pol: ⟨∀ *L*∈#$\mathcal{L}_{all}$ *?*$\mathcal{A}$. *nat-of-lit L* < *length x1c* ∧
x1c ! *nat-of-lit L* = *polarity* (*ys* @ *Propagated x2 C* # *zs*) *L*⟩ **and**
n-d: ⟨*no-dup* (*ys* @ *Propagated x2 C* # *zs*)⟩ **and**
lvls: ⟨∀ *L*∈#$\mathcal{L}_{all}$ *?*$\mathcal{A}$. *atm-of L* < *length x1d* ∧
x1d ! *atm-of L* = *get-level* (*ys* @ *Propagated x2 C* # *zs*) *L*⟩ **and**

⟨*undefined-lit ys* (*lit-of* (*Propagated x2 C*))⟩ **and**
⟨*undefined-lit zs* (*lit-of* (*Propagated x2 C*))⟩ **and**
inA:⟨∀ *L*∈*set* (*ys* @ *Propagated x2 C* # *zs*). *lit-of L* ∈# $\mathcal{L}_{all}$ *?*$\mathcal{A}$⟩ **and**
cs: ⟨*control-stack y* (*ys* @ *Propagated x2 C* # *zs*)⟩ **and**
⟨*literals-are-in-*$\mathcal{L}_{in}$*-trail ?*$\mathcal{A}$ (*ys* @ *Propagated x2 C* # *zs*)⟩ **and**
⟨*length* (*ys* @ *Propagated x2 C* # *zs*) < *uint32-max*⟩ **and**
⟨*length* (*ys* @ *Propagated x2 C* # *zs*) ≤ *uint32-max div 2* + *1*⟩ **and**
⟨*count-decided* (*ys* @ *Propagated x2 C* # *zs*) < *uint32-max*⟩ **and**
⟨*length* (*map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*))) =
length (*ys* @ *Propagated x2 C* # *zs*)⟩ **and**
bounded: ⟨*isasat-input-bounded ?*$\mathcal{A}$⟩ **and**
x1b: ⟨*x1b* = *map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*))⟩
using *pol* **unfolding** *trail-pol-alt-def*
by *blast*
have *lev-eq*: ⟨*get-level* (*ys* @ *Propagated x2 C* # *zs*) =
get-level (*ys* @ *Propagated x2 0* # *zs*)⟩
⟨*count-decided* (*ys* @ *Propagated x2 C* # *zs*) =
count-decided (*ys* @ *Propagated x2 0* # *zs*)⟩
by (*auto simp*: *get-level-cons-if get-level-append-if*)
have [*simp*]: ⟨*atm-of x2* < *length x1e*⟩
using *reasons x2 in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
by (*auto simp*: *ann-lits-split-reasons-def* $\mathcal{L}_{all}$*-all-init-atms all-init-atms-def*
$\mathcal{L}_{all}$*-atm-of-all-init-lits-of-mm*
simp del: *all-init-atms-def*[*symmetric*]
dest: *multi-member-split*)

have ⟨((*x1b*, *x1e*[*atm-of x2* := *0*]), *ys* @ *Propagated x2 0* # *zs*)
∈ *ann-lits-split-reasons ?*$\mathcal{A}$⟩
using *reasons n-d undefined-notin*
by (*auto simp*: *ann-lits-split-reasons-def x1b*
DECISION-REASON-def atm-of-eq-atm-of)
moreover have *n-d′*: ⟨*no-dup* (*ys* @ *Propagated x2 0* # *zs*)⟩
using *n-d* **by** *auto*
moreover have ⟨∀ *L*∈#$\mathcal{L}_{all}$ *?*$\mathcal{A}$.
nat-of-lit L < *length x1c* ∧
x1c ! *nat-of-lit L* = *polarity* (*ys* @ *Propagated x2 0* # *zs*) *L*⟩
using *pol* **by** (*auto simp*: *polarity-def*)
moreover have ⟨∀ *L*∈#$\mathcal{L}_{all}$ *?*$\mathcal{A}$.
atm-of L < *length x1d* ∧
x1d ! *atm-of L* = *get-level* (*ys* @ *Propagated x2 0* # *zs*) *L*⟩
using *lvls* **by** (*auto simp*: *get-level-append-if get-level-cons-if*)
moreover have ⟨*control-stack y* (*ys* @ *Propagated x2 0* # *zs*)⟩
using *cs* **apply** −
apply (*subst control-stack-alt-def*[*symmetric*, *OF n-d′*])
apply (*subst* (*asm*) *control-stack-alt-def*[*symmetric*, *OF n-d*])
unfolding *control-stack′-def lev-eq*
apply *normalize-goal*
apply (*intro ballI conjI*)
apply (*solves auto*)
unfolding *set-append list.set*(*2*) *Un-iff insert-iff*
apply (*rule disjE*, *assumption*)
subgoal for *L*
by (*drule-tac x* = *L* **in** *bspec*)
(*auto simp*: *nth-append nth-Cons split*: *nat.splits*)
apply (*rule disjE*[*of* ⟨- = -⟩], *assumption*)
subgoal for *L*

by (*auto simp*: *nth-append nth-Cons split*: *nat.splits*)
subgoal for *L*
by (*drule-tac x* = *L* **in** *bspec*)
(*auto simp*: *nth-append nth-Cons split*: *nat.splits*)

done
ultimately have
‹((*x1b*, *x1c*, *x1d*, *x1e*[*atm-of x2* := *0*], *x2d*), *ys* @ *Propagated x2 0* # *zs*)
∈ *trail-pol ?𝒜*›
using *n-d x2 inA bounded*
unfolding *trail-pol-def x2d*
by *simp*
moreover { **fix** *aaa ca*
have ‹*vmtf-*$\mathcal{L}_{all}$ (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 C* # *zs*) =
vmtf-$\mathcal{L}_{all}$ (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *vmtf-*$\mathcal{L}_{all}$*-def*)
then have ‹*isa-vmtf* (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 C* # *zs*) =
isa-vmtf (*all-init-atms aaa ca*) (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *isa-vmtf-def vmtf-def*
image-iff)
}
moreover { **fix** *D*
have ‹*get-level* (*ys* @ *Propagated x2 C* # *zs*) = *get-level* (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *get-level-append-if get-level-cons-if*)
then have ‹*counts-maximum-level* (*ys* @ *Propagated x2 C* # *zs*) *D* =
counts-maximum-level (*ys* @ *Propagated x2 0* # *zs*) *D*› **and**
‹*out-learned* (*ys* @ *Propagated x2 C* # *zs*) = *out-learned* (*ys* @ *Propagated x2 0* # *zs*)›
by (*auto simp*: *counts-maximum-level-def card-max-lvl-def*
out-learned-def intro!: *ext*)
}
ultimately show *?thesis*
using *assms*(*3*) **unfolding** *twl-st-heur-restart-ana-def*
by (*cases x2n*; *cases x2n′*)
(*auto simp*: *twl-st-heur-restart-def*
simp flip: *mset-map drop-map*)
qed

lemmas *trail-pol-replace-annot-in-trail-spec2* =
trail-pol-replace-annot-in-trail-spec[*of* ‹− -›, *simplified*]

lemma $\mathcal{L}_{all}$*-ball-all*:
‹(∀ *L* ∈# $\mathcal{L}_{all}$ (*all-atms N NUE*). *P L*) = (∀ *L* ∈# *all-lits N NUE*. *P L*)›
‹(∀ *L* ∈# $\mathcal{L}_{all}$ (*all-init-atms N NUE*). *P L*) = (∀ *L* ∈# *all-init-lits N NUE*. *P L*)›
by (*simp-all add*: $\mathcal{L}_{all}$*-all-atms-all-lits* $\mathcal{L}_{all}$*-all-init-atms*)

lemma *twl-st-heur-restart-ana-US-empty*:
‹*NO-MATCH* {#} *US* ⟹ (*S*, *M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *W*, *Q*) ∈ *twl-st-heur-restart-ana r* ⟷
(*S*, *M*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *W*, *Q*)
∈ *twl-st-heur-restart-ana r*›
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)

fun *equality-except-trail-empty-US-wl* :: ‹′*v twl-st-wl* ⇒ ′*v twl-st-wl* ⇒ *bool*› **where**
‹*equality-except-trail-empty-US-wl* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*)
(*M′*, *N′*, *D′*, *NE′*, *UE′*, *NS′*, *US′*, *WS′*, *Q′*) ⟷
N = *N′* ∧ *D* = *D′* ∧ *NE* = *NE′* ∧ *NS* = *NS′* ∧ *US* = {#} ∧ *UE* = *UE′* ∧ *WS* = *WS′* ∧ *Q* = *Q′*›

lemma *equality-except-conflict-wl-get-clauses-wl*:
‹*equality-except-conflict-wl S Y* $\Longrightarrow$ *get-clauses-wl S* = *get-clauses-wl Y*› **and**
equality-except-conflict-wl-get-trail-wl:
‹*equality-except-conflict-wl S Y* $\Longrightarrow$ *get-trail-wl S* = *get-trail-wl Y*› **and**
equality-except-trail-empty-US-wl-get-conflict-wl:
‹*equality-except-trail-empty-US-wl S Y* $\Longrightarrow$ *get-conflict-wl S* = *get-conflict-wl Y*› **and**
equality-except-trail-empty-US-wl-get-clauses-wl:
‹*equality-except-trail-empty-US-wl S Y* $\Longrightarrow$ *get-clauses-wl S* = *get-clauses-wl Y*›
by (*cases S*; *cases Y*; *solves auto*)+

lemma *isasat-replace-annot-in-trail-replace-annot-in-trail-spec*:
‹(((*L*, *C*), *S*), ((*L'*, *C'*), *S'*)) ∈ *Id* $\times_f$ *Id* $\times_f$ *twl-st-heur-restart-ana r* $\Longrightarrow$
isasat-replace-annot-in-trail L C S ≤
⇓{(*U*, *U'*). (*U*, *U'*) ∈ *twl-st-heur-restart-ana r* ∧
get-clauses-wl-heur U = *get-clauses-wl-heur S* ∧
get-vdom U = *get-vdom S* ∧
equality-except-trail-empty-US-wl U' S'}
(*replace-annot-wl L' C' S'*)›
unfolding *isasat-replace-annot-in-trail-def replace-annot-wl-def*
uncurry-def
apply *refine-rcg*
subgoal
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def* $\mathcal{L}_{all}$*-ball-all*
twl-st-heur-restart-def twl-st-heur-restart-ana-def replace-annot-wl-pre-def)
subgoal for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f*
x2f x1g x2g x1h x1i
x2h x1j x2i x1k x2j x1l
unfolding *replace-annot-wl-pre-def replace-annot-l-pre-def*
apply (*clarify dest*!: *split-list*[*of* ‹*Propagated - -*›])
apply (*rule RETURN-SPEC-refine*)
apply (*rule-tac x* = ‹(*ys* @ *Propagated L 0* # *zs*, *x1*, *x2*, *x1b*,
x1c, *x1d*, {#}, *x1f*, *x2f*)› **in** *exI*)
apply (*intro conjI*)
prefer *2*
apply (*rule-tac x* = ‹*ys* @ *Propagated L 0* # *zs*› **in** *exI*)
apply (*intro conjI*)
apply *blast*
by (*cases x1l*; *auto intro*!: *trail-pol-replace-annot-in-trail-spec*
trail-pol-replace-annot-in-trail-spec2
simp: *atm-of-eq-atm-of all-init-atms-def replace-annot-wl-pre-def*
$\mathcal{L}_{all}$*-ball-all replace-annot-l-pre-def state-wl-l-def*
twl-st-heur-restart-ana-US-empty
simp del: *all-init-atms-def*[*symmetric*])+
done

definition *remove-one-annot-true-clause-one-imp-wl-D-heur*
:: ‹*nat* ⇒ *twl-st-wl-heur* ⇒ (*nat* × *twl-st-wl-heur*) *nres*›
where
‹*remove-one-annot-true-clause-one-imp-wl-D-heur* = (*λi S. do* {
(*L*, *C*) ← *do* {
L ← *isa-trail-nth* (*get-trail-wl-heur S*) *i*;
C ← *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *L*;
RETURN (*L*, *C*)};
ASSERT(*C* ≠ *None* ∧ *i* + *1* ≤ *Suc* (*uint32-max div 2*));
if the C = *0 then RETURN* (*i*+*1*, *S*)
else do {

ASSERT($C \neq$ *None*);
$S \leftarrow$ *isasat-replace-annot-in-trail L* (*the C*) *S*;
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *the C*) $\wedge$ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) (*the C*));
$S \leftarrow$ *mark-garbage-heur2* (*the C*) *S*;
— $S \leftarrow$ *remove-all-annot-true-clause-imp-wl-D-heur L S*;
RETURN (*i*+*1*, *S*)
}
})›

definition *cdcl-twl-full-restart-wl-D-GC-prog-heur-post* :: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur* $\Rightarrow$ *bool*› **where**
‹*cdcl-twl-full-restart-wl-D-GC-prog-heur-post S T* $\longleftrightarrow$
($\exists S'$ T'. (S, S') $\in$ *twl-st-heur-restart* $\wedge$ (T, T') $\in$ *twl-st-heur-restart* $\wedge$
cdcl-twl-full-restart-wl-GC-prog-post S' T')›

definition *remove-one-annot-true-clause-imp-wl-D-heur-inv*
:: ‹*twl-st-wl-heur* $\Rightarrow$ (*nat* $\times$ *twl-st-wl-heur*) $\Rightarrow$ *bool*› **where**
‹*remove-one-annot-true-clause-imp-wl-D-heur-inv S* = (λ(*i*, *T*).
($\exists S'$ T'. (S, S') $\in$ *twl-st-heur-restart* $\wedge$ (T, T') $\in$ *twl-st-heur-restart* $\wedge$
remove-one-annot-true-clause-imp-wl-inv S' (*i*, T')))›

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*›
where
‹*remove-one-annot-true-clause-imp-wl-D-heur* = (λS. *do* {
ASSERT((*isa-length-trail-pre o get-trail-wl-heur*) *S*);
$k \leftarrow$ (*if count-decided-st-heur S* = *0*
then RETURN (*isa-length-trail* (*get-trail-wl-heur S*))
else get-pos-of-level-in-trail-imp (*get-trail-wl-heur S*) *0*);
(-, *S*) $\leftarrow$ $WHILE_T{}^{remove\text{-}one\text{-}annot\text{-}true\text{-}clause\text{-}imp\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S}$
(λ(*i*, *S*). $i < k$)
(λ(*i*, *S*). *remove-one-annot-true-clause-one-imp-wl-D-heur i S*)
(*0*, *S*);
RETURN S
})›

lemma *get-pos-of-level-in-trail-le-decomp*:
assumes
‹(*S*, *T*) $\in$ *twl-st-heur-restart*›
shows ‹*get-pos-of-level-in-trail* (*get-trail-wl T*) *0*
$\leq$ *SPEC*
(λk. $\exists$ *M1*. ($\exists$ *M2 K*.
(*Decided K* # *M1*, *M2*)
$\in$ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) $\wedge$
count-decided M1 = *0* $\wedge$ *k* = *length M1*)›
unfolding *get-pos-of-level-in-trail-def*
proof (*rule SPEC-rule*)
fix *x*
assume *H*: ‹$x <$ *length* (*get-trail-wl T*) $\wedge$
is-decided (*rev* (*get-trail-wl T*) ! *x*) $\wedge$
get-level (*get-trail-wl T*) (*lit-of* (*rev* (*get-trail-wl T*) ! *x*)) = *0* + *1*›
let *?M1* = ‹*rev* (*take x* (*rev* (*get-trail-wl T*)))›
let *?K* = ‹*Decided* ((*lit-of*(*rev* (*get-trail-wl T*) ! *x*)))›
let *?M2* = ‹*rev* (*drop* (*Suc x*) (*rev* (*get-trail-wl T*)))›
have *T*: ‹(*get-trail-wl T*) = *?M2* @ *?K* # *?M1*› **and**
K: ‹*Decided* (*lit-of ?K*) = *?K*›

apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* (*get-trail-wl T*) − *Suc x*›])
apply (*subst Cons-nth-drop-Suc*[*symmetric*])
using *H*
apply (*auto simp*: *rev-take rev-drop rev-nth*)
apply (*cases* ‹*rev* (*get-trail-wl T*) ! *x*›)
apply (*auto simp*: *rev-take rev-drop rev-nth*)
done
have *n-d*: ‹*no-dup* (*get-trail-wl T*)›
using *assms*(*1*)
by (*auto simp*: *twl-st-heur-restart-def*)
obtain *M2* **where**
‹(*?K # ?M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))›
using *get-all-ann-decomposition-ex*[*of* ‹*lit-of ?K*› *?M1 ?M2*]
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *T*[*symmetric*])
by *blast*
moreover have ‹*count-decided ?M1* = *0*›
using *n-d H*
by (*subst* (*asm*)(*1*) *T*, *subst* (*asm*)(*11*)*T*, *subst T*) *auto*
moreover have ‹*x* = *length ?M1*›
using *n-d H* **by** *auto*
ultimately show ‹∃ *M1*. (∃ *M2 K*. (*Decided K # M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) ∧
count-decided M1 = *0* ∧ *x* = *length M1* ›
by *blast*
qed

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *isa-length-trail* (*get-trail-wl-heur S*) = *length* (*get-trail-wl T*)›
unfolding *isa-length-trail-def twl-st-heur-restart-ana-def twl-st-heur-restart-def trail-pol-def*
by (*cases S*) (*auto dest*: *ann-lits-split-reasons-map-lit-of*)

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
fixes *S* :: *twl-st-wl-heur*
shows ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
unfolding *count-decided-st-def twl-st-heur-restart-ana-def trail-pol-def twl-st-heur-restart-def*
by (*cases S*) (*auto simp*: *count-decided-st-heur-def*)

lemma *twl-st-heur-restart-trailD*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹
(*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup M* ⟹ *k* < *length M* ⟹ *M* ! *k* = *Propagated L C* ⟹ *Decided L* ∉ *set M*›
apply (*auto dest*!: *split-list simp*: *nth-append nth-Cons defined-lit-def in-set-conv-nth*
split: *if-splits nat.splits*)
by (*metis no-dup-no-propa-and-dec nth-mem*)

lemma *remove-all-annot-true-clause-imp-wl-inv-length-cong*:
‹*remove-all-annot-true-clause-imp-wl-inv S xs T* ⟹
length xs = *length ys* ⟹ *remove-all-annot-true-clause-imp-wl-inv S ys T*›
by (*auto simp*: *remove-all-annot-true-clause-imp-wl-inv-def*
remove-all-annot-true-clause-imp-inv-def)

lemma *get-literal-and-reason*:
assumes
⟨$((k, S), k', T) \in$ *nat-rel* $\times_f$ *twl-st-heur-restart-ana r*⟩ **and**
⟨*remove-one-annot-true-clause-one-imp-wl-pre* k' T⟩ **and**
proped: ⟨*is-proped* (*rev* (*get-trail-wl* T) ! k')⟩
shows ⟨*do* {
$L \leftarrow$ *isa-trail-nth* (*get-trail-wl-heur* S) k;
$C \leftarrow$ *get-the-propagation-reason-pol* (*get-trail-wl-heur* S) L;
RETURN (L, C)
} $\leq \Downarrow \{((L, C), L', C').\ L = L' \wedge C' = \textit{the } C \wedge C \neq \textit{None}\}$
(*SPEC* (λp. *rev* (*get-trail-wl* T) ! k' = *Propagated* (*fst p*) (*snd p*)))⟩
proof −
have *n-d*: ⟨*no-dup* (*get-trail-wl* T)⟩ **and**
res: ⟨$((k, S), k', T) \in$ *nat-rel* $\times_f$ *twl-st-heur-restart*⟩
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
from *no-dup-nth-proped-dec-notin*[*OF this*(*1*), *of* ⟨*length* (*get-trail-wl* T) − *Suc* k'⟩]
have *dec-notin*: ⟨*Decided* (*lit-of* (*rev* (*fst* T) ! k')) $\notin$ *set* (*fst* T)⟩
using *proped assms*(*2*) **by** (*cases* T; *cases* ⟨*rev* (*get-trail-wl* T) ! k'⟩)
(*auto simp*: *twl-st-heur-restart-def state-wl-l-def*
remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
remove-one-annot-true-clause-one-imp-pre-def rev-nth
dest: *no-dup-nth-proped-dec-notin*)
have k': ⟨$k' <$ *length* (*get-trail-wl* T)⟩ **and** [*simp*]: ⟨*fst* T = *get-trail-wl* T⟩
using *proped assms*(*2*)
by (*cases* T; *auto simp*: *twl-st-heur-restart-def state-wl-l-def*
remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
remove-one-annot-true-clause-one-imp-pre-def; *fail*)+
define k'' **where** ⟨$k'' \equiv$ *length* (*get-trail-wl* T) − *Suc* k'⟩
have k'': ⟨$k'' <$ *length* (*get-trail-wl* T)⟩
using k' **by** (*auto simp*: k''*-def*)
have ⟨*rev* (*get-trail-wl* T) ! k' = *get-trail-wl* T ! k''⟩
using k' k'' **by** (*auto simp*: k''*-def nth-rev*)
then have ⟨*rev-trail-nth* (*fst* T) k' $\in\#$ $\mathcal{L}_{all}$ (*all-init-atms-st* T)⟩
using k'' *assms nth-mem*[*OF* k']
by (*auto simp*: *twl-st-heur-restart-ana-def rev-trail-nth-def*
trail-pol-alt-def twl-st-heur-restart-def)
then have *1*: ⟨(*SPEC* (λp. *rev* (*get-trail-wl* T) ! k' = *Propagated* (*fst p*) (*snd p*))) =
do {
$L \leftarrow$ *RETURN* (*rev-trail-nth* (*fst* T) k');
ASSERT($L \in\# \mathcal{L}_{all}$ (*all-init-atms-st* T));
$C \leftarrow$ (*get-the-propagation-reason* (*fst* T) L);
ASSERT($C \neq$ *None*);
RETURN (L, *the* C)
}⟩
using *proped dec-notin* k' *nth-mem*[*OF* k''] *no-dup-same-annotD*[*OF n-d*]
apply (*subst order-class.eq-iff*)
apply (*rule conjI*)
subgoal
unfolding *get-the-propagation-reason-def*
by (*cases* ⟨*rev* (*get-trail-wl* T) ! k'⟩)
(*auto simp*: *RES-RES-RETURN-RES rev-trail-nth-def*
get-the-propagation-reason-def lits-of-def rev-nth
RES-RETURN-RES
dest: *split-list*
simp flip: k''*-def*

intro!: *le-SPEC-bindI*[*of* - ‹*Some* (*mark-of* (*get-trail-wl* *T* ! *k''*))›])
  subgoal
    apply (*cases* ‹*rev* (*get-trail-wl* *T*) ! *k'*›)
    apply (*auto simp*: *RES-RES-RETURN-RES* *rev-trail-nth-def*
        get-the-propagation-reason-def *lits-of-def* *rev-nth*
RES-RETURN-RES
      simp flip: *k''-def*
      dest: *split-list*
      intro!: *exI*[*of* - ‹*Some* (*mark-of* (*rev* (*fst* *T*) ! *k'*))›])
  apply (*subst* *RES-ASSERT-moveout*)
  apply (*auto simp*: *RES-RETURN-RES*
      dest: *split-list*)
done
  done

show *?thesis*
  supply *RETURN-as-SPEC-refine*[*refine2 del*]
  apply (*subst 1*)
  apply (*refine-rcg*
    isa-trail-nth-rev-trail-nth[*THEN fref-to-Down-curry*, *unfolded comp-def*,
      of - - - - ‹*all-init-atms-st* *T*›]
    get-the-propagation-reason-pol[*THEN fref-to-Down-curry*, *unfolded comp-def*,
      of ‹*all-init-atms-st* *T*›])
  subgoal using k' **by** *auto*
  subgoal using *assms* **by** (*cases* *S*; *auto dest*: *twl-st-heur-restart-trailD*)
  subgoal by *auto*
  subgoal for *K* K'
    using *assms* **by** (*auto simp*: *twl-st-heur-restart-def* *twl-st-heur-restart-ana-def*)
  subgoal
    by *auto*
  done
qed

lemma *red-in-dom-number-of-learned-ge1*: ‹C' ∈# *dom-m baa* ⟹ ¬ *irred baa* C' ⟹ *Suc 0* ≤ *size* (*learned-clss-l baa*)›
  by (*auto simp*: *ran-m-def* *dest*!: *multi-member-split*)

lemma *mark-garbage-heur2-remove-and-add-cls-l*:
  ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ (*C*, C') ∈ *Id* ⟹
    mark-garbage-heur2 C S
      ≤ ⇓ (*twl-st-heur-restart-ana r*) (*remove-and-add-cls-wl* C' *T*)›
  unfolding *mark-garbage-heur2-def* *remove-and-add-cls-wl-def* *Let-def*
  apply (*cases S*; *cases T*)
  apply *refine-rcg*
  subgoal
    by (*auto simp*: *twl-st-heur-restart-def* *arena-lifting*
      valid-arena-extra-information-mark-to-delete'
      all-init-atms-fmdrop-add-mset-unit *learned-clss-l-l-fmdrop*
      learned-clss-l-l-fmdrop-irrelev *twl-st-heur-restart-ana-def* *ASSERT-refine-left*
      size-Diff-singleton *red-in-dom-number-of-learned-ge1* *intro*!: *ASSERT-leI*
    dest: *in-vdom-m-fmdropD*)
  subgoal
    by (*auto simp*: *twl-st-heur-restart-def* *arena-lifting*
      valid-arena-extra-information-mark-to-delete'
      all-init-atms-fmdrop-add-mset-unit *learned-clss-l-l-fmdrop*

learned-clss-l-l-fmdrop-irrelev twl-st-heur-restart-ana-def
size-Diff-singleton red-in-dom-number-of-learned-ge1
dest: *in-vdom-m-fmdropD*)
done

lemma *remove-one-annot-true-clause-one-imp-wl-pre-fst-le-uint32*:
assumes ‹(x, y) ∈ *nat-rel* $\times_f$ *twl-st-heur-restart-ana r*› **and**
‹*remove-one-annot-true-clause-one-imp-wl-pre* (*fst y*) (*snd y*)›
shows ‹*fst x* + *1* ≤ *Suc* (*uint32-max div 2*)›
proof −
have [*simp*]: ‹*fst y* = *fst x*›
using *assms* **by** (*cases x*, *cases y*) *auto*
have ‹*fst x* < *length* (*get-trail-wl* (*snd y*))›
using *assms* **apply** −
unfolding
remove-one-annot-true-clause-one-imp-wl-pre-def
remove-one-annot-true-clause-one-imp-pre-def
by *normalize-goal*+ *auto*
moreover have ‹(*get-trail-wl-heur* (*snd x*), *get-trail-wl* (*snd y*)) ∈ *trail-pol* (*all-init-atms-st* (*snd y*))›
using *assms*
by (*cases x*, *cases y*) (*simp add*: *twl-st-heur-restart-ana-def*
twl-st-heur-restart-def)
ultimately show ‹*?thesis*›
by (*auto simp add*: *trail-pol-alt-def*)
qed

lemma *remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D*:
‹(*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*,
uncurry remove-one-annot-true-clause-one-imp-wl) ∈
nat-rel $\times_f$ *twl-st-heur-restart-ana r* $\rightarrow_f$ ⟨*nat-rel* $\times_f$ *twl-st-heur-restart-ana r*⟩*nres-rel*›
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def*
remove-one-annot-true-clause-one-imp-wl-def case-prod-beta uncurry-def
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply (*refine-rcg get-literal-and-reason*[**where** *r*=*r*]
isasat-replace-annot-in-trail-replace-annot-in-trail-spec
[**where** *r*=*r*]
mark-garbage-heur2-remove-and-add-cls-l[**where** *r*=*r*])
subgoal by *auto*
subgoal unfolding *remove-one-annot-true-clause-one-imp-wl-pre-def*
by *auto*
subgoal
by (*rule remove-one-annot-true-clause-one-imp-wl-pre-fst-le-uint32*)
subgoal for *p pa*
by (*cases pa*)
(*auto simp*: *all-init-atms-def simp del*: *all-init-atms-def*[*symmetric*])
subgoal
by (*cases x*, *cases y*)
(*fastforce simp*: *twl-st-heur-restart-def*
trail-pol-alt-def)+
subgoal by *auto*
subgoal for *p pa*
by (*cases pa*; *cases p*; *cases x*; *cases y*)
(*auto simp*: *all-init-atms-def simp del*: *all-init-atms-def*[*symmetric*])

subgoal for *p pa S Sa*

unfolding *mark-garbage-pre-def*
arena-is-valid-clause-idx-def
prod.case
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac S*, *case-tac Sa*; *cases y*)
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
subgoal for *p pa S Sa*
unfolding *arena-is-valid-clause-vdom-def*
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac S*, *case-tac Sa*; *cases y*)
apply (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
done
subgoal
by *auto*
subgoal
by *auto*
subgoal
by (*cases x*, *cases y*) *fastforce*
done
done

definition *find-decomp-wl0* :: ‹*'v twl-st-wl* ⇒ *'v twl-st-wl* ⇒ *bool*› **where**
‹*find-decomp-wl0* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) (*M'*, *N'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*, *W'*).
($\exists$ *K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
count-decided M' = *0*) ∧
(*N'*, *D'*, *NE'*, *UE'*, *NS*, *US*, *Q'*, *W'*) = (*N*, *D*, *NE*, *UE*, *NS'*, *US'*, *Q*, *W*))›

definition *empty-Q-wl* :: ‹*'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*empty-Q-wl* = (λ(*M'*, *N*, *D*, *NE*, *UE*, *NS*, *US*, -, *W*). (*M'*, *N*, *D*, *NE*, *UE*, *NS*, {#}, {#}, *W*))›

definition *empty-US-wl* :: ‹*'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*empty-US-wl* = (λ(*M'*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*). (*M'*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *Q*, *W*))›

lemma *cdcl-twl-local-restart-wl-spec0-alt-def*:
‹*cdcl-twl-local-restart-wl-spec0* = (λ*S*. *do* {
ASSERT(*restart-abs-wl-pre2 S False*);
if count-decided (*get-trail-wl S*) > *0*
then do {
T ← *SPEC*(*find-decomp-wl0 S*);
RETURN (*empty-Q-wl T*)
} *else RETURN* (*empty-US-wl S*)})›
by (*intro ext*; *case-tac S*)
(*auto 5 3 simp*: *cdcl-twl-local-restart-wl-spec0-def*
RES-RETURN-RES2 image-iff RES-RETURN-RES empty-US-wl-def
find-decomp-wl0-def empty-Q-wl-def uncurry-def
intro!: *bind-cong*[*OF refl*]
dest: *get-all-ann-decomposition-exists-prepend*)

lemma *cdcl-twl-local-restart-wl-spec0*:
assumes *Sy*: ‹(*S*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*get-conflict-wl y* = *None*›

shows ‹*do* {
if count-decided-st-heur S > *0*
then do {
S ← *find-decomp-wl-st-int 0 S*;
empty-Q S
} *else RETURN S*
}
≤ ⇓ (*twl-st-heur-restart-ana r*) (*cdcl-twl-local-restart-wl-spec0 y*)›
proof −
define *upd* :: ‹- ⇒ - ⇒ *twl-st-wl-heur* ⇒ *twl-st-wl-heur*› **where**
‹*upd M′ vm* = (λ (-, *N*, *D*, *Q*, *W*, -, *clvls*, *cach*, *lbd*, *stats*).
(*M′*, *N*, *D*, *Q*, *W*, *vm*, *clvls*, *cach*, *lbd*, *stats*))›
for *M′* :: *trail-pol* **and** *vm*

have *find-decomp-wl-st-int-alt-def*:
‹*find-decomp-wl-st-int* = (λ*highest S*. *do*{
(*M′*, *vm*) ← *isa-find-decomp-wl-imp* (*get-trail-wl-heur S*) *highest* (*get-vmtf-heur S*);
RETURN (*upd M′ vm S*)
})›
unfolding *upd-def find-decomp-wl-st-int-def*
by (*auto intro*!: *ext*)

have [*refine0*]: ‹*do* {
(*M′*, *vm*) ←
isa-find-decomp-wl-imp (*get-trail-wl-heur S*) *0* (*get-vmtf-heur S*);
RETURN (*upd M′ vm S*)
} ≤ ⇓ {((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*,
slow-ema, *ccount*, *wasted*),
vdom, *avdom*, *lcount*, *opts*),
T).
((*M′*, *N′*, *D′*, *isa-length-trail M′*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*,
slow-ema, *restart-info-restart-done ccount*, *wasted*), *vdom*, *avdom*, *lcount*, *opts*),
(*empty-Q-wl T*)) ∈ *twl-st-heur-restart-ana r* ∧
isa-length-trail-pre M′} (*SPEC* (*find-decomp-wl0 y*))›
(**is** ‹- ≤ ⇓ *?A* -›)
if
‹*0* < *count-decided-st-heur S*› **and**
‹*0* < *count-decided* (*get-trail-wl y*)›
proof −
have *A*:
‹*?A* = {((*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*,
ccount, *wasted*),
vdom, *avdom*, *lcount*, *opts*),
T).
((*M′*, *N′*, *D′*, *length* (*get-trail-wl T*), *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*,
slow-ema, *restart-info-restart-done ccount*, *wasted*), *vdom*, *avdom*, *lcount*, *opts*),
(*empty-Q-wl T*)) ∈ *twl-st-heur-restart-ana r* ∧
isa-length-trail-pre M′}›
supply[[*goals-limit=1*]]
apply (*rule* ; *rule*)
subgoal for *x*
apply *clarify*
apply (*frule twl-st-heur-restart-isa-length-trail-get-trail-wl*)
by (*auto simp*: *trail-pol-def empty-Q-wl-def*
isa-length-trail-def
dest!: *ann-lits-split-reasons-map-lit-of*)

subgoal for *x*
apply *clarify*
apply (*frule twl-st-heur-restart-isa-length-trail-get-trail-wl*)
by (*auto simp*: *trail-pol-def empty-Q-wl-def*
isa-length-trail-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done

let *?A* = ⟨*all-init-atms-st y*⟩
have ⟨*get-vmtf-heur S* ∈ *isa-vmtf ?A* (*get-trail-wl y*)⟩**and**
n-d: ⟨*no-dup* (*get-trail-wl y*)⟩
using *Sy*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
then obtain *vm′* **where**
vm′: ⟨(*get-vmtf-heur S*, *vm′*) ∈ *Id* $\times_f$ *distinct-atoms-rel ?A*⟩ **and**
vm: ⟨*vm′* ∈ *vmtf* (*all-init-atms-st y*) (*get-trail-wl y*)⟩
unfolding *isa-vmtf-def*
by *force*

have *find-decomp-w-ns-pre*:
⟨*find-decomp-w-ns-pre* (*all-init-atms-st y*) ((*get-trail-wl y*, *0*), *vm′*)⟩
using *that assms vm′ vm* **unfolding** *find-decomp-w-ns-pre-def*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
dest: *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*)
have *1*: ⟨*isa-find-decomp-wl-imp* (*get-trail-wl-heur S*) *0* (*get-vmtf-heur S*) ≤
⇓ ({(*M*, *M′*). (*M*, *M′*) ∈ *trail-pol ?A* ∧ *count-decided M′* = *0*} $\times_f$ (*Id* $\times_f$ *distinct-atoms-rel ?A*))
(*find-decomp-w-ns ?A* (*get-trail-wl y*) *0 vm′*)⟩
apply (*rule order-trans*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*,
of ⟨*get-trail-wl y*⟩ *0 vm′* - - - *?A*])
subgoal using *that* **by** *auto*
subgoal
using *Sy vm′*
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
apply (*rule order-trans*, *rule ref-two-step′*)
apply (*rule find-decomp-wl-imp-find-decomp-wl′*[*THEN fref-to-Down-curry2*,
of ?A ⟨*get-trail-wl y*⟩ *0 vm′*])
subgoal by (*rule find-decomp-w-ns-pre*)
subgoal by *auto*
subgoal
using *n-d*
by (*fastforce simp*: *find-decomp-w-ns-def conc-fun-RES Image-iff*)
done
show *?thesis*
supply [[*goals-limit=1*]] **unfolding** *A*
apply (*rule bind-refine-res*[*OF - 1*[*unfolded find-decomp-w-ns-def conc-fun-RES*]])
apply (*case-tac y*, *cases S*)
apply *clarify*
apply (*rule RETURN-SPEC-refine*)
using *assms*
by (*auto simp*: *upd-def find-decomp-wl0-def*
intro!: *RETURN-SPEC-refine simp*: *twl-st-heur-restart-def out-learned-def*
empty-Q-wl-def twl-st-heur-restart-ana-def
intro: *isa-vmtfI isa-length-trail-pre dest*: *no-dup-appendD*)
qed

have *Sy'*: ‹(*S*, *empty-US-wl y*) ∈ *twl-st-heur-restart-ana r*›
using *Sy* **by** (*cases y*; *cases S*; *auto simp*: *twl-st-heur-restart-ana-def*
empty-US-wl-def twl-st-heur-restart-def)
show *?thesis*
unfolding *find-decomp-wl-st-int-alt-def*
cdcl-twl-local-restart-wl-spec0-alt-def
apply *refine-vcg*
subgoal
using *Sy* **by** (*auto simp*: *twl-st-heur-restart-count-decided-st-alt-def*)
subgoal
unfolding *empty-Q-def empty-Q-wl-def*
by *refine-vcg*
(*simp-all add*: *twl-st-heur-restart-isa-length-trail-get-trail-wl*)
subgoal
using *Sy'* **.**
done
qed

lemma *no-get-all-ann-decomposition-count-dec0*:
‹(∀ *M1*. (∀ *M2 K*. (*Decided K* # *M1*, *M2*) ∉ *set* (*get-all-ann-decomposition M*))) ⟷
count-decided M = *0*›
apply (*induction M rule*: *ann-lit-list-induct*)
subgoal by *auto*
subgoal for *L M*
by *auto*
subgoal for *L C M*
by (*cases* ‹*get-all-ann-decomposition M*›) *fastforce+*
done

lemma *get-pos-of-level-in-trail-decomp-iff*:
assumes ‹*no-dup M*›
shows ‹((∃ *M1 M2 K*.
(*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition M*) ∧
count-decided M1 = *0* ∧ *k* = *length M1*)) ⟷
k < *length M* ∧ *count-decided M* > *0* ∧ *is-decided* (*rev M* ! *k*) ∧ *get-level M* (*lit-of* (*rev M* ! *k*)) =
1›
(**is** ‹*?A* ⟷ *?B*›)
proof
assume *?A*
then obtain *K M1 M2* **where**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
[*simp*]: ‹*count-decided M1* = *0*› **and**
k-M1: ‹*length M1* = *k*›
by *auto*
then have ‹*k* < *length M*›
by *auto*
moreover have ‹*rev M* ! *k* = *Decided K*›
using *decomp*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *nth-append*
simp flip: *k-M1*)
moreover have ‹*get-level M* (*lit-of* (*rev M* ! *k*)) = *1*›
using *assms decomp*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *get-level-append-if nth-append*

simp flip: *k-M1*)
ultimately show *?B*
using *decomp* **by** *auto*
next
assume *?B*
define *K* **where** ‹*K* = *lit-of* (*rev M* ! *k*)›
obtain *M1 M2* **where**
M: ‹*M* = *M2* @ *Decided K* # *M1*› **and**
k-M1: ‹*length M1* = *k*›
apply (*subst* (*asm*) *append-take-drop-id*[*of* ‹*length M* − *Suc k*›, *symmetric*])
apply (*subst* (*asm*) *Cons-nth-drop-Suc*[*symmetric*])
unfolding *K-def*
subgoal using ‹*?B*› **by** *auto*
subgoal using ‹*?B*› *K-def* **by** (*cases* ‹*rev M* ! *k*›) (*auto simp*: *rev-nth*)
done
moreover have ‹*count-decided M1* = *0*›
using *assms* ‹*?B*› **unfolding** *M*
by (*auto simp*: *nth-append k-M1*)
ultimately show *?A*
using *get-all-ann-decomposition-ex*[*of K M1 M2*]
unfolding *M*
by *auto*
qed

lemma *remove-all-learned-subsumed-clauses-wl-id*:
‹(*x2a*, *x2*) ∈ *twl-st-heur-restart-ana r* ⟹
RETURN x2a
≤ ⇓ (*twl-st-heur-restart-ana r*)
(*remove-all-learned-subsumed-clauses-wl x2*)›
by (*cases x2a*; *cases x2*)
(*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
remove-all-learned-subsumed-clauses-wl-def)

lemma *remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D*:
‹(*remove-one-annot-true-clause-imp-wl-D-heur*, *remove-one-annot-true-clause-imp-wl*) ∈
twl-st-heur-restart-ana r $\rightarrow_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
unfolding *remove-one-annot-true-clause-imp-wl-def*
remove-one-annot-true-clause-imp-wl-D-heur-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
WHILEIT-refine[**where** *R* = ‹*nat-rel* $\times_r$ *twl-st-heur-restart-ana r*›]
remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D[*THEN*
fref-to-Down-curry])
subgoal by (*auto simp*: *trail-pol-alt-def isa-length-trail-pre-def*
twl-st-heur-restart-def twl-st-heur-restart-ana-def)
subgoal by (*auto simp*: *twl-st-heur-restart-isa-length-trail-get-trail-wl*
twl-st-heur-restart-count-decided-st-alt-def)
subgoal for *x y*
apply (*rule order-trans*)
apply (*rule get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*[*THEN fref-to-Down-curry*,
of ‹*get-trail-wl y*› *0* - - ‹*all-init-atms-st y*›])
subgoal by (*auto simp*: *get-pos-of-level-in-trail-pre-def*
twl-st-heur-restart-count-decided-st-alt-def)
subgoal by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
subgoal
apply (*subst get-pos-of-level-in-trail-decomp-iff*)

apply (*solves* ‹*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*›)
apply (*auto simp*: *get-pos-of-level-in-trail-def*
twl-st-heur-restart-count-decided-st-alt-def)
done
done
subgoal by *auto*
subgoal for $x\ y\ k\ k'\ T\ T'$
apply (*subst* (*asm*)(*12*) *surjective-pairing*)
apply (*subst* (*asm*)(*10*) *surjective-pairing*)
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-inv-def*
prod-rel-iff
apply (*subst* (*10*) *surjective-pairing*, *subst prod.case*)
by (*fastforce intro*: *twl-st-heur-restart-anaD simp*: *twl-st-heur-restart-anaD*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *remove-all-learned-subsumed-clauses-wl-id*)
done

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl2-D*:
‹(*mark-to-delete-clauses-wl-D-heur*, *mark-to-delete-clauses-wl2*) ∈
twl-st-heur-restart-ana r $\to_f$ ⟨*twl-st-heur-restart-ana r*⟩*nres-rel*›
proof –
have *mark-to-delete-clauses-wl2-D-alt-def*:
‹*mark-to-delete-clauses-wl2* = (λ*S0*. *do* {
ASSERT(*mark-to-delete-clauses-wl-pre S0*);
S ← *reorder-vdom-wl S0*;
xs ← *collect-valid-indices-wl S*;
l ← *SPEC*(λ-::*nat*. *True*);
(-, *S*, -) ← $WHILE_T{}^{mark\text{-}to\text{-}delete\text{-}clauses\text{-}wl2\text{-}inv\ S\ xs}$
(λ(*i*, *T*, *xs*). *i* < *length xs*)
(λ(*i*, *T*, *xs*). *do* {
b ← *RETURN* (*xs*!*i* ∈# *dom-m* (*get-clauses-wl T*));
if ¬*b then RETURN* (*i*, *T*, *delete-index-and-swap xs i*)
else do {
ASSERT(*0* < *length* (*get-clauses-wl T*∝(*xs*!*i*)));
ASSERT (*get-clauses-wl T* ∝ (*xs* ! *i*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st T*));
K ← *RETURN* (*get-clauses-wl T* ∝ (*xs* ! *i*) ! *0*);
b ← *RETURN* (); — propagation reason
can-del ← *SPEC*(λ*b*. *b* ⟶
(*Propagated* (*get-clauses-wl T*∝(*xs*!*i*)!*0*) (*xs*!*i*) ∉ *set* (*get-trail-wl T*)) ∧
¬*irred* (*get-clauses-wl T*) (*xs*!*i*) ∧ *length* (*get-clauses-wl T* ∝ (*xs*!*i*)) ≠ *2*);
ASSERT(*i* < *length xs*);
if can-del
then
RETURN (*i*, *mark-garbage-wl* (*xs*!*i*) *T*, *delete-index-and-swap xs i*)
else
RETURN (*i*+*1*, *T*, *xs*)
}
})
(*l*, *S*, *xs*);
remove-all-learned-subsumed-clauses-wl S
})›
unfolding *mark-to-delete-clauses-wl2-def reorder-vdom-wl-def bind-to-let-conv Let-def*
by (*force intro*!: *ext*)
have *mono*: ‹$g = g' \Longrightarrow$ *do* {*f*; *g*} = *do* {*f*; *g'*}›

$\langle(\bigwedge x.\ h\ x = h'\ x) \Longrightarrow do\ \{x \leftarrow f;\ h\ x\} = do\ \{x \leftarrow f;\ h'\ x\}\rangle$ **for** $f\ f'$ :: ⟨- *nres*⟩ **and** $g\ g'$ **and** $h\ h'$
by *auto*

have [*refine0*]: ⟨*RETURN* (*get-avdom x*) $\leq \Downarrow$ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-avdom x*} (*collect-valid-indices-wl y*)⟩
if
⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
⟨*mark-to-delete-clauses-wl-D-heur-pre x*⟩
for *x y*
proof −
show *?thesis* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed
have *init-rel*[*refine0*]: ⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r* ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
((*l*, *x*), *la*, *y*) ∈ *nat-rel* $\times_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *get-avdom S* = *get-avdom x*}⟩
for *x y l la*
by *auto*

define *reason-rel* **where**
⟨*reason-rel K x1a* ≡ {(*C*, - :: *unit*).
(*C* ≠ *None*) = (*Propagated K* (*the C*) ∈ *set* (*get-trail-wl x1a*)) ∧
(*C* = *None*) = (*Decided K* ∈ *set* (*get-trail-wl x1a*) ∨
K ∉ *lits-of-l* (*get-trail-wl x1a*)) ∧
(∀ *C1*. (*Propagated K C1* ∈ *set* (*get-trail-wl x1a*) ⟶ *C1* = *the C*))}⟩ **for** *K* :: ⟨*nat literal*⟩ **and** *x1a*
have *get-the-propagation-reason*:
⟨*get-the-propagation-reason-pol* (*get-trail-wl-heur x2b*) *L*
≤ *SPEC* (λ*D*. (*D*, ()) ∈ *reason-rel K x1a*)⟩
if
⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
⟨*mark-to-delete-clauses-wl-pre y*⟩ **and**
⟨*mark-to-delete-clauses-wl-D-heur-pre x*⟩ **and**
⟨(*S*, *Sa*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧
V = *y* ∧
(*mark-to-delete-clauses-wl-pre y* ⟶
mark-to-delete-clauses-wl-pre V) ∧
(*mark-to-delete-clauses-wl-D-heur-pre x* ⟶
mark-to-delete-clauses-wl-D-heur-pre U)}⟩ **and**
⟨(*ys*, *xs*) ∈ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-avdom S*}⟩ **and**
⟨(*l*, *la*) ∈ *nat-rel*⟩ **and**
⟨*la* ∈ {-. *True*}⟩ **and**
xa-x': ⟨(*xa*, *x'*)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩ **and**
⟨*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)⟩ **and**
⟨*case x' of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*⟩ **and**
⟨*x1b* < *length* (*get-avdom x2b*)⟩ **and**
⟨*access-vdom-at-pre x2b x1b*⟩ **and**
dom: ⟨(*b*, *ba*)
∈ {(*b*, *b'*).
(*b*, *b'*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}⟩
⟨¬ ¬ *b*⟩
⟨¬ ¬ *ba*⟩ **and**

⟨*0 < length (get-clauses-wl x1a ∝ (x2a ! x1))*⟩ **and**
⟨*access-lit-in-clauses-heur-pre ((x2b, get-avdom x2b ! x1b), 0)*⟩ **and**
st:
  ⟨*x2 = (x1a, x2a)*⟩
  ⟨*x' = (x1, x2)*⟩
  ⟨*xa = (x1b, x2b)*⟩ **and**
L: ⟨*get-clauses-wl x1a ∝ (x2a ! x1) ! 0 ∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a)*⟩ **and**
L': ⟨*(L, K)*
∈ {(L, L').
  (L, L') ∈ nat-lit-lit-rel ∧
  L' = get-clauses-wl x1a ∝ (x2a ! x1) ! 0}⟩
for *x y S Sa xs' xs l la xa x' x1 x2 x1a x2a x1' x2' x3 x1b ys x2b L K b ba*
proof −
  have *L*: ⟨*arena-lit (get-clauses-wl-heur x2b) (x2a ! x1) ∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a)*⟩
  using *L that* **by** (*auto simp*: *twl-st-heur-restart st arena-lifting dest*: *$\mathcal{L}_{all}$-init-all twl-st-heur-restart-anaD*)

  show *?thesis*
    apply (*rule order.trans*)
    apply (*rule get-the-propagation-reason-pol[THEN fref-to-Down-curry,*
      of ⟨*all-init-atms-st x1a*⟩ ⟨*get-trail-wl x1a*⟩
  ⟨*arena-lit (get-clauses-wl-heur x2b) (get-avdom x2b ! x1b + 0)*⟩*]*)
    subgoal
      using *xa-x' L L'* **by** (*auto simp add*: *twl-st-heur-restart-def st*)
    subgoal
      using *xa-x' L' dom* **by** (*auto simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def st arena-lifting*)
    using *that* **unfolding** *get-the-propagation-reason-def reason-rel-def* **apply** −
    by (*auto simp*: *twl-st-heur-restart lits-of-def get-the-propagation-reason-def*
      conc-fun-RES
      dest!: *twl-st-heur-restart-anaD dest*: *twl-st-heur-restart-same-annotD imageI[of - - lit-of]*)
qed
have ⟨*((M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur, vdom, avdom, lcount),*
    S')
    ∈ twl-st-heur-restart ⟹
  ((M', N', D', j, W', vm, clvls, cach, lbd, outl, stats, heur, vdom, avdom', lcount),
    S')
    ∈ twl-st-heur-restart⟩
  if ⟨*mset avdom' ⊆# mset avdom*⟩
  for *M' N' D' j W' vm clvls cach lbd outl stats fast-ema slow-ema*
    ccount vdom lcount S' avdom' avdom heur
  using *that* **unfolding** *twl-st-heur-restart-def*
  by *auto*
then have *mark-to-delete-clauses-wl-D-heur-pre-vdom'*:
  ⟨*mark-to-delete-clauses-wl-D-heur-pre (M', N', D', j, W', vm, clvls, cach, lbd, outl, stats,*
    heur, vdom, avdom', lcount) ⟹
   mark-to-delete-clauses-wl-D-heur-pre (M', N', D', j, W', vm, clvls, cach, lbd, outl, stats,
    heur, vdom, avdom, lcount)⟩
  if ⟨*mset avdom ⊆# mset avdom'*⟩
  for *M' N' D' j W' vm clvls cach lbd outl stats fast-ema slow-ema avdom avdom'*
    ccount vdom lcount heur
  using *that*
  unfolding *mark-to-delete-clauses-wl-D-heur-pre-def*
  by *metis*
have [*refine0*]:
  ⟨*sort-vdom-heur S ≤ ⇓ {(U, V). (U, V) ∈ twl-st-heur-restart-ana r ∧ V = T ∧*
    (mark-to-delete-clauses-wl-pre T ⟶ mark-to-delete-clauses-wl-pre V) ∧

(*mark-to-delete-clauses-wl-D-heur-pre S* ⟶ *mark-to-delete-clauses-wl-D-heur-pre U*)}
(*reorder-vdom-wl T*)›
if ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **for** *S T*
using *that* **unfolding** *reorder-vdom-wl-def sort-vdom-heur-def*
apply (*refine-rcg ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left*)
apply (*rule-tac N1* = ‹*get-clauses-wl T*› **and** *vdom1* = ‹(*get-vdom S*)› **in**
order-trans[*OF isa-remove-deleted-clauses-from-avdom-remove-deleted-clauses-from-avdom*,
unfolded Down-id-eq, *OF - - - remove-deleted-clauses-from-avdom*])
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of -* ‹*get-clauses-wl T*›] *dest*!: *set-mset-mono mset-subset-eqD*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def valid-arena-vdom-subset twl-st-heur-restart-def*
dest!: *size-mset-mono valid-arena-vdom-subset*)
subgoal
apply (*rewrite at* ‹- ≤ ¤› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of -* ‹*get-clauses-wl T*› ‹*get-vdom S*›])
by (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*
intro: *mark-to-delete-clauses-wl-D-heur-pre-vdom'*
dest: *mset-eq-setD*)
done
have *already-deleted*:
‹((*x1b*, *delete-index-vdom-heur x1b x2b*), *x1*, *x1a*,
delete-index-and-swap x2a x1)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧
V = *y* ∧
(*mark-to-delete-clauses-wl-pre y* ⟶
mark-to-delete-clauses-wl-pre V) ∧
(*mark-to-delete-clauses-wl-D-heur-pre x* ⟶
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**

⟨*la* ∈ {-. *True*}⟩ **and**
xx: ⟨(*xa*, *x'*)
∈ *nat-rel* $\times_f$ {(*Sa*, *T*, *xs*). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩ **and**
⟨*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)⟩ **and**
⟨*case x' of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*⟩ **and**
st:
⟨*x2* = (*x1a*, *x2a*)⟩
⟨*x'* = (*x1*, *x2*)⟩
⟨*xa* = (*x1b*, *x2b*)⟩ **and**
le: ⟨*x1b* < *length* (*get-avdom x2b*)⟩ **and**
⟨*access-vdom-at-pre x2b x1b*⟩ **and**
⟨(*b*, *ba*) ∈ {(*b*, *b'*). (*b*, *b'*) ∈ *bool-rel* ∧ *b* = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}⟩ **and**
⟨¬*ba*⟩
for *x y S xs l la xa x' xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa ba b*
proof −
show *?thesis*
using *xx le* **unfolding** *st*
by (*auto simp*: *twl-st-heur-restart-ana-def delete-index-vdom-heur-def*
twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
learned-clss-l-l-fmdrop size-remove1-mset-If
intro: *valid-arena-extra-information-mark-to-delete'*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
qed
have *get-learned-count-ge*: ⟨*Suc 0* ≤ *get-learned-count x2b*⟩
if
xy: ⟨(*x*, *y*) ∈ *twl-st-heur-restart-ana r*⟩ **and**
⟨(*xa*, *x'*)
∈ *nat-rel* $\times_f$
{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}⟩ **and**
⟨*x2* = (*x1a*, *x2a*)⟩ **and**
⟨*x'* = (*x1*, *x2*)⟩ **and**
⟨*xa* = (*x1b*, *x2b*)⟩ **and**
dom: ⟨(*b*, *ba*)
∈ {(*b*, *b'*).
(*b*, *b'*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}⟩
⟨¬ ¬ *b*⟩
⟨¬ ¬ *ba*⟩ **and**
⟨*MINIMUM-DELETION-LBD*
< *arena-lbd* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) ∧
arena-status (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) = *LEARNED* ∧
arena-length (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*) ≠ *2* ∧
¬ *marked-as-used* (*get-clauses-wl-heur x2b*) (*get-avdom x2b* ! *x1b*)⟩ **and**
⟨*can-del*⟩ **for** *x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1b x2b D can-del b ba*
proof −
have ⟨¬*irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)⟩ **and** ⟨(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*⟩
using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting*
dest!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena*)
then show *?thesis*
using *dom* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*)
qed
have *mop-clause-not-marked-to-delete-heur*:
⟨*mop-clause-not-marked-to-delete-heur x2b* (*get-avdom x2b* ! *x1b*)

≤ *SPEC*
(λ*c*. (*c*, *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))
∈ {(*b*, *b*′). (*b*,*b*′) ∈ *bool-rel* ∧ (*b* ⟷ *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))})›
if
‹(*xa*, *x*′)
∈ *nat-rel* $\times_f$
{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)› **and**
‹*case x*′ *of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl2-inv Sa xs x*′› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x*′ = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)›
for *x y S Sa uu xs l la xa x*′ *x1 x2 x1a x2a x1b x2b*
unfolding *mop-clause-not-marked-to-delete-heur-def*
apply *refine-vcg*
subgoal
using *that* **by** *blast*
subgoal
using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting dest*!: *twl-st-heur-restart-anaD*)
done

have *init*:
‹(*u*, *xs*) ∈ {(*xs*, *xs*′). *xs* = *xs*′ ∧ *xs* = *get-avdom S*} ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
(*S*, *Sa*) ∈ *twl-st-heur-restart-ana r* ⟹
((*l*, *S*), *la*, *Sa*, *xs*) ∈ *nat-rel* $\times_f$
{(*Sa*, (*T*, *xs*)). (*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}›
for *x y S Sa xs l la u*
by *auto*
have *mop-access-lit-in-clauses-heur*:
‹*mop-access-lit-in-clauses-heur x2b* (*get-avdom x2b* ! *x1b*) *0*
≤ *SPEC*
(λ*c*. (*c*, *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*)
∈ {(*L*, *L*′). (*L*, *L*′) ∈ *nat-lit-lit-rel* ∧ *L*′ = *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*})›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*mark-to-delete-clauses-wl-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-ana r* ∧
V = *y* ∧
(*mark-to-delete-clauses-wl-pre y* ⟶
mark-to-delete-clauses-wl-pre V) ∧
(*mark-to-delete-clauses-wl-D-heur-pre x* ⟶
mark-to-delete-clauses-wl-D-heur-pre U)}› **and**
‹(*uu*, *xs*) ∈ {(*xs*, *xs*′). *xs* = *xs*′ ∧ *xs* = *get-avdom S*}› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {*uu*. *True*}› **and**
‹*length* (*get-avdom S*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹(*xa*, *x*′)
∈ *nat-rel* $\times_f$

{(*Sa*, *T*, *xs*).
(*Sa*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *xs* = *get-avdom Sa*}› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-avdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl2-inv Sa xs x′*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*x1b* < *length* (*get-avdom x2b*)› **and**
‹*access-vdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-avdom x2b* ! *x1b*)› **and**
‹(*b*, *ba*)
∈ {(*b*, *b′*).
(*b*, *b′*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}› **and**
‹¬ ¬ *b*› **and**
‹¬ ¬ *ba*› **and**
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))› **and**
‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*
∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)› **and**
pre: ‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-avdom x2b* ! *x1b*), *0*)›
for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba*
unfolding *mop-access-lit-in-clauses-heur-def mop-arena-lit2-def*
apply *refine-vcg*
subgoal using *pre* **unfolding** *access-lit-in-clauses-heur-pre-def* **by** *simp*
subgoal using *that* **by** (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting*)
done

have *incr-restart-stat*: ‹*incr-restart-stat T*
≤ ⇓ (*twl-st-heur-restart-ana r*) (*remove-all-learned-subsumed-clauses-wl S*)›
if ‹(*T*, *S*) ∈ *twl-st-heur-restart-ana r*› **for** *S T i*
using *that*
by (*cases S*; *cases T*)
(*auto simp*: *conc-fun-RES incr-restart-stat-def*
twl-st-heur-restart-ana-def twl-st-heur-restart-def
remove-all-learned-subsumed-clauses-wl-def
RES-RETURN-RES)

have [*refine0*]: ‹*mark-clauses-as-unused-wl-D-heur i T* ≫= *incr-restart-stat*
≤ ⇓ (*twl-st-heur-restart-ana r*)
(*remove-all-learned-subsumed-clauses-wl S*)›
if ‹(*T*, *S*) ∈ *twl-st-heur-restart-ana r*› **for** *S T i*
apply (*cases S*)
apply (*rule bind-refine-res*[**where** *R* = *Id*, *simplified*])
defer
apply (*rule mark-clauses-as-unused-wl-D-heur*[*unfolded conc-fun-RES*, *OF that*, *of i*])
apply (*rule incr-restart-stat*[*THEN order-trans*, *of - S*])
by *auto*

show *?thesis*
supply *sort-vdom-heur-def*[*simp*] *twl-st-heur-restart-anaD*[*dest*] [[*goals-limit=1*]]
unfolding *mark-to-delete-clauses-wl-D-heur-alt-def mark-to-delete-clauses-wl2-D-alt-def*
access-lit-in-clauses-heur-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg sort-vdom-heur-reorder-vdom-wl*[*THEN fref-to-Down*])

subgoal
unfolding *mark-to-delete-clauses-wl-D-heur-pre-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y S T* **unfolding** *number-clss-to-keep-def* **by** (*cases S*) (*auto*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule init*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-vdom-at-pre-def*)
subgoal for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d*
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def*
prod.simps
by (*rule exI*[*of* - ‹*get-clauses-wl x2a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2d*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-get-avdom-nth-get-vdom*)
apply (*rule mop-clause-not-marked-to-delete-heur*; *assumption*)
subgoal for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b*
by (*auto simp*: *twl-st-heur-restart*)
subgoal
by (*rule already-deleted*)
subgoal for *x y - - - - - xs l la xa x′ x1 x2 x1a x2a*
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def
by (*rule bex-leI*[*of* - ‹*get-avdom x2a* ! *x1a*›], *simp*, *rule exI*[*of* - ‹*get-clauses-wl x1*›])
(*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal premises *p* **using** *p*(*7*−) **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule mop-access-lit-in-clauses-heur*; *assumption*)
apply (*rule get-the-propagation-reason*; *assumption*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*
twl-st-heur-restart-get-avdom-nth-get-vdom)
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *prod.simps*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart arena-dom-status-iff*
dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal
unfolding *marked-as-used-pre-def*
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart*)

subgoal
by (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting*)
subgoal by *fast*
subgoal for *x y S Sa - xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *mop-arena-length-st-def*
apply (*rule mop-arena-length*[*THEN fref-to-Down-curry*, *THEN order-trans*,
of ‹*get-clauses-wl x1a*› ‹*get-avdom x2b* ! *x1b*› - - ‹*set* (*get-vdom x2b*)›])
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-restart-valid-arena*)
subgoal
apply (*auto intro*!: *incr-wasted-st-twl-st ASSERT-leI*)
subgoal
unfolding *prod.simps mark-garbage-pre-def*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart dest*: *twl-st-heur-restart-valid-arena*)
subgoal
apply (*rule get-learned-count-ge*; *assumption?*; *fast?*)
apply *auto*
done
subgoal
by (*use arena-lifting*(*24*)[*of* ‹*get-clauses-wl-heur x2b*› - - ‹*get-avdom x2b* ! *x1*›] **in**
‹*auto intro*!: *incr-wasted-st mark-garbage-heur-wl-ana*
dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD*›)
done
done
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *dest*!: *set-mset-mono mset-subset-eqD*)
subgoal
by (*auto intro*!: *mark-unused-st-heur-ana*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset size-mset-mono*)
subgoal
by *auto*
done
qed

definition *iterate-over-VMTF* **where**
‹*iterate-over-VMTF* ≡ (λ*f* (*I* :: ′*a* ⇒ *bool*) (*ns* :: (*nat*, *nat*) *vmtf-node list*, *n*) *x*. *do* {
(-, *x*) ← $WHILE_T{}^{\lambda(n,\ x).\ I\ x}$
(λ(*n*, -). *n* ≠ *None*)
(λ(*n*, *x*). *do* {
ASSERT(*n* ≠ *None*);
let A = *the n*;
ASSERT(*A* < *length ns*);
ASSERT(*A* ≤ *uint32-max div 2*);
x ← *f A x*;
RETURN (*get-next* ((*ns* ! *A*)), *x*)
})

    (n, x);
   RETURN x
  })›

definition *iterate-over-$\mathcal{L}_{all}$* **where**
 ‹*iterate-over-$\mathcal{L}_{all}$* = (λ*f* $\mathcal{A}_0$ *I x*. *do* {
  $\mathcal{A}$ ← *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}_0$ ∧ *distinct-mset* $\mathcal{A}$);
  (-, *x*) ← *WHILE*$_T$$^{\lambda(-,\ x).\ I\ x}$
   (λ($\mathcal{B}$, -). $\mathcal{B}$ ≠ {#})
   (λ($\mathcal{B}$, *x*). *do* {
    ASSERT($\mathcal{B}$ ≠ {#});
    A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
    x ← *f A x*;
    RETURN (*remove1-mset A* $\mathcal{B}$, *x*)
   })
   ($\mathcal{A}$, *x*);
  RETURN x
 })›

lemma *iterate-over-VMTF-iterate-over-$\mathcal{L}_{all}$*:
 fixes *x* :: *'a*
 assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
  nempty: ‹$\mathcal{A}$ ≠ {#}› ‹*isasat-input-bounded* $\mathcal{A}$›
 shows ‹*iterate-over-VMTF f I* (*ns*, *Some fst-As*) *x* ≤ ⇓ *Id* (*iterate-over-$\mathcal{L}_{all}$ f* $\mathcal{A}$ *I x*)›
proof −
 obtain *xs' ys'* **where**
  vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
  ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
  ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
  vmtf-$\mathcal{L}$: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* ((*set xs'*, *set ys'*), *to-remove*)› **and**
  fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
  le: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*›
  using *vmtf* **unfolding** *vmtf-def*
  by *blast*
 define *zs* **where** ‹*zs* = *ys'* @ *xs'*›
 define *is-lasts* **where**
  ‹*is-lasts* $\mathcal{B}$ *n m* ⟷ *set-mset* $\mathcal{B}$ = *set* (*drop m zs*) ∧ *set-mset* $\mathcal{B}$ ⊆ *set-mset* $\mathcal{A}$ ∧
    distinct-mset $\mathcal{B}$ ∧
    card (*set-mset* $\mathcal{B}$) ≤ *length zs* ∧
    card (*set-mset* $\mathcal{B}$) + *m* = *length zs* ∧
    (*n* = *option-hd* (*drop m zs*)) ∧
    m ≤ *length zs*› **for** $\mathcal{B}$ **and** *n* :: ‹*nat option*› **and** *m*
 have *card-$\mathcal{A}$*: ‹*card* (*set-mset* $\mathcal{A}$) = *length zs*›
  ‹*set-mset* $\mathcal{A}$ = *set zs*› **and**
  nempty': ‹*zs* ≠ []› **and**
  dist-zs: ‹*distinct zs*›
  using *vmtf-$\mathcal{L}$ vmtf-ns-distinct*[*OF vmtf-ns*] *nempty*
  unfolding *vmtf-$\mathcal{L}_{all}$-def eq-commute*[*of* - ‹*atms-of* -›] *zs-def*
  by (*auto simp*: *atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ card-Un-disjoint distinct-card*)
 have *hd-zs-le*: ‹*hd zs* < *length ns*›
  using *vmtf-ns-le-length*[*OF vmtf-ns*, *of* ‹*hd zs*›] *nempty'*
  unfolding *zs-def*[*symmetric*]
  by *auto*
 have [*refine0*]: ‹
   (*the x1a*, *A*) ∈ *nat-rel* ⟹
   x = *x2b* ⟹

f (*the x1a*) $x2b \le \Downarrow$ *Id* (f A x)› **for** *x1a A x x2b*
by *auto*
define *iterate-over-VMTF2* **where**
‹*iterate-over-VMTF2* $\equiv$ (λf (I :: $'a \Rightarrow$ *bool*) (*vm* :: (*nat*, *nat*) *vmtf-node list*, n) x. *do* {
let - = *remdups-mset* $\mathcal{A}$;
(-, -, x) $\leftarrow$ $WHILE_T{}^{\lambda(n,m,\ x).\ I\ x}$
(λ(n, -, -). $n \neq$ *None*)
(λ(n, m, x). *do* {
ASSERT($n \neq$ *None*);
let A = *the* n;
ASSERT($A <$ *length ns*);
ASSERT($A \le$ *uint32-max div 2*);
$x \leftarrow f\ A\ x$;
RETURN (*get-next* ((*ns* ! A)), *Suc* m, x)
})
(n, *0*, x);
RETURN x
})›
have *iterate-over-VMTF2-alt-def*:
‹*iterate-over-VMTF2* $\equiv$ (λf (I :: $'a \Rightarrow$ *bool*) (*vm* :: (*nat*, *nat*) *vmtf-node list*, n) x. *do* {
(-, -, x) $\leftarrow$ $WHILE_T{}^{\lambda(n,m,\ x).\ I\ x}$
(λ(n, -, -). $n \neq$ *None*)
(λ(n, m, x). *do* {
ASSERT($n \neq$ *None*);
let A = *the* n;
ASSERT($A <$ *length ns*);
ASSERT($A \le$ *uint32-max div 2*);
$x \leftarrow f\ A\ x$;
RETURN (*get-next* ((*ns* ! A)), *Suc* m, x)
})
(n, *0*, x);
RETURN x
})›
unfolding *iterate-over-VMTF2-def* **by** *force*
have *nempty-iff*: ‹(*x1* $\neq$ *None*) = (*x1b* $\neq$ {#})›
if
‹(*remdups-mset* $\mathcal{A}$, $\mathcal{A}'$) $\in$ *Id*› **and**
H: ‹(x, x') $\in$ {((n, m, x), $\mathcal{A}'$, y). *is-lasts* $\mathcal{A}'$ n m $\wedge$ $x = y$}› **and**
‹*case* x *of* (n, m, *xa*) $\Rightarrow$ *I xa*› **and**
‹*case* x' *of* (*uu-*, x) $\Rightarrow$ *I* x› **and**
st[*simp*]:
‹*x2* = (*x1a*, *x2a*)›
‹x = (*x1*, *x2*)›
‹x' = (*x1b*, *xb*)›
for $\mathcal{A}'$ x x' *x1 x2 x1a x2a x1b xb*
proof
show ‹*x1b* $\neq$ {#}› **if** ‹*x1* $\neq$ *None*›
using *that H*
by (*auto simp*: *is-lasts-def*)
show ‹*x1* $\neq$ *None*› **if** ‹*x1b* $\neq$ {#}›
using *that H*
by (*auto simp*: *is-lasts-def*)
qed
have *IH*: ‹((*get-next* (*ns* ! *the x1a*), *Suc x1b*, *xa*), *remove1-mset A x1*, *xb*)
$\in$ {((n, m, x), $\mathcal{A}'$, y). *is-lasts* $\mathcal{A}'$ n m $\wedge$ $x = y$}›

if
‹(*remdups-mset* $\mathcal{A}$, $\mathcal{A}'$) ∈ *Id*› **and**
H: ‹(x, x') ∈ {((n, m, x), $\mathcal{A}'$, y). *is-lasts* $\mathcal{A}'$ n m ∧ x = y}› **and**
‹*case* x *of* (n, *uu-*, *uua-*) ⇒ n ≠ *None*› **and**
nempty: ‹*case* x' *of* ($\mathcal{B}$, *uu-*) ⇒ $\mathcal{B}$ ≠ {#}› **and**
‹*case* x *of* (n, m, *xa*) ⇒ *I xa*› **and**
‹*case* x' *of* (*uu-*, x) ⇒ *I x*› **and**
st:
‹x' = (*x1*, *x2*)›
‹*x2a* = (*x1b*, *x2b*)›
‹x = (*x1a*, *x2a*)›
‹(*xa*, *xb*) ∈ *Id*› **and**
‹*x1* ≠ {#}› **and**
‹*x1a* ≠ *None*› **and**
A: ‹(*the x1a*, *A*) ∈ *nat-rel*› **and**
‹*the x1a* < *length ns*›
for $\mathcal{A}'$ x x' *x1 x2 x1a x2a x1b x2b A xa xb*
proof −
have [*simp*]: ‹*distinct-mset x1*› ‹*x1b* < *length zs*›
using *H A nempty*
apply (*auto simp*: *st is-lasts-def simp flip*: *Cons-nth-drop-Suc*)
apply (*cases* ‹*x1b* = *length zs*›)
apply *auto*
done
then have [*simp*]: ‹*zs* ! *x1b* ∉ *set* (*drop* (*Suc x1b*) *zs*)›
by (*auto simp*: *in-set-drop-conv-nth nth-eq-iff-index-eq dist-zs*)
have [*simp*]: ‹*length zs* − *Suc x1b* + *x1b* = *length zs* ⟷ *False*›
using ‹*x1b* < *length zs*› **by** *presburger*
have ‹*vmtf-ns* (*take x1b zs* @ *zs* ! *x1b* # *drop* (*Suc x1b*) *zs*) *m ns*›
using *vmtf-ns*
by (*auto simp*: *Cons-nth-drop-Suc simp flip*: *zs-def*)
from *vmtf-ns-last-mid-get-next-option-hd*[*OF this*]
show *?thesis*
using *H A st*
by (*auto simp*: *st is-lasts-def dist-zs distinct-card distinct-mset-set-mset-remove1-mset simp flip*: *Cons-nth-drop-Suc*)
qed
have *WTF*[*simp*]: ‹*length zs* − *Suc 0* = *length zs* ⟷ *zs* = []›
by (*cases zs*) *auto*
have *zs2*: ‹*set* (xs' @ ys') = *set zs*›
by (*auto simp*: *zs-def*)
have *is-lasts-le*: ‹*is-lasts x1* (*Some A*) *x1b* ⟹ *A* < *length ns*› **for** *x2 xb x1b x1 A*
using *vmtf*-$\mathcal{L}$ *le nth-mem*[*of* ‹*x1b*› *zs*] **unfolding** *is-lasts-def prod.case vmtf*-$\mathcal{L}_{all}$-*def set-append*[*symmetric*]*zs-def*[*symmetric*] *zs2*
by (*auto simp*: *eq-commute*[*of* ‹*set zs*› ‹*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›] *hd-drop-conv-nth simp del*: *nth-mem*)
have *le-uint32-max*: ‹*the x1a* ≤ *uint32-max div 2*›
if
‹(*remdups-mset* $\mathcal{A}$, $\mathcal{A}'$) ∈ *Id*› **and**
‹(x, x') ∈ {((n, m, x), $\mathcal{A}'$, y). *is-lasts* $\mathcal{A}'$ n m ∧ x = y}› **and**
‹*case* x *of* (n, *uu-*, *uua-*) ⇒ n ≠ *None*› **and**
‹*case* x' *of* ($\mathcal{B}$, *uu-*) ⇒ $\mathcal{B}$ ≠ {#}› **and**
‹*case* x *of* (n, m, *xa*) ⇒ *I xa*› **and**
‹*case* x' *of* (*uu-*, x) ⇒ *I x*› **and**
‹x' = (*x1*, *x2*)› **and**
‹*x2a* = (*x1b*, *xb*)› **and**

⟨$x = (x1a, x2a)$⟩ **and**
⟨$x1 \neq \{\#\}$⟩ **and**
⟨$x1a \neq None$⟩ **and**
⟨$(the\ x1a,\ A) \in nat\text{-}rel$⟩ **and**
⟨$the\ x1a < length\ ns$⟩
for $\mathcal{A}'\ x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ xb\ A$
proof −
have ⟨$the\ x1a \in\#\ \mathcal{A}$⟩
using *that* **by** (*auto simp*: *is-lasts-def*)
then show *?thesis*
using *nempty* **by** (*auto dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
qed
have ⟨*iterate-over-VMTF2 f I* (*ns*, *Some fst-As*) $x \leq \Downarrow$ *Id* (*iterate-over-*$\mathcal{L}_{all}$ *f* $\mathcal{A}$ *I x*)⟩
unfolding *iterate-over-VMTF2-def iterate-over-*$\mathcal{L}_{all}$*-def prod.case*
apply (*refine-vcg WHILEIT-refine*[**where** R = ⟨$\{((n :: nat\ option,\ m::nat,\ x::'a),\ (\mathcal{A}' :: nat\ multiset,\ y)).$
$is\text{-}lasts\ \mathcal{A}'\ n\ m \wedge x = y\}$⟩])
subgoal by *simp*
subgoal by *simp*
subgoal
using *card-*$\mathcal{A}$ *fst-As nempty nempty′ hd-conv-nth*[*OF nempty′*] *hd-zs-le* **unfolding** *zs-def*[*symmetric*]
is-lasts-def
by (*simp-all add*: *eq-commute*[*of* ⟨*remdups-mset* -⟩])
subgoal by *auto*
subgoal for $\mathcal{A}'\ x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ xb$
by (*rule nempty-iff*)
subgoal by *auto*
subgoal for $\mathcal{A}'\ x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ xb$
by (*simp add*: *is-lasts-def in-set-dropI*)
subgoal for $\mathcal{A}'\ x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ xb$
by (*auto simp*: *is-lasts-le*)
subgoal by (*rule le-uint32-max*)
subgoal by *auto*
subgoal for $\mathcal{A}'\ x\ x'\ x1\ x2\ x1a\ x2a\ x1b\ x2b\ A\ xa\ xb$
by (*rule IH*)
subgoal by *auto*
done
moreover have ⟨*iterate-over-VMTF f I* (*ns*, *Some fst-As*) $x \leq \Downarrow$ *Id* (*iterate-over-VMTF2 f I* (*ns*, *Some fst-As*) *x*)⟩
unfolding *iterate-over-VMTF2-alt-def iterate-over-VMTF-def prod.case*
by (*refine-vcg WHILEIT-refine*[**where** R = ⟨$\{((n :: nat\ option,\ x::'a),\ (n' :: nat\ option,\ m'::nat,\ x'::'a)).$
$n = n' \wedge x = x'\}$⟩]) *auto*
ultimately show *?thesis*
by *simp*
qed

definition *arena-is-packed* :: ⟨*arena* ⇒ *nat clauses-l* ⇒ *bool*⟩ **where**
⟨*arena-is-packed arena N* ⟷ *length arena* = ($\sum C \in\#$ *dom-m N*. *length* ($N \propto C$) + *header-size* ($N \propto C$))⟩

lemma *arena-is-packed-empty*[*simp*]: ⟨*arena-is-packed* [] *fmempty*⟩
by (*auto simp*: *arena-is-packed-def*)

lemma *sum-mset-cong*:
⟨$(\bigwedge A.\ A \in\#\ M \Longrightarrow f\ A = g\ A) \Longrightarrow (\sum\ A \in\#\ M.\ f\ A) = (\sum\ A \in\#\ M.\ g\ A)$⟩
by (*induction M*) *auto*
lemma *arena-is-packed-append*:
assumes ⟨*arena-is-packed* (*arena*) *N*⟩ **and**
[*simp*]: ⟨*length C* = *length* (*fst C'*) + *header-size* (*fst C'*)⟩ **and**
[*simp*]: ⟨$a \notin\#$ *dom-m N*⟩
shows ⟨*arena-is-packed* (*arena* @ *C*) (*fmupd a C' N*)⟩
proof −
show *?thesis*
using *assms*(*1*) **by** (*auto simp*: *arena-is-packed-def*
intro!: *sum-mset-cong*)
qed

lemma *arena-is-packed-append-valid*:
assumes
in-dom: ⟨*fst C* $\in\#$ *dom-m x1a*⟩ **and**
valid0: ⟨*valid-arena x1c x1a vdom0*⟩ **and**
valid: ⟨*valid-arena x1d x2a* (*set x2d*)⟩ **and**
packed: ⟨*arena-is-packed x1d x2a*⟩ **and**
n: ⟨*n* = *header-size* (*x1a* $\propto$ (*fst C*))⟩
shows ⟨*arena-is-packed*
(*x1d* @
Misc.slice (*fst C* − *n*)
(*fst C* + *arena-length x1c* (*fst C*)) *x1c*)
(*fmupd* (*length x1d* + *n*) (*the* (*fmlookup x1a* (*fst C*))) *x2a*)⟩
proof −
have [*simp*]: ⟨*length x1d* + *n* $\notin\#$ *dom-m x2a*⟩
using *valid* **by** (*auto dest*: *arena-lifting*(*2*) *valid-arena-in-vdom-le-arena*
simp: *arena-is-valid-clause-vdom-def header-size-def*)
have [*simp*]: ⟨*arena-length x1c* (*fst C*) = *length* (*x1a* $\propto$ (*fst C*))⟩ ⟨*fst C* $\geq$ *n*⟩
⟨*fst C* − *n* < *length x1c*⟩ ⟨*fst C* < *length x1c*⟩
using *valid0 valid in-dom* **by** (*auto simp*: *arena-lifting n less-imp-diff-less*)
have [*simp*]: ⟨*length*
(*Misc.slice* (*fst C* − *n*)
(*fst C* + *length* (*x1a* $\propto$ (*fst C*))) *x1c*) =
length (*x1a* $\propto$ *fst C*) + *header-size* (*x1a* $\propto$ *fst C*)⟩
using *valid in-dom arena-lifting*(*10*)[*OF valid0*]
by (*fastforce simp*: *slice-len-min-If min-def arena-lifting*(*4*) *simp flip*: *n*)
show *?thesis*
by (*rule arena-is-packed-append*[*OF packed*]) *auto*
qed

definition *move-is-packed* :: ⟨*arena* ⇒ - ⇒ *arena* ⇒ - ⇒ *bool*⟩ **where**
⟨*move-is-packed* $arena_o$ N_o *arena N* ⟷
(($\sum$ *C*$\in\#$*dom-m* N_o. *length* ($N_o \propto C$) + *header-size* ($N_o \propto C$)) +
($\sum$ *C*$\in\#$*dom-m N*. *length* ($N \propto C$) + *header-size* ($N \propto C$)) $\leq$ *length* $arena_o$)⟩

definition *isasat-GC-clauses-prog-copy-wl-entry*
:: ⟨*arena* ⇒ (*nat watcher*) *list list* ⇒ *nat literal* ⇒
(*arena* × - × -) ⇒ (*arena* × (*arena* × - × -)) *nres*⟩
where
⟨*isasat-GC-clauses-prog-copy-wl-entry* = (λ*N0 W A* (*N'*, *vdm*, *avdm*). *do* {
ASSERT(*nat-of-lit A* < *length W*);
ASSERT(*length* (*W* ! *nat-of-lit A*) $\leq$ *length N0*);

let le = *length* (*W* ! *nat-of-lit A*);
(*i*, *N*, *N'*, *vdm*, *avdm*) ← $WHILE_T$
(λ(*i*, *N*, *N'*, *vdm*, *avdm*). *i* < *le*)
(λ(*i*, *N*, (*N'*, *vdm*, *avdm*)). *do* {
ASSERT(*i* < *length* (*W* ! *nat-of-lit A*));
let C = *fst* (*W* ! *nat-of-lit A* ! *i*);
ASSERT(*arena-is-valid-clause-vdom N C*);
let st = *arena-status N C*;
if st ≠ *DELETED then do* {
ASSERT(*arena-is-valid-clause-idx N C*);
ASSERT(*length N'* + (*if arena-length N C* > *4 then 5 else 4*) + *arena-length N C* ≤ *length N0*);
ASSERT(*length N* = *length N0*);
ASSERT(*length vdm* < *length N0*);
ASSERT(*length avdm* < *length N0*);
let D = *length N'* + (*if arena-length N C* > *4 then 5 else 4*);
N' ← *fm-mv-clause-to-new-arena C N N'*;
ASSERT(*mark-garbage-pre* (*N*, *C*));
RETURN (*i+1*, *extra-information-mark-to-delete N C*, *N'*, *vdm* @ [*D*],
(*if st* = *LEARNED then avdm* @ [*D*] *else avdm*))
} *else RETURN* (*i+1*, *N*, (*N'*, *vdm*, *avdm*))
}) (*0*, *N0*, (*N'*, *vdm*, *avdm*));
RETURN (*N*, (*N'*, *vdm*, *avdm*))
})›

definition *isasat-GC-entry* :: ‹-› **where**
‹*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena-old W'* = {((*arena*$_o$, (*arena*, *vdom*, *avdom*)), (N_o, *N*)). *valid-arena arena*$_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set vdom*) ∧ *vdom-m* $\mathcal{A}$ *W'* N_o ⊆ *vdom0* ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom* ∧
arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena*$_o$ = *length arena-old* ∧
move-is-packed arena$_o$ N_o *arena N*}›

definition *isasat-GC-refl* :: ‹-› **where**
‹*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena-old* = {((*arena*$_o$, (*arena*, *vdom*, *avdom*), *W*), (N_o, *N*, *W'*)). *valid-arena arena*$_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set vdom*) ∧
(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W'* N_o ⊆ *vdom0* ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom* ∧
arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena*$_o$ = *length arena-old* ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length arena*$_o$) ∧*move-is-packed arena*$_o$ N_o *arena N*}›

lemma *move-is-packed-empty*[*simp*]: ‹*valid-arena arena N vdom* ⟹ *move-is-packed arena N* [] *fmempty*›
by (*auto simp*: *move-is-packed-def valid-arena-ge-length-clauses*)

lemma *move-is-packed-append*:
assumes
dom: ‹*C* ∈# *dom-m x1a*› **and**
E: ‹*length E* = *length* (*x1a* ∝ *C*) + *header-size* (*x1a* ∝ *C*)› ‹(*fst E'*) = (*x1a* ∝ *C*)›
‹*n* = *header-size* (*x1a* ∝ *C*)› **and**
valid: ‹*valid-arena x1d x2a D'*› **and**
packed: ‹*move-is-packed x1c x1a x1d x2a*›
shows ‹*move-is-packed* (*extra-information-mark-to-delete x1c C*)
(*fmdrop C x1a*)
(*x1d* @ *E*)
(*fmupd* (*length x1d* + *n*) *E' x2a*)›
proof −
have [*simp*]: ‹(∑ *x*∈#*remove1-mset C*

    (*dom-m*
      x1a). *length*
        (*fst* (*the* (*if* $x = C$ *then None*
          else fmlookup x1a x))) +
        header-size
        (*fst* (*the* (*if* $x = C$ *then None*
          else fmlookup x1a x)))) =
  ($\sum x \in \#$*remove1-mset C*
    (*dom-m*
      x1a). *length*
        (*x1a* $\propto$ *x*) +
        header-size
        (*x1a* $\propto$ *x*))›
 by (*rule sum-mset-cong*)
 (*use distinct-mset-dom*[*of x1a*] **in** ‹*auto dest*!: *simp*: *distinct-mset-remove1-All*›)
have [*simp*]: ‹(*length x1d* + *header-size* (*x1a* $\propto$ *C*)) $\notin\#$ (*dom-m x2a*)›
 using *valid arena-lifting*(*2*) **by** *blast*
have [*simp*]: ‹($\sum x \in \#$(*dom-m x2a*). *length*
    (*fst* (*the* (*if length x1d* + *header-size* (*x1a* $\propto$ *C*) = *x*
      then Some E'
      else fmlookup x2a x))) +
    header-size
    (*fst* (*the* (*if length x1d* + *header-size* (*x1a* $\propto$ *C*) = *x*
      then Some E'
      else fmlookup x2a x)))) =
  ($\sum x \in \#$*dom-m x2a*. *length*
    (*x2a* $\propto$ *x*) +
    header-size
    (*x2a* $\propto$ *x*))›
 by (*rule sum-mset-cong*)
 (*use distinct-mset-dom*[*of x2a*] **in** ‹*auto dest*!: *simp*: *distinct-mset-remove1-All*›)
show *?thesis*
 using *packed dom E*
 by (*auto simp*: *move-is-packed-def split*: *if-splits dest*!: *multi-member-split*)
qed

definition *arena-header-size* :: ‹*arena* ⇒ *nat* ⇒ *nat*› **where**
‹*arena-header-size arena C* = (*if arena-length arena C* > *4 then 5 else 4*)›

lemma *valid-arena-header-size*:
 ‹*valid-arena arena N vdom* ⟹ *C* $\in\#$ *dom-m N* ⟹ *arena-header-size arena C* = *header-size* (*N* $\propto$ *C*)›

 by (*auto simp*: *arena-header-size-def header-size-def arena-lifting*)
lemma *isasat-GC-clauses-prog-copy-wl-entry*:
 assumes ‹*valid-arena arena N vdom0*› **and**
  ‹*valid-arena arena*′ *N*′ (*set vdom*)› **and**
  vdom: ‹*vdom-m* $\mathcal{A}$ *W N* ⊆ *vdom0*› **and**
  L: ‹*atm-of A* $\in\#$ $\mathcal{A}$› **and**
  L′*-L*: ‹(*A*′, *A*) ∈ *nat-lit-lit-rel*› **and**
  W: ‹(*W*′ , *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
  ‹*dom-m N*′ = *mset vdom*› ‹*distinct vdom*› **and**
  ‹*arena-is-packed arena*′ *N*′› **and**
  avdom: ‹*mset avdom* $\subseteq\#$ *mset vdom*› **and**
  r: ‹*length arena* = *r*› **and**
  le: ‹∀ *L* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W L*) ≤ *length arena*› **and**

packed: ‹*move-is-packed arena N arena' N'*›
shows ‹*isasat-GC-clauses-prog-copy-wl-entry arena W' A'* (*arena'*, *vdom*, *avdom*)
$\leq \Downarrow$ (*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena W*)
(*cdcl-GC-clauses-prog-copy-wl-entry N* (*W A*) *A N'*)›
(**is** ‹- $\leq \Downarrow$ (*?R*) -›)
proof −
have *A*: ‹*A'* = *A*› **and** *K*[*simp*]: ‹*W'* ! *nat-of-lit A* = *W A*›
using *L'-L L W* **apply** *auto*
by (*cases A*) (*auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *A-le*: ‹*nat-of-lit A* < *length W'*›
using *W L* **by** (*cases A*; *auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *length-slice*: ‹*C* ∈# *dom-m x1a* ⟹ *valid-arena x1c x1a vdom'* ⟹
length
(*Misc.slice* (*C* − *header-size* (*x1a* ∝ *C*))
(*C* + *arena-length x1c C*) *x1c*) =
arena-length x1c C + *header-size* (*x1a* ∝ *C*)› **for** *x1c x1a C vdom'*
using *arena-lifting*(*1*−*4*,*10*)[*of x1c x1a vdom' C*]
by (*auto simp*: *header-size-def slice-len-min-If min-def split*: *if-splits*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def cdcl-GC-clauses-prog-copy-wl-entry-def prod.case A*
arena-header-size-def[*symmetric*]
apply (*refine-vcg ASSERT-leI WHILET-refine*[**where** *R* = ‹*nat-rel* $\times_r$ *?R*›])
subgoal using *A-le* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal using *le L K* **by** (*cases A*) (*auto dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *assms* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal using *W L* **by** *auto*
subgoal by *auto*
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
using *vdom L*
unfolding *arena-is-valid-clause-vdom-def K isasat-GC-entry-def*
by (*cases A*)
(*force dest*!: *multi-member-split simp*: *vdom-m-def* $\mathcal{L}_{all}$*-add-mset*)+
subgoal
using *vdom L*
unfolding *arena-is-valid-clause-vdom-def K isasat-GC-entry-def*
by (*subst arena-dom-status-iff*)
(*cases A* ; *auto dest*!: *multi-member-split simp*: *arena-lifting arena-dom-status-iff*
vdom-m-def $\mathcal{L}_{all}$*-add-mset*; *fail*)+
subgoal
unfolding *arena-is-valid-clause-idx-def isasat-GC-entry-def*
by *auto*
subgoal unfolding *isasat-GC-entry-def move-is-packed-def arena-is-packed-def*
by (*auto simp*: *valid-arena-header-size arena-lifting dest*!: *multi-member-split*)
subgoal using *r* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal by (*auto dest*: *valid-arena-header-size simp*: *arena-lifting dest*!: *valid-arena-vdom-subset multi-member-split simp*: *arena-header-size-def isasat-GC-entry-def*
split: *if-splits*)
subgoal by (*auto simp*: *isasat-GC-entry-def dest*!: *size-mset-mono*)
subgoal
by (*force simp*: *isasat-GC-entry-def dest*: *arena-lifting*(*2*))
subgoal by (*auto simp*: *arena-header-size-def*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d D*
by (*rule order-trans*[*OF fm-mv-clause-to-new-arena*])
(*auto intro*: *valid-arena-extra-information-mark-to-delete'*
simp: *arena-lifting remove-1-mset-id-iff-notin*

```
        mark-garbage-pre-def isasat-GC-entry-def min-def
        valid-arena-header-size
    dest: in-vdom-m-fmdropD arena-lifting(2)
    intro!: arena-is-packed-append-valid subset-mset-trans-add-mset
     move-is-packed-append length-slice)
  subgoal
    by auto
  subgoal
    by auto
  done
 qed

definition isasat-GC-clauses-prog-single-wl
  :: ‹arena ⇒  (arena × - × -) ⇒ (nat watcher) list list ⇒ nat ⇒
       (arena × (arena × - × -) × (nat watcher) list list) nres›
where
‹isasat-GC-clauses-prog-single-wl = (λN0 N' WS A. do {
    let L = Pos A; ~~use phase saving instead~~
    ASSERT(nat-of-lit L < length WS);
    ASSERT(nat-of-lit (−L) < length WS);
    (N, (N', vdom, avdom)) ← isasat-GC-clauses-prog-copy-wl-entry N0 WS L N';
    let WS = WS[nat-of-lit L := []];
    ASSERT(length N = length N0);
    (N, N') ← isasat-GC-clauses-prog-copy-wl-entry N WS (−L) (N', vdom, avdom);
    let WS = WS[nat-of-lit (−L) := []];
    RETURN (N, N', WS)
  })›


lemma isasat-GC-clauses-prog-single-wl:
  assumes
    ‹(X, X') ∈ isasat-GC-refl 𝒜 vdom0 arena0› and
    X: ‹X = (arena, (arena', vdom, avdom), W)› ‹X' = (N, N', W')› and
    L: ‹A ∈# 𝒜› and
    st: ‹(A, A') ∈ Id› and st': ‹narena = (arena', vdom, avdom)› and
    ae: ‹length arena0 = length arena› and
    le-all: ‹∀ L ∈# ℒall 𝒜. length (W' L) ≤ length arena›
  shows ‹isasat-GC-clauses-prog-single-wl arena narena  W A
     ≤ ⇓ (isasat-GC-refl 𝒜 vdom0 arena0)
        (cdcl-GC-clauses-prog-single-wl N W' A' N')›
     (is ‹- ≤ ⇓ ?R -›)
proof −
  have H:
    ‹valid-arena arena N vdom0›
    ‹valid-arena arena' N' (set vdom)› and
    vdom: ‹vdom-m 𝒜 W' N ⊆ vdom0› and
    L: ‹A ∈# 𝒜› and
    eq: ‹A' = A› and
    WW': ‹(W, W') ∈ ⟨Id⟩map-fun-rel (D0 𝒜)› and
    vdom-dom: ‹dom-m N' = mset vdom› and
    dist: ‹distinct vdom› and
    packed: ‹arena-is-packed arena' N'› and
    avdom: ‹mset avdom ⊆# mset vdom› and
    packed2: ‹move-is-packed arena N arena' N'› and
    incl: ‹vdom-m 𝒜 W' N ⊆ vdom0›
    using assms X st by (auto simp: isasat-GC-refl-def)
```

have *vdom2*: ‹*vdom-m* $\mathcal{A}$ *W′ x1* ⊆ *vdom0* ⟹ *vdom-m* $\mathcal{A}$ (*W′*(*L* := [])) *x1* ⊆ *vdom0*› **for** *x1 L*
by (*force simp*: *vdom-m-def dest*!: *multi-member-split*)
have *vdom-m-upd*: ‹*x* ∈ *vdom-m* $\mathcal{A}$ (*W*(*Pos A* := [], *Neg A* := [])) *N* ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ *W N*› **for** *x W A N*
by (*auto simp*: *image-iff vdom-m-def dest*: *multi-member-split*)
have *vdom-m3*: ‹*x* ∈ *vdom-m* $\mathcal{A}$ *W a* ⟹ *dom-m a* ⊆# *dom-m b* ⟹ *dom-m b* ⊆# *dom-m c* ⟹*x* ∈ *vdom-m* $\mathcal{A}$ *W c*› **for** *a b c W x*
unfolding *vdom-m-def* **by** *auto*
have *W*: ‹(*W*[*2* ∗ *A* := [], *Suc* (*2* ∗ *A*) := []], *W′*(*Pos A* := [], *Neg A* := []))
∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **for** *A*
using *WW′* **unfolding** *map-fun-rel-def*
apply *clarify*
apply (*intro conjI*)
apply *auto*[]
apply (*drule multi-member-split*)
apply (*case-tac L*)
apply (*auto dest*!: *multi-member-split*)
done
have *le*: ‹*nat-of-lit* (*Pos A*) < *length W*› ‹*nat-of-lit* (*Neg A*) < *length W*›
using *WW′ L* **by** (*auto dest*!: *multi-member-split simp*: *map-fun-rel-def* $\mathcal{L}_{all}$-*add-mset*)
have [*refine0*]: ‹*RETURN* (*Pos A*) ≤ ⇓ *Id* (*RES* {*Pos A*, *Neg A*})› **by** *auto*
have *vdom-upD*:‹ *x* ∈ *vdom-m* $\mathcal{A}$ (*W′*(*Pos A* := [], *Neg A* := [])) *xd* ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ (λ*a*. *if a* = *Pos A then* [] *else W′ a*) *xd*›
for *W′ a A x xd*
by (*auto simp*: *vdom-m-def*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-single-wl-def*
cdcl-GC-clauses-prog-single-wl-def eq st′ isasat-GC-refl-def
apply (*refine-vcg*
isasat-GC-clauses-prog-copy-wl-entry[**where** *r*= ‹*length arena*› **and** $\mathcal{A}$ = $\mathcal{A}$])
subgoal using *le* **by** *auto*
subgoal using *le* **by** *auto*
apply (*rule H*(*1*); *fail*)
apply (*rule H*(*2*); *fail*)
subgoal using *incl* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW′* **by** *auto*
subgoal using *vdom-dom* **by** *blast*
subgoal using *dist* **by** *blast*
subgoal using *packed* **by** *blast*
subgoal using *avdom* **by** *blast*
subgoal by *blast*
subgoal using *le-all* **by** *auto*
subgoal using *packed2* **by** *auto*
subgoal using *ae* **by** (*auto simp*: *isasat-GC-entry-def*)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*rule vdom2*; *auto*)
supply *isasat-GC-entry-def*[*simp*]
subgoal using *WW′* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *L* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW′* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW′* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW′ le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)

subgoal using *WW′ le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW′ le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW′ le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *WW′ le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset*)
subgoal using *W ae le-all vdom* **by** (*auto simp*: *dest*!: *vdom-upD*)
done
qed

definition *isasat-GC-clauses-prog-wl2* **where**
‹*isasat-GC-clauses-prog-wl2* ≡ (λ(*ns* :: (*nat*, *nat*) *vmtf-node list*, *n*) *x0*. *do* {
  (-, *x*) ← $WHILE_T{}^{\lambda(n,\ x).\ length\ (fst\ x)\ =\ length\ (fst\ x0)}$
    (λ(*n*, -). *n* ≠ *None*)
    (λ(*n*, *x*). *do* {
      ASSERT(*n* ≠ *None*);
      let A = *the n*;
      ASSERT(*A* < *length ns*);
      ASSERT(*A* ≤ *uint32-max div 2*);
      x ← (λ($arena_o$, *arena*, *W*). *isasat-GC-clauses-prog-single-wl* $arena_o$ *arena W A*) *x*;
      RETURN (*get-next* ((*ns* ! *A*)), *x*)
    })
    (*n*, *x0*);
  RETURN x
})›

definition *cdcl-GC-clauses-prog-wl2* **where**
‹*cdcl-GC-clauses-prog-wl2* = (λ*N0* $\mathcal{A}$*0 WS*. *do* {
  $\mathcal{A}$ ← *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}$*0*);
  (-, (*N*, *N′*, *WS*)) ← $WHILE_T{}^{cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl\text{-}inv\ \mathcal{A}\ N0}$
    (λ($\mathcal{B}$, -). $\mathcal{B}$ ≠ {#})
    (λ($\mathcal{B}$, (*N*, *N′*, *WS*)). *do* {
      ASSERT($\mathcal{B}$ ≠ {#});
      A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
      (*N*, *N′*, *WS*) ← *cdcl-GC-clauses-prog-single-wl N WS A N′*;
      RETURN (*remove1-mset A* $\mathcal{B}$, (*N*, *N′*, *WS*))
    })
    ($\mathcal{A}$, (*N0*, *fmempty*, *WS*));
  RETURN (*N*, *N′*, *WS*)
})›

lemma *WHILEIT-refine-with-invariant-and-break*:
assumes *R0*: *I′ x′* ⟹ (*x*,*x′*)∈*R*
assumes *IREF*: ⋀*x x′*. ⟦ (*x*,*x′*)∈*R*; *I′ x′* ⟧ ⟹ *I x*
assumes *COND-REF*: ⋀*x x′*. ⟦ (*x*,*x′*)∈*R*; *I x*; *I′ x′* ⟧ ⟹ *b x* = *b′ x′*
assumes *STEP-REF*:
  ⋀*x x′*. ⟦ (*x*,*x′*)∈*R*; *b x*; *b′ x′*; *I x*; *I′ x′* ⟧ ⟹ *f x* ≤ ⇓*R* (*f′ x′*)
shows *WHILEIT I b f x* ≤⇓{(*x*, *x′*). (*x*, *x′*) ∈ *R* ∧ *I x* ∧ *I′ x′* ∧ ¬*b′ x′*} (*WHILEIT I′ b′ f′ x′*)
(**is** ‹- ≤ ⇓*?R′* -›)
  apply (*subst* (*2*) *WHILEIT-add-post-condition*)
  apply (*refine-vcg WHILEIT-refine-genR*[**where** *R′*=*R* **and** *R* = *?R′*])
  subgoal by (*auto intro*: *assms*)[]
  subgoal by (*auto intro*: *assms*)[]
  subgoal using *COND-REF* **by** (*auto*)
  subgoal by (*auto intro*: *assms*)[]
  subgoal by (*auto intro*: *assms*)[]

done

lemma *cdcl-GC-clauses-prog-wl-inv-cong-empty*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N* ({#}, *x*) $\Longrightarrow$ *cdcl-GC-clauses-prog-wl-inv* $\mathcal{B}$ *N* ({#}, *x*)›
by (*auto simp*: *cdcl-GC-clauses-prog-wl-inv-def*)

lemma *isasat-GC-clauses-prog-wl2*:
assumes ‹*valid-arena arena*$_o$ *N*$_o$ *vdom0*› **and**
‹*valid-arena arena N* (*set vdom*)› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W′ N*$_o$ ⊆ *vdom0*› **and**
vmtf: ‹((*ns*, *m*, *n*, *lst-As1*, *next-search1*), *to-remove1*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
nempty: ‹$\mathcal{A}$ ≠ {#}› **and**
W-W′: ‹(*W*, *W′*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and** *old*: ‹*old-arena* = []› **and**
le-all: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W′ L*) ≤ *length arena*$_o$›
shows
‹*isasat-GC-clauses-prog-wl2* (*ns*, *Some n*) (*arena*$_o$, (*old-arena*, [], []), *W*)
≤ ⇓ ({((*arena*$_o$′, (*arena*, *vdom*, *avdom*), *W*), (*N*$_o$′, *N*, *W′*)). *valid-arena arena*$_o$′ *N*$_o$′ *vdom0* ∧
valid-arena arena N (*set vdom*) ∧
(*W*, *W′*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W′ N*$_o$′ ⊆ *vdom0* ∧
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N*$_o$ ({#}, *N*$_o$′, *N*, *W′*) ∧ *dom-m N* = *mset vdom* ∧ *distinct vdom*
∧
arena-is-packed arena N ∧ *mset avdom* ⊆# *mset vdom* ∧ *length arena*$_o$′ = *length arena*$_o$})
(*cdcl-GC-clauses-prog-wl2 N*$_o$ $\mathcal{A}$ *W′*)›
proof −
define *f* **where**
‹*f A* ≡ (λ(*arena*$_o$, *arena*, *W*). *isasat-GC-clauses-prog-single-wl arena*$_o$ *arena W A*)› **for** *A* :: *nat*
let *?R* = ‹{((*𝒜′*, *arena*$_o$′, (*arena*, *vdom*), *W*), (*𝒜″*, *N*$_o$′, *N*, *W′*)). *𝒜′* = *𝒜″* ∧
((*arena*$_o$′, (*arena*, *vdom*), *W*), (*N*$_o$′, *N*, *W′*)) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena*$_o$ ∧
length arena$_o$′ = *length arena*$_o$}›
have *H*: ‹(*X*, *X′*) ∈ *?R* $\Longrightarrow$ *X* = (*x1*, *x2*) $\Longrightarrow$ *x2* = (*x3*, *x4*) $\Longrightarrow$ *x4* = (*x5*, *x6*) $\Longrightarrow$
X′ = (*x1′*, *x2′*) $\Longrightarrow$ *x2′* = (*x3′*, *x4′*) $\Longrightarrow$ *x4′* = (*x5′*, *x6′*) $\Longrightarrow$
((*x3*, (*fst x5*, *fst* (*snd x5*), *snd* (*snd x5*)), *x6*), (*x3′*, *x5′*, *x6′*)) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena*$_o$›
for *X X′ A B x1 x1′ x2 x2′ x3 x3′ x4 x4′ x5 x5′ x6 x6′ x0 x0′ x x′*
supply [[*show-types*]]
by *auto*
have *isasat-GC-clauses-prog-wl-alt-def*:
‹*isasat-GC-clauses-prog-wl2 n x0* = *iterate-over-VMTF f* (λ*x*. *length* (*fst x*) = *length* (*fst x0*)) *n x0*›
for *n x0*
unfolding *f-def isasat-GC-clauses-prog-wl2-def iterate-over-VMTF-def* **by** (*cases n*) (*auto intro*!:
ext)
show *?thesis*
unfolding *isasat-GC-clauses-prog-wl-alt-def prod.case f-def*[*symmetric*] *old*
apply (*rule order-trans*[*OF iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$[*OF vmtf nempty bounded*]])
unfolding *Down-id-eq iterate-over-*$\mathcal{L}_{all}$*-def cdcl-GC-clauses-prog-wl2-def f-def*
apply (*refine-vcg WHILEIT-refine-with-invariant-and-break*[**where** *R* = *?R*]
isasat-GC-clauses-prog-single-wl)
subgoal by *fast*
subgoal using *assms* **by** (*auto simp*: *valid-arena-empty isasat-GC-refl-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *assumption*; *fail*)
apply (*rule refl*)+

subgoal by (*auto simp add*: *cdcl-GC-clauses-prog-wl-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal using *le-all* **by** (*auto simp*: *isasat-GC-refl-def split*: *prod.splits*)
subgoal by (*auto simp*: *isasat-GC-refl-def*)
subgoal by (*auto simp*: *isasat-GC-refl-def*
dest: *cdcl-GC-clauses-prog-wl-inv-cong-empty*)
done
qed

lemma *cdcl-GC-clauses-prog-wl-alt-def*:
‹*cdcl-GC-clauses-prog-wl* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*). *do* {
ASSERT(*cdcl-GC-clauses-pre-wl* (*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*));
(*N*, *N'*, *WS*) ← *cdcl-GC-clauses-prog-wl2 N0* (*all-init-atms N0* (*NE*+*NS*)) *WS*;
RETURN (*M*, *N'*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*)
})›
proof −
have [*refine0*]: ‹(*x1c*, *x1*) ∈ *Id* ⟹ *RES* (*set-mset x1c*)
≤ ⇓ *Id* (*RES* (*set-mset x1*))› **for** *x1 x1c*
by *auto*
have [*refine0*]: ‹(*xa*, *x'*) ∈ *Id* ⟹
x2a = (*x1b*, *x2b*) ⟹
x2 = (*x1a*, *x2a*) ⟹
x' = (*x1*, *x2*) ⟹
x2d = (*x1e*, *x2e*) ⟹
x2c = (*x1d*, *x2d*) ⟹
xa = (*x1c*, *x2c*) ⟹
(*A*, *Aa*) ∈ *Id* ⟹
cdcl-GC-clauses-prog-single-wl x1d x2e A x1e
≤ ⇓ *Id*
(*cdcl-GC-clauses-prog-single-wl x1a x2b Aa x1b*)›
for *𝒜 x xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e A aaa Aa*
by *auto*
show *?thesis*
unfolding *cdcl-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl2-def*
while.imonad3

apply (*intro ext*)
apply (*clarsimp simp add*: *while.imonad3*)
apply (*subst order-class.eq-iff*[*of* ‹(- :: - *nres*)›])
apply (*intro conjI*)
subgoal
by (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*]) (*refine-rcg WHILEIT-refine*[**where** *R* = *Id*], *auto*)
subgoal
by (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*]) (*refine-rcg WHILEIT-refine*[**where** *R* = *Id*], *auto*)
done
qed

definition *isasat-GC-clauses-prog-wl* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
‹*isasat-GC-clauses-prog-wl* = (λ(*M'*, *N'*, *D'*, *j*, *W'*, ((*ns*, *st*, *fst-As*, *lst-As*, *nxt*), *to-remove*), *clvls*, *cach*, *lbd*, *outl*, *stats*,
heur, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
ASSERT(*old-arena* = []);
(*N*, (*N'*, *vdom*, *avdom*), *WS*) ← *isasat-GC-clauses-prog-wl2* (*ns*, *Some fst-As*) (*N'*, (*old-arena*, *take 0 vdom*, *take 0 avdom*), *W'*);

RETURN (*M′*, *N′*, *D′*, *j*, *WS*, ((*ns*, *st*, *fst-As*, *lst-As*, *nxt*), *to-remove*), *clvls*, *cach*, *lbd*, *outl*, *incr-GC stats*, *set-zero-wasted heur*,
vdom, *avdom*, *lcount*, *opts*, *take 0 N*)
})›

lemma *length-watched-le″*:
assumes
xb-x′a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-restart*› **and**
prop-inv: ‹*correct-watching″ x1*›
shows ‹∀ *x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
proof
fix *x2*
assume *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
have ‹*correct-watching″ x1*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*all-init-atms correct-watching″.simps*
simp flip: *all-init-lits-def all-init-lits-alt-def*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x′a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$-*atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
using *x2 xb-x′a* **unfolding** *all-init-atms-def all-init-lits-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x′a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)

have ‹*vdom-m* (*all-init-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def all-atms-def*[*symmetric*])
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur-restart-def simp flip*: *all-init-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*4*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*)›

by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

lemma *isasat-GC-clauses-prog-wl*:
‹(*isasat-GC-clauses-prog-wl*, *cdcl-GC-clauses-prog-wl*) ∈
twl-st-heur-restart $\rightarrow_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}⟩*nres-rel*›
(**is** ‹- ∈ *?T* $\rightarrow_f$ -›)
proof−
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹
valid-arena x1g x1a (*set x1z*)›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹
isasat-input-bounded (*all-init-atms x1a* (*x1c* + *NS*))›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹
vdom-m (*all-init-atms x1a* (*x1c*+*NS*)) *x2e x1a* ⊆ *set x1z*›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*, *x1r*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹
all-init-atms x1a (*x1c*+*NS*) ≠ {#}›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹

((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *set* (*fst x2o*)) ∈ *vmtf* (*all-init-atms x1a* (*x1c*+*NS*)) *x1*⟩
⟨⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US*.
((*x1f*, *x1g*, *x1h*, *x1i*, *x1j*, ((*x1m*, *x1n*, *x1o*, *x1p*, *x2n*), *x2o*), *x1q*,
x1s, *x1t*, *x1w*, *x1x*, *x1y*, *x1z*, *x1aa*, *x1ab*, *x2ab*),
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹ (*x1j*, *x2e*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms x1a* (*x1c*+*NS*)))⟩
unfolding *twl-st-heur-restart-def isa-vmtf-def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
by *auto*
have *H*: ⟨*vdom-m* (*all-init-atms x1a x1c*) *x2ad x1ad* ⊆ *set x2af*⟩
if
empty: ⟨∀ *A*∈#*all-init-atms x1a x1c*. *x2ad* (*Pos A*) = [] ∧ *x2ad* (*Neg A*) = []⟩ **and**
rem: ⟨*GC-remap*** (*x1a*, *Map.empty*, *fmempty*) (*fmempty*, *m*, *x1ad*)⟩ **and**
⟨*dom-m x1ad* = *mset x2af*⟩
for *m* :: ⟨*nat* ⇒ *nat option*⟩ **and** *y* :: ⟨*nat literal multiset*⟩ **and** *x* :: ⟨*nat*⟩ **and**
x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab x1ac x1ad x2ad x1ae x1ag x2af x2ag
proof −
have ⟨*xa* ∈# $\mathcal{L}_{all}$ (*all-init-atms x1a x1c*) ⟹ *x2ad xa* = []⟩ **for** *xa*
using *empty* **by** (*cases xa*) (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show *?thesis*
using ⟨*dom-m x1ad* = *mset x2af*⟩
by (*auto simp*: *vdom-m-def*)
qed
have *H'*: ⟨*mset x2ag* ⊆# *mset x1ah* ⟹ *x* ∈ *set x2ag* ⟹ *x* ∈ *set x1ah*⟩ **for** *x2ag x1ah x*
by (*auto dest*: *mset-eq-setD*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl-alt-def take-0*
apply (*intro frefI nres-relI*)
apply (*refine-vcg isasat-GC-clauses-prog-wl2*[**where** $\mathcal{A}$ = ⟨*all-init-atms* - -⟩]; *remove-dummy-vars*)
subgoal
by (*clarsimp simp add*: *twl-st-heur-restart-def*
cdcl-GC-clauses-prog-wl-inv-def H H'
rtranclp-GC-remap-all-init-atms
rtranclp-GC-remap-learned-clss-l)
subgoal
unfolding *cdcl-GC-clauses-pre-wl-def*
by (*drule length-watched-le''*)
(*clarsimp-all simp add*: *twl-st-heur-restart-def*
cdcl-GC-clauses-prog-wl-inv-def H H'
rtranclp-GC-remap-all-init-atms
rtranclp-GC-remap-learned-clss-l)
subgoal
by (*clarsimp simp add*: *twl-st-heur-restart-def*
cdcl-GC-clauses-prog-wl-inv-def H H'
rtranclp-GC-remap-all-init-atms
rtranclp-GC-remap-learned-clss-l)
done
qed

definition *cdcl-remap-st* :: ⟨*'v twl-st-wl* ⇒ *'v twl-st-wl nres*⟩ **where**
⟨*cdcl-remap-st* = (λ(*M*, *N0*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*, *WS'*).
(*M'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*) = (*M*, *D*, *NE*, *UE*, *NS*, *US*, *Q*) ∧

($\exists$ *m. GC-remap*** (*N0*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *N'*)) $\wedge$
0 $\notin\#$ *dom-m N'*))›

definition *rewatch-spec* :: ‹*nat twl-st-wl* $\Rightarrow$ *nat twl-st-wl nres*› **where**
‹*rewatch-spec* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*, *WS'*).
(*M'*, *N'*, *D'*, *NE'*, *UE'*, *NS'*, *US'*, *Q'*) = (*M*, *N*, *D*, *NE*, *UE*, *NS*, {#}, *Q*) $\wedge$
correct-watching' (*M*, *N'*, *D*, *NE*, *UE*, *NS'*, *US*, *Q'*, *WS'*) $\wedge$
literals-are-$\mathcal{L}_{in}$*'* (*M*, *N'*, *D*, *NE*, *UE*, *NS'*, *US*, *Q'*, *WS'*)))›

lemma *blits-in-*$\mathcal{L}_{in}$*'-restart-wl-spec0'*:
‹*literals-are-*$\mathcal{L}_{in}$*'* (*a*, *aq*, *ab*, *ac*, *ad*, *ae*, *af*, *Q*, *b*) $\Longrightarrow$
literals-are-$\mathcal{L}_{in}$*'* (*a*, *aq*, *ab*, *ac*, *ad*, *ae*, *af*, {#}, *b*)›
by (*auto simp*: *literals-are-*$\mathcal{L}_{in}$*'-empty blits-in-*$\mathcal{L}_{in}$*'-restart-wl-spec0*)

lemma *cdcl-GC-clauses-wl-D-alt-def*:
‹*cdcl-GC-clauses-wl* = (λ*S*. *do* {
ASSERT(*cdcl-GC-clauses-pre-wl S*);
let b = *True*;
if b then do {
S $\leftarrow$ *cdcl-remap-st S*;
S $\leftarrow$ *rewatch-spec S*;
RETURN S
}
else remove-all-learned-subsumed-clauses-wl S})›
supply [[*goals-limit=1*]]
unfolding *cdcl-GC-clauses-wl-def*
by (*fastforce intro*!: *ext simp*: *RES-RES-RETURN-RES2 cdcl-remap-st-def*
RES-RES9-RETURN-RES uncurry-def image-iff cdcl-remap-st-def
RES-RETURN-RES-RES2 RES-RETURN-RES RES-RES2-RETURN-RES rewatch-spec-def
rewatch-spec-def remove-all-learned-subsumed-clauses-wl-def
literals-are-$\mathcal{L}_{in}$*'-empty blits-in-*$\mathcal{L}_{in}$*'-restart-wl-spec0'*
intro!: *bind-cong-nres intro*: *literals-are-*$\mathcal{L}_{in}$*'-empty*(*4*))

definition *isasat-GC-clauses-pre-wl-D* :: ‹*twl-st-wl-heur* $\Rightarrow$ *bool*› **where**
‹*isasat-GC-clauses-pre-wl-D S* $\longleftrightarrow$ (
$\exists$ *T*. (*S*, *T*) $\in$ *twl-st-heur-restart* $\wedge$ *cdcl-GC-clauses-pre-wl T*
)›

definition *isasat-GC-clauses-wl-D* :: ‹*twl-st-wl-heur* $\Rightarrow$ *twl-st-wl-heur nres*› **where**
‹*isasat-GC-clauses-wl-D* = (λ*S*. *do* {
ASSERT(*isasat-GC-clauses-pre-wl-D S*);
let b = *True*;
if b then do {
T $\leftarrow$ *isasat-GC-clauses-prog-wl S*;
ASSERT(*length* (*get-clauses-wl-heur T*) $\leq$ *length* (*get-clauses-wl-heur S*));
ASSERT($\forall$ *i* $\in$ *set* (*get-vdom T*). *i* < *length* (*get-clauses-wl-heur S*));
U $\leftarrow$ *rewatch-heur-st T*;
RETURN U
}
else RETURN S})›

lemma *cdcl-GC-clauses-prog-wl2-st*:
assumes ‹(*T*, *S*) $\in$ *state-wl-l None*›

‹correct-watching'' T ∧ cdcl-GC-clauses-pre S ∧
set-mset (dom-m (get-clauses-wl T)) ⊆ clauses-pointed-to
(Neg ‘ set-mset (all-init-atms-st T) ∪
Pos ‘ set-mset (all-init-atms-st T))
(get-watched-wl T) ∧
literals-are-$\mathcal{L}_{in}$' T› **and**
‹get-clauses-wl T = N0'›
shows
‹cdcl-GC-clauses-prog-wl T ≤
⇓ {((M', N'', D', NE', UE', NS', US', Q', WS'), (N, N')).
(M', D', NE', UE', NS', US', Q') = (get-trail-wl T, get-conflict-wl T, get-unit-init-clss-wl T,
get-unit-learned-clss-wl T, get-subsumed-init-clauses-wl T, get-subsumed-learned-clauses-wl T,
literals-to-update-wl T) ∧ N'' = N ∧
(∀ L∈#all-init-lits-st T. WS' L = []) ∧
all-init-lits-st T = all-init-lits N (NE'+NS') ∧
(∃ m. GC-remap** (get-clauses-wl T, Map.empty, fmempty)
(fmempty, m, N))}
(SPEC(λ(N'::(nat, 'a literal list × bool) fmap, m).
GC-remap** (N0', (λ-. None), fmempty) (fmempty, m, N') ∧
0 ∉# dom-m N'))›
using cdcl-GC-clauses-prog-wl2[of ‹get-trail-wl T› ‹get-clauses-wl T› ‹get-conflict-wl T›
‹get-unit-init-clss-wl T› ‹get-unit-learned-clss-wl T› ‹get-subsumed-init-clauses-wl T›
‹get-subsumed-learned-clauses-wl T› ‹literals-to-update-wl T›
‹get-watched-wl T› S N0'] assms
by (cases T) auto

lemma correct-watching''-clauses-pointed-to:
assumes
xa-xb: ‹(xa, xb) ∈ state-wl-l None› **and**
corr: ‹correct-watching'' xa› **and**
pre: ‹cdcl-GC-clauses-pre xb› **and**
L: ‹literals-are-$\mathcal{L}_{in}$' xa›
shows ‹set-mset (dom-m (get-clauses-wl xa))
⊆ clauses-pointed-to
(Neg ‘
set-mset
(all-init-atms-st xa) ∪
Pos ‘
set-mset
(all-init-atms-st xa))
(get-watched-wl xa)›
(**is** ‹- ⊆ ?A›)
proof
let ?$\mathcal{A}$ = ‹all-init-atms (get-clauses-wl xa) (get-unit-init-clss-wl xa)›
fix C
assume C: ‹C ∈# dom-m (get-clauses-wl xa)›
obtain M N D NE UE NS US Q W **where**
xa: ‹xa = (M, N, D, NE, UE, NS, US, Q, W)›
by (cases xa)
obtain x **where**
xb-x: ‹(xb, x) ∈ twl-st-l None› **and**
‹twl-list-invs xb› **and**
struct-invs: ‹twl-struct-invs x› **and**
‹get-conflict-l xb = None› **and**
‹clauses-to-update-l xb = {#}› **and**
‹count-decided (get-trail-l xb) = 0› **and**

⟨∀ L∈set (get-trail-l xb). mark-of L = 0⟩
using *pre* **unfolding** *cdcl-GC-clauses-pre-def* **by** *fast*
have ⟨*twl-st-inv x*⟩
using *xb-x C struct-invs*
by (*auto simp*: *twl-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*)
then have *le0*: ⟨*get-clauses-wl xa* ∝ *C* ≠ []⟩
using *xb-x C xa-xb*
by (*cases x*; *cases* ⟨*irred N C*⟩)
(*auto simp*: *twl-struct-invs-def twl-st-inv.simps*
twl-st-l-def state-wl-l-def xa ran-m-def conj-disj-distribR
Collect-disj-eq Collect-conv-if
dest!: *multi-member-split*)
then have *le*: ⟨*N* ∝ *C* ! *0* ∈ *set* (*watched-l* (*N* ∝ *C*))⟩
by (*cases* ⟨*N* ∝ *C*⟩) (*auto simp*: *xa*)
have *eq*: ⟨*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N NE*)) =
set-mset (*all-lits-of-mm* (*mset* ‘# *init-clss-lf N* + *NE*))⟩
by (*auto simp del*: *all-init-atms-def*[*symmetric*]
simp: *all-init-atms-def xa* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*[*symmetric*]
all-init-lits-def)

have *H*: ⟨*get-clauses-wl xa* ∝ *C* ! *0* ∈# *all-init-lits-st xa*⟩
using *L C le0* **apply** −
unfolding *all-init-atms-def*[*symmetric*] *all-init-lits-def*[*symmetric*]
apply (*subst literals-are-*$\mathcal{L}_{in}$′*-literals-are-*$\mathcal{L}_{in}$*-iff*(*4*)[*OF xa-xb xb-x struct-invs*])
apply (*cases* ⟨*N* ∝ *C*⟩; *auto simp*: *literals-are-*$\mathcal{L}_{in}$*-def all-lits-def ran-m-def eq*
all-lits-of-mm-add-mset is-$\mathcal{L}_{all}$*-def xa all-lits-of-m-add-mset*
$\mathcal{L}_{all}$*-all-atms-all-lits*
dest!: *multi-member-split*)
done

moreover {
have ⟨{#*i* ∈# *fst* ‘# *mset* (*W* (*N* ∝ *C* ! *0*)). *i* ∈# *dom-m N*#} =
add-mset C {#*Ca* ∈# *remove1-mset C* (*dom-m N*). *N* ∝ *C* ! *0* ∈ *set* (*watched-l* (*N* ∝ *Ca*))#}⟩
using *corr H C le* **unfolding** *xa*
by (*auto simp*: *clauses-pointed-to-def correct-watching*′′*.simps xa*
simp flip: *all-init-atms-def all-init-lits-def all-init-atms-alt-def*
all-init-lits-alt-def
simp: *clause-to-update-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*)
from *arg-cong*[*OF this*, *of set-mset*] **have** ⟨*C* ∈ *fst* ‘ *set* (*W* (*N* ∝ *C* ! *0*))⟩
using *corr H C le* **unfolding** *xa*
by (*auto simp*: *clauses-pointed-to-def correct-watching*′′*.simps xa*
simp: *all-init-atms-def all-init-lits-def clause-to-update-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*) **}**
ultimately show ⟨*C* ∈ *?A*⟩
by (*cases* ⟨*N* ∝ *C* ! *0*⟩)
(*auto simp*: *clauses-pointed-to-def correct-watching*′′*.simps xa*
simp flip: *all-init-lits-def all-init-atms-alt-def*
all-init-lits-alt-def
simp: *clause-to-update-def all-init-atms-def*
simp del: *all-init-atms-def*[*symmetric*]
dest!: *multi-member-split*)
qed

```
abbreviation isasat-GC-clauses-rel where
  ‹isasat-GC-clauses-rel y ≡ {(S, T). (S, T) ∈ twl-st-heur-restart ∧
          (∀L∈#all-init-lits-st y. get-watched-wl T L = [])∧
          get-trail-wl T = get-trail-wl y ∧
          get-conflict-wl T = get-conflict-wl y ∧
          get-unit-init-clss-wl T = get-unit-init-clss-wl y ∧
          get-unit-learned-clss-wl T = get-unit-learned-clss-wl y ∧
          get-subsumed-init-clauses-wl T = get-subsumed-init-clauses-wl y ∧
          get-subsumed-learned-clauses-wl T = get-subsumed-learned-clauses-wl y ∧
          (∃ m. GC-remap** (get-clauses-wl y, (λ-. None), fmempty) (fmempty, m, get-clauses-wl T)) ∧
          arena-is-packed (get-clauses-wl-heur S) (get-clauses-wl T)}›

lemma ref-two-step'': ‹R ⊆ R' ⟹ A ≤ B ⟹ ⇓ R A ≤ ⇓ R' B›
  by (simp add: weaken-⇓ ref-two-step')

lemma isasat-GC-clauses-prog-wl-cdcl-remap-st:
  assumes
    ‹(x, y) ∈ twl-st-heur-restart''' r› and
    ‹cdcl-GC-clauses-pre-wl y›
  shows ‹isasat-GC-clauses-prog-wl x ≤ ⇓ (isasat-GC-clauses-rel y) (cdcl-remap-st y)›
proof -
  have xy: ‹(x, y) ∈ twl-st-heur-restart›
    using assms(1) by fast
  have H: ‹isasat-GC-clauses-rel y =
    {(S, T). (S, T) ∈ twl-st-heur-restart ∧ arena-is-packed (get-clauses-wl-heur S) (get-clauses-wl T)}
O
    {(S, T). S = T ∧ (∀L∈#all-init-lits-st y. get-watched-wl T L = [])∧
          get-trail-wl T = get-trail-wl y ∧
          get-conflict-wl T = get-conflict-wl y ∧
          get-unit-init-clss-wl T = get-unit-init-clss-wl y ∧
          get-unit-learned-clss-wl T = get-unit-learned-clss-wl y ∧
          get-subsumed-init-clauses-wl T = get-subsumed-init-clauses-wl y ∧
          get-subsumed-learned-clauses-wl T = get-subsumed-learned-clauses-wl y ∧
          (∃ m. GC-remap** (get-clauses-wl y, (λ-. None), fmempty) (fmempty, m, get-clauses-wl T))}›
    by blast
  show ?thesis
    using assms apply -
    apply (rule order-trans[OF isasat-GC-clauses-prog-wl[THEN fref-to-Down]])
    subgoal by fast
    apply (rule xy)
    unfolding conc-fun-chain[symmetric] H
    apply (rule ref-two-step')
    unfolding cdcl-GC-clauses-pre-wl-D-def cdcl-GC-clauses-pre-wl-def
    apply normalize-goal+
    apply (rule order-trans[OF cdcl-GC-clauses-prog-wl2-st])
    apply assumption
    subgoal for xa
      using assms(2) by (simp add: correct-watching''-clauses-pointed-to
        cdcl-GC-clauses-pre-wl-def)
    apply (rule refl)
    subgoal by (auto simp: cdcl-remap-st-def conc-fun-RES split: prod.splits)
    done
qed

fun correct-watching''' :: ‹- ⇒ 'v twl-st-wl ⇒ bool› where
```

‹*correct-watching‴* $\mathcal{A}$ (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*) ⟷
(∀ *L* ∈# *all-lits-of-mm* $\mathcal{A}$.
distinct-watched (*W L*) ∧
(∀ (*i*, *K*, *b*)∈#*mset* (*W L*).
i ∈# *dom-m N* ∧ *K* ∈ *set* (*N* ∝ *i*) ∧ *K* ≠ *L* ∧
correctly-marked-as-binary N (*i*, *K*, *b*)) ∧
fst '# *mset* (*W L*) = *clause-to-update L* (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, {#}, {#})))›

declare *correct-watching‴.simps*[*simp del*]

lemma *correct-watching‴-add-clause*:
assumes
corr: ‹*correct-watching‴* $\mathcal{A}$ ((*a*, *aa*, *CD*, *ac*, *ad*, *NS*, *US*, *Q*, *b*))› **and**
leC: ‹*2* ≤ *length C*› **and**
i-notin[*simp*]: ‹*i* ∉# *dom-m aa*› **and**
dist[*iff*]: ‹*C* ! *0* ≠ *C* ! *Suc 0*›
shows ‹*correct-watching‴* $\mathcal{A}$
((*a*, *fmupd i* (*C*, *red*) *aa*, *CD*, *ac*, *ad*, *NS*, *US*, *Q*, *b*
(*C* ! *0* := *b* (*C* ! *0*) @ [(*i*, *C* ! *Suc 0*, *length C* = *2*)],
C ! *Suc 0* := *b* (*C* ! *Suc 0*) @ [(*i*, *C* ! *0*, *length C* = *2*)]))))›
proof −
have [*iff*]: ‹*C* ! *Suc 0* ≠ *C* ! *0*›
using ‹*C* ! *0* ≠ *C* ! *Suc 0*› **by** *argo*
have [*iff*]: ‹*C* ! *Suc 0* ∈# *all-lits-of-m* (*mset C*)› ‹*C* ! *0* ∈# *all-lits-of-m* (*mset C*)›
‹*C* ! *Suc 0* ∈ *set C*› ‹ *C* ! *0* ∈ *set C*› ‹*C* ! *0* ∈ *set* (*watched-l C*)› ‹*C* ! *Suc 0* ∈ *set* (*watched-l C*)›
using *leC* **by** (*force intro*!: *in-clause-in-all-lits-of-m nth-mem simp*: *in-set-conv-iff*
intro: *exI*[*of - 0*] *exI*[*of -* ‹*Suc 0*›])+
have [*dest*!]: ‹⋀*L*. *L* ≠ *C* ! *0* ⟹ *L* ≠ *C* ! *Suc 0* ⟹ *L* ∈ *set* (*watched-l C*) ⟹ *False*›
by (*cases C*; *cases* ‹*tl C*›; *auto*)+
have *i*: ‹*i* ∉ *fst* ' *set* (*b L*)› **if** ‹*L*∈#*all-lits-of-mm* $\mathcal{A}$›**for** *L*
using *corr i-notin that* **unfolding** *correct-watching‴.simps*
by *force*
have [*iff*]: ‹(*i*,*c*, *d*) ∉ *set* (*b L*)› **if** ‹*L*∈#*all-lits-of-mm* $\mathcal{A}$› **for** *L c d*
using *i*[*of L*, *OF that*] **by** (*auto simp*: *image-iff*)
then show *?thesis*
using *corr*
by (*force simp*: *correct-watching‴.simps ran-m-mapsto-upd-notin*
all-lits-of-mm-add-mset all-lits-of-mm-union clause-to-update-mapsto-upd-notin correctly-marked-as-binary.simps
split: *if-splits*)
qed

lemma *rewatch-correctness*:
assumes *empty*: ‹⋀*L*. *L* ∈# *all-lits-of-mm* $\mathcal{A}$ ⟹ *W L* = []› **and**
H[*dest*]: ‹⋀*x*. *x* ∈# *dom-m N* ⟹ *distinct* (*N* ∝ *x*) ∧ *length* (*N* ∝ *x*) ≥ *2*› **and**
incl: ‹*set-mset* (*all-lits-of-mm* (*mset* '# *ran-mf N*)) ⊆ *set-mset* (*all-lits-of-mm* $\mathcal{A}$)›
shows
‹*rewatch N W* ≤ *SPEC*(λ*W*. *correct-watching‴* $\mathcal{A}$ (*M*, *N*, *C*, *NE*, *UE*, *NS*, *US*, *Q*, *W*))›
proof −
define *I* **where**
‹*I* ≡ λ(*a* :: *nat list*) (*b* :: *nat list*) *W*.
correct-watching‴ $\mathcal{A}$ ((*M*, *fmrestrict-set* (*set a*) *N*, *C*, *NE*, *UE*, *NS*, *US*, *Q*, *W*))›
have *I0*: ‹*set-mset* (*dom-m N*) ⊆ *set x* ∧ *distinct x* ⟹ *I* [] *x W*› **for** *x*
using *empty* **unfolding** *I-def* **by** (*auto simp*: *correct-watching‴.simps*
all-blits-are-in-problem-init.simps clause-to-update-def
all-lits-of-mm-union)

have *le*: ‹*length* (*σ L*) < *size* (*dom-m N*)›
if ‹*correct-watching'''* $\mathcal{A}$ (*M*, *fmrestrict-set* (*set l1*) *N*, *C*, *NE*, *UE*, *NS*, *US*, *Q*, *σ*)› **and**
‹*set-mset* (*dom-m N*) ⊆ *set x* ∧ *distinct x*› **and**
‹*x* = *l1* @ *xa* # *l2*› ‹*xa* ∈# *dom-m N*› ‹*L* ∈ *set* (*N* ∝ *xa*)›
for *L l1 σ xa l2 x*
proof −
have *1*: ‹*card* (*set l1*) ≤ *length l1*›
by (*auto simp*: *card-length*)
have ‹*L* ∈# *all-lits-of-mm* $\mathcal{A}$›
using *that incl in-clause-in-all-lits-of-m*[*of L* ‹*mset* (*N* ∝ *xa*)›]
by (*auto simp*: *correct-watching'''.simps dom-m-fmrestrict-set' ran-m-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset atm-of-all-lits-of-m
in-all-lits-of-mm-ain-atms-of-iff
dest!: *multi-member-split*)
then have ‹*distinct-watched* (*σ L*)› **and** ‹*fst* ‘ *set* (*σ L*) ⊆ *set l1* ∩ *set-mset* (*dom-m N*)›
using *that incl*
by (*auto simp*: *correct-watching'''.simps dom-m-fmrestrict-set' dest*!: *multi-member-split*)
then have ‹*length* (*map fst* (*σ L*)) ≤ *card* (*set l1* ∩ *set-mset* (*dom-m N*))›
using *1* **by** (*subst distinct-card*[*symmetric*])
(*auto simp*: *distinct-watched-alt-def intro*!: *card-mono intro*: *order-trans*)
also have ‹... < *card* (*set-mset* (*dom-m N*))›
using *that* **by** (*auto intro*!: *psubset-card-mono*)
also have ‹... = *size* (*dom-m N*)›
by (*simp add*: *distinct-mset-dom distinct-mset-size-eq-card*)
finally show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *rewatch-def*
apply (*refine-vcg*
nfoldli-rule[**where** *I* = ‹*I*›])
subgoal by (*rule I0*)
subgoal using *assms* **unfolding** *I-def* **by** *auto*
subgoal for *x xa l1 l2 σ* **using** *H*[*of xa*] **unfolding** *I-def* **apply** −
by (*rule*, *subst* (*asm*)*nth-eq-iff-index-eq*)
linarith+
subgoal for *x xa l1 l2 σ* **unfolding** *I-def* **by** (*rule le*) (*auto intro*!: *nth-mem*)
subgoal for *x xa l1 l2 σ* **unfolding** *I-def* **by** (*drule le*[**where** *L* = ‹*N* ∝ *xa* ! *1*›]) (*auto simp*: *I-def*
dest!: *le*)
subgoal for *x xa l1 l2 σ*
unfolding *I-def*
by (*cases* ‹*the* (*fmlookup N xa*)›)
(*auto intro*!: *correct-watching'''-add-clause simp*: *dom-m-fmrestrict-set'*)
subgoal
unfolding *I-def*
by *auto*
subgoal by *auto*
subgoal unfolding *I-def*
by (*auto simp*: *fmlookup-restrict-set-id'*)
done
qed

inductive-cases *GC-remapE*: ‹*GC-remap* (*a*, *aa*, *b*) (*ab*, *ac*, *ba*)›
lemma *rtranclp-GC-remap-ran-m-remap*:
‹$GC\text{-}remap^{**}$ (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈# *dom-m old* ⟹ *C* ∉# *dom-m old'* ⟹
m' C ≠ *None* ∧
fmlookup new' (*the* (*m' C*)) = *fmlookup old C*›

apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *a aa b ab ac ba*
apply (*cases* ‹$C \notin\#$ *dom-m a*›)
apply (*auto dest*: *GC-remap-ran-m-remap GC-remap-ran-m-no-rewrite-map GC-remap-ran-m-no-rewrite*)
apply (*metis GC-remap-ran-m-no-rewrite-fmap GC-remap-ran-m-no-rewrite-map in-dom-m-lookup-iff option.sel*)
using *GC-remap-ran-m-remap rtranclp-GC-remap-ran-m-no-rewrite* **by** *fastforce*
done

lemma *GC-remap-ran-m-exists-earlier*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ $C \in\#$ *dom-m new'* $\Longrightarrow$ $C \notin\#$ *dom-m new* $\Longrightarrow$
$\exists D$. *m' D* = *Some C* $\wedge$ $D \in\#$ *dom-m old* $\wedge$
fmlookup new' C = *fmlookup old D*›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) *auto*

lemma *rtranclp-GC-remap-ran-m-exists-earlier*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ $C \in\#$ *dom-m new'* $\Longrightarrow$ $C \notin\#$ *dom-m new* $\Longrightarrow$
$\exists D$. *m' D* = *Some C* $\wedge$ $D \in\#$ *dom-m old* $\wedge$
fmlookup new' C = *fmlookup old D*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
apply (*auto dest*: *GC-remap-ran-m-exists-earlier*)
apply (*case-tac* ‹$C \in\#$ *dom-m b*›)
apply (*auto elim*!: *GC-remapE split*: *if-splits*)
apply *blast*
using *rtranclp-GC-remap-ran-m-no-new-map rtranclp-GC-remap-ran-m-no-rewrite*
by (*metis fst-conv*)

lemma $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*:
‹*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N NE*)) = *set-mset* (*all-init-lits N NE*)›
unfolding $\mathcal{L}_{all}$*-all-init-atms* **..**

lemma *rewatch-heur-st-correct-watching*:
assumes
pre: ‹*cdcl-GC-clauses-pre-wl y*› **and**
S-T: ‹(*S*, *T*) $\in$ *isasat-GC-clauses-rel y*›
shows ‹*rewatch-heur-st S* $\leq \Downarrow$ (*twl-st-heur-restart'''* (*length* (*get-clauses-wl-heur S*)))
(*rewatch-spec T*)›
proof −
obtain *M N D NE UE NS US Q W* **where**
T: ‹*T* = (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*)›
by (*cases T*) *auto*

obtain *M' N' D' j W' vm clvls cach lbd outl stats fast-ema slow-ema ccount vdom avdom lcount opts* **where**
S: ‹*S* = (*M'*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*, *slow-ema*, *ccount*),
vdom, *avdom*, *lcount*, *opts*)›
by (*cases S*) *auto*

have
valid: ‹*valid-arena N' N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
dom-m-vdom: ‹*set-mset* (*dom-m N*) $\subseteq$ *set vdom*› **and**

W: ‹(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms-st T*))› **and**
empty: ‹⋀*L*. *L* ∈# *all-init-lits-st y* ⟹ *W L* = []› **and**
NUE:‹*get-unit-init-clss-wl y* = *NE* ›
‹*get-unit-learned-clss-wl y* = *UE*›
‹*get-trail-wl y* = *M*›
‹*get-subsumed-init-clauses-wl y* = *NS*›
‹*get-subsumed-learned-clauses-wl y* = *US*›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def S T*)
obtain *m* **where**
m: ‹*GC-remap*** (*get-clauses-wl y*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *N*)›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def S T*)
obtain *x xa xb* **where**
y-x: ‹(*y*, *x*) ∈ *Id*› ‹*x* = *y*› **and**
lits-y: ‹*literals-are-*$\mathcal{L}_{in}$*'* *y*› **and**
x-xa: ‹(*x*, *xa*) ∈ *state-wl-l None*› **and**
‹*correct-watching'' x*› **and**
xa-xb: ‹(*xa*, *xb*) ∈ *twl-st-l None*› **and**
‹*twl-list-invs xa*› **and**
struct-invs: ‹*twl-struct-invs xb*› **and**
‹*get-conflict-l xa* = *None*› **and**
‹*clauses-to-update-l xa* = {#}› **and**
‹*count-decided* (*get-trail-l xa*) = *0*› **and**
‹∀ *L*∈*set* (*get-trail-l xa*). *mark-of L* = *0*›
using *pre*
unfolding *cdcl-GC-clauses-pre-wl-def*
cdcl-GC-clauses-pre-def
by *blast*
have [*iff*]:
‹*distinct-mset* (*mset* (*watched-l C*) + *mset* (*unwatched-l C*)) ⟷ *distinct C*› **for** *C*
unfolding *mset-append*[*symmetric*]
by *auto*

have ‹*twl-st-inv xb*›
using *xa-xb struct-invs*
by (*auto simp*: *twl-struct-invs-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*)
then have *A*:
‹⋀*C*. *C* ∈# *dom-m* (*get-clauses-wl x*) ⟹ *distinct* (*get-clauses-wl x* ∝ *C*) ∧ *2* ≤ *length* (*get-clauses-wl x* ∝ *C*)›
using *xa-xb x-xa*
by (*cases x*; *cases* ‹*irred* (*get-clauses-wl x*) *C*›)
(*auto simp*: *twl-struct-invs-def twl-st-inv.simps*
twl-st-l-def state-wl-l-def ran-m-def conj-disj-distribR
Collect-disj-eq Collect-conv-if
dest!: *multi-member-split*
split: *if-splits*)
have *struct-wf*:
‹*C* ∈# *dom-m N* ⟹ *distinct* (*N* ∝ *C*) ∧ *2* ≤ *length* (*N* ∝ *C*)› **for** *C*
using *rtranclp-GC-remap-ran-m-exists-earlier*[*OF m*, *of* ‹*C*›] *A y-x*
by (*auto simp*: *T dest*:)

have *eq-UnD*: ‹*A* = *A'* ∪ *A''* ⟹ *A'* ⊆ *A*› **for** *A A' A''*
by *blast*

have *eq3*: ‹*all-init-lits* (*get-clauses-wl y*) (*NE*+*NS*) = *all-init-lits N* (*NE*+*NS*)›

using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *all-init-lits-def*)
moreover have ‹*all-lits-st y* = *all-lits-st T*›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*] *rtranclp-GC-remap-learned-clss-l-old-new*[*OF m*]
apply (*auto simp*: *all-init-lits-def T NUE all-lits-def*)
by (*metis NUE*(*1*) *NUE*(*2*) *all-clss-l-ran-m all-lits-def get-unit-clauses-wl-alt-def*)
ultimately have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-init-atms N* (*NE*+*NS*)) (*mset* ‘# *ran-mf N*)›
using *literals-are-*$\mathcal{L}_{in}$′*-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*OF x-xa xa-xb struct-invs*] *lits-y*
rtranclp-GC-remap-init-clss-l-old-new[*OF m*]
rtranclp-GC-remap-learned-clss-l-old-new[*OF m*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def* $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*
y-x literals-are-$\mathcal{L}_{in}$′*-def literals-are-*$\mathcal{L}_{in}$*-def all-lits-def*[*of N*] *T*
get-unit-clauses-wl-alt-def all-lits-def atm-of-eq-atm-of
is-$\mathcal{L}_{all}$*-def NUE all-init-atms-def all-init-lits-def all-atms-def conj-disj-distribR*
in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def atm-of-all-lits-of-mm
ex-disj-distrib Collect-disj-eq atms-of-def $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
dest!: *multi-member-split*[*of* - ‹*ran-m* -›]
split: *if-splits*
simp del: *all-init-atms-def*[*symmetric*] *all-atms-def*[*symmetric*])

have *eq*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N* (*NE*+*NS*))) = *set-mset* (*all-init-lits-st y*)›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *T all-init-lits-def NUE*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
then have *vd*: ‹*vdom-m* (*all-init-atms N* (*NE*+*NS*)) *W N* ⊆ *set-mset* (*dom-m N*)›
using *empty dom-m-vdom*
by (*auto simp*: *vdom-m-def*)
have ‹{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, {#}, {#}).
i ∈# *dom-m N*#} =
{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, {#}, {#}).
True#}› **for** *L*
by (*rule filter-mset-cong2*) (*auto simp*: *clause-to-update-def*)
then have *corr2*: ‹*correct-watching*′′′
({#*mset* (*fst x*). *x* ∈# *init-clss-l* (*get-clauses-wl y*)#} + *NE* + *NS*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, *Q*, *W*′*a*) ⟹
correct-watching′ (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*,*NS*, *US*, *Q*, *W*′*a*)› **for** *W*′*a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *correct-watching*′′′*.simps correct-watching*′*.simps*)
have *eq2*: ‹*all-init-lits* (*get-clauses-wl y*) (*NE*+*NS*) = *all-init-lits N* (*NE*+*NS*)›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *T all-init-lits-def NUE*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
have ‹*i* ∈# *dom-m N* ⟹ *set* (*N* ∝ *i*) ⊆ *set-mset* (*all-init-lits N* (*NE*+*NS*))› **for** *i*
using *lits* **by** (*auto dest*!: *multi-member-split split-list*
simp: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def ran-m-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits*)
then have *blit2*: ‹*correct-watching*′′′
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE* + *NS*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, *Q*, *W*′*a*) ⟹
blits-in-$\mathcal{L}_{in}$′ (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, *Q*, *W*′*a*)› **for** *W*′*a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
unfolding *correct-watching*′′′*.simps blits-in-*$\mathcal{L}_{in}$′*-def eq2*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits all-init-lits-alt-def*[*symmetric*]
by (*fastforce simp*: *correct-watching*′′′*.simps blits-in-*$\mathcal{L}_{in}$′*-def*
simp: *eq* $\mathcal{L}_{all}$*-all-init-atms eq2*

dest!: *multi-member-split*[*of* - ‹*all-init-lits N* (*NE*+*NS*)›]
dest: *mset-eq-setD*)
have ‹*correct-watching'''*
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + (*NE* + *NS*))
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, *Q*, *W'a*) ⟹
vdom-m (*all-init-atms N* (*NE*+*NS*)) *W'a N* ⊆ *set-mset* (*dom-m N*)› **for** *W'a*
unfolding *correct-watching'''.simps blits-in-*$\mathcal{L}_{in}$*'-def*
$\mathcal{L}_{all}$*-all-init-atms-all-init-lits all-init-lits-def*[*symmetric*]
all-init-lits-alt-def[*symmetric*]
using *eq eq3*
by (*force simp*: *correct-watching'''.simps vdom-m-def NUE*
$\mathcal{L}_{all}$*-all-init-atms*)
then have *st*: ‹(*x*, *W'a*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-init-atms N* (*NE*+*NS*))) ⟹
correct-watching'''
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE* + *NS*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, *US*, *Q*, *W'a*) ⟹
((*M'*, *N'*, *D'*, *j*, *x*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, (*fast-ema*,
slow-ema, *ccount*), *vdom*, *avdom*, *lcount*, *opts*),
M, *N*, *get-conflict-wl y*, *NE*, *UE*, *NS*, {#}, *Q*, *W'a*)
∈ *twl-st-heur-restart*› **for** *W'a m x*
using *S-T dom-m-vdom*
by (*auto simp*: *S T twl-st-heur-restart-def y-x NUE ac-simps*)
have *truc*: ‹*xa* ∈# *all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *learned-clss-l N*#} + (*UE* + *US*)) ⟹
xa ∈# *all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *init-clss-l N*#} + (*NE* + *NS*))› **for** *xa*
using *lits-y eq3 rtranclp-GC-remap-learned-clss-l*[*OF m*]
unfolding *literals-are-*$\mathcal{L}_{in}$*'-def all-init-lits-def NUE*
all-lits-of-mm-union all-init-lits-def $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*
by *auto*

show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *rewatch-heur-st-def T S*
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal by (*auto dest*!: *valid-arena-vdom-subset simp*: *twl-st-heur-restart-def*)
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist dom-m-vdom W*[*unfolded T*, *simplified*] *lits*])
apply (*solves simp*)
apply (*rule vd*)
apply (*rule order-trans*[*OF ref-two-step'*])
apply (*rule rewatch-correctness*[**where** *M=M* **and** *N=N* **and** *NE=NE* **and** *UE=UE* **and** *C=D*
and *Q=Q* **and**
NS=NS **and** *US=US*])
apply (*rule empty*[*unfolded all-init-lits-def*]; *assumption*)
apply (*rule struct-wf*; *assumption*)
subgoal using *lits eq2* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-init-atms-def all-init-lits-def*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
simp del: *all-init-atms-def*[*symmetric*])
apply (*subst conc-fun-RES*)
apply (*rule order.refl*)
by (*fastforce simp*: *rewatch-spec-def RETURN-RES-refine-iff NUE*
literals-are-in-$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*'-def add.assoc*
intro: *corr2 blit2 st dest*: *truc*)

qed

lemma *GC-remap-dom-m-subset*:
⟨*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m old'* ⊆# *dom-m old*⟩
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m-subset*:
⟨*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m old'* ⊆# *dom-m old*⟩
apply (*induction rule*: *rtranclp-induct*[*of r* ⟨(-, -, -)⟩ ⟨(-, -, -)⟩, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*] **by** *auto*
done

lemma *GC-remap-mapping-unchanged*:
⟨*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *C* ∈ *dom m* $\Longrightarrow$ *m' C* = *m C*⟩
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) *auto*

lemma *rtranclp-GC-remap-mapping-unchanged*:
⟨*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *C* ∈ *dom m* $\Longrightarrow$ *m' C* = *m C*⟩
apply (*induction rule*: *rtranclp-induct*[*of r* ⟨(-, -, -)⟩ ⟨(-, -, -)⟩, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-unchanged*[*of old1 m1 new1 old2 m2 new2*, *of C*]
by (*auto dest*: *GC-remap-mapping-unchanged simp*: *dom-def intro*!: *image-mset-cong2*)
done

lemma *GC-remap-mapping-dom-extended*:
⟨*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)⟩
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-mapping-dom-extended*:
⟨*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)⟩
apply (*induction rule*: *rtranclp-induct*[*of r* ⟨(-, -, -)⟩ ⟨(-, -, -)⟩, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-dom-extended*[*of old1 m1 new1 old2 m2 new2*]
GC-remap-dom-m-subset[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-dom-m-subset[*of old m new old1 m1 new1*]
by (*auto dest*: *GC-remap-mapping-dom-extended simp*: *dom-def mset-subset-eq-exists-conv*)
done

lemma *GC-remap-dom-m*:
⟨*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* − *dom-m old'*)⟩
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m*:
⟨*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) $\Longrightarrow$ *dom-m new'* = *dom-m new* + *the* '# *m'* '# (*dom-m old* − *dom-m old'*)⟩
apply (*induction rule*: *rtranclp-induct*[*of r* ⟨(-, -, -)⟩ ⟨(-, -, -)⟩, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*

using *GC-remap-dom-m*[*of old1 m1 new1 old2 m2 new2*] *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-dom-m-subset[*of old m new old1 m1 new1*]
GC-remap-mapping-unchanged[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-mapping-dom-extended[*of old m new old1 m1 new1*]
by (*auto dest*: *simp*: *mset-subset-eq-exists-conv intro*!: *image-mset-cong2*)
done

lemma *isasat-GC-clauses-rel-packed-le*:
assumes
xy: ‹$(x, y) \in$ *twl-st-heur-restart'''* *r*› **and**
ST: ‹$(S, T) \in$ *isasat-GC-clauses-rel y*›
shows ‹*length* (*get-clauses-wl-heur S*) $\leq$ *length* (*get-clauses-wl-heur x*)› **and**
‹$\forall C \in$ *set* (*get-vdom S*). $C <$ *length* (*get-clauses-wl-heur x*)›
proof −
obtain *m* **where**
‹$(S, T) \in$ *twl-st-heur-restart*› **and**
‹$\forall L \in \#$*all-init-lits-st y*. *get-watched-wl T L* = []› **and**
‹*get-trail-wl T* = *get-trail-wl y*› **and**
‹*get-conflict-wl T* = *get-conflict-wl y*› **and**
‹*get-unit-init-clss-wl T* = *get-unit-init-clss-wl y*› **and**
‹*get-unit-learned-clss-wl T* = *get-unit-learned-clss-wl y*› **and**
remap: ‹*GC-remap*** (*get-clauses-wl y*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *get-clauses-wl T*)› **and**
packed: ‹*arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*)›
using *ST* **by** *auto*
have ‹*valid-arena* (*get-clauses-wl-heur x*) (*get-clauses-wl y*) (*set* (*get-vdom x*))›
using *xy* **unfolding** *twl-st-heur-restart-def* **by** (*cases x*; *cases y*) *auto*
from *valid-arena-ge-length-clauses*[*OF this*]
have ‹($\sum C \in \#$*dom-m* (*get-clauses-wl y*). *length* (*get-clauses-wl y* $\propto$ *C*) +
header-size (*get-clauses-wl y* $\propto$ *C*)) $\leq$ *length* (*get-clauses-wl-heur x*)›
(**is** ‹*?A* $\leq$ -›) **.**
moreover have ‹*?A* = ($\sum C \in \#$*dom-m* (*get-clauses-wl T*). *length* (*get-clauses-wl T* $\propto$ *C*) +
header-size (*get-clauses-wl T* $\propto$ *C*))›
using *rtranclp-GC-remap-ran-m-remap*[*OF remap*]
by (*auto simp*: *rtranclp-GC-remap-dom-m*[*OF remap*] *intro*!: *sum-mset-cong*)
ultimately show *le*: ‹*length* (*get-clauses-wl-heur S*) $\leq$ *length* (*get-clauses-wl-heur x*)›
using *packed* **unfolding** *arena-is-packed-def* **by** *simp*

have ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
using *ST* **unfolding** *twl-st-heur-restart-def* **by** (*cases S*; *cases T*) *auto*
then show ‹$\forall C \in$ *set* (*get-vdom S*). $C <$ *length* (*get-clauses-wl-heur x*)›
using *le*
by (*auto dest*: *valid-arena-in-vdom-le-arena*)
qed

lemma *isasat-GC-clauses-wl-D*:
‹(*isasat-GC-clauses-wl-D*, *cdcl-GC-clauses-wl*)
$\in$ *twl-st-heur-restart'''* *r* $\rightarrow_f$ ⟨*twl-st-heur-restart''''* *r*⟩*nres-rel*›
unfolding *isasat-GC-clauses-wl-D-def cdcl-GC-clauses-wl-D-alt-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg isasat-GC-clauses-prog-wl-cdcl-remap-st*[**where** *r*=*r*]
rewatch-heur-st-correct-watching)
subgoal unfolding *isasat-GC-clauses-pre-wl-D-def* **by** *blast*
subgoal by *fast*
subgoal by (*rule isasat-GC-clauses-rel-packed-le*)

subgoal by (*rule isasat-GC-clauses-rel-packed-le*(*2*))
apply *assumption*+
subgoal by (*auto*)
subgoal by (*auto*)
done

definition *cdcl-twl-full-restart-wl-D-GC-heur-prog* **where**
‹*cdcl-twl-full-restart-wl-D-GC-heur-prog S0* = *do* {
S ← *do* {
if count-decided-st-heur S0 > *0*
then do {
S ← *find-decomp-wl-st-int 0 S0*;
empty-Q S
} *else RETURN S0*
};
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
T ← *remove-one-annot-true-clause-imp-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S0*));
U ← *mark-to-delete-clauses-wl-D-heur T*;
ASSERT(*length* (*get-clauses-wl-heur U*) = *length* (*get-clauses-wl-heur S0*));
V ← *isasat-GC-clauses-wl-D U*;
RETURN V
}›

lemma
cdcl-twl-full-restart-wl-GC-prog-pre-heur:
‹*cdcl-twl-full-restart-wl-GC-prog-pre T* ⟹
(*S*, *T*) ∈ *twl-st-heur‴ r* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart-ana r*› (**is** ‹- ⟹ - *?A*›) **and**
cdcl-twl-full-restart-wl-D-GC-prog-post-heur:
‹*cdcl-twl-full-restart-wl-GC-prog-post S0 T* ⟹
(*S*, *T*) ∈ *twl-st-heur* ⟷ (*S*, *T*) ∈ *twl-st-heur-restart*› (**is** ‹- ⟹ - *?B*›)
proof −
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong phase-saving-cong*
cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong

show ‹*cdcl-twl-full-restart-wl-GC-prog-pre T* ⟹ *?A*›
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def cdcl-twl-full-restart-l-GC-prog-pre-def*
apply *normalize-goal*+
apply (*rule iffI*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*simp add*: *twl-st-heur-def twl-st-heur-restart-ana-def*
twl-st-heur-restart-def del: *isasat-input-nempty-def*)
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]

vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
by (*cases S*; *cases T*)
(*simp add*: *twl-st-heur-def twl-st-heur-restart-ana-def*
twl-st-heur-restart-def del: *isasat-input-nempty-def*)
done
show ‹*cdcl-twl-full-restart-wl-GC-prog-post S0 T* ⟹ *?B*›
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-GC-prog-post-def*
cdcl-twl-full-restart-wl-GC-prog-post-def
cdcl-twl-full-restart-l-GC-prog-pre-def
apply *normalize-goal+*
subgoal for *S0′ T′ S0″*
apply (*rule iffI*)
subgoal
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T T′*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
cdcl-twl-restart-l-invs[*of S0′ S0″ T′*]
apply −
apply (*clarsimp simp del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*; *cases T′*)
apply (*simp add*: *twl-st-heur-def twl-st-heur-restart-def del*: *isasat-input-nempty-def*)
using *isa-vmtf-cong option-lookup-clause-rel-cong trail-pol-cong heuristic-rel-cong*
by *presburger*
subgoal
using *literals-are-*$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T T′*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
cdcl-twl-restart-l-invs[*of S0′ S0″ T′*]
apply −
apply (*cases S*; *cases T*)
by (*clarsimp simp add*: *twl-st-heur-def twl-st-heur-restart-def*
simp del: *isasat-input-nempty-def*)
done
done

qed

lemma *cdcl-twl-full-restart-wl-D-GC-heur-prog*:
‹(*cdcl-twl-full-restart-wl-D-GC-heur-prog*, *cdcl-twl-full-restart-wl-GC-prog*) ∈
twl-st-heur‴ r $\rightarrow_f$ ⟨*twl-st-heur⁗ r*⟩*nres-rel*›
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-def*
cdcl-twl-full-restart-wl-GC-prog-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg cdcl-twl-local-restart-wl-spec0*
remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D[**where** *r=r*, *THEN fref-to-Down*]
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl2-D[**where** *r=r*, *THEN fref-to-Down*]
isasat-GC-clauses-wl-D[**where** *r=r*, *THEN fref-to-Down*])
apply (*subst* (*asm*) *cdcl-twl-full-restart-wl-GC-prog-pre-heur*, *assumption*)
apply *assumption*
subgoal
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def*
cdcl-twl-full-restart-l-GC-prog-pre-def
by *normalize-goal+ auto*

subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply *assumption*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal for *x y*
by (*blast dest*: *cdcl-twl-full-restart-wl-D-GC-prog-post-heur*)
done

definition *end-of-restart-phase* :: ⟨*restart-heuristics* ⇒ *64 word*⟩ **where**
⟨*end-of-restart-phase* = (λ(-, -, (*restart-phase*,- ,- , *end-of-phase*, -), -).
end-of-phase)⟩

definition *end-of-restart-phase-st* :: ⟨*twl-st-wl-heur* ⇒ *64 word*⟩ **where**
⟨*end-of-restart-phase-st* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*).
end-of-restart-phase heur)⟩

definition *end-of-rephasing-phase-st* :: ⟨*twl-st-wl-heur* ⇒ *64 word*⟩ **where**
⟨*end-of-rephasing-phase-st* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*).
end-of-rephasing-phase-heur heur)⟩

Using $a + (1::'a)$ ensures that we do not get stuck with 0.

fun *incr-restart-phase-end* :: ⟨*restart-heuristics* ⇒ *restart-heuristics*⟩ **where**
⟨*incr-restart-phase-end* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*), *wasted*) =
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase* + *length-phase*, (*length-phase* ∗ *3*) >> *1*), *wasted*)⟩

definition *update-restart-phases* :: ⟨*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*⟩ **where**
⟨*update-restart-phases* = (λ(*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*). *do* {
heur ← *RETURN* (*incr-restart-phase heur*);
heur ← *RETURN* (*incr-restart-phase-end heur*);
RETURN (*M′*, *N′*, *D′*, *j*, *W′*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*, *heur*,
vdom, *avdom*, *lcount*, *opts*, *old-arena*)
})⟩

definition *update-all-phases* :: ⟨*twl-st-wl-heur* ⇒ *nat* ⇒ (*twl-st-wl-heur* × *nat*) *nres*⟩ **where**
⟨*update-all-phases* = (λ*S n*. *do* {
let lcount = *get-learned-count S*;
end-of-restart-phase ← *RETURN* (*end-of-restart-phase-st S*);
S ← (*if end-of-restart-phase* > *of-nat lcount then RETURN S else update-restart-phases S*);
S ← (*if end-of-rephasing-phase-st S* > *of-nat lcount then RETURN S else rephase-heur-st S*);
RETURN (*S*, *n*)
})⟩

definition *restart-prog-wl-D-heur*
:: *twl-st-wl-heur* ⇒ *nat* ⇒ *bool* ⇒ (*twl-st-wl-heur* × *nat*) *nres*
where
⟨*restart-prog-wl-D-heur S n brk* = *do* {
b ← *restart-required-heur S n*;

    if $\neg$*brk* $\wedge$ *b* = *FLAG-GC-restart*
    then do {
        T $\leftarrow$ *cdcl-twl-full-restart-wl-D-GC-heur-prog S*;
        RETURN (*T*, *n+1*)
    }
    else if $\neg$*brk* $\wedge$ *b* = *FLAG-restart*
    then do {
        T $\leftarrow$ *cdcl-twl-restart-wl-heur S*;
        RETURN (*T*, *n+1*)
    }
    else update-all-phases S n
  }›

lemma *restart-required-heur-restart-required-wl*:
  ‹(*uncurry restart-required-heur*, *uncurry restart-required-wl*) $\in$
    twl-st-heur $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*restart-flag-rel*⟩*nres-rel*›
    unfolding *restart-required-heur-def restart-required-wl-def uncurry-def Let-def*
      restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def FLAG-no-restart-def
      GC-required-heur-def
    by (*intro frefI nres-relI*)
      (*auto simp*: *twl-st-heur-def get-learned-clss-wl-def RETURN-RES-refine-iff*)

lemma *restart-required-heur-restart-required-wl0*:
  ‹(*uncurry restart-required-heur*, *uncurry restart-required-wl*) $\in$
    twl-st-heur''' *r* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*restart-flag-rel*⟩*nres-rel*›
    unfolding *restart-required-heur-def restart-required-wl-def uncurry-def Let-def*
      restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def FLAG-no-restart-def
      GC-required-heur-def
    by (*intro frefI nres-relI*)
     (*auto simp*: *twl-st-heur-def get-learned-clss-wl-def RETURN-RES-refine-iff*)

lemma *heuristic-rel-incr-restartI*[*intro*!]:
  ‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase-end heur*)›
  by (*auto simp*: *heuristic-rel-def*)

lemma *update-all-phases-Pair*:
  ‹(*uncurry update-all-phases*, *uncurry* (*RETURN oo Pair*)) $\in$
    twl-st-heur'''' *r* $\times_f$ *nat-rel* $\rightarrow_f$ ⟨*twl-st-heur''''* *r* $\times_f$ *nat-rel*⟩*nres-rel*›
proof –
 have [*refine0*]: ‹$(S, S') \in$ *twl-st-heur''''* *r* $\Longrightarrow$ *update-restart-phases S* $\leq$ *SPEC*($\lambda S.$ $(S, S') \in$ *twl-st-heur''''*
r)›
    for *S* :: *twl-st-wl-heur* **and** *S'* :: ‹*nat twl-st-wl*›
    unfolding *update-all-phases-def update-restart-phases-def*
    by (*auto simp*: *twl-st-heur'-def twl-st-heur-def*
        intro!: *rephase-heur-st-spec*[*THEN order-trans*]
        simp del: *incr-restart-phase-end.simps incr-restart-phase.simps*)
  have [*refine0*]: ‹$(S, S') \in$ *twl-st-heur''''* *r* $\Longrightarrow$ *rephase-heur-st S* $\leq$ *SPEC*($\lambda S.$ $(S, S') \in$ *twl-st-heur''''*
r)›
    for *S* :: *twl-st-wl-heur* **and** *S'* :: ‹*nat twl-st-wl*›
    unfolding *update-all-phases-def rephase-heur-st-def*
    apply (*cases S'*)
    apply (*refine-vcg rephase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
    apply (*clarsimp-all simp*: *twl-st-heur'-def twl-st-heur-def*)
    done
  have *Pair-alt-def*: ‹*RETURN* $\circ\circ$ *Pair* = (λS *n*. *do* {*S* $\leftarrow$ *RETURN S*; *S* $\leftarrow$ *RETURN S*; *RETURN*

$(S, n)\})$›
    by (*auto intro*!: *ext*)

  show *?thesis*
    supply[[*goals-limit=1*]]
    unfolding *update-all-phases-def Pair-alt-def*
    apply (*subst* (*1*) *bind-to-let-conv*)
    apply (*subst* (*1*) *Let-def*)
    apply (*subst* (*1*) *Let-def*)
    apply (*intro frefI nres-relI*)
    apply (*case-tac x rule*:*prod.exhaust*)
    apply (*simp only*: *uncurry-def prod.case*)
    apply *refine-vcg*
    subgoal by *simp*
    subgoal by *simp*
    subgoal by *simp*
    done
qed

lemma *restart-prog-wl-D-heur-restart-prog-wl-D*:
  ‹(*uncurry2 restart-prog-wl-D-heur*, *uncurry2 restart-prog-wl*) ∈
    twl-st-heur''' $r \times_f$ *nat-rel* $\times_f$ *bool-rel* $\rightarrow_f$ ⟨*twl-st-heur''''* $r \times_f$ *nat-rel*⟩*nres-rel*›
proof −
  have [*refine0*]: ‹*GC-required-heur S n* ≤ *SPEC* (λ-. *True*)› **for** *S n*
    by (*auto simp*: *GC-required-heur-def*)
  show *?thesis*
   supply *RETURN-as-SPEC-refine*[*refine2 del*]
    unfolding *restart-prog-wl-D-heur-def restart-prog-wl-def uncurry-def*
    apply (*intro frefI nres-relI*)
    apply (*refine-rcg*
        restart-required-heur-restart-required-wl0[**where** *r=r*, *THEN fref-to-Down-curry*]
        cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog[**where** *r=r*, *THEN fref-to-Down*]
        cdcl-twl-full-restart-wl-D-GC-heur-prog[**where** *r=r*, *THEN fref-to-Down*]
        update-all-phases-Pair[**where** *r=r*, *THEN fref-to-Down-curry*, *unfolded comp-def*])
    subgoal by *auto*
    subgoal by (*auto simp*: *restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def*
      FLAG-no-restart-def)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by (*auto simp*: *restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def*
      FLAG-no-restart-def)
    subgoal by *auto*
    subgoal by *auto*
    subgoal
      by *auto*
    done
 qed

lemma *restart-prog-wl-D-heur-restart-prog-wl-D2*:
  ‹(*uncurry2 restart-prog-wl-D-heur*, *uncurry2 restart-prog-wl*) ∈
  twl-st-heur $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\rightarrow_f$ ⟨*twl-st-heur* $\times_f$ *nat-rel*⟩*nres-rel*›
  apply (*intro frefI nres-relI*)
  apply (*rule-tac r2* = ‹*length*(*get-clauses-wl-heur* (*fst* (*fst x*)))› **and** *x'1* = ‹*y*› **in**
    order-trans[*OF restart-prog-wl-D-heur-restart-prog-wl-D*[*THEN fref-to-Down*]])
  apply *fast*

apply (*auto intro*!: *conc-fun-R-mono*)
done

definition *isasat-trail-nth-st* :: ‹*twl-st-wl-heur* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isasat-trail-nth-st S i* = *isa-trail-nth* (*get-trail-wl-heur S*) *i*›

lemma *isasat-trail-nth-st-alt-def*:
‹*isasat-trail-nth-st* = (λ(*M*, -) *i*. *isa-trail-nth M i*)›
by (*auto simp*: *isasat-trail-nth-st-def intro*!: *ext*)

definition *get-the-propagation-reason-pol-st* :: ‹*twl-st-wl-heur* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-the-propagation-reason-pol-st S i* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *i*›

lemma *get-the-propagation-reason-pol-st-alt-def*:
‹*get-the-propagation-reason-pol-st* = (λ(*M*, -) *i*. *get-the-propagation-reason-pol M i*)›
by (*auto simp*: *get-the-propagation-reason-pol-st-def intro*!: *ext*)

definition *rewatch-heur-st-pre* :: ‹*twl-st-wl-heur* ⇒ *bool*› **where**
‹*rewatch-heur-st-pre S* ⟷ (∀ *i* < *length* (*get-vdom S*). *get-vdom S* ! *i* ≤ *sint64-max*)›

lemma *isasat-GC-clauses-wl-D-rewatch-pre*:
assumes
‹*length* (*get-clauses-wl-heur x*) ≤ *sint64-max*› **and**
‹*length* (*get-clauses-wl-heur xc*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹∀ *i* ∈ *set* (*get-vdom xc*). *i* ≤ *length* (*get-clauses-wl-heur x*)›
shows ‹*rewatch-heur-st-pre xc*›
using *assms*
unfolding *rewatch-heur-st-pre-def all-set-conv-all-nth*
by *auto*

lemma *li-uint32-maxdiv2-le-unit32-max*: ‹*a* ≤ *uint32-max div 2* + *1* ⟹ *a* ≤ *uint32-max*›
by (*auto simp*: *uint32-max-def*)

end
theory *IsaSAT-Arena-Sorting-LLVM*
imports *IsaSAT-Sorting-LLVM*
begin
definition *idx-cdom* :: *arena* ⇒ *nat set* **where**
idx-cdom arena ≡ {*i*. *valid-sort-clause-score-pre-at arena i*}

definition *mop-clause-score-less* **where**
‹*mop-clause-score-less arena i j* = *do* {
ASSERT(*valid-sort-clause-score-pre-at arena i*);
ASSERT(*valid-sort-clause-score-pre-at arena j*);
RETURN (*clause-score-ordering* (*clause-score-extract arena i*) (*clause-score-extract arena j*))
}›

sepref-register *clause-score-extract*

sepref-def (**in** −) *clause-score-extract-code*
is ‹*uncurry* (*RETURN oo clause-score-extract*)›
:: ‹[*uncurry valid-sort-clause-score-pre-at*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} → *uint32-nat-assn* $\times_a$ *uint32-nat-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-extract-def valid-sort-clause-score-pre-at-def*

apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-def (**in** −) *clause-score-ordering-code*
is ‹*uncurry* (*RETURN oo clause-score-ordering*)›
:: ‹(*uint32-nat-assn* $\times_a$ *uint32-nat-assn*)^k $*_a$ (*uint32-nat-assn* $\times_a$ *uint32-nat-assn*)^k $\rightarrow_a$ *bool1-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-ordering-def*
by *sepref*

sepref-register *mop-clause-score-less clause-score-less clause-score-ordering*
sepref-def *mop-clause-score-less-impl*
is ‹*uncurry2 mop-clause-score-less*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *mop-clause-score-less-def*
by *sepref*

interpretation *LBD*: *weak-ordering-on-lt* **where**
C = *idx-cdom vs* **and**
less = *clause-score-less vs*
by *unfold-locales*
(*auto simp*: *clause-score-less-def clause-score-extract-def*
clause-score-ordering-def split: *if-splits*)

interpretation *LBD*: *parameterized-weak-ordering idx-cdom clause-score-less*
mop-clause-score-less
by *unfold-locales*
(*auto simp*: *mop-clause-score-less-def*
idx-cdom-def clause-score-less-def)

global-interpretation *LBD*: *parameterized-sort-impl-context*
woarray-assn snat-assn eoarray-assn snat-assn snat-assn
return return
eo-extract-impl
array-upd
idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
arena-fast-assn
defines
LBD-is-guarded-insert-impl = *LBD.is-guarded-param-insert-impl*
and *LBD-is-unguarded-insert-impl* = *LBD.is-unguarded-param-insert-impl*
and *LBD-unguarded-insertion-sort-impl* = *LBD.unguarded-insertion-sort-param-impl*
and *LBD-guarded-insertion-sort-impl* = *LBD.guarded-insertion-sort-param-impl*
and *LBD-final-insertion-sort-impl* = *LBD.final-insertion-sort-param-impl*

and *LBD-pcmpo-idxs-impl* = *LBD.pcmpo-idxs-impl*
and *LBD-pcmpo-v-idx-impl* = *LBD.pcmpo-v-idx-impl*
and *LBD-pcmpo-idx-v-impl* = *LBD.pcmpo-idx-v-impl*
and *LBD-pcmp-idxs-impl* = *LBD.pcmp-idxs-impl*

and *LBD-mop-geth-impl* = *LBD.mop-geth-impl*
and *LBD-mop-seth-impl* = *LBD.mop-seth-impl*
and *LBD-sift-down-impl* = *LBD.sift-down-impl*
and *LBD-heapify-btu-impl* = *LBD.heapify-btu-impl*
and *LBD-heapsort-impl* = *LBD.heapsort-param-impl*

and *LBD-qsp-next-l-impl* = *LBD.qsp-next-l-impl*
and *LBD-qsp-next-h-impl* = *LBD.qsp-next-h-impl*
and *LBD-qs-partition-impl* = *LBD.qs-partition-impl*

and *LBD-partition-pivot-impl* = *LBD.partition-pivot-impl*
and *LBD-introsort-aux-impl* = *LBD.introsort-aux-param-impl*
and *LBD-introsort-impl* = *LBD.introsort-param-impl*
and *LBD-move-median-to-first-impl* = *LBD.move-median-to-first-param-impl*

apply *unfold-locales*
apply (*rule eo-hnr-dep*)+
unfolding *GEN-ALGO-def refines-param-relp-def*
by (*rule mop-clause-score-less-impl.refine*)

global-interpretation
LBD-it: *pure-eo-adapter sint64-nat-assn vdom-fast-assn arl-nth arl-upd*
defines *LBD-it-eo-extract-impl* = *LBD-it.eo-extract-impl*
apply (*rule al-pure-eo*)
by *simp*

global-interpretation *LBD-it*: *parameterized-sort-impl-context*
vdom-fast-assn LBD-it.eo-assn sint64-nat-assn
return return
LBD-it-eo-extract-impl
arl-upd
idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
arena-fast-assn
defines
LBD-it-is-guarded-insert-impl = *LBD-it.is-guarded-param-insert-impl*
and *LBD-it-is-unguarded-insert-impl* = *LBD-it.is-unguarded-param-insert-impl*
and *LBD-it-unguarded-insertion-sort-impl* = *LBD-it.unguarded-insertion-sort-param-impl*
and *LBD-it-guarded-insertion-sort-impl* = *LBD-it.guarded-insertion-sort-param-impl*
and *LBD-it-final-insertion-sort-impl* = *LBD-it.final-insertion-sort-param-impl*

and *LBD-it-pcmpo-idxs-impl* = *LBD-it.pcmpo-idxs-impl*
and *LBD-it-pcmpo-v-idx-impl* = *LBD-it.pcmpo-v-idx-impl*
and *LBD-it-pcmpo-idx-v-impl* = *LBD-it.pcmpo-idx-v-impl*
and *LBD-it-pcmp-idxs-impl* = *LBD-it.pcmp-idxs-impl*

and *LBD-it-mop-geth-impl* = *LBD-it.mop-geth-impl*
and *LBD-it-mop-seth-impl* = *LBD-it.mop-seth-impl*
and *LBD-it-sift-down-impl* = *LBD-it.sift-down-impl*
and *LBD-it-heapify-btu-impl* = *LBD-it.heapify-btu-impl*
and *LBD-it-heapsort-impl* = *LBD-it.heapsort-param-impl*
and *LBD-it-qsp-next-l-impl* = *LBD-it.qsp-next-l-impl*
and *LBD-it-qsp-next-h-impl* = *LBD-it.qsp-next-h-impl*
and *LBD-it-qs-partition-impl* = *LBD-it.qs-partition-impl*

and *LBD-it-partition-pivot-impl* = *LBD-it.partition-pivot-impl*
and *LBD-it-introsort-aux-impl* = *LBD-it.introsort-aux-param-impl*
and *LBD-it-introsort-impl* = *LBD-it.introsort-param-impl*
and *LBD-it-move-median-to-first-impl* = *LBD-it.move-median-to-first-param-impl*

apply *unfold-locales*
unfolding *GEN-ALGO-def refines-param-relp-def*
apply (*rule mop-clause-score-less-impl.refine*)
done

lemmas [*llvm-inline*] = *LBD-it.eo-extract-impl-def*[*THEN meta-fun-cong*, *THEN meta-fun-cong*]

print-named-simpset *llvm-inline*
export-llvm
LBD-heapsort-impl :: - ⇒ - ⇒ -
LBD-introsort-impl :: - ⇒ - ⇒ -

end
theory *IsaSAT-Restart-Heuristics-LLVM*
imports *IsaSAT-Restart-Heuristics IsaSAT-Setup-LLVM*
IsaSAT-VMTF-LLVM IsaSAT-Rephase-LLVM
IsaSAT-Arena-Sorting-LLVM
begin

hide-fact (**open**) *Sepref-Rules.frefI*
no-notation *Sepref-Rules.fref* ($[-]_{fd}$ - → - [*0*,*60*,*60*] *60*)
no-notation *Sepref-Rules.freft* (- $\rightarrow_{fd}$ - [*60*,*60*] *60*)
no-notation *Sepref-Rules.freftnd* (- $\rightarrow_{f}$ - [*60*,*60*] *60*)
no-notation *Sepref-Rules.frefnd* ($[-]_{f}$ - → - [*0*,*60*,*60*] *60*)

sepref-def *FLAG-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-restart*)›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *word-assn*›
unfolding *FLAG-restart-def*
by *sepref*

sepref-def *FLAG-no-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-no-restart*)›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *word-assn*›
unfolding *FLAG-no-restart-def*
by *sepref*

sepref-def *FLAG-GC-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-GC-restart*)›
:: ‹$unit\text{-}assn^k \rightarrow_a$ *word-assn*›
unfolding *FLAG-GC-restart-def*
by *sepref*

lemma *current-restart-phase-alt-def*:
‹*current-restart-phase* = (λ(*fast-ema*, *slow-ema*,
(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), -).
restart-phase)›
by *auto*

sepref-def *current-restart-phase-impl*
is ‹*RETURN o current-restart-phase*›
:: ‹$heuristic\text{-}assn^k \rightarrow_a$ *word-assn*›
unfolding *current-restart-phase-alt-def heuristic-assn-def*

by *sepref*

sepref-def *get-restart-phase-imp*
is ⟨(*RETURN o get-restart-phase*)⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *get-restart-phase-def isasat-bounded-assn-def*
by *sepref*

sepref-def *end-of-restart-phase-impl*
is ⟨*RETURN o end-of-restart-phase*⟩
:: ⟨$heuristic\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *end-of-restart-phase-def heuristic-assn-def*
by *sepref*

sepref-def *end-of-restart-phase-st-impl*
is ⟨*RETURN o end-of-restart-phase-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *end-of-restart-phase-st-def isasat-bounded-assn-def*
by *sepref*

sepref-def *end-of-rephasing-phase-impl*
is ⟨*RETURN o end-of-rephasing-phase*⟩
:: ⟨$phase\text{-}heur\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *end-of-rephasing-phase-def heuristic-assn-def*
by *sepref*

sepref-def *end-of-rephasing-phase-heur-impl*
is ⟨*RETURN o end-of-rephasing-phase-heur*⟩
:: ⟨$heuristic\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *end-of-rephasing-phase-heur-def heuristic-assn-def*
by *sepref*

sepref-def *end-of-rephasing-phase-st-impl*
is ⟨*RETURN o end-of-rephasing-phase-st*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a word\text{-}assn$⟩
unfolding *end-of-rephasing-phase-st-def isasat-bounded-assn-def*
by *sepref*

lemma *incr-restart-phase-end-alt-def*:
⟨*incr-restart-phase-end* = (λ(*fast-ema*, *slow-ema*,
(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*), *wasted*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase* + *length-phase*,
(*length-phase* ∗ *3*) >> *1*), *wasted*))⟩
by *auto*

sepref-def *incr-restart-phase-end-impl*
is ⟨*RETURN o incr-restart-phase-end*⟩
:: ⟨$heuristic\text{-}assn^d \rightarrow_a heuristic\text{-}assn$⟩
supply[[*goals-limit*=*1*]]
unfolding *heuristic-assn-def incr-restart-phase-end-alt-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *incr-restart-phase-alt-def*:
⟨*incr-restart-phase* = (λ(*fast-ema*, *slow-ema*,

(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*))›
by *auto*

sepref-def *incr-restart-phase-impl*
is ‹*RETURN o incr-restart-phase*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
unfolding *heuristic-assn-def incr-restart-phase-alt-def*
by *sepref*

sepref-register *incr-restart-phase incr-restart-phase-end*
update-restart-phases update-all-phases

sepref-def *update-restart-phases-impl*
is ‹*update-restart-phases*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *update-restart-phases-def isasat-bounded-assn-def*
fold-tuple-optimizations
by *sepref*

sepref-def *update-all-phases-impl*
is ‹*uncurry update-all-phases*›
:: ‹*isasat-bounded-assn*^{d} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$
isasat-bounded-assn $\times_a$ *uint64-nat-assn*›
unfolding *update-all-phases-def*
fold-tuple-optimizations
by *sepref*

sepref-def *find-local-restart-target-level-fast-code*
is ‹*uncurry find-local-restart-target-level-int*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *vmtf-remove-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-int-inv-def*
length-uint32-nat-def vmtf-remove-assn-def trail-pol-fast-assn-def
apply (*annot-unat-const TYPE*(*32*))
apply (*rewrite at stamp* (¤) *annot-index-of-atm*)
apply (*rewrite* **in** (- ! -) *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite* **in** (- ! ¤) *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite* **in** (¤ < *length* -) *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

sepref-def *find-local-restart-target-level-st-fast-code*
is ‹*find-local-restart-target-level-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
supply [[*goals-limit=1*]] *length-rev*[*simp del*]
unfolding *find-local-restart-target-level-st-alt-def isasat-bounded-assn-def PR-CONST-def*
fold-tuple-optimizations
by *sepref*

sepref-def *empty-Q-fast-code*
is ‹*empty-Q*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-Q-def isasat-bounded-assn-def fold-tuple-optimizations*
heuristic-assn-def

by *sepref*

sepref-register *cdcl-twl-local-restart-wl-D-heur*
empty-Q find-decomp-wl-st-int

find-theorems *count-decided-st-heur name*:*refine*
sepref-def *cdcl-twl-local-restart-wl-D-heur-fast-code*
is ‹*cdcl-twl-local-restart-wl-D-heur*›
:: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a isasat\text{-}bounded\text{-}assn$›
unfolding *cdcl-twl-local-restart-wl-D-heur-def PR-CONST-def*
fold-tuple-optimizations
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-register *upper-restart-bound-not-reached*

sepref-def *upper-restart-bound-not-reached-fast-impl*
is ‹(*RETURN o upper-restart-bound-not-reached*)›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
unfolding *upper-restart-bound-not-reached-def PR-CONST-def isasat-bounded-assn-def*
fold-tuple-optimizations
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-register *lower-restart-bound-not-reached*
sepref-def *lower-restart-bound-not-reached-impl*
is ‹(*RETURN o lower-restart-bound-not-reached*)›
:: ‹$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$›
unfolding *lower-restart-bound-not-reached-def PR-CONST-def isasat-bounded-assn-def*
fold-tuple-optimizations
supply [[*goals-limit* = *1*]]
by *sepref*

find-theorems *sort-spec*

definition *lbd-sort-clauses-raw* :: ‹*arena* ⇒ *vdom* ⇒ *nat* ⇒ *nat* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses-raw arena N* = *pslice-sort-spec idx-cdom clause-score-less arena N*›

definition *lbd-sort-clauses* :: ‹*arena* ⇒ *vdom* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses arena N* = *lbd-sort-clauses-raw arena N 0* (*length N*)›

lemmas *LBD-introsort*[*sepref-fr-rules*] =
LBD-it.introsort-param-impl-correct[*unfolded lbd-sort-clauses-raw-def*[*symmetric*] *PR-CONST-def*]

lemma *quicksort-clauses-by-score-sort*:
‹(*lbd-sort-clauses*, *sort-clauses-by-score*) ∈
Id → *Id* → ⟨*Id*⟩*nres-rel*›
apply (*intro fun-relI nres-relI*)
subgoal for *arena arena′ vdom vdom′*
by (*auto simp*: *lbd-sort-clauses-def lbd-sort-clauses-raw-def sort-clauses-by-score-def*
pslice-sort-spec-def le-ASSERT-iff idx-cdom-def slice-rel-def br-def
conc-fun-RES sort-spec-def
eq-commute[*of -* ‹*length vdom′*›]
intro!: *ASSERT-leI slice-sort-spec-refine-sort*[*THEN order-trans*, *of - vdom vdom*])

done

sepref-register *lbd-sort-clauses-raw*
sepref-def *lbd-sort-clauses-impl*
is ‹*uncurry lbd-sort-clauses*›
:: ‹$arena\text{-}fast\text{-}assn^k *_a vdom\text{-}fast\text{-}assn^d \rightarrow_a vdom\text{-}fast\text{-}assn$›
supply[[*goals-limit=1*]]
unfolding *lbd-sort-clauses-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemmas [*sepref-fr-rules*] =
lbd-sort-clauses-impl.refine[*FCOMP quicksort-clauses-by-score-sort*]

sepref-register *remove-deleted-clauses-from-avdom arena-status DELETED*

sepref-def *remove-deleted-clauses-from-avdom-fast-code*
is ‹*uncurry isa-remove-deleted-clauses-from-avdom*›
:: ‹$[\lambda(N,\ vdom).\ length\ vdom \leq sint64\text{-}max]_a\ arena\text{-}fast\text{-}assn^k *_a vdom\text{-}fast\text{-}assn^d \rightarrow vdom\text{-}fast\text{-}assn$›
supply [[*goals-limit=1*]]
unfolding *isa-remove-deleted-clauses-from-avdom-def*
convert-swap gen-swap if-conn(*4*)
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-def *sort-vdom-heur-fast-code*
is ‹*sort-vdom-heur*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$›
supply *sort-clauses-by-score-invI*[*intro*]
[[*goals-limit=1*]]
unfolding *sort-vdom-heur-def isasat-bounded-assn-def*
by *sepref*

sepref-register *max-restart-decision-lvl*

sepref-def *minimum-number-between-restarts-impl*
is ‹*uncurry0* (*RETURN minimum-number-between-restarts*)›
:: ‹$unit\text{-}assn^k \rightarrow_a word\text{-}assn$›
unfolding *minimum-number-between-restarts-def*
by *sepref*

sepref-def *uint32-nat-assn-impl*
is ‹*uncurry0* (*RETURN max-restart-decision-lvl*)›
:: ‹$unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$›
unfolding *max-restart-decision-lvl-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-def *GC-EVERY-impl*
is ‹*uncurry0* (*RETURN GC-EVERY*)›
:: ‹$unit\text{-}assn^k \rightarrow_a word\text{-}assn$›
unfolding *GC-EVERY-def*
by *sepref*

sepref-def *get-reductions-count-fast-code*
  is ⟨*RETURN o get-reductions-count*⟩
  :: ⟨*isasat-bounded-assn*^{k} $\rightarrow_a$ *word-assn*⟩
  unfolding *get-reduction-count-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-register *get-reductions-count*

lemma *of-nat-snat*:
  (*id*,*of-nat*) ∈ *snat-rel'* *TYPE*('*a*::*len2*) → *word-rel*
  by (*auto simp*: *snat-rel-def snat.rel-def in-br-conv snat-eq-unat*)

sepref-def *GC-required-heur-fast-code*
  is ⟨*uncurry GC-required-heur*⟩
  :: ⟨*isasat-bounded-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*⟩
  supply [[*goals-limit*=*1*]] *of-nat-snat*[*sepref-import-param*]
  unfolding *GC-required-heur-def GC-EVERY-def*
  apply (*annot-snat-const TYPE*(*64*))
  by *sepref*

sepref-register *ema-get-value get-fast-ema-heur get-slow-ema-heur*
sepref-def *restart-required-heur-fast-code*
  is ⟨*uncurry restart-required-heur*⟩
  :: ⟨*isasat-bounded-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $\rightarrow_a$ *word-assn*⟩
  supply [[*goals-limit*=*1*]]
  unfolding *restart-required-heur-def*
  apply (*rewrite* **in** ⟨⌑ < -⟩ *unat-const-fold*(*3*)[**where** '*a*=*32*])
  apply (*rewrite* **in** ⟨(- >> *32*) < ⌑⟩ *annot-unat-unat-upcast*[**where** '*l*=*64*])
  apply (*annot-snat-const TYPE*(*64*))
  by *sepref*

sepref-register *isa-trail-nth isasat-trail-nth-st*

sepref-def *isasat-trail-nth-st-code*
  is ⟨*uncurry isasat-trail-nth-st*⟩
  :: ⟨*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *unat-lit-assn*⟩
  supply [[*goals-limit*=*1*]]
  unfolding *isasat-trail-nth-st-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-register *get-the-propagation-reason-pol-st*

sepref-def *get-the-propagation-reason-pol-st-code*
  is ⟨*uncurry get-the-propagation-reason-pol-st*⟩
  :: ⟨*isasat-bounded-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *snat-option-assn'* *TYPE*(*64*)⟩
  supply [[*goals-limit*=*1*]]
  unfolding *get-the-propagation-reason-pol-st-alt-def isasat-bounded-assn-def*
  by *sepref*

sepref-register *isasat-replace-annot-in-trail*
sepref-def *isasat-replace-annot-in-trail-code*

is ⟨*uncurry2 isasat-replace-annot-in-trail*⟩
:: ⟨*unat-lit-assn*^{k} $*_a$ (*sint64-nat-assn*)^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *isasat-replace-annot-in-trail-def isasat-bounded-assn-def*
trail-pol-fast-assn-def
apply (*annot-snat-const TYPE*(*64*))
apply (*rewrite at list-update - - - annot-index-of-atm*)
by *sepref*

sepref-register *mark-garbage-heur2*
sepref-def *mark-garbage-heur2-code*
is ⟨*uncurry mark-garbage-heur2*⟩
:: ⟨[λ(*C*, *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) $\wedge$ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*]$_a$
sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *mark-garbage-heur2-def isasat-bounded-assn-def*
fold-tuple-optimizations
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

sepref-register *remove-one-annot-true-clause-one-imp-wl-D-heur*
term *mark-garbage-heur2*
sepref-def *remove-one-annot-true-clause-one-imp-wl-D-heur-code*
is ⟨*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*⟩
:: ⟨*sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def*
isasat-trail-nth-st-def[*symmetric*] *get-the-propagation-reason-pol-st-def*[*symmetric*]
fold-tuple-optimizations
apply (*rewrite* **in** ⟨- = ⌑⟩ *snat-const-fold*(*1*)[**where** *'a=64*])
apply (*annot-snat-const TYPE*(*64*))
by *sepref*
sepref-register *mark-clauses-as-unused-wl-D-heur*

sepref-def *access-vdom-at-fast-code*
is ⟨*uncurry* (*RETURN oo access-vdom-at*)⟩
:: ⟨[*uncurry access-vdom-at-pre*]$_a$ *isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow$ *sint64-nat-assn*⟩
unfolding *access-vdom-at-alt-def access-vdom-at-pre-def isasat-bounded-assn-def*
supply [[*goals-limit = 1*]]
by *sepref*

sepref-register *remove-one-annot-true-clause-imp-wl-D-heur*

sepref-def *remove-one-annot-true-clause-imp-wl-D-heur-code*
is ⟨*remove-one-annot-true-clause-imp-wl-D-heur*⟩
:: ⟨*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]]
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-def*
isasat-length-trail-st-def[*symmetric*] *get-pos-of-level-in-trail-imp-st-def*[*symmetric*]
apply (*rewrite at* (⌑, -) *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

lemma *length-ll*[*def-pat-rules*]: ‹*length-ll*$*xs*$*i* ≡ *op-list-list-llen*$*xs*$*i*›
  by (*auto simp*: *length-ll-def*)

lemma [*def-pat-rules*]: ‹*nth-rll* ≡ *op-list-list-idx*›
  by (*auto simp*: *nth-rll-def*[*abs-def*] *op-list-list-idx-def intro*!: *ext*)

sepref-register *length-ll extra-information-mark-to-delete nth-rll*
  LEARNED

lemma *isasat-GC-clauses-prog-copy-wl-entry-alt-def*:
‹*isasat-GC-clauses-prog-copy-wl-entry* = (λ*N0 W A* (*N'*, *vdm*, *avdm*). *do* {
    ASSERT(*nat-of-lit A* < *length W*);
    ASSERT(*length* (*W* ! *nat-of-lit A*) ≤ *length N0*);
    let le = *length* (*W* ! *nat-of-lit A*);
    (*i*, *N*, *N'*, *vdm*, *avdm*) ← *WHILE*$_T$
      (λ(*i*, *N*, *N'*, *vdm*, *avdm*). *i* < *le*)
      (λ(*i*, *N*, (*N'*, *vdm*, *avdm*)). *do* {
        ASSERT(*i* < *length* (*W* ! *nat-of-lit A*));
        let (*C*, -, -) = (*W* ! *nat-of-lit A* ! *i*);
        ASSERT(*arena-is-valid-clause-vdom N C*);
        let st = *arena-status N C*;
        if st ≠ *DELETED then do* {
          ASSERT(*arena-is-valid-clause-idx N C*);
          ASSERT(*length N'* + (*if arena-length N C* > *4 then 5 else 4*) + *arena-length N C* ≤ *length N0*);
          ASSERT(*length N* = *length N0*);
          ASSERT(*length vdm* < *length N0*);
          ASSERT(*length avdm* < *length N0*);
          let D = *length N'* + (*if arena-length N C* > *4 then 5 else 4*);
          N' ← *fm-mv-clause-to-new-arena C N N'*;
          ASSERT(*mark-garbage-pre* (*N*, *C*));
    RETURN (*i+1*, *extra-information-mark-to-delete N C*, *N'*, *vdm* @ [*D*],
            (*if st* = *LEARNED then avdm* @ [*D*] *else avdm*))
        } *else RETURN* (*i+1*, *N*, (*N'*, *vdm*, *avdm*))
      }) (*0*, *N0*, (*N'*, *vdm*, *avdm*));
    RETURN (*N*, (*N'*, *vdm*, *avdm*))
  })›
proof −
  have *H*: ‹(*let* (*a*, -, -) = *c in f a*) = (*let a* = *fst c in f a*)› **for** *a c f*
    by (*cases c*) (*auto simp*: *Let-def*)
  show *?thesis*
    unfolding *isasat-GC-clauses-prog-copy-wl-entry-def H*
    ..
qed

sepref-def *isasat-GC-clauses-prog-copy-wl-entry-code*
  is ‹*uncurry3 isasat-GC-clauses-prog-copy-wl-entry*›
  :: ‹[λ(((*N*, -), -), -). *length N* ≤ *sint64-max*]$_a$
    arena-fast-assn^d $*_a$ *watchlist-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $*_a$
      (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*)^d →
    (*arena-fast-assn* $\times_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*))›
  supply [[*goals-limit=1*]] *if-splits*[*split*] *length-ll-def*[*simp*]
  unfolding *isasat-GC-clauses-prog-copy-wl-entry-alt-def nth-rll-def*[*symmetric*]
    length-ll-def[*symmetric*] *if-conn*(*4*)
  apply (*annot-snat-const TYPE*(*64*))

by *sepref*

sepref-register *isasat-GC-clauses-prog-copy-wl-entry*

lemma *shorten-taken-op-list-list-take*:
‹$W[L := []] = $ *op-list-list-take W L 0*›
by (*auto simp*:)

sepref-def *isasat-GC-clauses-prog-single-wl-code*
is ‹*uncurry3 isasat-GC-clauses-prog-single-wl*›
:: ‹$[\lambda(((N, -), -), A).\ A \leq$ *uint32-max div 2* $\wedge$ *length* $N \leq$ *sint64-max*$]_a$
arena-fast-assn^d $*_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*)^d $*_a$ *watchlist-fast-assn*^{d} $*_a$ *atom-assn*^{k} $\rightarrow$
(*arena-fast-assn* $\times_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*) $\times_a$ *watchlist-fast-assn*)›
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-single-wl-def*
shorten-taken-op-list-list-take
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

definition *isasat-GC-clauses-prog-wl2′* **where**
‹*isasat-GC-clauses-prog-wl2′ ns fst′* = (*isasat-GC-clauses-prog-wl2* (*ns*, *fst′*))›

sepref-register *isasat-GC-clauses-prog-wl2 isasat-GC-clauses-prog-single-wl*
sepref-def *isasat-GC-clauses-prog-wl2-code*
is ‹*uncurry2 isasat-GC-clauses-prog-wl2′*›
:: ‹$[\lambda((-, -), (N, -)).$ *length* $N \leq$ *sint64-max*$]_a$
(*array-assn vmtf-node-assn*)^k $*_a$ (*atom.option-assn*)^k $*_a$
(*arena-fast-assn* $\times_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*) $\times_a$ *watchlist-fast-assn*)^d $\rightarrow$
(*arena-fast-assn* $\times_a$ (*arena-fast-assn* $\times_a$ *vdom-fast-assn* $\times_a$ *vdom-fast-assn*) $\times_a$ *watchlist-fast-assn*)›
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl2-def isasat-GC-clauses-prog-wl2′-def prod.case*
atom.fold-option
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
by *sepref*

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted*›
:: ‹*heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
unfolding *heuristic-assn-def set-zero-wasted-def*
by *sepref*

sepref-register *isasat-GC-clauses-prog-wl isasat-GC-clauses-prog-wl2′ rewatch-heur-st*
sepref-def *isasat-GC-clauses-prog-wl-code*
is ‹*isasat-GC-clauses-prog-wl*›
:: ‹$[\lambda S.$ *length* (*get-clauses-wl-heur S*) $\leq$ *sint64-max*$]_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl-def isasat-bounded-assn-def*
isasat-GC-clauses-prog-wl2′-def[*symmetric*] *vmtf-remove-assn-def*
atom.fold-option fold-tuple-optimizations
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemma *rewatch-heur-st-pre-alt-def*:
⟨*rewatch-heur-st-pre S* ⟷ (∀ *i* ∈ *set* (*get-vdom S*). *i* ≤ *sint64-max*)⟩
by (*auto simp*: *rewatch-heur-st-pre-def all-set-conv-nth*)

sepref-def *rewatch-heur-st-code*
is ⟨*rewatch-heur-st*⟩
:: ⟨[λ*S*. *rewatch-heur-st-pre S* ∧ *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *append-ll-def*[*simp*]
unfolding *isasat-GC-clauses-prog-wl-def isasat-bounded-assn-def*
rewatch-heur-st-def Let-def rewatch-heur-st-pre-alt-def
by *sepref*

sepref-register *isasat-GC-clauses-wl-D*

sepref-def *isasat-GC-clauses-wl-D-code*
is ⟨*isasat-GC-clauses-wl-D*⟩
:: ⟨[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *sint64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*⟩
supply [[*goals-limit=1*]] *isasat-GC-clauses-wl-D-rewatch-pre*[*intro*!]
unfolding *isasat-GC-clauses-wl-D-def*
by *sepref*

sepref-register *number-clss-to-keep*

sepref-register *access-vdom-at*

lemma [*sepref-fr-rules*]:
⟨(*return o id*, *RETURN o unat*) ∈ *word64-assn*^{k} →$_a$ *uint64-nat-assn*⟩
proof −
have [*simp*]: ⟨(λ*s*. ∃*xa*. (↑(*xa* = *unat x*) ∧* ↑(*xa* = *unat x*)) *s*) = ↑*True*⟩
by (*intro ext*)
(*auto intro*!: *exI*[*of* - ⟨*unat x*⟩] *simp*: *pure-true-conv pure-part-pure-eq pred-lift-def*
simp flip: *import-param-3*)
show *?thesis*
apply *sepref-to-hoare*
apply (*vcg*)
apply (*auto simp*: *unat-rel-def unat.rel-def br-def pred-lift-def ENTAILS-def pure-true-conv simp flip*: *import-param-3 pure-part-def*)
done
qed

sepref-def *number-clss-to-keep-fast-code*
is ⟨*number-clss-to-keep-impl*⟩
:: ⟨*isasat-bounded-assn*^{k} →$_a$ *sint64-nat-assn*⟩
supply [[*goals-limit = 1*]]
unfolding *number-clss-to-keep-impl-def isasat-bounded-assn-def*
fold-tuple-optimizations
apply (*rewrite at If* - - ⌑ *annot-unat-snat-conv*)
apply (*rewrite at If* (⌑ ≤-) *annot-snat-unat-conv*)
by *sepref*

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
⟨(*number-clss-to-keep-impl*, *number-clss-to-keep*) ∈ *Sepref-Rules.freft Id* (λ-. ⟨*nat-rel*⟩*nres-rel*)⟩
by (*auto simp*: *number-clss-to-keep-impl-def number-clss-to-keep-def Let-def intro*!: *Sepref-Rules.frefI nres-relI*)

lemma *number-clss-to-keep-fast-code-refine*[*sepref-fr-rules*]:
‹(*number-clss-to-keep-fast-code*, *number-clss-to-keep*) ∈ (*isasat-bounded-assn*)^k $\rightarrow_a$ *snat-assn*›
using *hfcomp*[*OF number-clss-to-keep-fast-code.refine*
number-clss-to-keep-impl-number-clss-to-keep, *simplified*]
by *auto*

sepref-def *mark-clauses-as-unused-wl-D-heur-fast-code*
is ‹*uncurry mark-clauses-as-unused-wl-D-heur*›
:: ‹[λ(-, *S*). *length* (*get-avdom S*) ≤ *sint64-max*]$_a$
sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *length-avdom-def*[*simp*]
unfolding *mark-clauses-as-unused-wl-D-heur-def*
mark-unused-st-heur-def[*symmetric*]
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

experiment
begin
export-llvm *restart-required-heur-fast-code*
access-vdom-at-fast-code
isasat-GC-clauses-wl-D-code
end

end
theory *IsaSAT-Restart*
imports *IsaSAT-Restart-Heuristics IsaSAT-CDCL*
begin

Chapter 20

Full CDCL with Restarts

definition *cdcl-twl-stgy-restart-abs-wl-heur-inv* **where**
‹*cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 *brk* T $n \longleftrightarrow$
  $(\exists\, S_0{}'\ T'.\ (S_0,\ S_0{}') \in$ *twl-st-heur* $\wedge\ (T,\ T') \in$ *twl-st-heur* $\wedge$
    cdcl-twl-stgy-restart-abs-wl-inv $S_0{}'$ *brk* T' $n)$›

definition *cdcl-twl-stgy-restart-prog-wl-heur*
  :: *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*
where
‹*cdcl-twl-stgy-restart-prog-wl-heur* S_0 = *do* {
  (*brk*, T, -) ← $WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ brk\ T\ n}$
    (λ(*brk*, -). ¬*brk*)
    (λ(*brk*, *S*, *n*).
    do {
      T ← *unit-propagation-outer-loop-wl-D-heur S*;
      (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
      (*T*, *n*) ← *restart-prog-wl-D-heur T n brk*;
      RETURN (*brk*, *T*, *n*)
    })
    (*False*, S_0::*twl-st-wl-heur*, *0*);
  RETURN T
}›

lemma *cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D*:
‹(*cdcl-twl-stgy-restart-prog-wl-heur*, *cdcl-twl-stgy-restart-prog-wl*) ∈
  twl-st-heur $\rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
proof −
  show *?thesis*
    unfolding *cdcl-twl-stgy-restart-prog-wl-heur-def cdcl-twl-stgy-restart-prog-wl-def*
    apply (*intro frefI nres-relI*)
    apply (*refine-rcg*
        restart-prog-wl-D-heur-restart-prog-wl-D2[*THEN fref-to-Down-curry2*]
        cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[*THEN fref-to-Down*]
        cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D[*THEN fref-to-Down*]
        unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D[*THEN fref-to-Down*]
        WHILEIT-refine[**where** *R* = ‹*bool-rel* $\times_r$ *twl-st-heur* $\times_r$ *nat-rel*›])
    subgoal by *auto*
    subgoal unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fastforce*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *fast-number-of-iterations* :: ‹- ⇒ *bool*› **where**
‹*fast-number-of-iterations n* ⟷ *n* < *uint64-max* >> *1*›

definition *isasat-fast-slow* :: ‹*twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*› **where**
[*simp*]: ‹*isasat-fast-slow S* = *RETURN S*›

definition *cdcl-twl-stgy-restart-prog-early-wl-heur*
:: *twl-st-wl-heur* ⇒ *twl-st-wl-heur nres*
where
‹*cdcl-twl-stgy-restart-prog-early-wl-heur* S_0 = *do* {
  ebrk ← *RETURN* (¬*isasat-fast* S_0);
  (*ebrk*, *brk*, *T*, *n*) ←
  $WHILE_T$λ(*ebrk*, *brk*, *T*, *n*). *cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 *brk T n* ∧ (¬*ebrk* ⟶*isasat-fast T*) ∧ *length* (*get-c*
    (λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
    (λ(*ebrk*, *brk*, *S*, *n*).
    do {
      ASSERT(¬*brk* ∧ ¬*ebrk*);
      ASSERT(*length* (*get-clauses-wl-heur S*) ≤ *uint64-max*);
      T ← *unit-propagation-outer-loop-wl-D-heur S*;
      ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *uint64-max*);
      ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*));
      (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
      ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *uint64-max*);
      (*T*, *n*) ← *restart-prog-wl-D-heur T n brk*;
ebrk ← *RETURN* (¬*isasat-fast T*);
      RETURN (*ebrk*, *brk*, *T*, *n*)
    })
    (*ebrk*, *False*, S_0::*twl-st-wl-heur*, *0*);
  ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *uint64-max* ∧
      get-old-arena T = []);
  if ¬*brk then do* {
    T ← *isasat-fast-slow T*;
    (*brk*, *T*, -) ← $WHILE_T$λ(*brk*, *T*, *n*). *cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 *brk T n*
      (λ(*brk*, -). ¬*brk*)
      (λ(*brk*, *S*, *n*).
      do {
        T ← *unit-propagation-outer-loop-wl-D-heur S*;
        (*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
        (*T*, *n*) ← *restart-prog-wl-D-heur T n brk*;
        RETURN (*brk*, *T*, *n*)
      })
      (*False*, *T*, *n*);
    RETURN T
  }
  else isasat-fast-slow T
}›

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:

assumes r: ‹$r \leq$ *uint64-max*›
shows ‹(*cdcl-twl-stgy-restart-prog-early-wl-heur*, *cdcl-twl-stgy-restart-prog-early-wl*) $\in$
twl-st-heur''' $r \rightarrow_f$ ⟨*twl-st-heur*⟩*nres-rel*›
proof −
have *cdcl-twl-stgy-restart-prog-early-wl-alt-def*:
‹*cdcl-twl-stgy-restart-prog-early-wl* S_0 = *do* {
ebrk ← *RES UNIV*;
(*ebrk*, *brk*, *T*, *n*) ← $WHILE_T{}^{\lambda(-,\ brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}inv\ S_0\ brk\ T\ n}$
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(-, *brk*, *S*, *n*).
do {
T ← *unit-propagation-outer-loop-wl S*;
(*brk*, *T*) ← *cdcl-twl-o-prog-wl T*;
(*T*, *n*) ← *restart-prog-wl T n brk*;
ebrk ← *RES UNIV*;
RETURN (*ebrk*, *brk*, *T*, *n*)
})
(*ebrk*, *False*, S_0::*nat twl-st-wl*, *0*);
if ¬*brk then do* {
T ← *RETURN T*;
(*brk*, *T*, -) ← $WHILE_T{}^{\lambda(brk,\ T,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}inv\ S_0\ brk\ T\ n}$
(λ(*brk*, -). ¬*brk*)
(λ(*brk*, *S*, *n*).
do {
T ← *unit-propagation-outer-loop-wl S*;
(*brk*, *T*) ← *cdcl-twl-o-prog-wl T*;
(*T*, *n*) ← *restart-prog-wl T n brk*;
RETURN (*brk*, *T*, *n*)
})
(*False*, *T*::*nat twl-st-wl*, *n*);
RETURN T
}
else RETURN T
}› **for** S_0
unfolding *cdcl-twl-stgy-restart-prog-early-wl-def nres-monad1* **by** *auto*
have [*refine0*]: ‹*RETURN* (¬*isasat-fast x*) ≤ ⇓
{(*b*, *b'*). *b* = *b'* ∧ (*b* = (¬*isasat-fast x*))} (*RES UNIV*)›
for *x*
by (*auto intro*: *RETURN-RES-refine*)
have [*refine0*]: ‹*isasat-fast-slow x1e*
≤ ⇓ {(*S*, *S'*). *S* = *x1e* ∧ *S'* = *x1b*}
(*RETURN x1b*)›
for *x1e x1b*
proof −
show *?thesis*
unfolding *isasat-fast-slow-def* **by** *auto*
qed
have *twl-st-heur''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur* ⟹
(*x1e*, *x1b*)
∈ *twl-st-heur''*
(*dom-m* (*get-clauses-wl x1b*))
(*length* (*get-clauses-wl-heur x1e*))›
for *x1e x1b*
by (*auto simp*: *twl-st-heur'-def*)
have *twl-st-heur'''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur''* $\mathcal{D}$ *r* ⟹

(x1e, x1b)
∈ *twl-st-heur''' r*›
for *x1e x1b r* $\mathcal{D}$
by (*auto simp*: *twl-st-heur'-def*)
have *H*: ‹(*xb*, *x'a*)
∈ *bool-rel* $\times_f$
twl-st-heur'''' (*length* (*get-clauses-wl-heur x1e*) + *6* + *uint32-max div 2*) ⟹
x'a = (*x1f*, *x2f*) ⟹
xb = (*x1g*, *x2g*) ⟹
(*x1g*, *x1f*) ∈ *bool-rel* ⟹
(*x2e*, *x2b*) ∈ *nat-rel* ⟹
(((*x2g*, *x2e*), *x1g*), (*x2f*, *x2b*), *x1f*)
∈ *twl-st-heur'''* (*length* (*get-clauses-wl-heur x2g*)) $\times_f$
nat-rel $\times_f$
bool-rel› **for** *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb x'a x1f x2f x1g x2g*
by *auto*
have *abs-inv*: ‹(*x*, *y*) ∈ *twl-st-heur''' r* ⟹
(*ebrk*, *ebrka*) ∈ {(*b*, *b'*). *b* = *b'* ∧ *b* = (¬ *isasat-fast x*)} ⟹
(*xb*, *x'a*) ∈ *bool-rel* $\times_f$ (*twl-st-heur* $\times_f$ *nat-rel*) ⟹
case x'a of
(*brk*, *xa*, *xb*) ⇒
cdcl-twl-stgy-restart-abs-wl-inv y brk xa xb ⟹
x2f = (*x1g*, *x2g*) ⟹
xb = (*x1f*, *x2f*) ⟹
cdcl-twl-stgy-restart-abs-wl-heur-inv x x1f x1g x2g›
for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb x'a x1f x2f x1g x2g*
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fastforce*

show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*]
unfolding *cdcl-twl-stgy-restart-prog-early-wl-heur-def*
cdcl-twl-stgy-restart-prog-early-wl-alt-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry2*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
WHILEIT-refine[**where** *R* = ‹*bool-rel* $\times_r$ *twl-st-heur* $\times_r$ *nat-rel*›]
WHILEIT-refine[**where** *R* = ‹{((*ebrk*, *brk*, *T*,*n*), (*ebrk'*, *brk'*, *T'*, *n'*)).
(*ebrk* = *ebrk'*) ∧ (*brk* = *brk'*) ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧ *n* = *n'* ∧
(¬*ebrk* ⟶ *isasat-fast T*) ∧ *length* (*get-clauses-wl-heur T*) ≤ *uint64-max*}›])
subgoal using *r* **by** *auto*
subgoal
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur'''*; *assumption*)

subgoal by (*auto simp*: *isasat-fast-def uint64-max-def sint64-max-def uint32-max-def*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*subst* (*asm*)(*2*) *twl-st-heur-def*) *force*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule abs-inv*)
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
apply (*rule twl-st-heur'''*; *assumption*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *isasat-fast-slow-def*)
done
qed

lemma *mark-unused-st-heur*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-unused-st-heur-def*
all-init-atms-def[*symmetric*])
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused valid-arena-arena-decr-act*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹*D* = *Some C* ⟷ *D* ≠ *None* ∧ ((*the D*) = *C*)›
by *auto*

lemma (**in** −) *isasat-fast-alt-def*:
‹*RETURN o isasat-fast* = (λ(*M*, *N*, -). *RETURN* (*length N* ≤ *sint64-max* − (*uint32-max div 2* + *6*)))›
unfolding *isasat-fast-def*
by (*auto intro*!:*ext*)

definition *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
:: *twl-st-wl-heur* ⇒ (*bool* × *twl-st-wl-heur*) *nres*
where
‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur* S_0 = *do* {
ebrk ← *RETURN* (¬*isasat-fast* S_0);
(*ebrk*, *brk*, *T*, *n*) ←

```
  WHILE_T^λ(ebrk, brk, T, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S0 brk T n ∧        (¬ebrk ⟶isasat-fast T ∧ n < uint64-m
    (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
    (λ(ebrk, brk, S, n).
    do {
      ASSERT(¬brk ∧ ¬ebrk);
      ASSERT(length (get-clauses-wl-heur S) ≤ sint64-max);
      T ← unit-propagation-outer-loop-wl-D-heur S;
      ASSERT(length (get-clauses-wl-heur T) ≤ sint64-max);
      ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S));
      (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
      ASSERT(length (get-clauses-wl-heur T) ≤ sint64-max);
      (T, n) ← restart-prog-wl-D-heur T n brk;
ebrk ← RETURN (¬(isasat-fast T ∧ n < uint64-max));
      RETURN (ebrk, brk, T, n)
    })
    (ebrk, False, S0::twl-st-wl-heur, 0);
  RETURN (brk, T)
}›
```

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D*:

```
  assumes r: ‹r ≤ uint64-max›
  shows ‹(cdcl-twl-stgy-restart-prog-bounded-wl-heur, cdcl-twl-stgy-restart-prog-bounded-wl) ∈
   twl-st-heur''' r →_f ⟨bool-rel ×_r twl-st-heur⟩nres-rel›
proof –
  have cdcl-twl-stgy-restart-prog-bounded-wl-alt-def:
  ‹cdcl-twl-stgy-restart-prog-bounded-wl S0 = do {
    ebrk ← RES UNIV;
    (ebrk, brk, T, n) ← WHILE_T^λ(-, brk, T, n). cdcl-twl-stgy-restart-abs-wl-inv S0 brk T n
      (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
      (λ(-, brk, S, n).
      do {
        T ← unit-propagation-outer-loop-wl S;
        (brk, T) ← cdcl-twl-o-prog-wl T;
        (T, n) ← restart-prog-wl T n brk;
        ebrk ← RES UNIV;
        RETURN (ebrk, brk, T, n)
      })
      (ebrk, False, S0::nat twl-st-wl, 0);
    RETURN (brk, T)
  }› for S0
    unfolding cdcl-twl-stgy-restart-prog-bounded-wl-def nres-monad1 by auto
  have [refine0]: ‹RETURN (¬(isasat-fast x ∧ n < uint64-max)) ≤ ⇓
    {(b, b'). b = b' ∧ (b = (¬(isasat-fast x ∧ n < uint64-max)))} (RES UNIV)›
     ‹RETURN (¬isasat-fast x) ≤ ⇓
    {(b, b'). b = b' ∧ (b = (¬(isasat-fast x ∧ 0 < uint64-max)))} (RES UNIV)›
    for x n
    by (auto intro: RETURN-RES-refine simp: uint64-max-def)
  have [refine0]: ‹isasat-fast-slow x1e
    ≤ ⇓ {(S, S'). S = x1e ∧ S' = x1b}
   (RETURN x1b)›
    for x1e x1b
  proof –
    show ?thesis
      unfolding isasat-fast-slow-def by auto
  qed
```

have *twl-st-heur''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur* ⟹
  (*x1e*, *x1b*)
  ∈ *twl-st-heur''*
      (*dom-m* (*get-clauses-wl x1b*))
      (*length* (*get-clauses-wl-heur x1e*))›
  for *x1e x1b*
  by (*auto simp*: *twl-st-heur'-def*)
have *twl-st-heur'''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur''* 𝒟 *r* ⟹
  (*x1e*, *x1b*)
  ∈ *twl-st-heur''' r*›
  for *x1e x1b r* 𝒟
  by (*auto simp*: *twl-st-heur'-def*)
have *H*: ‹(*xb*, *x'a*)
  ∈ *bool-rel* $\times_f$
    twl-st-heur'''' (*length* (*get-clauses-wl-heur x1e*) + *6* + *uint32-max div 2*) ⟹
  x'a = (*x1f*, *x2f*) ⟹
  xb = (*x1g*, *x2g*) ⟹
  (*x1g*, *x1f*) ∈ *bool-rel* ⟹
  (*x2e*, *x2b*) ∈ *nat-rel* ⟹
  (((*x2g*, *x2e*), *x1g*), (*x2f*, *x2b*), *x1f*)
  ∈ *twl-st-heur'''* (*length* (*get-clauses-wl-heur x2g*)) $\times_f$
    nat-rel $\times_f$
    bool-rel› **for** *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb*
     x'a x1f x2f x1g x2g
  by *auto*
have *abs-inv*: ‹(*x*, *y*) ∈ *twl-st-heur''' r* ⟹
  (*ebrk*, *ebrka*) ∈ {(*b*, *b'*). *b* = *b'* ∧ *b* = (¬ *isasat-fast x* ∧ *x2g* < *uint64-max*)} ⟹
  (*xb*, *x'a*) ∈ *bool-rel* $\times_f$ (*twl-st-heur* $\times_f$ *nat-rel*) ⟹
  case x'a of
  (*brk*, *xa*, *xb*) ⇒
    cdcl-twl-stgy-restart-abs-wl-inv y brk xa xb ⟹
  x2f = (*x1g*, *x2g*) ⟹
  xb = (*x1f*, *x2f*) ⟹
  cdcl-twl-stgy-restart-abs-wl-heur-inv x x1f x1g x2g›
 for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
     x1e x2e T Ta xb x'a x1f x2f x1g x2g
  unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def*
  apply (*rule-tac x=y* **in** *exI*)
  by *fastforce*
show *?thesis*
  supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*]
  unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def*
    cdcl-twl-stgy-restart-prog-bounded-wl-alt-def
  apply (*intro frefI nres-relI*)
  apply (*refine-rcg*
      restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry2*]
      cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
      unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
      WHILEIT-refine[**where** *R* = ‹{((*ebrk*, *brk*, *T*,*n*), (*ebrk'*, *brk'*, *T'*, *n'*)).
  (*ebrk* = *ebrk'*) ∧ (*brk* = *brk'*) ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧ *n* = *n'* ∧
    (¬*ebrk* ⟶ *isasat-fast T* ∧ *n* < *uint64-max*) ∧
          (¬*ebrk* ⟶ *length* (*get-clauses-wl-heur T*) ≤ *sint64-max*)}›])
  subgoal using *r* **by** (*auto simp*: *sint64-max-def isasat-fast-def uint32-max-def*)
  subgoal
    unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def* **by** *fast*
  subgoal by *auto*

subgoal by *auto*
subgoal by (*auto simp*: *sint64-max-def isasat-fast-def uint32-max-def*)
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur'''*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def uint64-max-def uint32-max-def sint64-max-def*)
apply (*rule H*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

end
theory *IsaSAT-Restart-LLVM*
imports *IsaSAT-Restart IsaSAT-Restart-Heuristics-LLVM IsaSAT-CDCL-LLVM*
begin

sepref-register *mark-to-delete-clauses-wl-D-heur*

sepref-def *MINIMUM-DELETION-LBD-impl*
is ⟨*uncurry0* (*RETURN MINIMUM-DELETION-LBD*)⟩
:: ⟨$unit\text{-}assn^k \rightarrow_a uint32\text{-}nat\text{-}assn$⟩
unfolding *MINIMUM-DELETION-LBD-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-register *delete-index-and-swap mop-mark-garbage-heur*

sepref-def *mark-to-delete-clauses-wl-D-heur-fast-impl*
is ⟨*mark-to-delete-clauses-wl-D-heur*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
unfolding *mark-to-delete-clauses-wl-D-heur-def*
access-vdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
get-the-propagation-reason-heur-def[*symmetric*]
clause-is-learned-heur-def[*symmetric*]
clause-lbd-heur-def[*symmetric*]
access-length-heur-def[*symmetric*]
short-circuit-conv mark-to-delete-clauses-wl-D-heur-is-Some-iff
marked-as-used-st-def[*symmetric*] *if-conn*(*4*)
fold-tuple-optimizations
mop-arena-lbd-st-def[*symmetric*]
mop-marked-as-used-st-def[*symmetric*]
mop-arena-status-st-def[*symmetric*]
mop-arena-length-st-def[*symmetric*]
supply [[*goals-limit* = *1*]] *of-nat-snat*[*sepref-import-param*]
length-avdom-def[*symmetric*, *simp*] *access-vdom-at-def*[*simp*]
apply (*annot-snat-const TYPE*(*64*))

by *sepref*

sepref-register *cdcl-twl-full-restart-wl-prog-heur*

sepref-def *cdcl-twl-full-restart-wl-prog-heur-fast-code*
is ⟨*cdcl-twl-full-restart-wl-prog-heur*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
unfolding *cdcl-twl-full-restart-wl-prog-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *cdcl-twl-restart-wl-heur-fast-code*
is ⟨*cdcl-twl-restart-wl-heur*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
unfolding *cdcl-twl-restart-wl-heur-def*
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code*
is ⟨*cdcl-twl-full-restart-wl-D-GC-heur-prog*⟩
:: ⟨$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \rightarrow isasat\text{-}bounded\text{-}assn$⟩
supply [[*goals-limit* = *1*]]
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-def*
apply (*annot-unat-const TYPE*(*32*))
by *sepref*

sepref-register *restart-required-heur cdcl-twl-restart-wl-heur*

sepref-def *restart-prog-wl-D-heur-fast-code*
is ⟨*uncurry2* (*restart-prog-wl-D-heur*)⟩
:: ⟨$[\lambda((S,\ n),\ \text{-}).\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq sint64\text{-}max \wedge n < uint64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k *_a bool1\text{-}assn^k \rightarrow isasat\text{-}bounded\text{-}assn \times_a uint64\text{-}nat\text{-}assn$⟩
unfolding *restart-prog-wl-D-heur-def*
supply [[*goals-limit* = *1*]]
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

definition *isasat-fast-bound* **where**
⟨*isasat-fast-bound* = *uint64-max* − (*uint32-max div 2* + *6*)⟩

lemma *isasat-fast-bound-alt-def*:
⟨*isasat-fast-bound* = *18446744071562067962*⟩
by (*auto simp*: *br-def isasat-fast-bound-def*
uint64-max-def uint32-max-def)

sepref-register *isasat-fast*
sepref-def *isasat-fast-code*
is ⟨*RETURN o isasat-fast*⟩
:: ⟨$isasat\text{-}bounded\text{-}assn^k \rightarrow_a bool1\text{-}assn$⟩
unfolding *isasat-fast-alt-def isasat-fast-bound-def*[*symmetric*]
isasat-fast-bound-alt-def
supply [[*goals-limit* = *1*]]
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

sepref-register *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
sepref-def *cdcl-twl-stgy-restart-prog-wl-heur-fast-code*
is ‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur*›
:: ‹$[\lambda S.\ \textit{isasat-fast}\ S]_a\ \textit{isasat-bounded-assn}^d \rightarrow \textit{bool1-assn} \times_a \textit{isasat-bounded-assn}$›
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def*
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
apply (*annot-unat-const TYPE*(*64*))
by *sepref*

experiment
begin
export-llvm *opts-reduction-st-fast-code*
opts-restart-st-fast-code
get-conflict-count-since-last-restart-heur-fast-code
get-fast-ema-heur-fast-code
get-slow-ema-heur-fast-code
get-learned-count-fast-code
count-decided-st-heur-pol-fast
upper-restart-bound-not-reached-fast-impl
minimum-number-between-restarts-impl
restart-required-heur-fast-code
cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code
cdcl-twl-restart-wl-heur-fast-code
cdcl-twl-full-restart-wl-prog-heur-fast-code
cdcl-twl-local-restart-wl-D-heur-fast-code

end

end
theory *IsaSAT*
imports *IsaSAT-Restart IsaSAT-Initialisation*
begin

Chapter 21

Full IsaSAT

We now combine all the previous definitions to prove correctness of the complete SAT solver:

1. We initialise the arena part of the state;
2. Then depending on the options and the number of clauses, we either use the bounded version or the unbounded version. Once have if decided which one, we initiale the watch lists;
3. After that, we can run the CDCL part of the SAT solver;
4. Finally, we extract the trail from the state.

   Remark that the statistics and the options are unchecked: the number of propagations might overflows (but they do not impact the correctness of the whole solver). Similar restriction applies on the options: setting the options might not do what you expect to happen, but the result will still be correct.

21.1 Correctness Relation

We cannot use *cdcl-twl-stgy-restart* since we do not always end in a final state for *cdcl-twl-stgy*.

definition *conclusive-TWL-run* :: ‹*'v twl-st* ⇒ *'v twl-st nres*› **where**
‹*conclusive-TWL-run S* =
SPEC(λT. $\exists\, n\; n'$. *cdcl-twl-stgy-restart-with-leftovers*** (S, n) (T, n') $\wedge$ *final-twl-state* T)›

definition *conclusive-TWL-run-bounded* :: ‹*'v twl-st* ⇒ (*bool* × *'v twl-st*) *nres*› **where**
‹*conclusive-TWL-run-bounded S* =
SPEC(λ(*brk*, T). $\exists\, n\; n'$. *cdcl-twl-stgy-restart-with-leftovers*** (S, n) (T, n') $\wedge$
(*brk* $\longrightarrow$ *final-twl-state* T))›

To get a full CDCL run:

- either we fully apply *cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy* (up to restarts)
- or we can stop early.

definition *conclusive-CDCL-run* **where**
‹*conclusive-CDCL-run CS T U* $\longleftrightarrow$
($\exists\, n\; n'$. *cdcl*$_W$*-restart-mset.cdcl*$_W$*-restart-stgy*** (T, n) (U, n') $\wedge$

no-step $cdcl_W$*-restart-mset.*$cdcl_W$ (U)) ∨
($CS \neq \{\#\}$ ∧ *conflicting* $U \neq$ *None* ∧ *count-decided* (*trail* U) = *0* ∧
unsatisfiable (*set-mset* CS))⟩

lemma *cdcl-twl-stgy-restart-restart-prog-spec*: ⟨*twl-struct-invs* S ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog S ≤ *conclusive-TWL-run* S⟩
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-restart-prog-spec*; *assumption?*)
unfolding *conclusive-TWL-run-def twl-restart-def*
by *auto*

lemma *cdcl-twl-stgy-restart-prog-bounded-spec*: ⟨*twl-struct-invs* S ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog-bounded S ≤ *conclusive-TWL-run-bounded* S⟩
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-prog-bounded-spec*; *assumption?*)
unfolding *conclusive-TWL-run-bounded-def twl-restart-def*
by *auto*

lemma *cdcl-twl-stgy-restart-restart-prog-early-spec*: ⟨*twl-struct-invs* S ⟹
twl-stgy-invs S ⟹
clauses-to-update S = {#} ⟹
get-conflict S = *None* ⟹
cdcl-twl-stgy-restart-prog-early S ≤ *conclusive-TWL-run* S⟩
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-prog-early-spec*; *assumption?*)
unfolding *conclusive-TWL-run-def twl-restart-def*
by *auto*

lemma $cdcl_W$*-ex-*$cdcl_W$*-stgy*:
⟨$cdcl_W$*-restart-mset.*$cdcl_W$ S T ⟹ ∃ U. $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* S U⟩
by (*meson* $cdcl_W$*-restart-mset.*$cdcl_W$*.cases* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.simps*)

lemma *rtranclp-*$cdcl_W$*-*$cdcl_W$*-init-state*:
⟨$cdcl_W$*-restart-mset.*$cdcl_W^{**}$ (*init-state* {#}) S ⟷ S = *init-state* {#}⟩
unfolding *rtranclp-unfold*
by (*subst tranclp-unfold-begin*)
(*auto simp*: $cdcl_W$*-stgy-*$cdcl_W$*-init-state-empty-no-step*
$cdcl_W$*-stgy-*$cdcl_W$*-init-state*
simp del: *init-state.simps*
dest: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-*$cdcl_W$ $cdcl_W$*-ex-*$cdcl_W$*-stgy*)

definition *init-state-l* :: ⟨*'v twl-st-l-init*⟩ **where**
⟨*init-state-l* = (([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})⟩

definition *to-init-state-l* :: ⟨*nat twl-st-l-init* ⇒ *nat twl-st-l-init*⟩ **where**
⟨*to-init-state-l* S = S⟩

definition *init-state0* :: ⟨*'v twl-st-init*⟩ **where**

‹*init-state0* = (([], {#}, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›

definition *to-init-state0* :: ‹*nat twl-st-init* ⇒ *nat twl-st-init*› **where**
‹*to-init-state0 S* = *S*›

lemma *init-dt-pre-init*:
assumes *dist*: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
using *dist* **apply** −
unfolding *init-dt-pre-def to-init-state-l-def init-state-l-def*
by (*rule exI*[*of* - ‹(([], {#}, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›])
(*auto simp*: *twl-st-l-init-def twl-init-invs*)

This is the specification of the SAT solver:

definition *SAT* :: ‹*nat clauses* ⇒ *nat* $cdcl_W$*-restart-mset nres*› **where**
‹*SAT CS* = *do*{
let T = *init-state CS*;
SPEC (*conclusive-CDCL-run CS T*)
}›

definition *init-dt-spec0* :: ‹′*v clause-l list* ⇒ ′*v twl-st-init* ⇒ ′*v twl-st-init* ⇒ *bool*› **where**
‹*init-dt-spec0 CS SOC T*′ ⟷
(
twl-struct-invs-init T′ ∧
clauses-to-update-init T′ = {#} ∧
(∀ *s*∈*set* (*get-trail-init T*′). ¬*is-decided s*) ∧
(*get-conflict-init T*′ = *None* ⟶
literals-to-update-init T′ = *uminus* ‘# *lit-of* ‘# *mset* (*get-trail-init T*′)) ∧
(*mset* ‘# *mset CS* + *clause* ‘# (*get-init-clauses-init SOC*) + *other-clauses-init SOC* +
get-unit-init-clauses-init SOC + *get-subsumed-init-clauses-init SOC* =
clause ‘# (*get-init-clauses-init T*′) + *other-clauses-init T*′ +
get-unit-init-clauses-init T′ + *get-subsumed-init-clauses-init T*′) ∧
get-learned-clauses-init SOC = *get-learned-clauses-init T*′ ∧
get-subsumed-learned-clauses-init SOC = *get-subsumed-learned-clauses-init T*′ ∧
get-unit-learned-clauses-init T′ = *get-unit-learned-clauses-init SOC* ∧
twl-stgy-invs (*fst T*′) ∧
(*other-clauses-init T*′ ≠ {#} ⟶ *get-conflict-init T*′ ≠ *None*) ∧
({#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T*′ ≠ *None*) ∧
(*get-conflict-init SOC* ≠ *None* ⟶ *get-conflict-init SOC* = *get-conflict-init T*′))›

21.2 Refinements of the Whole SAT Solver

We do not add the refinement steps in separate files, since the form is very specific to the SAT solver we want to generate (and needs to be updated if it changes).

definition *SAT0* :: ‹*nat clause-l list* ⇒ *nat twl-st nres*› **where**
‹*SAT0 CS* = *do*{
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state0*;
T ← *SPEC* (*init-dt-spec0 CS* (*to-init-state0 S*));
let T = *fst T*;
if get-conflict T ≠ *None*
then RETURN T

else if CS = [] *then RETURN* (*fst init-state0*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update T* = {#});
ASSERT(*clause* '# (*get-clauses T*) + *unit-clss T* + *subsumed-clauses T* = *mset* '# *mset CS*);
ASSERT(*get-learned-clss T* = {#});
ASSERT(*subsumed-learned-clss T* = {#});
cdcl-twl-stgy-restart-prog T
}
}
else do {
let S = *init-state0*;
T ← *SPEC* (*init-dt-spec0 CS* (*to-init-state0 S*));
failed ← *SPEC* (λ- :: *bool*. *True*);
if failed then do {
T ← *SPEC* (*init-dt-spec0 CS* (*to-init-state0 S*));
let T = *fst T*;
if get-conflict T ≠ *None*
then RETURN T
else if CS = [] *then RETURN* (*fst init-state0*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update T* = {#});
ASSERT(*clause* '# (*get-clauses T*) + *unit-clss T* + *subsumed-clauses T* = *mset* '# *mset CS*);
ASSERT(*get-learned-clss T* = {#});
cdcl-twl-stgy-restart-prog T
}
} *else do* {
let T = *fst T*;
if get-conflict T ≠ *None*
then RETURN T
else if CS = [] *then RETURN* (*fst init-state0*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update T* = {#});
ASSERT(*clause* '# (*get-clauses T*) + *unit-clss T* + *subsumed-clauses T* = *mset* '# *mset CS*);
ASSERT(*get-learned-clss T* = {#});
cdcl-twl-stgy-restart-prog-early T
}
}
}
})

lemma *SAT0-SAT*:
assumes ‹*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*›
shows ‹*SAT0 CS* ≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*} (*SAT* (*mset* '# *mset CS*))›
proof −
have *conflict-during-init*: ‹*RETURN* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*}
(*SPEC* (*conclusive-CDCL-run* (*mset* '# *mset CS*)
(*init-state* (*mset* '# *mset CS*))))›
if
spec: ‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
confl: ‹*get-conflict* (*fst T*) ≠ *None*›
for *T*
proof −

let *?CS* = ‹*mset* '# *mset CS*›
have
struct-invs: ‹*twl-struct-invs-init T*› **and**
‹*clauses-to-update-init T* = {#}› **and**
count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus '# *lit-of* '# *mset* (*get-trail-init T*)› **and**
clss: ‹*mset* '# *mset CS* +
clause '# *get-init-clauses-init* (*to-init-state0 init-state0*) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) +
get-subsumed-init-clauses-init (*to-init-state0 init-state0*) =
clause '# *get-init-clauses-init T* + *other-clauses-init T* +
get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T*› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›
‹*get-unit-learned-clauses-init T* =
get-unit-learned-clauses-init (*to-init-state0 init-state0*)›
‹*get-subsumed-learned-clauses-init T* =
get-subsumed-learned-clauses-init (*to-init-state0 init-state0*)› **and**
‹*twl-stgy-invs* (*fst T*)› **and**
‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
‹{#} ∈# *mset* '# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
using *spec* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
Set.mem-Collect-eq **apply** −
apply *normalize-goal*+
by *metis*+

have *count-dec*: ‹*count-decided* (*get-trail* (*fst T*)) = *0*›
using *count-dec* **unfolding** *count-decided-0-iff* **by** (*auto simp*: *twl-st-init*
twl-st-wl-init)

have *le*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* ($state_W$*-of-init T*)› **and**
all-struct-invs:
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of-init T*)›
using *struct-invs* **unfolding** *twl-struct-invs-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast*+
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting* ($state_W$*-of-init T*)›
using *struct-invs* **unfolding** *twl-struct-invs-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast*
have ‹*unsatisfiable* (*set-mset* (*mset* '# *mset* (*rev CS*)))›
using *conflict-of-level-unsatisfiable*[*OF all-struct-invs*] *count-dec confl*
learned le clss
by (*auto simp*: *clauses-def mset-take-mset-drop-mset′ twl-st-init twl-st-wl-init*
image-image to-init-state0-def init-state0-def ac-simps
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def ac-simps*
twl-st-l-init)
then have *unsat*[*simp*]: ‹*unsatisfiable* (*mset* ' *set CS*)›
by *auto*
then have [*simp*]: ‹*CS* ≠ []›
by (*auto simp del*: *unsat*)

show *?thesis*
unfolding *conclusive-CDCL-run-def*
apply (*rule RETURN-SPEC-refine*)
apply (*rule exI*[*of* - ‹$state_W$*-of* (*fst T*)›])
apply (*intro conjI*)
subgoal
by *auto*
subgoal
apply (*rule disjI2*)
using *struct-invs learned count-dec clss confl*
by (*clarsimp simp*: *twl-st-init twl-st-wl-init twl-st-l-init*)
done
qed

have *empty-clauses*: ‹*RETURN* (*fst init-state0*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*}
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
(*init-state* (*mset* ‘# *mset CS*))))›
if
‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
‹*CS* = []›
for *T*
proof −
have [*dest*]: ‹$cdcl_W$*-restart-mset.*$cdcl_W$ ([], {#}, {#}, *None*) (*a*, *aa*, *ab*, *b*) ⟹ *False*›
for *a aa ab b*
by (*metis* $cdcl_W$*-restart-mset.*$cdcl_W$*.cases* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.conflict′*
$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.propagate′* $cdcl_W$*-restart-mset.other′*
$cdcl_W$*-stgy-*$cdcl_W$*-init-state-empty-no-step init-state.simps*)
show *?thesis*
by (*rule RETURN-RES-refine*, *rule exI*[*of* - ‹*init-state* {#}›])
(*use that* **in** ‹*auto simp*: *conclusive-CDCL-run-def init-state0-def*›)
qed

have *extract-atms-clss-nempty*: ‹*extract-atms-clss CS* {} ≠ {}›
if
‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
‹*CS* ≠ []›
for *T*
proof −
show *?thesis*
using *that*
by (*cases T*; *cases CS*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
qed

have *cdcl-twl-stgy-restart-prog*: ‹*cdcl-twl-stgy-restart-prog* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*}
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
(*init-state* (*mset* ‘# *mset CS*))))› (**is** *?G1*) **and**
cdcl-twl-stgy-restart-prog-early: ‹*cdcl-twl-stgy-restart-prog-early* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = $state_W$*-of S*}

(*SPEC*
  (*conclusive-CDCL-run* (*mset* ‘# *mset CS*)
    (*init-state* (*mset* ‘# *mset CS*))))› (**is** *?G2*)
 if
  spec: ‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
  confl: ‹¬ *get-conflict* (*fst T*) ≠ *None*› **and**
  CS-nempty[*simp*]: ‹*CS* ≠ []› **and**
  ‹*extract-atms-clss CS* {} ≠ {}› **and**
  ‹*clause* ‘# *get-clauses* (*fst T*) + *unit-clss* (*fst T*) + *subsumed-clauses* (*fst T*) =
    mset ‘# *mset CS*› **and**
  ‹*get-learned-clss* (*fst T*) = {#}›
 for *T*
proof −
 let *?CS* = ‹*mset* ‘# *mset CS*›
 have
  struct-invs: ‹*twl-struct-invs-init T*› **and**
  clss-to-upd: ‹*clauses-to-update-init T* = {#}› **and**
  count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
  ‹*get-conflict-init T* = *None* ⟶
   literals-to-update-init T =
   uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
  clss: ‹*mset* ‘# *mset CS* +
   clause ‘# *get-init-clauses-init* (*to-init-state0 init-state0*) +
   other-clauses-init (*to-init-state0 init-state0*) +
   get-unit-init-clauses-init (*to-init-state0 init-state0*) +
   get-subsumed-init-clauses-init (*to-init-state0 init-state0*) =
   clause ‘# *get-init-clauses-init T* + *other-clauses-init T* +
   get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T*› **and**
  learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
     get-learned-clauses-init T›
   ‹*get-unit-learned-clauses-init T* =
     get-unit-learned-clauses-init (*to-init-state0 init-state0*)›
   ‹*get-subsumed-learned-clauses-init T* =
     get-subsumed-learned-clauses-init (*to-init-state0 init-state0*)› **and**
  stgy-invs: ‹*twl-stgy-invs* (*fst T*)› **and**
  oth: ‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
  ‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
  ‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
   get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
  using *spec* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
   Set.mem-Collect-eq **apply** −
  apply *normalize-goal*+
  by *metis*+
 have *struct-invs*: ‹*twl-struct-invs* (*fst T*)›
  by (*rule twl-struct-invs-init-twl-struct-invs*)
   (*use struct-invs oth confl* **in** ‹*auto simp*: *twl-st-init*›)
 have *clss-to-upd*: ‹*clauses-to-update* (*fst T*) = {#}›
  using *clss-to-upd* **by** (*auto simp*: *twl-st-init*)

 have *conclusive-le*: ‹*conclusive-TWL-run* (*fst T*)
 ≤ ⇓ {(*S*, *T*). *T* = $state_W$-*of S*}
   (*SPEC*
    (*conclusive-CDCL-run* (*mset* ‘# *mset CS*) (*init-state* (*mset* ‘# *mset CS*))))›
  unfolding *IsaSAT.conclusive-TWL-run-def*
 proof (*rule RES-refine*)
  fix *Ta*

assume *s*: ‹*Ta* ∈ {*Ta*.
$\exists$ *n n'*.
*cdcl-twl-stgy-restart-with-leftovers*** (*fst T*, *n*) (*Ta*, *n'*) ∧
final-twl-state Ta}›
then obtain *n n'* **where**
twl: ‹*cdcl-twl-stgy-restart-with-leftovers*** (*fst T*, *n*) (*Ta*, *n'*)› **and**
final: ‹*final-twl-state Ta*›
by *blast*
have *stgy-T-Ta*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst T*), *n*) ($state_W$*-of Ta*, *n'*)›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst T*), *n*) ($state_W$*-of Ta*, *n'*)›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

have *struct-invs-x*: ‹*twl-struct-invs Ta*›
using *twl struct-invs rtranclp-cdcl-twl-stgy-restart-with-leftovers-twl-struct-invs*[*OF twl*]
by *simp*
then have *all-struct-invs-x*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of Ta*)›
unfolding *twl-struct-invs-def*
by *blast*

have *M-lev*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*)

have *learned'*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* ([], *mset* ‘# *mset CS*, {#}, *None*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
by *auto*
have *ent*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*)
define *MW* **where** ‹*MW* ≡ *get-trail-init T*›
have *CS-clss*: ‹$cdcl_W$*-restart-mset.clauses* ($state_W$*-of* (*fst T*)) = *mset* ‘# *mset CS*›
using *learned clss oth confl* **unfolding** *clauses-def to-init-state0-def init-state0-def*
$cdcl_W$*-restart-mset.clauses-def*
by (*cases T*) *auto*
have *n-d*: ‹*no-dup MW*› **and**
propa: ‹⋀*L mark a b*. *a* @ *Propagated L mark* # *b* = *MW* ⟹
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*› **and**
clss-in-clss: ‹*set* (*get-all-mark-of-propagated MW*) ⊆ *set-mset ?CS*›
using *struct-invs* **unfolding** *twl-struct-invs-def twl-struct-invs-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
clauses-def MW-def clss to-init-state0-def init-state0-def CS-clss[*symmetric*]
by ((*cases T*; *auto*)+)[*3*]

have *count-dec'*: ‹∀ *L*∈*set MW*. ¬*is-decided L*›
using *count-dec* **unfolding** *MW-def twl-st-init* **by** *auto*
have *st-W*: ‹$state_W$*-of* (*fst T*) = (*MW*, *?CS*, {#}, *None*)›
using *clss learned confl oth*
by (*cases T*) (*auto simp*: *state-wl-l-init-def state-wl-l-def twl-st-l-init-def*
mset-take-mset-drop-mset mset-take-mset-drop-mset' clauses-def MW-def
added-only-watched-def state-wl-l-init'-def
to-init-state0-def init-state0-def

simp del: *all-clss-l-ran-m*
simp: *all-clss-lf-ran-m*[*symmetric*])

have *0*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** ([], *?CS*, {#}, *None*)
(*MW*, *?CS*, {#}, *None*)›
using *n-d count-dec′ propa clss-in-clss*
proof (*induction MW*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons K MW*) **note** *IH* = *this*(*1*) **and** *H* = *this*(*2*−) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *propa* = *this*(*4*) **and** *clss-in-clss* = *this*(*5*)
let *?init* = ‹([], *mset* ‘# *mset CS*, {#}, *None*)›
let *?int* = ‹(*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
let *?final* = ‹(*K* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
obtain *L C* **where**
K: ‹*K* = *Propagated L C*›
using *dec* **by** (*cases K*) *auto*
term *?init*

have *1*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** *?init ?int*›
apply (*rule IH*)
subgoal using *n-d* **by** *simp*
subgoal using *dec* **by** *simp*
subgoal for *M2 L′ mark M1*
using *K propa*[*of* ‹*K* # *M2*› *L′ mark M1*]
by (*auto split*: *if-splits*)
subgoal using *clss-in-clss* **by** (*auto simp*: *K*)
done
have ‹*MW* ⊨*as CNot* (*remove1-mset L C*)› **and** ‹*L* ∈# *C*›
using *propa*[*of* ‹[]› *L C* ‹*MW*›]
by (*auto simp*: *K*)
moreover have ‹*C* ∈# $cdcl_W$*-restart-mset.clauses* (*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *clss-in-clss* **by** (*auto simp*: *K clauses-def split*: *if-splits*)
ultimately have ‹$cdcl_W$*-restart-mset.propagate ?int*
(*Propagated L C* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *n-d* **apply** −
apply (*rule* $cdcl_W$*-restart-mset.propagate-rule*[*of* - ‹*C*› *L*])
by (*auto simp*: *K*)
then have *2*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy ?int ?final*›
by (*auto simp add*: *K dest*!: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.propagate′*)

show *?case*
apply (*rule rtranclp.rtrancl-into-rtrancl*[*OF 1*])
apply (*rule 2*)
.
qed

with $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*[*OF 0*, *of n*]
have *stgy*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** (([], *mset* ‘# *mset CS*, {#}, *None*), *n*)
($state_W$*-of Ta*, *n′*)›
using *stgy-T-Ta* **unfolding** *st-W* **by** *simp*

have *entailed*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of Ta*)›
apply (*rule* $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-learned-clauses-entailed*)
apply (*rule* $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*[*OF stgy*, *unfolded fst-conv*])

```
  apply (rule learned')
 apply (rule M-lev)
apply (rule ent)
done

     consider
       (ns) ‹no-step cdcl-twl-stgy Ta› |
       (stop) ‹get-conflict Ta ≠ None› and ‹count-decided (get-trail Ta) = 0›
       using final unfolding final-twl-state-def by auto
     then show ‹∃ s'∈Collect (conclusive-CDCL-run (mset '# mset CS)
              (init-state (mset '# mset CS))).
          (Ta, s') ∈ {(S, T). T = state_W-of S}›
     proof cases
       case ns
       from no-step-cdcl-twl-stgy-no-step-cdcl_W-stgy[OF this struct-invs-x]
       have ‹no-step cdcl_W-restart-mset.cdcl_W (state_W-of Ta)›
   by (blast dest: cdcl_W-ex-cdcl_W-stgy)
       then show ?thesis
  apply −
  apply (rule bexI[of - ‹state_W-of Ta›])
         using twl stgy s
         unfolding conclusive-CDCL-run-def
         by auto
     next
       case stop
       have ‹unsatisfiable (set-mset (init-clss (state_W-of Ta)))›
         apply (rule conflict-of-level-unsatisfiable)
            apply (rule all-struct-invs-x)
         using entailed stop by (auto simp: twl-st)
       then have ‹unsatisfiable (mset ' set CS)›
         using cdcl_W-restart-mset.rtranclp-cdcl_W-restart-init-clss[symmetric, OF
            cdcl_W-restart-mset.rtranclp-cdcl_W-restart-stgy-cdcl_W-restart[OF stgy]]
         by auto

       then show ?thesis
         using stop
         by (auto simp: twl-st-init twl-st conclusive-CDCL-run-def)
     qed
   qed
   show ?G1
     apply (rule cdcl-twl-stgy-restart-restart-prog-spec[THEN order-trans])
         apply (rule struct-invs; fail)
        apply (rule stgy-invs; fail)
       apply (rule clss-to-upd; fail)
      apply (use confl in fast; fail)
     apply (rule conclusive-le)
     done
   show ?G2
     apply (rule cdcl-twl-stgy-restart-restart-prog-early-spec[THEN order-trans])
         apply (rule struct-invs; fail)
        apply (rule stgy-invs; fail)
       apply (rule clss-to-upd; fail)
      apply (use confl in fast; fail)
     apply (rule conclusive-le)
     done
 qed
```

show *?thesis*
  unfolding *SAT0-def SAT-def*
  apply (*refine-vcg lhs-step-If*)
  subgoal for *b T*
    by (*rule conflict-during-init*)
  subgoal by (*rule empty-clauses*)
  subgoal for *b T*
    by (*rule extract-atms-clss-nempty*)
  subgoal for *b T*
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal for *b T*
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal for *b T*
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal for *b T*
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal for *b T*
    by (*rule cdcl-twl-stgy-restart-prog*)
  subgoal for *b T*
    by (*rule conflict-during-init*)
  subgoal by (*rule empty-clauses*)
  subgoal for *b T*
    by (*rule extract-atms-clss-nempty*)
  subgoal premises *p* **for** *b - - T*
    using *p*(*6*−)
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal premises *p* **for** *b - - T*
    using *p*(*6*−)
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal premises *p* **for** *b - - T*
    using *p*(*6*−)
    by (*cases T*)
      (*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
        extract-atms-clss-alt-def)
  subgoal for *b T*
    by (*rule cdcl-twl-stgy-restart-prog*)
  subgoal for *b T*
    by (*rule conflict-during-init*)
  subgoal by (*rule empty-clauses*)
  subgoal for *b T*
    by (*rule extract-atms-clss-nempty*)
  subgoal for *b T*
    by (*cases T*)

(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def extract-atms-clss-alt-def*)
subgoal for *b T*
by (*cases T*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def extract-atms-clss-alt-def*)
subgoal for *b T*
by (*cases T*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def extract-atms-clss-alt-def*)
subgoal for *b T*
by (*rule cdcl-twl-stgy-restart-prog-early*)
done
qed

definition *SAT-l* :: ‹*nat clause-l list* ⇒ *nat twl-st-l nres*› **where**
‹*SAT-l CS = do*{
b ← SPEC(λ-::*bool. True*);
if b then do {
let S = init-state-l;
T ← init-dt CS (*to-init-state-l S*);
let T = fst T;
if get-conflict-l T ≠ None
then RETURN T
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* + *get-subsumed-clauses-l T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-l T
}
}
else do {
let S = init-state-l;
T ← init-dt CS (*to-init-state-l S*);
failed ← SPEC (λ- :: *bool. True*);
if failed then do {
T ← init-dt CS (*to-init-state-l S*);
let T = fst T;
if get-conflict-l T ≠ None
then RETURN T
else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* + *get-subsumed-clauses-l T* = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-l T
}
} *else do* {
let T = fst T;
if get-conflict-l T ≠ None
then RETURN T

else if CS = [] *then RETURN* (*fst init-state-l*)
else do {
  ASSERT (*extract-atms-clss CS* {} ≠ {});
  ASSERT (*clauses-to-update-l T* = {#});
  ASSERT(*mset* '# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* +
  get-subsumed-clauses-l T = *mset* '# *mset CS*);
  ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
  cdcl-twl-stgy-restart-prog-early-l T
}
}
}
})›

lemma *SAT-l-SAT0*:
assumes *dist*: ‹*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*›
shows ‹*SAT-l CS* ≤ ⇓ {(*T*,*T'*). (*T*, *T'*) ∈ *twl-st-l None*} (*SAT0 CS*)›
proof −
have *inj*: ‹*inj* (*uminus* :: - *literal* ⇒ -)›
by (*auto simp*: *inj-on-def*)
have [*simp*]: ‹{#− *lit-of x*. *x* ∈# *A*#} = {#− *lit-of x*. *x* ∈# *B*#} ⟷
{#*lit-of x*. *x* ∈# *A*#} = {#*lit-of x*. *x* ∈# *B*#}› **for** *A B* :: ‹(*nat literal*, *nat literal*,
nat) *annotated-lit multiset*›
unfolding *multiset.map-comp*[*unfolded comp-def*, *symmetric*]
apply (*subst inj-image-mset-eq-iff* [*of uminus*])
apply (*rule inj*)
by (*auto simp*: *inj-on-def*)[]
have *get-unit-twl-st-l*: ‹(*s*, *x*) ∈ *twl-st-l-init* ⟹ *get-learned-unit-clauses-l-init s* = {#} ⟹
learned-clss-l (*get-clauses-l-init s*) = {#} ⟹
get-subsumed-learned-clauses-l-init s = {#} ⟹
{#*mset* (*fst x*). *x* ∈# *ran-m* (*get-clauses-l-init s*)#} +
(*get-unit-clauses-l-init s* + *get-subsumed-init-clauses-l-init s*) =
clause '# *get-init-clauses-init x* + *get-unit-init-clauses-init x* +
get-subsumed-init-clauses-init x› **for** *s x*
apply (*cases s*; *cases x*)
apply (*auto simp*: *twl-st-l-init-def mset-take-mset-drop-mset'*)
by (*metis* (*mono-tags*, *lifting*) *add.right-neutral all-clss-l-ran-m*)

have *init-dt-pre*: ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
by (*rule init-dt-pre-init*) (*use dist* **in** *auto*)

have *init-dt-spec0*: ‹*init-dt CS* (*to-init-state-l init-state-l*)
≤ ⇓{((*T*),*T'*). (*T*, *T'*) ∈ *twl-st-l-init* ∧ *twl-list-invs* (*fst T*) ∧
clauses-to-update-l (*fst T*) = {#}}
(*SPEC* (*init-dt-spec0 CS* (*to-init-state0 init-state0*)))›
apply (*rule init-dt-full*[*THEN order-trans*])
subgoal by (*rule init-dt-pre*)
subgoal
apply (*rule RES-refine*)
unfolding *init-dt-spec-def Set.mem-Collect-eq init-dt-spec0-def*
to-init-state-l-def init-state-l-def
to-init-state0-def init-state0-def
apply *normalize-goal*+
apply (*rule-tac x*=*x* **in** *bexI*)
subgoal for *s x* **by** (*auto simp*: *twl-st-l-init*)
subgoal for *s x*
unfolding *Set.mem-Collect-eq*

by (*simp-all add*: *twl-st-init twl-st-l-init twl-st-l-init-no-decision-iff get-unit-twl-st-l*)
done
done

have *init-state0*: ⟨(*fst init-state-l*, *fst init-state0*) ∈ {(T, T'). (T, T') ∈ *twl-st-l None*}⟩
by (*auto simp*: *twl-st-l-def init-state0-def init-state-l-def*)
show *?thesis*
unfolding *SAT-l-def SAT0-def*
apply (*refine-vcg init-dt-spec0*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
init-dt-spec0-def)
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ⟨*fst Ta*⟩,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def mset-take-mset-drop-mset′ intro*!: *conc-fun-R-mono*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal for *b ba - - - - T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
init-dt-spec0-def)
subgoal for *b ba - - - - T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba - - - - T Ta*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba - - - - T Ta*
by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ⟨*fst Ta*⟩,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def intro*!: *conc-fun-R-mono*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal by *auto*
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*

by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
by (*rule cdcl-twl-stgy-restart-prog-early-l-cdcl-twl-stgy-restart-prog-early*[*THEN fref-to-Down*, *of* - ‹*fst Ta*›,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def intro*!: *conc-fun-R-mono*)
done
qed

definition *SAT-wl* :: ‹*nat clause-l list* ⇒ *nat twl-st-wl nres*› **where**
‹*SAT-wl CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(*distinct-mset-set* (*mset* ‘ *set CS*));
let $\mathcal{A}_{in}{}'$ = *extract-atms-clss CS* {};
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN T
else if CS = [] *then RETURN* (([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*))
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}{}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* +
get-subsumed-clauses-wl T = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
cdcl-twl-stgy-restart-prog-wl (*finalise-init T*)
}
}
else do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
let T = *from-init-state T*;
failed ← *SPEC* (λ- :: *bool*. *True*);
if failed then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN T
else if CS = [] *then RETURN* (([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*))
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}{}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* +
get-subsumed-clauses-wl T = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
cdcl-twl-stgy-restart-prog-wl (*finalise-init T*)
}
} *else do* {
if get-conflict-wl T ≠ *None*
then RETURN T

else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, λ-. undefined))
else do {
ASSERT (extract-atms-clss CS {} ≠ {});
ASSERT(isasat-input-bounded-nempty (mset-set $\mathcal{A}_{in}{}'$));
ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
get-subsumed-clauses-wl T = mset ‘# mset CS);
ASSERT(learned-clss-l (get-clauses-wl T) = {#});
T ← rewatch-st (finalise-init T);
cdcl-twl-stgy-restart-prog-early-wl T
}
}
}
}›

lemma *SAT-l-alt-def*:
‹SAT-l CS = do{
$\mathcal{A}$ ← RETURN (); ~~atoms~~
b ← SPEC(λ-::bool. True);
if b then do {
let S = init-state-l;
$\mathcal{A}$ ← RETURN (); ~~initialisation~~
T ← init-dt CS (to-init-state-l S); ~~rewatch~~
let T = fst T;
if get-conflict-l T ≠ None
then RETURN T
else if CS = [] then RETURN (fst init-state-l)
else do {
ASSERT (extract-atms-clss CS {} ≠ {});
ASSERT (clauses-to-update-l T = {#});
ASSERT(mset ‘# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
get-subsumed-clauses-l T = mset ‘# mset CS);
ASSERT(learned-clss-l (get-clauses-l T) = {#});
cdcl-twl-stgy-restart-prog-l T
}
}
else do {
let S = init-state-l;
$\mathcal{A}$ ← RETURN (); ~~initialisation~~
T ← init-dt CS (to-init-state-l S);
failed ← SPEC (λ- :: bool. True);
if failed then do {
let S = init-state-l;
$\mathcal{A}$ ← RETURN (); ~~initialisation~~
T ← init-dt CS (to-init-state-l S);
let T = T;
if get-conflict-l-init T ≠ None
then RETURN (fst T)
else if CS = [] then RETURN (fst init-state-l)
else do {
ASSERT (extract-atms-clss CS {} ≠ {});
ASSERT (clauses-to-update-l (fst T) = {#});
ASSERT(mset ‘# ran-mf (get-clauses-l (fst T)) + get-unit-clauses-l (fst T) +
get-subsumed-clauses-l (fst T) = mset ‘# mset CS);
ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
let T = fst T;

```
        cdcl-twl-stgy-restart-prog-l T
      }
    } else do {
      let T = T;
      if get-conflict-l-init T ≠ None
      then RETURN (fst T)
      else if CS = [] then RETURN (fst init-state-l)
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT (clauses-to-update-l (fst T) = {#});
        ASSERT(mset `# ran-mf (get-clauses-l (fst T)) + get-unit-clauses-l (fst T) +
          get-subsumed-clauses-l (fst T) = mset `# mset CS);
        ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
        let T = fst T;
        cdcl-twl-stgy-restart-prog-early-l T
      }
    }
  }
}›
unfolding SAT-l-def by (auto cong: if-cong Let-def twl-st-l-init)

lemma init-dt-wl-full-init-dt-wl-spec-full:
  assumes ‹init-dt-wl-pre CS S› and  ‹init-dt-pre CS S'› and
   ‹(S, S') ∈ state-wl-l-init› and ‹∀ C∈set CS. distinct C›
  shows ‹init-dt-wl-full CS S ≤ ⇓ {(S, S'). (fst S, fst S') ∈ state-wl-l None} (init-dt CS S')›
proof -
  have init-dt-wl: ‹init-dt-wl CS S ≤ SPEC (λT. RETURN T ≤ ⇓ state-wl-l-init (init-dt CS S') ∧
     init-dt-wl-spec CS S T)›
    apply (rule SPEC-rule-conjI)
    apply (rule order-trans)
    apply (rule init-dt-wl-init-dt[of S S'])
    subgoal by (rule assms)
    subgoal by (rule assms)
    apply (rule no-fail-spec-le-RETURN-itself)
    subgoal
      apply (rule SPEC-nofail)
      apply (rule order-trans)
      apply (rule ref-two-step')
      apply (rule init-dt-full)
      using assms by (auto simp: conc-fun-RES init-dt-wl-pre-def)
    subgoal
      apply (rule order-trans)
      apply (rule init-dt-wl-init-dt-wl-spec)
      apply (rule assms)
      apply simp
      done
    done

  show ?thesis
    unfolding init-dt-wl-full-def
    apply (rule specify-left)
    apply (rule init-dt-wl)
    subgoal for x
      apply (cases x, cases ‹fst x›)
      apply (simp only: prod.case fst-conv)
      apply normalize-goal+
```

apply (*rule specify-left*)
apply (*rule-tac M =aa* **and** *N=ba* **and** *C=c* **and** *NE=d* **and** *UE=e* **and** *NS=f* **and** *US=g* **and** *Q=h* **in**
rewatch-correctness[*OF - init-dt-wl-spec-rewatch-pre*])
subgoal by *rule*
apply (*assumption*)
apply (*auto*)[*3*]
apply (*cases* ‹*init-dt CS S'*›)
apply (*auto simp*: *RETURN-RES-refine-iff state-wl-l-def state-wl-l-init-def state-wl-l-init'-def*)
done
done
qed

lemma *init-dt-wl-pre*:
assumes *dist*: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
unfolding *init-dt-wl-pre-def to-init-state-def init-state-wl-def*
apply (*rule exI*[*of -* ‹(([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›])
apply (*intro conjI*)
apply (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def*)[]
unfolding *init-dt-pre-def*
apply (*rule exI*[*of -* ‹(([], {#}, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}), {#})›])
using *dist* **by** (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def twl-st-l-init-def twl-init-invs*)[]

lemma *SAT-wl-SAT-l*:
assumes
dist: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*› **and**
bounded: ‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*SAT-wl CS* ≤ ⇓ {(*T*,*T'*). (*T*, *T'*) ∈ *state-wl-l None*} (*SAT-l CS*)›
proof −
have *extract-atms-clss*: ‹(*extract-atms-clss CS* {}, ()) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}›
by *auto*
have *init-dt-wl-pre*: ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
by (*rule init-dt-wl-pre*) (*use dist* **in** *auto*)

have *init-rel*: ‹(*to-init-state init-state-wl*, *to-init-state-l init-state-l*)
∈ *state-wl-l-init*›
by (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def twl-st-l-init-def twl-init-invs to-init-state-def init-state-wl-def init-state-l-def to-init-state-l-def*)[]

— The following stlightly strange theorem allows to reuse the definition and the correctness of *init-dt-wl-heur-full*, which was split in the definition for purely refinement-related reasons.
define *init-dt-wl-rel* **where**
‹*init-dt-wl-rel S* ≡ ({(*T*, *T'*). *RETURN T* ≤ *init-dt-wl' CS S* ∧ *T'* = ()})› **for** *S*
have *init-dt-wl'*:
‹*init-dt-wl' CS S* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ (*init-dt-wl-rel S*))›
if
‹*init-dt-wl-pre CS S*› **and**
‹(*S*, *S'*) ∈ *state-wl-l-init*› **and**
‹∀ *C*∈*set CS*. *distinct C*›
for *S S'*
proof −

have [*simp*]: ‹(U, U') ∈ ({(T, T'). *RETURN* T ≤ *init-dt-wl'* *CS* S ∧ *remove-watched* T = T'} *O* *state-wl-l-init*) ⟷ ((U, U') ∈ {(T, T'). *remove-watched* T = T'} *O* *state-wl-l-init* ∧ *RETURN* U ≤ *init-dt-wl'* *CS* S)› **for** S S' U U'
by *auto*
have H: ‹A ≤ ⇓ ({(S, S'). P S S'}) B ⟷ A ≤ ⇓ ({(S, S'). *RETURN* S ≤ A ∧ P S S'}) B›
for A B P R
by (*simp add*: *pw-conc-inres pw-conc-nofail pw-le-iff p2rel-def*)
have *nofail*: ‹*nofail* (*init-dt-wl'* *CS* S)›
apply (*rule SPEC-nofail*)
apply (*rule order-trans*)
apply (*rule init-dt-wl'-spec*[*unfolded conc-fun-RES*])
using *that* **by** *auto*
have H: ‹A ≤ ⇓ ({(S, S'). P S S'} *O* R) B ⟷ A ≤ ⇓ ({(S, S'). *RETURN* S ≤ A ∧ P S S'} *O* R) B›
for A B P R
by (*smt Collect-cong H case-prod-cong conc-fun-chain*)
show *?thesis*
unfolding *init-dt-wl-rel-def*
using *that*
by (*auto simp*: *nofail no-fail-spec-le-RETURN-itself*)
qed

have *rewatch-st*: ‹*rewatch-st* (*from-init-state* T) ≤
⇓ ({(S, S'). (S, *fst* S') ∈ *state-wl-l None* ∧ *correct-watching* S ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init* S)) (*finalise-init* S)})
(*init-dt* *CS* (*to-init-state-l init-state-l*))›
(**is** ‹- ≤ ⇓ *?rewatch* -›)
if ‹(*extract-atms-clss* *CS* {}, $\mathcal{A}$) ∈ {(x, -). x = *extract-atms-clss* *CS* {}}› **and**
‹(T, *Ta*) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for T *Ta* **and** $\mathcal{A}$ :: *unit*
proof −
have *le-wa*: ‹⇓ {(T, T'). T = *append-empty-watched* T'} A =
(*do* {S ← A; *RETURN* (*append-empty-watched* S)})› **for** A R
by (*cases* A)
(*auto simp*: *conc-fun-RES RES-RETURN-RES image-iff*)
have *init'*: ‹*init-dt-pre* *CS* (*to-init-state-l init-state-l*)›
by (*rule init-dt-pre-init*) (*use assms* **in** *auto*)
have H: ‹*do* {T ← *RETURN* T; *rewatch-st* (*from-init-state* T)} ≤
⇓{(S', T'). S' = *fst* T'} (*init-dt-wl-full* *CS* (*to-init-state init-state-wl*))›
using *that* **unfolding** *init-dt-wl-full-def init-dt-wl-rel-def init-dt-wl'-def* **apply** −
apply (*rule bind-refine*[*of* - ‹{(T', T''). T' = *append-empty-watched* T''}›])
apply (*subst le-wa*)
apply (*auto simp*: *rewatch-st-def from-init-state-def intro*!: *bind-refine*[*of* - *Id*])
done
have [*intro*]: ‹*correct-watching-init* (*af*, *ag*, *None*, *ai*, *aj*, *NS*, *US*, {#}, *ba*) ⟹
blits-in-$\mathcal{L}_{in}$ (*af*, *ag*, *ah*, *ai*, *aj*, *NS*, *US*, *ak*, *ba*)› **for** *af ag ah ai aj ak ba NS US*
by (*auto simp*: *correct-watching-init.simps blits-in-*$\mathcal{L}_{in}$*-def*
all-blits-are-in-problem-init.simps all-lits-def
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-all-lits-of-mm-ain-atms-of-iff*
atm-of-all-lits-of-mm)

have ‹*rewatch-st* (*from-init-state* T)
≤ ⇓ {(S, S'). (S, *fst* S') ∈ *state-wl-l None*}
(*init-dt* *CS* (*to-init-state-l init-state-l*))›
apply (*rule* H[*simplified*, *THEN order-trans*])
apply (*rule order-trans*)

apply (*rule ref-two-step'*)
apply (*rule init-dt-wl-full-init-dt-wl-spec-full*)
subgoal by (*rule init-dt-wl-pre*)
apply (*rule init'*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def init-state-l-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal using *assms* **by** *auto*
by (*auto intro*!: *conc-fun-R-mono simp*: *conc-fun-chain*)

moreover have ‹*rewatch-st* (*from-init-state T*) $\leq$ *SPEC* (λS. *correct-watching S* $\wedge$ *literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init S*)) (*finalise-init S*))›
apply (*rule H*[*simplified*, *THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule Watched-Literals-Watch-List-Initialisation.init-dt-wl-full-init-dt-wl-spec-full*)
subgoal by (*rule init-dt-wl-pre*)
by (*auto simp*: *conc-fun-RES init-dt-wl-spec-full-def correct-watching-init-correct-watching finalise-init-def literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* $\mathcal{L}_{all}$*-all-atms-all-lits*)

ultimately show *?thesis*
by (*rule add-invar-refineI-P*)
qed
have *cdcl-twl-stgy-restart-prog-wl-D*: ‹*cdcl-twl-stgy-restart-prog-wl* (*finalise-init U*)
$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-l* (*fst U'*))›
if
‹(*extract-atms-clss CS* {}, ($\mathcal{A}$::*unit*)) $\in$ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
UU': ‹(*U*, *U'*) $\in$ *?rewatch*› **and**
‹¬ *get-conflict-wl U* $\neq$ *None*› **and**
‹¬ *get-conflict-l* (*fst U'*) $\neq$ *None*› **and**
‹*CS* $\neq$ []› **and**
‹*CS* $\neq$ []› **and**
‹*extract-atms-clss CS* {} $\neq$ {}› **and**
‹*clauses-to-update-l* (*fst U'*) = {#}› **and**
‹*mset* ‘# *ran-mf* (*get-clauses-l* (*fst U'*)) + *get-unit-clauses-l* (*fst U'*) + *get-subsumed-clauses-l* (*fst U'*) =
mset ‘# *mset CS*› **and**
‹*learned-clss-l* (*get-clauses-l* (*fst U'*)) = {#}› **and**
‹*extract-atms-clss CS* {} $\neq$ {}› **and**
‹*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))› **and**
‹*mset* ‘# *ran-mf* (*get-clauses-wl U*) + *get-unit-clauses-wl U* + *get-subsumed-clauses-wl U*=
mset ‘# *mset CS*›
for $\mathcal{A}$ *T Ta U U'*
proof −
have *1*: ‹ $\{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$ = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init U*)) (*finalise-init U*)›
using *UU'* **by** (*auto simp*: *finalise-init-def*)
show *?thesis*
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using *UU'* **by** (*auto simp*: *finalise-init-def*)
qed

have *conflict-during-init*:

‹((([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*), *fst init-state-l*)
∈ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None*}›
by (*auto simp*: *init-state-l-def state-wl-l-def*)

have *init-init-dt*: ‹*RETURN* (*from-init-state T*)
≤ ⇓ ({(*S*, *S'*). *S* = *fst S'*} *O* {(*S* :: *nat twl-st-wl-init-full*, *S'* :: *nat twl-st-wl-init*).
remove-watched S = *S'*} *O state-wl-l-init*)
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- ≤ ⇓ *?init-dt* - ›)
if
‹(*extract-atms-clss CS* {}, (𝒜::*unit*)) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
‹(*T*, *Ta*) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for 𝒜 *T Ta*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *that* **by** (*auto simp*: *init-dt-wl-rel-def from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*RETURN T*)›
by (*auto simp*: *RETURN-refine from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*init-dt-wl' CS* (*to-init-state init-state-wl*))›
apply (*rule 2*[*THEN order-trans*])
apply (*rule ref-two-step'*)
apply (*rule 1*)
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule 2*)
unfolding *conc-fun-chain*[*symmetric*]
apply (*rule ref-two-step'*)
unfolding *conc-fun-chain*
apply (*rule init-dt-wl'-init-dt*)
apply (*rule init-dt-wl-pre*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def*
init-state-l-def state-wl-l-init-def state-wl-l-init'-def)
subgoal using *assms* **by** *auto*
done
qed

have *rewatch-st-fst*: ‹*rewatch-st* (*finalise-init* (*from-init-state T*))
≤ *SPEC* (λ*c*. (*c*, *fst Ta*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *state-wl-l None* ∧ *correct-watching S* ∧ *blits-in-*$\mathcal{L}_{in}$ *S*})›
(**is** ‹- ≤ *SPEC ?rewatch*›)
if

‹(*extract-atms-clss CS* {}, 𝒜) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
T: ‹(*T*, 𝒜') ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)› **and**
T-Ta: ‹(*from-init-state T*, *Ta*)
∈ {(*S*, *S'*). *S* = *fst S'*} *O*
{(*S*, *S'*). *remove-watched S* = *S'*} *O state-wl-l-init*› **and**
‹¬ *get-conflict-wl* (*from-init-state T*) ≠ *None*› **and**
‹¬ *get-conflict-l-init Ta* ≠ *None*›
for 𝒜 *b ba T* 𝒜' *Ta bb bc*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *T* **unfolding** *init-dt-wl-rel-def* **by** *auto*
have *2*: ‹*RETURN T* ≤ ⇓ {(*S*, *S'*). *remove-watched S* = *S'*}
(*SPEC* (*init-dt-wl-spec CS* (*to-init-state init-state-wl*)))›

using *order-trans*[*OF 1 init-dt-wl′-spec*[*OF init-dt-wl-pre*]] **.**

have *empty-watched*: ‹*get-watched-wl* (*finalise-init* (*from-init-state T*)) = (λ-. [])›
using *1 2 init-dt-wl′-spec*[*OF init-dt-wl-pre*]
by (*cases T*; *cases* ‹*init-dt-wl CS* (*init-state-wl*, {#})›)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init′-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl′-def RES-RETURN-RES)

have *1*: ‹*length* (*aa* ∝ *x*) ≥ *2*› ‹*distinct* (*aa* ∝ *x*)›
if
struct: ‹*twl-struct-invs-init*
((*af*,
{#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#},
{#}, *y*, *ac*, {#}, *NS*, *US*, {#}, *ae*),
OC)› **and**
x: ‹*x* ∈# *dom-m aa*› **and**
learned: ‹*learned-clss-l aa* = {#}›
for *af aa y ac ae x OC NS US*
proof −
have *irred*: ‹*irred aa x*›
using *that* **by** (*cases* ‹*fmlookup aa x*›) (*auto simp*: *ran-m-def dest*!: *multi-member-split*
split: *if-splits*)
have ‹*Multiset.Ball*
({#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#} +
{#})
struct-wf-twl-cls›
using *struct* **unfolding** *twl-struct-invs-init-def fst-conv twl-st-inv.simps*
by *fast*
then show ‹*length* (*aa* ∝ *x*) ≥ *2*› ‹*distinct* (*aa* ∝ *x*)›
using *x learned in-ran-mf-clause-inI*[*OF x*, *of True*] *irred*
by (*auto simp*: *mset-take-mset-drop-mset′ dest*!: *multi-member-split*[*of x*]
split: *if-splits*)
qed
have *min-len*: ‹*x* ∈# *dom-m* (*get-clauses-wl* (*finalise-init* (*from-init-state T*))) ⟹
distinct (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*) ∧
2 ≤ *length* (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*)›
for *x*
using *2*
by (*cases T*)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init′-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl′-def RES-RETURN-RES
init-dt-spec-def init-state-wl-def twl-st-l-init-def
intro: *1*)

show *?thesis*
apply (*rule rewatch-st-correctness*[*THEN order-trans*])
subgoal by (*rule empty-watched*)
subgoal by (*rule min-len*)
subgoal using *T-Ta* **by** (*auto simp*: *finalise-init-def*
state-wl-l-init-def state-wl-l-init′-def state-wl-l-def

correct-watching-init-correct-watching
correct-watching-init-blits-in-$\mathcal{L}_{in}$)
done
qed

have *cdcl-twl-stgy-restart-prog-wl-D2*: ‹*cdcl-twl-stgy-restart-prog-wl* U'
$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-l* (*fst* T'))› (**is** *?A*) **and**
cdcl-twl-stgy-restart-prog-early-wl-D2: ‹*cdcl-twl-stgy-restart-prog-early-wl* U'
$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-early-l* (*fst* T'))› (**is** *?B*)

if
U': ‹(U', *fst* T') $\in \{(S,\ T).\ (S,\ T) \in$ *state-wl-l None* $\wedge$ *correct-watching* S $\wedge$ *blits-in-$\mathcal{L}_{in}$* $S\}$›
for $\mathcal{A}$ b b' T $\mathcal{A}'$ T' c c' U'
proof −
have *1*: ‹ $\{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$ = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-$\mathcal{L}_{in}$* (*all-atms-st* (U')) (U')›
using U' **by** (*auto simp*: *finalise-init-def correct-watching.simps*)
show *?A*
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
show *?B*
apply (*rule cdcl-twl-stgy-restart-prog-early-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
qed
have *all-le*: ‹$\forall$ $C\in$*set CS*. $\forall$ $L\in$*set* C. *nat-of-lit* $L \leq$ *uint32-max*›
proof (*intro ballI*)
fix C L
assume ‹$C \in$ *set CS*› **and** ‹$L \in$ *set* C›
then have ‹$L \in\#$ $\mathcal{L}_{all}$ (*mset-set* ($\bigcup$ $C\in$*set CS*. *atm-of* ‘ *set* C))›
by (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)
then show ‹*nat-of-lit* $L \leq$ *uint32-max*›
using *assms* **by** *auto*
qed
have [*simp*]: ‹(*Tc*, *fst Td*) $\in$ *state-wl-l None* $\Longrightarrow$
get-conflict-l-init Td = *get-conflict-wl Tc*› **for** *Tc Td*
by (*cases Tc*; *cases Td*; *auto simp*: *state-wl-l-def*)
show *?thesis*
unfolding *SAT-wl-def SAT-l-alt-def*
apply (*refine-vcg extract-atms-clss init-dt-wl′ init-rel*)
subgoal using *assms* **unfolding** *extract-atms-clss-alt-def* **by** *auto*
subgoal using *assms* **unfolding** *distinct-mset-set-def* **by** *auto*
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)

subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def simp del*: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule init-init-dt*; *assumption*)
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def simp del*: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
unfolding *twl-st-l-init*[*symmetric*]
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
by (*cases U'*; *cases U*)
(*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def state-wl-l-def*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal by (*rule conflict-during-init*)

subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def simp del*: *isasat-input-bounded-def*)
subgoal for $\mathcal{A}$ *b ba U* $\mathcal{A}'$ *T' bb bc*
by (*cases U*; *cases T'*)
(*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T* $\mathcal{A}'$ *T' bb bc*
by (*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
apply (*rule rewatch-st-fst*; *assumption*)
subgoal by (*rule cdcl-twl-stgy-restart-prog-early-wl-D2*)
done
qed

definition *extract-model-of-state* **where**
⟨*extract-model-of-state U = Some* (*map lit-of* (*get-trail-wl U*))⟩

definition *extract-model-of-state-heur* **where**
⟨*extract-model-of-state-heur U = Some* (*fst* (*get-trail-wl-heur U*))⟩

definition *extract-stats* **where**
[*simp*]: ⟨*extract-stats U = None*⟩

definition *extract-stats-init* **where**
[*simp*]: ⟨*extract-stats-init = None*⟩

definition *IsaSAT* :: ⟨*nat clause-l list* ⇒ *nat literal list option nres*⟩ **where**
⟨*IsaSAT CS* = *do*{
S ← *SAT-wl CS*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}⟩

lemma *IsaSAT-alt-def*:
⟨*IsaSAT CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(*distinct-mset-set* (*mset* ‘ *set CS*));
let $\mathcal{A}_{in}'$ = *extract-atms-clss CS* {};
- ← *RETURN* ();
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN (*extract-stats T*)
else if CS = [] *then RETURN* (*Some* [])
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* +
get-subsumed-clauses-wl T = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
T ← *RETURN* (*finalise-init T*);
S ← *cdcl-twl-stgy-restart-prog-wl* (*T*);
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}
}
else do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
failed ← *SPEC* (λ- :: *bool*. *True*);
if failed then do {
let S = *init-state-wl*;
T ← *init-dt-wl′ CS* (*to-init-state S*);
T ← *rewatch-st* (*from-init-state T*);
if get-conflict-wl T ≠ *None*
then RETURN (*extract-stats T*)
else if CS = [] *then RETURN* (*Some* [])
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}'$));
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* +
get-subsumed-clauses-wl T = *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
let T = *finalise-init T*;
S ← *cdcl-twl-stgy-restart-prog-wl T*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}
} *else do* {
let T = *from-init-state T*;
if get-conflict-wl T ≠ *None*

```
          then RETURN (extract-stats T)
          else if CS = [] then RETURN (Some [])
          else do {
            ASSERT (extract-atms-clss CS {} ≠ {});
            ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
            ASSERT(mset '# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
              get-subsumed-clauses-wl T = mset '# mset CS);
            ASSERT(learned-clss-l (get-clauses-wl T) = {#});
            T ← rewatch-st T;
      T ← RETURN (finalise-init T);
            S ← cdcl-twl-stgy-restart-prog-early-wl T;
            RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
          }
        }
      }
   }› (is ‹?A = ?B›) for CS opts
proof -
  have H: ‹A = B ⟹ A ≤ ⇓ Id B› for A B
    by auto
  have 1: ‹?A ≤ ⇓ Id ?B›
    unfolding IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto

    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    apply (rule H; solves auto)
    subgoal by auto
    done

  have 2: ‹?B ≤ ⇓ Id ?A›
```

unfolding *IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws Let-def finalise-init-def*
apply (*refine-vcg*)
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
apply (*rule H*; *solves auto*)
subgoal by *auto*
done

show *?thesis*
using *1 2* **by** *simp*
qed

definition *extract-model-of-state-stat* :: ‹*twl-st-wl-heur* ⇒ *bool* × *nat literal list* × *stats*› **where**
‹*extract-model-of-state-stat U* =
(*False*, (*fst* (*get-trail-wl-heur U*)),
(λ(*M*, -, -, -, - ,- ,- ,-, -, -, *stat*, -, -). *stat*) *U*)›

definition *extract-state-stat* :: ‹*twl-st-wl-heur* ⇒ *bool* × *nat literal list* × *stats*› **where**
‹*extract-state-stat U* =
(*True*, [],
(λ(*M*, -, -, -, - ,- ,- ,-, -, -, *stat*, -, -). *stat*) *U*)›

definition *empty-conflict* :: ‹*nat literal list option*› **where**
‹*empty-conflict* = *Some* []›

definition *empty-conflict-code* :: ‹(*bool* × - *list* × *stats*) *nres*› **where**
‹*empty-conflict-code* = *do*{
let M0 = [];
RETURN (*False*, *M0*, (*0*, *0*, *0*, *0*, *0*, *0*, *0*,

0))}⟩

definition *empty-init-code* :: ⟨*bool* × *- list* × *stats*⟩ **where**
⟨*empty-init-code* = (*True*, [], (*0*, *0*, *0*, *0*,
0, *0*, *0*, *0*))⟩

definition *convert-state* **where**
⟨*convert-state - S* = *S*⟩

definition *IsaSAT-use-fast-mode* **where**
⟨*IsaSAT-use-fast-mode* = *True*⟩

definition *isasat-fast-init* :: ⟨*twl-st-wl-heur-init* ⇒ *bool*⟩ **where**
⟨*isasat-fast-init S* ⟷ (*length* (*get-clauses-wl-heur-init S*) ≤ *sint64-max* − (*uint32-max div 2* + *6*))⟩

definition *IsaSAT-heur* :: ⟨*opts* ⇒ *nat clause-l list* ⇒ (*bool* × *nat literal list* × *stats*) *nres*⟩ **where**
⟨*IsaSAT-heur opts CS* = *do*{
  ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
  ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*);
  let $\mathcal{A}_{in}{}'$ = *mset-set* (*extract-atms-clss CS* {});
  ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}{}'$);
  ASSERT(*distinct-mset* $\mathcal{A}_{in}{}'$);
  let $\mathcal{A}_{in}{}''$ = *virtual-copy* $\mathcal{A}_{in}{}'$;
  let b = *opts-unbounded-mode opts*;
  if b
  then do {
    S ← *init-state-wl-heur* $\mathcal{A}_{in}{}'$;
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur True CS S*;
 T ← *rewatch-heur-st T*;
    let T = *convert-state* $\mathcal{A}_{in}{}''$ *T*;
    if ¬*get-conflict-wl-is-None-heur-init T*
    then RETURN (*empty-init-code*)
    else if CS = [] *then empty-conflict-code*
    else do {
      ASSERT($\mathcal{A}_{in}{}''$ ≠ {#});
      ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}{}''$);
      - ← *isasat-information-banner T*;
       ASSERT((λ(*M′*, *N′*, *D′*, *Q′*, *W′*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As ≠ *None* ∧
        lst-As ≠ *None*) *T*);
      T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
      U ← *cdcl-twl-stgy-restart-prog-wl-heur T*;
      RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
        else extract-state-stat U)
     }
  }
  else do {
    S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}{}'$;
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur False CS S*;
    let failed = *is-failed-heur-init T* ∨ ¬*isasat-fast-init T*;
    if failed then do {
      let $\mathcal{A}_{in}{}'$ = *mset-set* (*extract-atms-clss CS* {});
      S ← *init-state-wl-heur* $\mathcal{A}_{in}{}'$;
      (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur True CS S*;

let T = convert-state $\mathcal{A}_{in}''$ *T*;
T ← *rewatch-heur-st T*;
if ¬*get-conflict-wl-is-None-heur-init T*
then RETURN (*empty-init-code*)
else if CS = [] *then empty-conflict-code*
else do {
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
- ← *isasat-information-banner T*;
ASSERT((λ(*M*′, *N*′, *D*′, *Q*′, *W*′, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As ≠ *None* ∧
lst-As ≠ *None*) *T*);
T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
U ← *cdcl-twl-stgy-restart-prog-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else do {
let T = convert-state $\mathcal{A}_{in}''$ *T*;
if ¬*get-conflict-wl-is-None-heur-init T*
then RETURN (*empty-init-code*)
else if CS = [] *then empty-conflict-code*
else do {
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
- ← *isasat-information-banner T*;
ASSERT((λ(*M*′, *N*′, *D*′, *Q*′, *W*′, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As ≠ *None* ∧
lst-As ≠ *None*) *T*);
ASSERT(*rewatch-heur-st-fast-pre T*);
T ← *rewatch-heur-st-fast T*;
ASSERT(*isasat-fast-init T*);
T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
ASSERT(*isasat-fast T*);
U ← *cdcl-twl-stgy-restart-prog-early-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
}
})›

lemma *fref-to-Down-unRET-uncurry0-SPEC*:
assumes ‹(λ-. (*f*), λ-. (*RETURN g*)) ∈ $[P]_f$ *unit-rel* → ⟨*B*⟩*nres-rel*› **and** ‹*P* ()›
shows ‹*f* ≤ *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)›
proof −
have [*simp*]: ‹*RES* (B^{-1} ‘‘ {*g*}) = *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)›
by *auto*
show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *fref-to-Down-unRET-SPEC*:

assumes ‹$(f,\ RETURN\ o\ g) \in [P]_f\ A \to \langle B\rangle nres\text{-}rel$› **and**
‹$P\ y$› **and**
‹$(x,\ y) \in A$›
shows ‹$f\ x \leq SPEC\ (\lambda c.\ (c,\ g\ y) \in B)$›
proof −
have [*simp*]: ‹$RES\ (B^{-1}\ ``\ \{g\}) = SPEC\ (\lambda c.\ (c,\ g) \in B)$› **for** *g*
by *auto*
show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *fref-to-Down-unRET-curry-SPEC*:
assumes ‹$(uncurry\ f,\ uncurry\ (RETURN\ oo\ g)) \in [P]_f\ A \to \langle B\rangle nres\text{-}rel$› **and**
‹$P\ (x,\ y)$› **and**
‹$((x',\ y'),\ (x,\ y)) \in A$›
shows ‹$f\ x'\ y' \leq SPEC\ (\lambda c.\ (c,\ g\ x\ y) \in B)$›
proof −
have [*simp*]: ‹$RES\ (B^{-1}\ ``\ \{g\}) = SPEC\ (\lambda c.\ (c,\ g) \in B)$› **for** *g*
by *auto*
show *?thesis*
using *assms*
unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *all-lits-of-mm-empty-iff*: ‹$all\text{-}lits\text{-}of\text{-}mm\ A = \{\#\} \longleftrightarrow (\forall\ C\ \in\#\ A.\ C = \{\#\})$›
apply (*induction A*)
subgoal by *auto*
subgoal by (*auto simp*: *all-lits-of-mm-add-mset*)
done

lemma *all-lits-of-mm-extract-atms-clss*:
‹$L \in\# (all\text{-}lits\text{-}of\text{-}mm\ (mset\ `\#\ mset\ CS)) \longleftrightarrow atm\text{-}of\ L \in extract\text{-}atms\text{-}clss\ CS\ \{\}$›
by (*induction CS*)
(*auto simp*: *extract-atms-clss-alt-def all-lits-of-mm-add-mset*
in-all-lits-of-m-ain-atms-of-iff)

lemma *IsaSAT-heur-alt-def*:
‹*IsaSAT-heur opts CS* = *do*{
$ASSERT(isasat\text{-}input\text{-}bounded\ (mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\})))$;
$ASSERT(\forall\ C\in set\ CS.\ \forall\ L\in set\ C.\ nat\text{-}of\text{-}lit\ L \leq uint32\text{-}max)$;
$let\ \mathcal{A}_{in}{}' = mset\text{-}set\ (extract\text{-}atms\text{-}clss\ CS\ \{\})$;
$ASSERT(isasat\text{-}input\text{-}bounded\ \mathcal{A}_{in}{}')$;
$ASSERT(distinct\text{-}mset\ \mathcal{A}_{in}{}')$;
$let\ \mathcal{A}_{in}{}'' = virtual\text{-}copy\ \mathcal{A}_{in}{}'$;
$let\ b = opts\text{-}unbounded\text{-}mode\ opts$;
if b
then do {
$S \leftarrow init\text{-}state\text{-}wl\text{-}heur\ \mathcal{A}_{in}{}'$;
$(T::twl\text{-}st\text{-}wl\text{-}heur\text{-}init) \leftarrow\ init\text{-}dt\text{-}wl\text{-}heur\ True\ CS\ S$;
$T \leftarrow rewatch\text{-}heur\text{-}st\ T$;
$let\ T = convert\text{-}state\ \mathcal{A}_{in}{}''\ T$;
$if\ \neg get\text{-}conflict\text{-}wl\text{-}is\text{-}None\text{-}heur\text{-}init\ T$

then RETURN (empty-init-code)
else if CS = [] then empty-conflict-code
else do {
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
ASSERT((λ(M', N', D', Q', W', ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As $\neq$ *None* $\wedge$
lst-As $\neq$ *None*) *T*);
T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
U $\leftarrow$ *cdcl-twl-stgy-restart-prog-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else do {
S $\leftarrow$ *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur False CS S*;
failed $\leftarrow$ *RETURN* (*is-failed-heur-init T* $\vee$ $\neg$*isasat-fast-init T*);
if failed then do {
S $\leftarrow$ *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) $\leftarrow$ *init-dt-wl-heur True CS S*;
T $\leftarrow$ *rewatch-heur-st T*;
let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
if $\neg$*get-conflict-wl-is-None-heur-init T*
then RETURN (empty-init-code)
else if CS = [] then empty-conflict-code
else do {
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
ASSERT((λ(M', N', D', Q', W', ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As $\neq$ *None* $\wedge$
lst-As $\neq$ *None*) *T*);
T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
U $\leftarrow$ *cdcl-twl-stgy-restart-prog-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else do {
let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
if $\neg$*get-conflict-wl-is-None-heur-init T*
then RETURN (empty-init-code)
else if CS = [] then empty-conflict-code
else do {
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
ASSERT((λ(M', N', D', Q', W', ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*).
fst-As $\neq$ *None* $\wedge$
lst-As $\neq$ *None*) *T*);
ASSERT(*rewatch-heur-st-fast-pre T*);
T $\leftarrow$ *rewatch-heur-st-fast T*;
ASSERT(*isasat-fast-init T*);
T $\leftarrow$ *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
ASSERT(*isasat-fast T*);
U $\leftarrow$ *cdcl-twl-stgy-restart-prog-early-wl-heur T*;
RETURN (*if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*

else extract-state-stat U)
}
}
}
}›
by (*auto simp*: *init-state-wl-heur-fast-def IsaSAT-heur-def isasat-init-fast-slow-alt-def convert-state-def isasat-information-banner-def cong*: *if-cong*)

abbreviation *rewatch-heur-st-rewatch-st-rel* **where**
‹*rewatch-heur-st-rewatch-st-rel CS U V* ≡
{(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-subsumed-init-clauses-wl (*fst T*) = *get-subsumed-init-clauses-wl* (*fst V*) ∧
get-subsumed-learned-clauses-wl (*fst T*) = *get-subsumed-learned-clauses-wl* (*fst V*) ∧
get-unit-init-clss-wl (*fst T*) = *get-unit-init-clss-wl* (*fst V*) ∧
get-unit-learned-clss-wl (*fst T*) = *get-unit-learned-clss-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})}›

lemma *rewatch-heur-st-rewatch-st*:
assumes
UV: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
shows ‹*rewatch-heur-st U* ≤
⇓(*rewatch-heur-st-rewatch-st-rel CS U V*)
(*rewatch-st* (*from-init-state V*))›
proof −
let *?𝒜* = ‹(*mset-set* (*extract-atms-clss CS* {}))›
obtain *M′ arena D′ j W′ vm φ clvls cach lbd vdom M N D NE UE NS US Q W OC failed* **where**
U: ‹*U* = ((*M′*, *arena*, *D′*, *j*, *W′*, *vm*, *φ*, *clvls*, *cach*, *lbd*, *vdom*, *failed*))› **and**
V: ‹*V* = ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *Q*, *W*), *OC*)›
by (*cases U*; *cases V*) *auto*
have *valid*: ‹*valid-arena arena N* (*set vdom*)› **and**
dist: ‹*distinct vdom*› **and**
vdom-N: ‹*mset vdom* = *dom-m N*› **and**
watched: ‹(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 *?𝒜*)› **and**
lall: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm ?𝒜* (*mset* ‘# *ran-mf N*)› **and**
vdom: ‹*vdom-m ?𝒜 W N* ⊆ *set-mset* (*dom-m N*)›
using *UV* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def U V distinct-mset-dom empty-watched-def vdom-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-lits-of-mm-union simp flip*: *distinct-mset-mset-distinct*)

show *?thesis*
using *UV*
unfolding *rewatch-heur-st-def rewatch-st-def*
apply (*simp only*: *prod.simps from-init-state-def fst-conv nres-monad1 U V*)
apply *refine-vcg*
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def dest*: *valid-arena-vdom-subset*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist - watched lall*])
subgoal by *simp*
subgoal using *vdom-N*[*symmetric*] **by** *simp*
subgoal by (*auto simp*: *vdom-m-def*)

subgoal by (*auto simp*: *U V twl-st-heur-parsing-def Collect-eq-comp'*
twl-st-heur-parsing-no-WL-def)
done
qed

lemma *rewatch-heur-st-rewatch-st2*:
assumes
T: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
shows ‹*rewatch-heur-st-fast*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
≤ ⇓ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})})
(*rewatch-st* (*from-init-state V*))›
proof –
have
UV: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
unfolding *convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def*
by (*rule rewatch-heur-st-rewatch-st*[*of U V*, *THEN order-trans*])
(*auto intro*!: *conc-fun-R-mono simp*: *Collect-eq-comp'*
twl-st-heur-parsing-def)
qed

lemma *rewatch-heur-st-rewatch-st3*:
assumes
T: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
failed: ‹¬*is-failed-heur-init U*›
shows ‹*rewatch-heur-st-fast*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
≤ ⇓ (*rewatch-heur-st-rewatch-st-rel CS U V*)
(*rewatch-st* (*from-init-state V*))›
proof –
have
UV: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
using *T failed* **by** (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
unfolding *convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def*
by (*rule rewatch-heur-st-rewatch-st*[*of U V*, *THEN order-trans*])
(*auto intro*!: *conc-fun-R-mono simp*: *Collect-eq-comp'*
twl-st-heur-parsing-def)
qed

```
abbreviation option-with-bool-rel :: ‹((bool × 'a) × 'a option) set› where
  ‹option-with-bool-rel ≡ {((b, s), s'). (b = is-None s') ∧ (¬b ⟶ s = the s')}›

definition model-stat-rel :: ‹((bool × nat literal list × 'a) × nat literal list option) set› where
  ‹model-stat-rel = {((b, M', s), M). ((b, rev M'), M) ∈ option-with-bool-rel}›

lemma IsaSAT-heur-IsaSAT:
  ‹IsaSAT-heur b CS ≤ ⇓model-stat-rel (IsaSAT CS)›
proof -
  have init-dt-wl-heur: ‹init-dt-wl-heur True CS S ≤
      ⇓(twl-st-heur-parsing-no-WL 𝒜 True O {(S, T). S = remove-watched T ∧
          get-watched-wl (fst T) = (λ-. [])})
        (init-dt-wl' CS T)›
    if
      ‹case (CS, T) of
       (CS, S) ⇒
  (∀ C∈set CS. literals-are-in-ℒin 𝒜 (mset C)) ∧
  distinct-mset-set (mset ‘ set CS)› and
      ‹((CS, S), CS, T) ∈ ⟨Id⟩list-rel ×f twl-st-heur-parsing-no-WL 𝒜 True›
  for 𝒜 CS T S
  proof -
    show ?thesis
      apply (rule init-dt-wl-heur-init-dt-wl[THEN fref-to-Down-curry, of 𝒜 CS T CS S,
        THEN order-trans])
      apply (rule that(1))
      apply (rule that(2))
      apply (cases ‹init-dt-wl CS T›)
      apply (force simp: init-dt-wl'-def RES-RETURN-RES conc-fun-RES
        Image-iff)+
      done
  qed
  have init-dt-wl-heur-b: ‹init-dt-wl-heur False CS S ≤
      ⇓(twl-st-heur-parsing-no-WL 𝒜 False O {(S, T). S = remove-watched T ∧
          get-watched-wl (fst T) = (λ-. [])})
        (init-dt-wl' CS T)›
    if
      ‹case (CS, T) of
       (CS, S) ⇒
  (∀ C∈set CS. literals-are-in-ℒin 𝒜 (mset C)) ∧
  distinct-mset-set (mset ‘ set CS)› and
      ‹((CS, S), CS, T) ∈ ⟨Id⟩list-rel ×f twl-st-heur-parsing-no-WL 𝒜 True›
  for 𝒜 CS T S
  proof -
    show ?thesis
      apply (rule init-dt-wl-heur-init-dt-wl[THEN fref-to-Down-curry, of 𝒜 CS T CS S,
        THEN order-trans])
      apply (rule that(1))
      using that(2) apply (force simp: twl-st-heur-parsing-no-WL-def)
      apply (cases ‹init-dt-wl CS T›)
      apply (force simp: init-dt-wl'-def RES-RETURN-RES conc-fun-RES
        Image-iff)+
      done
  qed
  have virtual-copy: ‹(virtual-copy 𝒜, ()) ∈ {(ℬ, c). c = () ∧ ℬ = 𝒜}› for ℬ 𝒜
    by (auto simp: virtual-copy-def)
  have input-le: ‹∀ C∈set CS. ∀ L∈set C. nat-of-lit L ≤ uint32-max›
```

if ‹*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {}))›
proof (*intro ballI*)
fix *C L*
assume ‹$C \in$ *set CS*› **and** ‹$L \in$ *set C*›
then have ‹$L \in\#$ $\mathcal{L}_{all}$ (*mset-set* (*extract-atms-clss CS* {}))›
by (*auto simp*: *extract-atms-clss-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit* $L \leq$ *uint32-max*›
using *that* **by** *auto*
qed
have *lits-C*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*mset-set* (*extract-atms-clss CS* {})) (*mset C*)›
if ‹$C \in$ *set CS*› **for** *C CS*
using *that*
by (*force simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
in-all-lits-of-m-ain-atms-of-iff extract-atms-clss-alt-def
atm-of-eq-atm-of)
have *init-state-wl-heur*: ‹*isasat-input-bounded* $\mathcal{A} \Longrightarrow$
init-state-wl-heur $\mathcal{A} \leq$ *SPEC* ($\lambda c.$ (*c*, *init-state-wl*) $\in$
{(*S*, *S'*). (*S*, *S'*) $\in$ *twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *True*})› **for** $\mathcal{A}$
apply (*rule init-state-wl-heur-init-state-wl*[*THEN fref-to-Down-unRET-uncurry0-SPEC*,
of $\mathcal{A}$, *THEN order-trans*])
apply (*auto*)
done

let *?TT* = ‹*rewatch-heur-st-rewatch-st-rel CS*›
have *get-conflict-wl-is-None-heur-init*: ‹(*Tb*, *Tc*) $\in$ *?TT U V* $\Longrightarrow$
($\neg$ *get-conflict-wl-is-None-heur-init Tb*) = (*get-conflict-wl Tc* $\neq$ *None*)› **for** *Tb Tc U V*
by (*cases V*) (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *get-conflict-wl-is-None-heur-init3*: ‹(*T*, *Ta*)
$\in$ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* $\wedge$ *get-watched-wl* (*fst T*) = (λ-. [])} $\Longrightarrow$
(*failed*, *faileda*)
$\in$ {(*b*, *b'*). *b* = *b'* $\wedge$ *b* = (*is-failed-heur-init T* $\vee$ $\neg$ *isasat-fast-init T*)} $\Longrightarrow$ $\neg$*failed* $\Longrightarrow$
($\neg$ *get-conflict-wl-is-None-heur-init T*) = (*get-conflict-wl* (*fst Ta*) $\neq$ *None*)› **for** *T Ta failed faileda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-heur-parsing-no-WL-def*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *finalise-init-nempty*: ‹*x1i* $\neq$ *None*› ‹*x1j* $\neq$ *None*›
if
T: ‹(*Tb*, *Tc*) $\in$ *?TT U V*› **and**
nempty: ‹*extract-atms-clss CS* {} $\neq$ {}› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb* =
(*x1*, *x2*)›

for *ba S T Ta Tb Tc uu x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k U V*
proof −
show ‹*x1i* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
show ‹*x1j* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
qed

have *banner*: ‹*isasat-information-banner*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(-, -). *True*})› **for** *Tb*
by (*auto simp*: *isasat-information-banner-def*)

have *finalise-init-code*: ‹*finalise-init-code b*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?A*) **and**
finalise-init-code3: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?B*)
if
T: ‹(*Tb*, *Tc*) ∈ *?TT U V*› **and**
confl: ‹¬ *get-conflict-wl Tc* ≠ *None*› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
clss-CS: ‹*mset* ‘# *ran-mf* (*get-clauses-wl Tc*) + *get-unit-clauses-wl Tc* + *get-subsumed-clauses-wl Tc* =
mset ‘# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl Tc*) = {#}›
for *ba S T Ta Tb Tc u v U V*
proof −
have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl* **by** *auto*
have *2*: ‹*all-atms-st Tc* ≠ {#}›
using *clss-CS nempty* **unfolding** *all-lits-def add.assoc*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def extract-atms-clss-alt-def
all-lits-of-mm-empty-iff)
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *clss-CS nempty* **unfolding** *all-lits-def add.assoc*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
isasat-input-bounded-nempty-def
atm-of-all-lits-of-mm extract-atms-clss-alt-def atms-of-ms-def)
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-init-cong

vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$*-cong*[*THEN H*]
D_0*-cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$*-cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$*-all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)
show *?A*
by (*rule finalise-init-finalise-init*[*THEN fref-to-Down-unRET-curry-SPEC*, *of b*])
(*use 1 2 learned 4* **in** *auto*)
then show *?B* **unfolding** *convert-state-def* **by** *auto*
qed

have *get-conflict-wl-is-None-heur-init2*: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)) =
(*get-conflict-wl* (*from-init-state V*) ≠ *None*)› **for** *U V*
by (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def twl-st-heur-parsing-no-WL-def
option-lookup-clause-rel-def convert-state-def from-init-state-def)

have *finalise-init2*: ‹*x1i* ≠ *None*› ‹*x1j* ≠ *None*›
if
T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *b O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T* =
(*x1*, *x2*)›
for *uu ba S T Ta baa uua uub x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f*
x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k b
proof –
show ‹*x1i* ≠ *None*›

using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
show ‹*x1j* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
qed

have *rewatch-heur-st-fast-pre*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
for *uu ba S T Ta baa uua uub*
proof −
have ‹*valid-arena* (*get-clauses-wl-heur-init T*) (*get-clauses-wl* (*fst Ta*))
(*set* (*get-vdom-heur-init T*))›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
using *length-le* **unfolding** *rewatch-heur-st-fast-pre-def convert-state-def*
isasat-fast-init-def uint64-max-def uint32-max-def
by (*auto dest*: *valid-arena-in-vdom-le-arena*)
qed
have *rewatch-heur-st-fast-pre2*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
and
failed: ‹¬*is-failed-heur-init T*›
for *uu ba S T Ta baa uua uub*
proof −
have
Ta: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
using *failed T* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
from *rewatch-heur-st-fast-pre*[*OF this length-le*]
show *?thesis* **.**
qed
have *finalise-init-code2*: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ {(*S'*, *T'*).
(*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S'*})›
if

Ta: ⟨(*T*, *Ta*)
 ∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
 {(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}⟩ **and**
confl: ⟨¬ *get-conflict-wl* (*from-init-state Ta*) ≠ *None*⟩ **and**
⟨*CS* ≠ []⟩ **and**
nempty: ⟨*extract-atms-clss CS* {} ≠ {}⟩ **and**
⟨*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))⟩ **and**
clss-CS: ⟨*mset* ‘# *ran-mf* (*get-clauses-wl* (*from-init-state Ta*)) +
 get-unit-clauses-wl (*from-init-state Ta*) + *get-subsumed-clauses-wl* (*from-init-state Ta*) =
 mset ‘# *mset CS*⟩ **and**
learned: ⟨*learned-clss-l* (*get-clauses-wl* (*from-init-state Ta*)) = {#}⟩ **and**
⟨*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})) ≠ {#}⟩ **and**
⟨*isasat-input-bounded-nempty*
 (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))⟩ **and**
⟨*case convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T of*
 (*M*′, *N*′, *D*′, *Q*′, *W*′, *xa*, *xb*) ⇒
 (*case xa of*
 (*x*, *xa*) ⇒
 (*case x of*
 (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ⇒
 λ*to-remove* (φ, *clvls*). *fst-As* ≠ *None* ∧ *lst-As* ≠ *None*)
 xa)
 xb⟩ **and**
T: ⟨(*Tb*, *Tc*) ∈ *?TT T Ta*⟩ **and**
failed: ⟨¬*is-failed-heur-init T*⟩
for *uu ba S T Ta baa uua uub V W b Tb Tc*
proof −
have
Ta: ⟨(*T*, *Ta*)
 ∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
 {(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}⟩
 using *failed Ta* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)

have *1*: ⟨*get-conflict-wl Tc* = *None*⟩
 using *confl T* **by** (*auto simp*: *from-init-state-def*)
have *Ta-Tc*: ⟨*all-atms-st Tc* = *all-atms-st* (*from-init-state Ta*)⟩
 using *T Ta*
 unfolding *all-lits-alt-def mem-Collect-eq prod.case relcomp.simps*
 all-atms-def add.assoc **apply** −
 apply *normalize-goal*+
 by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
 twl-st-heur-parsing-no-WL-def twl-st-heur-parsing-def
 from-init-state-def)
moreover have *3*: ⟨*set-mset* (*all-atms-st* (*from-init-state Ta*)) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))⟩
 unfolding *all-lits-alt-def mem-Collect-eq prod.case relcomp.simps*
 all-atms-def clss-CS[*unfolded add.assoc*] **apply** −
 by (*auto simp*: *extract-atms-clss-alt-def*
 atm-of-all-lits-of-mm atms-of-ms-def)
ultimately have *2*: ⟨*all-atms-st Tc* ≠ {#}⟩
 using *nempty*
 by *auto*
have *3*: ⟨*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))⟩
 unfolding *Ta-Tc 3* **..**

have *H*: ⟨*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*⟩ **for** *A B x*

by *auto*
have *H′*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-init-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H′*]
phase-saving-cong[*THEN H′*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$-*cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$-*all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)

show *?thesis*
apply (*rule finalise-init-finalise-init-full*[*unfolded conc-fun-RETURN*,
THEN order-trans])
by (*use 1 2 learned 4 T* **in** ‹*auto simp*: *from-init-state-def convert-state-def*›)
qed
have *isasat-fast*: ‹*isasat-fast Td*›
if
fast: ‹¬ ¬ *isasat-fast-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))
T)› **and**
Tb: ‹(*Tb*, *Tc*) ∈ *?TT T Ta*› **and**
Td: ‹(*Td*, *Te*)
∈ {(*S′*, *T′*).
(*S′*, *T′*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S′*}›
for *uu ba S T Ta baa uua uub Tb Tc Td Te*
proof −
show *?thesis*
using *fast Td Tb*
by (*auto simp*: *convert-state-def isasat-fast-init-def sint64-max-def*
uint32-max-def uint64-max-def isasat-fast-def)
qed
define *init-succesfull* **where** ‹*init-succesfull T* = *RETURN* (*is-failed-heur-init T* ∨ ¬*isasat-fast-init T*)› **for** *T*
define *init-succesfull2* **where** ‹*init-succesfull2* = *SPEC* (λ- :: *bool*. *True*)›
have [*refine*]: ‹*init-succesfull T* ≤ ⇓ {(*b*, *b′*). (*b* = *b′*) ∧ (*b* ⟷ *is-failed-heur-init T* ∨ ¬*isasat-fast-init T*)}
init-succesfull2› **for** *T*
by (*auto simp*: *init-succesfull-def init-succesfull2-def intro*!: *RETURN-RES-refine*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *IsaSAT-heur-alt-def IsaSAT-alt-def init-succesfull-def*[*symmetric*]
apply (*rewrite at* ‹*do* {- ← *init-dt-wl′* - -; - ← (¤ :: *bool nres*); *If* - - - }› *init-succesfull2-def*[*symmetric*])

apply (*refine-vcg virtual-copy init-state-wl-heur banner*
cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D[*THEN fref-to-Down*])
subgoal by (*rule input-le*)
subgoal by (*rule distinct-mset-mset-set*)
subgoal by *auto*
subgoal by *auto*
apply (*rule init-dt-wl-heur*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
apply (*rule rewatch-heur-st-rewatch-st*; *assumption*)
subgoal unfolding *convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init*)
subgoal by (*auto simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*auto simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by *simp*
subgoal by (*rule finalise-init-nempty*)
subgoal by (*rule finalise-init-nempty*)
apply (*rule finalise-init-code*; *assumption*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*27*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)

apply (*rule init-dt-wl-heur-b*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
subgoal by *fast*
apply (*rule init-dt-wl-heur*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
apply (*rule rewatch-heur-st-rewatch-st*; *assumption*)
subgoal unfolding *convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init*)
subgoal by (*auto simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)

subgoal by *simp*
subgoal by (*rule finalise-init-nempty*)
subgoal by (*rule finalise-init-nempty*)
apply (*rule finalise-init-code*; *assumption*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** *- ba S T Ta Tb Tc u v*
using *p*(*34*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal unfolding *from-init-state-def convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init3*)
subgoal by (*simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*rule finalise-init2*)
subgoal by (*rule finalise-init2*)
subgoal for *uu ba S T Ta baa uua*
by (*rule rewatch-heur-st-fast-pre2*; *assumption?*) (*simp-all add*: *convert-state-def*)
apply (*rule rewatch-heur-st-rewatch-st3*; *assumption?*)
subgoal by *auto*
subgoal by (*clarsimp simp add*: *isasat-fast-init-def convert-state-def*)
apply (*rule finalise-init-code2*; *assumption?*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
apply (*rule-tac r1* = ‹*length* (*get-clauses-wl-heur Td*)› **in** *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-p*
THEN fref-to-Down])
subgoal by (*auto simp*: *isasat-fast-def sint64-max-def uint64-max-def uint32-max-def*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** *- ba S T Ta Tb Tc u v*
using *p*(*33*)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done
qed

definition *length-get-clauses-wl-heur-init* **where**
‹*length-get-clauses-wl-heur-init S* = *length* (*get-clauses-wl-heur-init S*)›

lemma *length-get-clauses-wl-heur-init-alt-def*:
‹*RETURN o length-get-clauses-wl-heur-init* = (λ(-, *N*,-). *RETURN* (*length N*))›
unfolding *length-get-clauses-wl-heur-init-def*
by *auto*

definition *model-if-satisfiable* :: ⟨*nat clauses* ⇒ *nat literal list option nres*⟩ **where**
⟨*model-if-satisfiable CS* = *SPEC* (λM.
if satisfiable (*set-mset CS*) *then* $M \neq$ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*)⟩

definition *SAT′* :: ⟨*nat clauses* ⇒ *nat literal list option nres*⟩ **where**
⟨*SAT′ CS* = *do* {
T ← *SAT CS*;
RETURN(*if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*)
}
⟩

lemma *SAT-model-if-satisfiable*:
⟨(*SAT′*, *model-if-satisfiable*) ∈ [λ*CS*. (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id*→ ⟨*Id*⟩*nres-rel*⟩
(**is** ⟨- ∈[λ*CS*. *?P CS*]$_f$ *Id* → -⟩)
proof −
have *H*: ⟨$cdcl_W$*-restart-mset*.$cdcl_W$*-stgy-invariant* (*init-state CS*)⟩
⟨$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*init-state CS*)⟩
if ⟨*?P CS*⟩ **for** *CS*
using *that* **by** (*auto simp*:
twl-struct-invs-def twl-st-inv.simps $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.no-strange-atm-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-M-level-inv-def*
$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clause-alt-def* $cdcl_W$*-restart-mset.no-smaller-propa-def*
past-invs.simps clauses-def twl-list-invs-def twl-stgy-invs-def clause-to-update-def
$cdcl_W$*-restart-mset*.$cdcl_W$*-stgy-invariant-def*
$cdcl_W$*-restart-mset.no-smaller-confl-def*
distinct-mset-set-def)
have *H*: ⟨*s* ∈ {*M*. *if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*}⟩
if
dist: ⟨*Multiset.Ball CS distinct-mset*⟩ **and**
[*simp*]: ⟨*CS′* = *CS*⟩ **and**
s: ⟨*s* ∈ (λ*T*. *if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-run CS′* (*init-state CS′*))⟩
for *s* :: ⟨*nat literal list option*⟩ **and** *CS CS′*
proof −
obtain *T* **where**
s: ⟨(*s* = *Some* (*map lit-of* (*trail T*)) ∧ *conflicting T* = *None*) ∨
(*s* = *None* ∧ *conflicting T* ≠ *None*)⟩ **and**
conc: ⟨*conclusive-CDCL-run CS′* ([], *CS′*, {#}, *None*) *T*⟩
using *s* **by** *auto force*
consider
n n′ **where** ⟨$cdcl_W$*-restart-mset*.$cdcl_W$*-restart-stgy*** (([], *CS′*, {#}, *None*), *n*) (*T*, *n′*)⟩
⟨*no-step* $cdcl_W$*-restart-mset*.$cdcl_W$ *T*⟩ |
⟨*CS′* ≠ {#}⟩ **and** ⟨*conflicting T* ≠ *None*⟩ **and** ⟨*backtrack-lvl T* = *0*⟩ **and**
⟨*unsatisfiable* (*set-mset CS′*)⟩
using *conc* **unfolding** *conclusive-CDCL-run-def*
by *auto*
then show *?thesis*
proof *cases*
case (*1 n n′*) **note** *st* = *this*(*1*) **and** *ns* = *this*(*2*)
have ⟨*no-step* $cdcl_W$*-restart-mset*.$cdcl_W$*-stgy T*⟩
using *ns* $cdcl_W$*-restart-mset*.$cdcl_W$*-stgy-*$cdcl_W$ **by** *blast*
then have *full-T*: ⟨*full* $cdcl_W$*-restart-mset*.$cdcl_W$*-stgy T T*⟩
unfolding *full-def* **by** *blast*

have *invs*: ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*stgy-invariant T*›
‹$cdcl_W$-*restart-mset.cdcl*$_W$-*all-struct-inv T*›
using *st* $cdcl_W$-*restart-mset.rtranclp-cdcl*$_W$-*restart-dcl*$_W$-*all-struct-inv*[*OF st*]
$cdcl_W$-*restart-mset.rtranclp-cdcl*$_W$-*restart-dcl*$_W$-*stgy-invariant*[*OF st*]
H[*OF dist*] **by** *auto*
have *res*: ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*restart*** ([], *CS'*, {#}, *None*) *T*›
using $cdcl_W$-*restart-mset.rtranclp-cdcl*$_W$-*restart-stgy-cdcl*$_W$-*restart*[*OF st*] **by** *simp*
have *ent*: ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*learned-clauses-entailed-by-init T*›
using $cdcl_W$-*restart-mset.rtranclp-cdcl*$_W$-*learned-clauses-entailed*[*OF res*] *H*[*OF dist*]
unfolding ‹*CS'* = *CS*› $cdcl_W$-*restart-mset.cdcl*$_W$-*learned-clauses-entailed-by-init-def*
$cdcl_W$-*restart-mset.cdcl*$_W$-*all-struct-inv-def*
by *simp*
have [*simp*]: ‹*init-clss T* = *CS*›
using $cdcl_W$-*restart-mset.rtranclp-cdcl*$_W$-*restart-init-clss*[*OF res*] **by** *simp*
show *?thesis*
using $cdcl_W$-*restart-mset.full-cdcl*$_W$-*stgy-inv-normal-form*[*OF full-T invs ent*] *s*
by (*auto simp*: *true-annots-true-cls lits-of-def*)
next
case *2*
moreover have ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*learned-clauses-entailed-by-init* (*init-state CS*)›
unfolding $cdcl_W$-*restart-mset.cdcl*$_W$-*learned-clauses-entailed-by-init-def*
by *auto*
ultimately show *?thesis*
using *H*[*OF dist*] $cdcl_W$-*restart-mset.full-cdcl*$_W$-*stgy-inv-normal-form*[*of* ‹*init-state CS*›
‹*init-state CS*›] *s*
by *auto*
qed
qed
show *?thesis*
unfolding *SAT'-def model-if-satisfiable-def SAT-def Let-def*
apply (*intro frefI nres-relI*)
subgoal for *CS' CS*
unfolding *RES-RETURN-RES*
apply (*rule RES-refine*)
unfolding *pair-in-Id-conv bex-triv-one-point1 bex-triv-one-point2*
by (*rule H*)
done
qed

lemma *SAT-model-if-satisfiable'*:
‹(*uncurry* (λ-. *SAT'*), *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id* $\times_r$ *Id*→ ⟨*Id*⟩*nres-rel*›
using *SAT-model-if-satisfiable* **by** (*auto simp*: *fref-def*)

definition *SAT-l'* **where**
‹*SAT-l' CS* = *do*{
S ← *SAT-l CS*;
RETURN (*if get-conflict-l S* = *None then Some* (*map lit-of* (*get-trail-l S*)) *else None*)
}›

definition *SAT0'* **where**
‹*SAT0' CS* = *do*{
S ← *SAT0 CS*;
RETURN (*if get-conflict S* = *None then Some* (*map lit-of* (*get-trail S*)) *else None*)
}›

lemma *twl-st-l-map-lit-of*[*twl-st-l*, *simp*]:
⟨$(S,\ T) \in$ *twl-st-l b* $\Longrightarrow$ *map lit-of* (*get-trail-l S*) = *map lit-of* (*get-trail T*)⟩
by (*auto simp*: *twl-st-l-def convert-lits-l-map-lit-of*)

lemma *ISASAT-SAT-l′*:
assumes ⟨*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*⟩ **and**
⟨*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ' *set C*))⟩
shows ⟨*IsaSAT CS* ≤ ⇓ *Id* (*SAT-l′ CS*)⟩
unfolding *IsaSAT-def SAT-l′-def*
apply *refine-vcg*
apply (*rule SAT-wl-SAT-l*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

lemma *SAT-l′-SAT0′*:
assumes ⟨*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*⟩
shows ⟨*SAT-l′ CS* ≤ ⇓ *Id* (*SAT0′ CS*)⟩
unfolding *SAT-l′-def SAT0′-def*
apply *refine-vcg*
apply (*rule SAT-l-SAT0*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

lemma *SAT0′-SAT′*:
assumes ⟨*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*⟩
shows ⟨*SAT0′ CS* ≤ ⇓ *Id* (*SAT′* (*mset* '# *mset CS*))⟩
unfolding *SAT′-def SAT0′-def*
apply *refine-vcg*
apply (*rule SAT0-SAT*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def twl-st-l twl-st*)
done

lemma *IsaSAT-heur-model-if-sat*:
assumes ⟨∀ *C* ∈# *mset* '# *mset CS*. *distinct-mset C*⟩ **and**
⟨*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ' *set C*))⟩
shows ⟨*IsaSAT-heur opts CS* ≤ ⇓ *model-stat-rel* (*model-if-satisfiable* (*mset* '# *mset CS*))⟩
apply (*rule IsaSAT-heur-IsaSAT*[*THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule ISASAT-SAT-l′*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-l′-SAT0′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT0′-SAT′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-model-if-satisfiable*[*THEN fref-to-Down*, *of* ‹*mset* ‘# *mset CS*›])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule conc-fun-R-mono*)
apply (*auto simp*: *model-stat-rel-def*)
done

lemma *IsaSAT-heur-model-if-sat′*: ‹(*uncurry IsaSAT-heur*, *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*) ∧
(∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*)]$_f$
Id $\times_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨*model-stat-rel*⟩*nres-rel*›
proof −
have *H*: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
if *CS-p*: ‹∀ *C*∈#*CS′*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*› **and**
‹(*CS*, *CS′*) ∈ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*›
for *CS CS′*
unfolding *isasat-input-bounded-def*
proof
fix *L*
assume *L*: ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
then obtain *C* **where**
L: ‹*C*∈*set CS* ∧ (*L* ∈*set C* ∨ − *L* ∈ *set C*)›
apply (*cases L*)
apply (*auto simp*: *extract-atms-clss-alt-def uint32-max-def*
$\mathcal{L}_{all}$-*def*)+
apply (*metis literal.exhaust-sel*)+
done
have ‹*nat-of-lit L* ≤ *uint32-max* ∨ *nat-of-lit* (−*L*) ≤ *uint32-max*›
using *L CS-p that* **by** (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff′*)
then show ‹*nat-of-lit L* ≤ *uint32-max*›
using *L*
by (*cases L*) (*auto simp*: *extract-atms-clss-alt-def uint32-max-def*)
qed
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *uncurry-def*
apply *clarify*
subgoal for *o1 o2 o3 CS o1′ o2′ o3′ CS′*
apply (*rule IsaSAT-heur-model-if-sat*[*THEN order-trans*, *of CS* - ‹(*o1*, *o2*, *o3*)›])
subgoal by (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff′*)

subgoal by (*rule H*) *auto*
apply (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
done
done
qed

21.3 Refinements of the Whole Bounded SAT Solver

This is the specification of the SAT solver:

definition *SAT-bounded* :: ‹*nat clauses* ⇒ (*bool* × *nat* $cdcl_W$*-restart-mset*) *nres*› **where**
‹*SAT-bounded CS* = *do*{
T ← *SPEC*(λ*T*. *T* = *init-state CS*);
finished ← *SPEC*(λ-. *True*);
if ¬*finished then*
RETURN (*finished*, *T*)
else
SPEC (λ(*b*, *U*). *b* ⟶ *conclusive-CDCL-run CS T U*)
}›

definition *SAT0-bounded* :: ‹*nat clause-l list* ⇒ (*bool* × *nat twl-st*) *nres*› **where**
‹*SAT0-bounded CS* = *do*{
let (*S* :: *nat twl-st-init*) = *init-state0*;
T ← *SPEC* (λ*T*. *init-dt-spec0 CS* (*to-init-state0 S*) *T*);
finished ← *SPEC*(λ-. *True*);
if ¬*finished then do* {
RETURN (*False*, *fst init-state0*)
} *else do* {
let T = *fst T*;
if get-conflict T ≠ *None*
then RETURN (*True*, *T*)
else if CS = [] *then RETURN* (*True*, *fst init-state0*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update T* = {#});
ASSERT(*clause* '# (*get-clauses T*) + *unit-clss T* + *subsumed-clauses T* = *mset* '# *mset CS*);
ASSERT(*get-learned-clss T* = {#});
cdcl-twl-stgy-restart-prog-bounded T
}
}
}›

lemma *SAT0-bounded-SAT-bounded*:
assumes ‹*Multiset.Ball* (*mset* '# *mset CS*) *distinct-mset*›
shows ‹*SAT0-bounded CS* ≤ ⇓ ({((*b*, *S*), (*b'*, *T*)). *b* = *b'* ∧ (*b* ⟶ *T* = $state_W$*-of S*)}) (*SAT-bounded* (*mset* '# *mset CS*))›
(**is** ‹- ≤ ⇓*?A* -›)
proof −
have *conflict-during-init*:
‹*RETURN* (*True*, *fst T*)
≤ ⇓ {((*b*, *S*), *b'*, *T*). *b* = *b'* ∧ (*b* ⟶ *T* = $state_W$*-of S*)}
(*SPEC* (λ(*b*, *U*). *b* ⟶ *conclusive-CDCL-run* (*mset* '# *mset CS*) *S U*))›
if

TS: ‹(*T*, *S*)
∈ {(*S*, *T*).
(*init-dt-spec0 CS* (*to-init-state0 init-state0*) *S*) ∧
(*T* = *init-state* (*mset* ‘# *mset CS*))}› **and**
‹¬ ¬ *failed'*› **and**
‹¬ ¬ *failed*› **and**
confl: ‹*get-conflict* (*fst T*) ≠ *None*›
for *bS bT failed T failed' S*
proof −
let *?CS* = ‹*mset* ‘# *mset CS*›
have *failed*[*simp*]: ‹*failed*› ‹*failed'*› ‹*failed* = *True*› ‹*failed'* = *True*›
using *that*
by *auto*
have
struct-invs: ‹*twl-struct-invs-init T*› **and**
‹*clauses-to-update-init T* = {#}› **and**
count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
clss: ‹*mset* ‘# *mset CS* +
clause ‘# *get-init-clauses-init* (*to-init-state0 init-state0*) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) +
get-subsumed-init-clauses-init (*to-init-state0 init-state0*) =
clause ‘# *get-init-clauses-init T* + *other-clauses-init T* +
get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T*› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›
‹*get-unit-learned-clauses-init T* =
get-unit-learned-clauses-init (*to-init-state0 init-state0*)›
‹*get-subsumed-learned-clauses-init T* =
get-subsumed-learned-clauses-init (*to-init-state0 init-state0*)› **and**
‹*twl-stgy-invs* (*fst T*)› **and**
‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
using *TS* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
Set.mem-Collect-eq prod.case failed simp-thms **apply** −
apply *normalize-goal*+
by *metis*+

have *count-dec*: ‹*count-decided* (*get-trail* (*fst T*)) = *0*›
using *count-dec* **unfolding** *count-decided-0-iff* **by** (*auto simp*: *twl-st-init*
twl-st-wl-init)

have *le*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* ($state_W$*-of-init T*)› **and**
all-struct-invs:
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of-init T*)›
using *struct-invs* **unfolding** *twl-struct-invs-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast*+
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting* ($state_W$*-of-init T*)›
using *struct-invs* **unfolding** *twl-struct-invs-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*

by *fast*
have ‹*unsatisfiable* (*set-mset* (*mset* '# *mset* (*rev CS*)))›
using *conflict-of-level-unsatisfiable*[*OF all-struct-invs*] *count-dec confl learned le clss*
by (*auto simp*: *clauses-def mset-take-mset-drop-mset' twl-st-init twl-st-wl-init image-image to-init-state0-def init-state0-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def ac-simps twl-st-l-init*)
then have *unsat*[*simp*]: ‹*unsatisfiable* (*mset* ' *set CS*)›
by *auto*
then have [*simp*]: ‹$CS \neq []$›
by (*auto simp del*: *unsat*)
show *?thesis*
unfolding *conclusive-CDCL-run-def*
apply (*rule RETURN-SPEC-refine*)
apply (*rule exI*[*of* - ‹(*True*, $state_W$*-of* (*fst T*))›])
apply (*intro conjI*)
subgoal
by *auto*
subgoal
using *struct-invs learned count-dec clss confl*
by (*clarsimp simp*: *twl-st-init twl-st-wl-init twl-st-l-init*)
done
qed

have *empty-clauses*: ‹*RETURN* (*True*, *fst init-state0*)
$\leq \Downarrow$ *?A*
(*SPEC*
(λ(*b*, *U*). *b* $\longrightarrow$ *conclusive-CDCL-run* (*mset* '# *mset CS*) *S U*))›
if
TS: ‹(*T*, *S*)
$\in$ {(*S*, *T*).
(*init-dt-spec0 CS* (*to-init-state0 init-state0*) *S*) $\wedge$
(*T* = *init-state* (*mset* '# *mset CS*))}› **and**
[*simp*]: ‹*CS* = []›
for *bS bT failed T failed' S*
proof −
let *?CS* = ‹*mset* '# *mset CS*›
have [*dest*]: ‹$cdcl_W$*-restart-mset.*$cdcl_W$ ([], {#}, {#}, *None*) (*a*, *aa*, *ab*, *b*) $\Longrightarrow$ *False*›
for *a aa ab b*
by (*metis* $cdcl_W$*-restart-mset.*$cdcl_W$*.cases* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.conflict'* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.propagate'* $cdcl_W$*-restart-mset.other'* $cdcl_W$*-stgy-*$cdcl_W$*-init-state-empty-no-step init-state.simps*)
show *?thesis*
by (*rule RETURN-RES-refine*, *rule exI*[*of* - ‹(*True*, *init-state* {#})›])
(*use that* **in** ‹*auto simp*: *conclusive-CDCL-run-def init-state0-def*›)
qed

have *extract-atms-clss-nempty*: ‹*extract-atms-clss CS* {} ≠ {}›
if
TS: ‹(*T*, *S*)
$\in$ {(*S*, *T*).
(*init-dt-spec0 CS* (*to-init-state0 init-state0*) *S*) $\wedge$
(*T* = *init-state* (*mset* '# *mset CS*))}› **and**
‹$CS \neq []$› **and**
‹¬*get-conflict* (*fst T*) ≠ *None*›

for *bS bT failed T failed′ S*
proof −
show *?thesis*
using *that*
by (*cases T*; *cases CS*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
qed

have *cdcl-twl-stgy-restart-prog*: ‹*cdcl-twl-stgy-restart-prog-bounded* (*fst T*)
≤ ⇓ {((*b*, *S*), *b*′, *T*). *b* = *b*′ ∧ (*b* ⟶ *T* = $state_W$*-of S*)}
(*SPEC* (λ(*b*, *U*). *b* ⟶ *conclusive-CDCL-run* (*mset* ‘# *mset CS*) *S U*))› (**is** *?G1*)
if
bT-bS: ‹(*T*, *S*)
∈ {(*S*, *T*).
(*init-dt-spec0 CS* (*to-init-state0 init-state0*) *S*) ∧
(*T* = *init-state* (*mset* ‘# *mset CS*))}› **and**
‹*CS* ≠ []› **and**
confl: ‹¬*get-conflict* (*fst T*) ≠ *None*› **and**
CS-nempty[*simp*]: ‹*CS* ≠ []› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*clause* ‘# *get-clauses* (*fst T*) + *unit-clss* (*fst T*) + *subsumed-clauses* (*fst T*) = *mset* ‘# *mset CS*›
and
‹*get-learned-clss* (*fst T*) = {#}›
for *bS bT failed T failed′ S*
proof −
let *?CS* = ‹*mset* ‘# *mset CS*›

have
struct-invs: ‹*twl-struct-invs-init T*› **and**
clss-to-upd: ‹*clauses-to-update-init T* = {#}› **and**
count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
clss: ‹*mset* ‘# *mset CS* +
clause ‘# *get-init-clauses-init* (*to-init-state0 init-state0*) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) +
get-subsumed-init-clauses-init (*to-init-state0 init-state0*) =
clause ‘# *get-init-clauses-init T* + *other-clauses-init T* +
get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T*› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›
‹*get-unit-learned-clauses-init T* =
get-unit-learned-clauses-init (*to-init-state0 init-state0*)›
‹*get-subsumed-learned-clauses-init T* =
get-subsumed-learned-clauses-init (*to-init-state0 init-state0*)› **and**
stgy-invs: ‹*twl-stgy-invs* (*fst T*)› **and**
oth: ‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
using *bT-bS* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
Set.mem-Collect-eq simp-thms prod.case **apply** −

apply *normalize-goal+*
by *metis+*
have *struct-invs*: ‹*twl-struct-invs* (*fst* T)›
by (*rule twl-struct-invs-init-twl-struct-invs*)
(*use struct-invs oth confl* **in** ‹*auto simp*: *twl-st-init*›)
have *clss-to-upd*: ‹*clauses-to-update* (*fst* T) = {#}›
using *clss-to-upd* **by** (*auto simp*: *twl-st-init*)

have *conclusive-le*: ‹*conclusive-TWL-run* (*fst* T)
$\leq \Downarrow \{(S,\ T).\ T = state_W\text{-}of\ S\}$
(*SPEC*
(*conclusive-CDCL-run* (*mset* ‘# *mset CS*) (*init-state* (*mset* ‘# *mset CS*))))›
unfolding *IsaSAT.conclusive-TWL-run-def*
proof (*rule RES-refine*)
fix *Ta*
assume *s*: ‹*Ta* ∈ {*Ta*.
$\exists\, n\ n'$.
cdcl-twl-stgy-restart-with-leftovers** (*fst* T, n) (*Ta*, n') ∧
final-twl-state Ta}›
then obtain $n\ n'$ **where**
twl: ‹*cdcl-twl-stgy-restart-with-leftovers*** (*fst* T, n) (*Ta*, n')› **and**
final: ‹*final-twl-state Ta*›
by *blast*
have *stgy-T-Ta*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst* T), n) ($state_W$*-of Ta*, n')›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** ($state_W$*-of* (*fst* T), n) ($state_W$*-of Ta*, n')›
using *rtranclp-cdcl-twl-stgy-restart-with-leftovers-*$cdcl_W$*-restart-stgy*[*OF twl*] *struct-invs*
stgy-invs **by** *simp*

have *struct-invs-x*: ‹*twl-struct-invs Ta*›
using *twl struct-invs rtranclp-cdcl-twl-stgy-restart-with-leftovers-twl-struct-invs*[*OF twl*]
by *simp*
then have *all-struct-invs-x*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of Ta*)›
unfolding *twl-struct-invs-def*
by *blast*

have *M-lev*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*)

have *learned'*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* ([], *mset* ‘# *mset CS*, {#}, *None*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
by *auto*
have *ent*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ([], *mset* ‘# *mset CS*, {#}, *None*)›
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*)
define *MW* **where** ‹*MW* ≡ *get-trail-init* T›
have *CS-clss*: ‹$cdcl_W$*-restart-mset.clauses* ($state_W$*-of* (*fst* T)) = *mset* ‘# *mset CS*›
using *learned clss oth confl* **unfolding** *clauses-def to-init-state0-def init-state0-def*
$cdcl_W$*-restart-mset.clauses-def*
by (*cases* T) *auto*
have *n-d*: ‹*no-dup MW*› **and**
propa: ‹$\bigwedge$*L mark a b*. *a* @ *Propagated L mark* # *b* = *MW* ⟹
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*› **and**

clss-in-clss: ‹*set* (*get-all-mark-of-propagated MW*) ⊆ *set-mset ?CS*›
using *struct-invs* **unfolding** *twl-struct-invs-def twl-struct-invs-init-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause-alt-def*
clauses-def MW-def clss to-init-state0-def init-state0-def CS-clss[*symmetric*]
by ((*cases T*; *auto*)+)[*3*]

have *count-dec′*: ‹∀ *L*∈*set MW*. ¬*is-decided L*›
using *count-dec* **unfolding** *MW-def twl-st-init* **by** *auto*
have *st-W*: ‹*state*$_W$*-of* (*fst T*) = (*MW*, *?CS*, {#}, *None*)›
using *clss learned confl oth*
by (*cases T*) (*auto simp*: *state-wl-l-init-def state-wl-l-def twl-st-l-init-def*
mset-take-mset-drop-mset mset-take-mset-drop-mset′ clauses-def MW-def
added-only-watched-def state-wl-l-init′-def
to-init-state0-def init-state0-def
simp del: *all-clss-l-ran-m*
simp: *all-clss-lf-ran-m*[*symmetric*])

have *0*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy*** ([], *?CS*, {#}, *None*)
(*MW*, *?CS*, {#}, *None*)›
using *n-d count-dec′ propa clss-in-clss*
proof (*induction MW*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons K MW*) **note** *IH* = *this*(*1*) **and** *H* = *this*(*2*−) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *propa* = *this*(*4*) **and** *clss-in-clss* = *this*(*5*)
let *?init* = ‹([], *mset* ‘# *mset CS*, {#}, *None*)›
let *?int* = ‹(*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
let *?final* = ‹(*K* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
obtain *L C* **where**
K: ‹*K* = *Propagated L C*›
using *dec* **by** (*cases K*) *auto*
term *?init*

have *1*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy*** *?init ?int*›
apply (*rule IH*)
subgoal using *n-d* **by** *simp*
subgoal using *dec* **by** *simp*
subgoal for *M2 L′ mark M1*
using *K propa*[*of* ‹*K* # *M2*› *L′ mark M1*]
by (*auto split*: *if-splits*)
subgoal using *clss-in-clss* **by** (*auto simp*: *K*)
done
have ‹*MW* ⊨*as CNot* (*remove1-mset L C*)› **and** ‹*L* ∈# *C*›
using *propa*[*of* ‹[]› *L C* ‹*MW*›]
by (*auto simp*: *K*)
moreover have ‹*C* ∈# *cdcl*$_W$*-restart-mset.clauses* (*MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *clss-in-clss* **by** (*auto simp*: *K clauses-def split*: *if-splits*)
ultimately have ‹*cdcl*$_W$*-restart-mset.propagate ?int*
(*Propagated L C* # *MW*, *mset* ‘# *mset CS*, {#}, *None*)›
using *n-d* **apply** −
apply (*rule cdcl*$_W$*-restart-mset.propagate-rule*[*of* - ‹*C*› *L*])
by (*auto simp*: *K*)
then have *2*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy ?int ?final*›
by (*auto simp add*: *K dest*!: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy.propagate′*)

show *?case*
apply (*rule rtranclp.rtrancl-into-rtrancl*[*OF 1*])
apply (*rule 2*)
.
qed

with $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-stgy-*$cdcl_W$*-restart-stgy*[*OF 0*, *of n*]
have *stgy*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart-stgy*** (([], *mset* '# *mset CS*, {#}, *None*), *n*)
(*state*$_W$*-of Ta*, *n'*)›
using *stgy-T-Ta* **unfolding** *st-W* **by** *simp*

have *entailed*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*state*$_W$*-of Ta*)›
apply (*rule* $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-learned-clauses-entailed*)
apply (*rule* $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*[*OF stgy*, *unfolded fst-conv*])
apply (*rule learned'*)
apply (*rule M-lev*)
apply (*rule ent*)
done

consider
(*ns*) ‹*no-step cdcl-twl-stgy Ta*› |
(*stop*) ‹*get-conflict Ta* ≠ *None*› **and** ‹*count-decided* (*get-trail Ta*) = *0*›
using *final* **unfolding** *final-twl-state-def* **by** *auto*
then show ‹∃ *s'*∈*Collect* (*conclusive-CDCL-run* (*mset* '# *mset CS*)
(*init-state* (*mset* '# *mset CS*))).
(*Ta*, *s'*) ∈ {(*S*, *T*). *T* = *state*$_W$*-of S*}›
proof *cases*
case *ns*
from *no-step-cdcl-twl-stgy-no-step-*$cdcl_W$*-stgy*[*OF this struct-invs-x*]
have ‹*no-step* $cdcl_W$*-restart-mset.*$cdcl_W$ (*state*$_W$*-of Ta*)›
by (*blast dest*: $cdcl_W$*-ex-*$cdcl_W$*-stgy*)
then show *?thesis*
apply −
apply (*rule bexI*[*of* - ‹*state*$_W$*-of Ta*›])
using *twl stgy s*
unfolding *conclusive-CDCL-run-def*
by *auto*
next
case *stop*
have ‹*unsatisfiable* (*set-mset* (*init-clss* (*state*$_W$*-of Ta*)))›
apply (*rule conflict-of-level-unsatisfiable*)
apply (*rule all-struct-invs-x*)
using *entailed stop* **by** (*auto simp*: *twl-st*)
then have ‹*unsatisfiable* (*mset* ' *set CS*)›
using $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-init-clss*[*symmetric*, *OF*
$cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*[*OF stgy*]]
by *auto*

then show *?thesis*
using *stop*
by (*auto simp*: *twl-st-init twl-st conclusive-CDCL-run-def*)
qed
qed
then have *conclusive-le*: ‹*conclusive-TWL-run-bounded* (*fst T*)
≤ ⇓ {((*b*, *S*), *b'*, *T*). *b* = *b'* ∧ (*b* ⟶ *T* = *state*$_W$*-of S*)}

(*SPEC* (λ(*b*, *U*). *b* ⟶ *conclusive-CDCL-run* (*mset* '# *mset CS*) *S U*))›
using *bT-bS*
unfolding *conclusive-TWL-run-bounded-def*
conclusive-TWL-run-def conc-fun-RES
less-eq-nres.simps subset-iff **apply** −
apply (*intro allI*)
apply (*rename-tac t*)
apply (*drule-tac x=* ‹(*snd t*)› **in** *spec*)
by (*fastforce*)

show *?G1*
apply (*rule cdcl-twl-stgy-restart-prog-bounded-spec*[*THEN order-trans*])
apply (*rule struct-invs*; *fail*)
apply (*rule stgy-invs*; *fail*)
apply (*rule clss-to-upd*; *fail*)
apply (*use confl* **in** ‹*simp add*: *twl-st-init*›; *fail*)
apply (*rule conclusive-le*)
done
qed

The following does not relate anything, because the initialisation is already doing some steps.

have [*refine0*]:
‹*SPEC*
(λ*T*. *init-dt-spec0 CS* (*to-init-state0 init-state0*) *T*)
≤ ⇓ {(*S*, *T*).
(*init-dt-spec0 CS* (*to-init-state0 init-state0*) *S*) ∧
(*T* = *init-state* (*mset* '# *mset CS*))}
(*SPEC* (λ*T*. *T* = *init-state* (*mset* '# *mset CS*)))›
by (*rule RES-refine*)
(*auto simp*: *init-state0-def to-init-state0-def*
extract-atms-clss-alt-def intro!:)[]
show *?thesis*
unfolding *SAT0-bounded-def SAT-bounded-def*
apply (*subst Let-def*)
apply (*refine-vcg*)
subgoal by (*auto simp*: *RETURN-def intro*!: *RES-refine*)
subgoal by (*auto simp*: *RETURN-def intro*!: *RES-refine*)
apply (*rule lhs-step-If*)
subgoal
by (*rule conflict-during-init*)
apply (*rule lhs-step-If*)
subgoal
by (*rule empty-clauses*) *assumption*+
apply (*intro ASSERT-leI*)
subgoal for *b T*
by (*rule extract-atms-clss-nempty*)
subgoal for *S T*
by (*cases S*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *S T*
by (*cases S*)
(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *S T*
by (*cases S*)

(*auto simp*: *init-state0-def to-init-state0-def init-dt-spec0-def*
extract-atms-clss-alt-def)
subgoal for *S T*
by (*rule cdcl-twl-stgy-restart-prog*)
done
qed

definition *SAT-l-bounded* :: ‹*nat clause-l list* ⇒ (*bool* × *nat twl-st-l*) *nres*› **where**
‹*SAT-l-bounded CS* = *do*{
let S = *init-state-l*;
T ← *init-dt CS* (*to-init-state-l S*);
finished ← *SPEC* (λ- :: *bool*. *True*);
if ¬*finished then do* {
RETURN (*False*, *fst init-state-l*)
} *else do* {
let T = *fst T*;
if get-conflict-l T ≠ *None*
then RETURN (*True*, *T*)
else if CS = [] *then RETURN* (*True*, *fst init-state-l*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});
ASSERT (*clauses-to-update-l T* = {#});
ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l T*) + *get-unit-clauses-l T* + *get-subsumed-clauses-l T*= *mset* ‘# *mset CS*);
ASSERT(*learned-clss-l* (*get-clauses-l T*) = {#});
cdcl-twl-stgy-restart-prog-bounded-l T
}

}
}›

lemma *SAT-l-bounded-SAT0-bounded*:
assumes *dist*: ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹*SAT-l-bounded CS* ≤ ⇓ {((*b*, *T*),(*b*′, *T*′)). *b*=*b*′ ∧ (*b* ⟶ (*T*, *T*′) ∈ *twl-st-l None*)} (*SAT0-bounded CS*)›
proof −
have *inj*: ‹*inj* (*uminus* :: - *literal* ⇒ -)›
by (*auto simp*: *inj-on-def*)
have [*simp*]: ‹{#− *lit-of x*. *x* ∈# *A*#} = {#− *lit-of x*. *x* ∈# *B*#} ⟷
{#*lit-of x*. *x* ∈# *A*#} = {#*lit-of x*. *x* ∈# *B*#}› **for** *A B* :: ‹(*nat literal*, *nat literal*,
nat) *annotated-lit multiset*›
unfolding *multiset.map-comp*[*unfolded comp-def*, *symmetric*]
apply (*subst inj-image-mset-eq-iff* [*of uminus*])
apply (*rule inj*)
by (*auto simp*: *inj-on-def*)[]
have *get-unit-twl-st-l*: ‹(*s*, *x*) ∈ *twl-st-l-init* ⟹ *get-learned-unit-clauses-l-init s* = {#} ⟹
learned-clss-l (*get-clauses-l-init s*) = {#} ⟹
{#*mset* (*fst x*). *x* ∈# *ran-m* (*get-clauses-l-init s*)#} +
(*get-unit-clauses-l-init s* + *get-subsumed-init-clauses-l-init s*) =
clause ‘# *get-init-clauses-init x* + (*get-unit-init-clauses-init x* +
get-subsumed-init-clauses-init x)› **for** *s x*
apply (*cases s*; *cases x*)
apply (*auto simp*: *twl-st-l-init-def mset-take-mset-drop-mset*′)
by (*metis* (*mono-tags*, *lifting*) *add.right-neutral all-clss-l-ran-m*)

have *init-dt-pre*: ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›

by (*rule init-dt-pre-init*) (*use dist* **in** *auto*)

have *init-dt-spec0*: ‹*init-dt CS* (*to-init-state-l init-state-l*)
≤ ⇓{((*T*),*T'*). (*T*, *T'*) ∈ *twl-st-l-init* ∧ *twl-list-invs* (*fst T*) ∧
clauses-to-update-l (*fst T*) = {#}}
(*SPEC* (*init-dt-spec0 CS* (*to-init-state0 init-state0*)))›
apply (*rule init-dt-full*[*THEN order-trans*])
subgoal by (*rule init-dt-pre*)
subgoal
apply (*rule RES-refine*)
unfolding *init-dt-spec-def Set.mem-Collect-eq init-dt-spec0-def*
to-init-state-l-def init-state-l-def
to-init-state0-def init-state0-def
apply *normalize-goal+*
apply (*rule-tac x=x* **in** *bexI*)
subgoal for *s x* **by** (*auto simp*: *twl-st-l-init*)
subgoal for *s x*
unfolding *Set.mem-Collect-eq*
by (*simp-all add*: *twl-st-init twl-st-l-init twl-st-l-init-no-decision-iff get-unit-twl-st-l*)
done
done

have *init-state0*: ‹ ((*True*, *fst init-state-l*), *True*, *fst init-state0*)
∈ {((*b*, *T*), *b'*, *T'*). *b*=*b'* ∧ (*b* ⟶ (*T*, *T'*) ∈ *twl-st-l None*)}›
by (*auto simp*: *twl-st-l-def init-state0-def init-state-l-def*)

show *?thesis*
unfolding *SAT-l-bounded-def SAT0-bounded-def*
apply (*refine-vcg init-dt-spec0*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*
init-dt-spec0-def)
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
subgoal for *T Ta finished finisheda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'*)
subgoal for *T Ta finished finisheda*
by (*rule cdcl-twl-stgy-restart-prog-bounded-l-cdcl-twl-stgy-restart-prog-bounded*[*THEN fref-to-Down*,
of - ‹*fst Ta*›,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def mset-take-mset-drop-mset' intro*!: *conc-fun-R-mono*)
done
qed

definition *SAT-wl-bounded* :: ‹*nat clause-l list* ⇒ (*bool* × *nat twl-st-wl*) *nres*› **where**
‹*SAT-wl-bounded CS* = *do*{

ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
ASSERT(distinct-mset-set (mset ‘ set CS));
let $\mathcal{A}_{in}'$ = extract-atms-clss CS {};
let S = init-state-wl;
T ← init-dt-wl' CS (to-init-state S);
let T = from-init-state T;
finished ← SPEC (λ- :: bool. True);
if ¬finished then do {
RETURN(finished, T)
} else do {
if get-conflict-wl T ≠ None
then RETURN (True, T)
else if CS = [] then RETURN (True, ([], fmempty, None, {#}, {#}, {#}, {#}, {#}, λ-. undefined))
else do {
ASSERT (extract-atms-clss CS {} ≠ {});
ASSERT(isasat-input-bounded-nempty (mset-set $\mathcal{A}_{in}'$));
ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T + get-subsumed-clauses-wl T = mset ‘# mset CS);
ASSERT(learned-clss-l (get-clauses-wl T) = {#});
T ← rewatch-st (finalise-init T);
cdcl-twl-stgy-restart-prog-bounded-wl T
}
}
}›

lemma *SAT-l-bounded-alt-def*:
‹SAT-l-bounded CS = do{
$\mathcal{A}$ ← RETURN (); ~~atoms~~
let S = init-state-l;
$\mathcal{A}$ ← RETURN (); ~~initialisation~~
T ← init-dt CS (to-init-state-l S);
failed ← SPEC (λ- :: bool. True);
if ¬failed then do {
RETURN(failed, fst init-state-l)
} else do {
let T = T;
if get-conflict-l-init T ≠ None
then RETURN (True, fst T)
else if CS = [] then RETURN (True, fst init-state-l)
else do {
ASSERT (extract-atms-clss CS {} ≠ {});
ASSERT (clauses-to-update-l (fst T) = {#});
ASSERT(mset ‘# ran-mf (get-clauses-l (fst T)) + get-unit-clauses-l (fst T) + get-subsumed-clauses-l (fst T) = mset ‘# mset CS);
ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
let T = fst T;
cdcl-twl-stgy-restart-prog-bounded-l T
}
}
}›
unfolding *SAT-l-bounded-def* **by** (*auto cong*: *if-cong Let-def twl-st-l-init*)

lemma *SAT-wl-bounded-SAT-l-bounded*:
assumes
dist: ‹Multiset.Ball (mset ‘# mset CS) distinct-mset› **and**

bounded: ‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*SAT-wl-bounded CS* ≤ ⇓ {((*b*, *T*),(*b*′, *T*′)). *b* =*b*′ ∧ (*b* ⟶ (*T*, *T*′) ∈ *state-wl-l None*)} (*SAT-l-bounded CS*)›
proof −
have *extract-atms-clss*: ‹(*extract-atms-clss CS* {}, ()) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}›
by *auto*
have *init-dt-wl-pre*: ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
by (*rule init-dt-wl-pre*) (*use dist* **in** *auto*)

have *init-rel*: ‹(*to-init-state init-state-wl*, *to-init-state-l init-state-l*)
∈ *state-wl-l-init*›
by (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init*′*-def*
twl-st-l-init-def twl-init-invs to-init-state-def init-state-wl-def
init-state-l-def to-init-state-l-def)[]

— The following stlightly strange theorem allows to reuse the definition and the correctness of *init-dt-wl-heur-full*, which was split in the definition for purely refinement-related reasons.
define *init-dt-wl-rel* **where**
‹*init-dt-wl-rel S* ≡ ({(*T*, *T*′). *RETURN T* ≤ *init-dt-wl*′ *CS S* ∧ *T*′ = ()})› **for** *S*
have *init-dt-wl*′:
‹*init-dt-wl*′ *CS S* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ (*init-dt-wl-rel S*))›
if
‹*init-dt-wl-pre CS S*› **and**
‹(*S*, *S*′) ∈ *state-wl-l-init*› **and**
‹∀ *C*∈*set CS*. *distinct C*›
for *S S*′
proof −
have [*simp*]: ‹(*U*, *U*′) ∈ ({(*T*, *T*′). *RETURN T* ≤ *init-dt-wl*′ *CS S* ∧ *remove-watched T* = *T*′} *O*
state-wl-l-init) ⟷ ((*U*, *U*′) ∈ {(*T*, *T*′). *remove-watched T* = *T*′} *O*
state-wl-l-init ∧ *RETURN U* ≤ *init-dt-wl*′ *CS S*)› **for** *S S*′ *U U*′
by *auto*
have *H*: ‹*A* ≤ ⇓ ({(*S*, *S*′). *P S S*′}) *B* ⟷ *A* ≤ ⇓ ({(*S*, *S*′). *RETURN S* ≤ *A* ∧ *P S S*′}) *B*›
for *A B P R*
by (*simp add*: *pw-conc-inres pw-conc-nofail pw-le-iff p2rel-def*)
have *nofail*: ‹*nofail* (*init-dt-wl*′ *CS S*)›
apply (*rule SPEC-nofail*)
apply (*rule order-trans*)
apply (*rule init-dt-wl*′*-spec*[*unfolded conc-fun-RES*])
using *that* **by** *auto*
have *H*: ‹*A* ≤ ⇓ ({(*S*, *S*′). *P S S*′} *O R*) *B* ⟷ *A* ≤ ⇓ ({(*S*, *S*′). *RETURN S* ≤ *A* ∧ *P S S*′} *O R*) *B*›
for *A B P R*
by (*smt Collect-cong H case-prod-cong conc-fun-chain*)
show *?thesis*
unfolding *init-dt-wl-rel-def*
using *that*
by (*auto simp*: *nofail no-fail-spec-le-RETURN-itself*)
qed

have *conflict-during-init*:
‹((*True*, ([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*)), (*True*, *fst init-state-l*))
∈ {((*b*, *T*), *b*′, *T*′). *b* = *b*′ ∧ (*b* ⟶ (*T*, *T*′) ∈ *state-wl-l None*)}›
by (*auto simp*: *init-state-l-def state-wl-l-def*)

have *init-init-dt*: ‹*RETURN* (*from-init-state T*)
≤ ⇓ ({(*S*, *S*′). *S* = *fst S*′} *O* {(*S* :: *nat twl-st-wl-init-full*, *S*′ :: *nat twl-st-wl-init*).

remove-watched S = S'} O *state-wl-l-init*)
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- $\leq \Downarrow$ *?init-dt* - ›)
if
‹(*extract-atms-clss CS* {}, ($\mathcal{A}$::*unit*)) $\in$ {(x, -). x = *extract-atms-clss CS* {}}› **and**
‹(T, *Ta*) $\in$ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for $\mathcal{A}$ T *Ta*
proof −
have *1*: ‹*RETURN* $T \leq$ *init-dt-wl'* *CS* (*to-init-state init-state-wl*)›
using *that* **by** (*auto simp*: *init-dt-wl-rel-def from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state* T) $\leq \Downarrow$ {(S, S'). S = *fst* S'} (*RETURN* T)›
by (*auto simp*: *RETURN-refine from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state* T) $\leq \Downarrow$ {(S, S'). S = *fst* S'} (*init-dt-wl'* *CS* (*to-init-state init-state-wl*))›
apply (*rule 2*[*THEN order-trans*])
apply (*rule ref-two-step'*)
apply (*rule 1*)
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule 2*)
unfolding *conc-fun-chain*[*symmetric*]
apply (*rule ref-two-step'*)
unfolding *conc-fun-chain*
apply (*rule init-dt-wl'-init-dt*)
apply (*rule init-dt-wl-pre*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def*
init-state-l-def state-wl-l-init-def state-wl-l-init'-def)
subgoal using *assms* **by** *auto*
done
qed

have *cdcl-twl-stgy-restart-prog-wl-D2*: ‹*cdcl-twl-stgy-restart-prog-bounded-wl* U'
$\leq \Downarrow$ {((b, T), (b', T')). b =b' $\wedge$ ($b \longrightarrow$ (T, T') $\in$ *state-wl-l None*)}
(*cdcl-twl-stgy-restart-prog-bounded-l* (*fst* T'))› (**is** *?A*)
if
U': ‹(U', *fst* T') $\in$ {(S, T). (S, T) $\in$ *state-wl-l None* $\wedge$ *correct-watching* S $\wedge$ *blits-in-*$\mathcal{L}_{in}$ S}›
for $\mathcal{A}$ b b' T $\mathcal{A}'$ T' c c' U'
proof −
have *1*: ‹ {(T, T'). (T, T') $\in$ *state-wl-l None*} = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (U')) (U')›
using U' **by** (*auto simp*: *finalise-init-def correct-watching.simps*)
show *?A*
apply (*rule cdcl-twl-stgy-restart-prog-bounded-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def intro*!: *conc-fun-R-mono*)

qed

have *rewatch-st-fst*: ‹*rewatch-st* (*finalise-init* (*from-init-state* T))
$\leq$ *SPEC* (λc. (c, *fst Ta*) $\in$ {(S, T). (S, T) $\in$ *state-wl-l None* $\wedge$ *correct-watching* S $\wedge$ *blits-in-*$\mathcal{L}_{in}$ S})›
(**is** ‹- $\leq$ *SPEC* *?rewatch*›)
if

⟨(*extract-atms-clss CS* {}, $\mathcal{A}$) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}⟩ **and**
T: ⟨(*T*, $\mathcal{A}'$) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)⟩ **and**
T-Ta: ⟨(*from-init-state T*, *Ta*)
∈ {(*S*, *S'*). *S* = *fst S'*} *O*
{(*S*, *S'*). *remove-watched S* = *S'*} *O state-wl-l-init*⟩ **and**
⟨¬ *get-conflict-wl* (*from-init-state T*) ≠ *None*⟩ **and**
⟨¬ *get-conflict-l-init Ta* ≠ *None*⟩
for $\mathcal{A}$ *b ba T* $\mathcal{A}'$ *Ta bb bc*
proof −
have *1*: ⟨*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)⟩
using *T* **unfolding** *init-dt-wl-rel-def* **by** *auto*
have *2*: ⟨*RETURN T* ≤ ⇓ {(*S*, *S'*). *remove-watched S* = *S'*}
(*SPEC* (*init-dt-wl-spec CS* (*to-init-state init-state-wl*)))⟩
using *order-trans*[*OF 1 init-dt-wl'-spec*[*OF init-dt-wl-pre*]] **.**

have *empty-watched*: ⟨*get-watched-wl* (*finalise-init* (*from-init-state T*)) = (λ-. [])⟩
using *1 2 init-dt-wl'-spec*[*OF init-dt-wl-pre*]
by (*cases T*; *cases* ⟨*init-dt-wl CS* (*init-state-wl*, {#})⟩)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init'-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl'-def RES-RETURN-RES)

have *1*: ⟨*length* (*aa* ∝ *x*) ≥ *2*⟩ ⟨*distinct* (*aa* ∝ *x*)⟩
if
struct: ⟨*twl-struct-invs-init*
((*af*,
{#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#},
{#}, *y*, *ac*, {#}, *NS*, *US*, {#}, *ae*),
OC)⟩ **and**
x: ⟨*x* ∈# *dom-m aa*⟩ **and**
learned: ⟨*learned-clss-l aa* = {#}⟩
for *af aa y ac ae x OC NS US*
proof −
have *irred*: ⟨*irred aa x*⟩
using *that* **by** (*cases* ⟨*fmlookup aa x*⟩) (*auto simp*: *ran-m-def dest*!: *multi-member-split*
split: *if-splits*)
have ⟨*Multiset.Ball*
({#*TWL-Clause* (*mset* (*watched-l* (*fst x*))) (*mset* (*unwatched-l* (*fst x*)))
. *x* ∈# *init-clss-l aa*#} +
{#})
struct-wf-twl-cls⟩
using *struct* **unfolding** *twl-struct-invs-init-def fst-conv twl-st-inv.simps*
by *fast*
then show ⟨*length* (*aa* ∝ *x*) ≥ *2*⟩ ⟨*distinct* (*aa* ∝ *x*)⟩
using *x learned in-ran-mf-clause-inI*[*OF x*, *of True*] *irred*
by (*auto simp*: *mset-take-mset-drop-mset'* *dest*!: *multi-member-split*[*of x*]
split: *if-splits*)
qed
have *min-len*: ⟨ *x* ∈# *dom-m* (*get-clauses-wl* (*finalise-init* (*from-init-state T*))) ⟹
distinct (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*) ∧
2 ≤ *length* (*get-clauses-wl* (*finalise-init* (*from-init-state T*)) ∝ *x*)⟩
for *x*
using *2*

by (*cases T*)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init'-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl'-def RES-RETURN-RES
init-dt-spec-def init-state-wl-def twl-st-l-init-def
intro: *1*)

show *?thesis*
apply (*rule rewatch-st-correctness*[*THEN order-trans*])
subgoal by (*rule empty-watched*)
subgoal by (*rule min-len*)
subgoal using *T-Ta* **by** (*auto simp*: *finalise-init-def*
state-wl-l-init-def state-wl-l-init'-def state-wl-l-def
correct-watching-init-correct-watching
correct-watching-init-blits-in-$\mathcal{L}_{in}$)
done
qed

have *all-le*: ‹∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*›
proof (*intro ballI*)
fix *C L*
assume ‹*C* ∈ *set CS*› **and** ‹*L* ∈ *set C*›
then have ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit L* ≤ *uint32-max*›
using *assms* **by** *auto*
qed
have [*simp*]: ‹(*Tc*, *fst Td*) ∈ *state-wl-l None* ⟹
get-conflict-l-init Td = *get-conflict-wl Tc*› **for** *Tc Td*
by (*cases Tc*; *cases Td*; *auto simp*: *state-wl-l-def*)
show *?thesis*
unfolding *SAT-wl-bounded-def SAT-l-bounded-alt-def*
apply (*refine-vcg extract-atms-clss init-dt-wl' init-rel*)
subgoal using *assms* **unfolding** *extract-atms-clss-alt-def* **by** *auto*
subgoal using *assms* **unfolding** *distinct-mset-set-def* **by** *auto*
subgoal by (*rule init-dt-wl-pre*)
subgoal using *dist* **by** *auto*
apply (*rule init-init-dt*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*
state-wl-l-def)
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def*
simp del: *isasat-input-bounded-def*)
subgoal by (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def state-wl-l-init'-def*
state-wl-l-init-def
simp del: *isasat-input-bounded-def*)
subgoal by (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def state-wl-l-init'-def*
state-wl-l-init-def
simp del: *isasat-input-bounded-def*)
apply (*rule rewatch-st-fst*; *assumption*)
subgoal for $\mathcal{A}$ *T* $\mathcal{A}'$ *Ta finished finished'*

unfolding *twl-st-l-init*[*symmetric*]
by (*rule cdcl-twl-stgy-restart-prog-wl-D2*)
done
qed

definition *SAT-bounded′* :: ‹*nat clauses* ⇒ (*bool* × *nat literal list option*) *nres*› **where**
‹*SAT-bounded′ CS* = *do* {
(*b*, *T*) ← *SAT-bounded CS*;
RETURN(*b*, *if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*)
}
›

definition *model-if-satisfiable-bounded* :: ‹*nat clauses* ⇒ (*bool* × *nat literal list option*) *nres*› **where**
‹*model-if-satisfiable-bounded CS* = *SPEC* (λ(*b*, *M*). *b* ⟶
(*if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*))›

lemma *SAT-bounded-model-if-satisfiable*:
‹(*SAT-bounded′*, *model-if-satisfiable-bounded*) ∈ [λ*CS*. (∀ *C* ∈# *CS*. *distinct-mset C*)]$_f$ *Id*→
⟨{((*b*, *S*), (*b′*, *T*)). *b* = *b′* ∧ (*b* ⟶ *S* = *T*)}⟩*nres-rel*›
(**is** ‹- ∈[λ*CS*. *?P CS*]$_f$ *Id* → -›)
proof −
have *H*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*init-state CS*)›
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*init-state CS*)›
if ‹*?P CS*› **for** *CS*
using *that* **by** (*auto simp*:
twl-struct-invs-def twl-st-inv.simps cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-restart-mset.no-strange-atm-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-learned-clause-alt-def cdcl*$_W$*-restart-mset.no-smaller-propa-def*
past-invs.simps clauses-def twl-list-invs-def twl-stgy-invs-def clause-to-update-def
cdcl$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant-def*
cdcl$_W$*-restart-mset.no-smaller-confl-def*
distinct-mset-set-def)
have *H*: ‹*s* ∈ {*M*. *if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*}›
if
dist: ‹*Multiset.Ball CS distinct-mset*› **and**
[*simp*]: ‹*CS′* = *CS*› **and**
s: ‹*s* ∈ (λ*T*. *if conflicting T* = *None then Some* (*map lit-of* (*trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-run CS′* (*init-state CS′*))›
for *s* :: ‹*nat literal list option*› **and** *CS CS′*
proof −
obtain *T* **where**
s: ‹(*s* = *Some* (*map lit-of* (*trail T*)) ∧ *conflicting T* = *None*) ∨
(*s* = *None* ∧ *conflicting T* ≠ *None*)› **and**
conc: ‹*conclusive-CDCL-run CS′* ([], *CS′*, {#}, *None*) *T*›
using *s* **by** *auto force*
consider
n n′ **where** ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-restart-stgy*** (([], *CS′*, {#}, *None*), *n*) (*T*, *n′*)›
‹*no-step cdcl*$_W$*-restart-mset.cdcl*$_W$ *T*› |
‹*CS′* ≠ {#}› **and** ‹*conflicting T* ≠ *None*› **and** ‹*backtrack-lvl T* = *0*› **and**
‹*unsatisfiable* (*set-mset CS′*)›
using *conc* **unfolding** *conclusive-CDCL-run-def*
by *auto*

then show *?thesis*
proof *cases*
case (*1 n n'*) **note** *st* = *this*(*1*) **and** *ns* = *this*(*2*)
have ‹*no-step* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy T*›
using *ns* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-*$cdcl_W$ **by** *blast*
then have *full-T*: ‹*full* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy T T*›
unfolding *full-def* **by** *blast*

have *invs*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant T*›
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv T*›
using *st* $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-dcl*$_W$*-all-struct-inv*[*OF st*]
$cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-dcl*$_W$*-stgy-invariant*[*OF st*]
H[*OF dist*] **by** *auto*
have *res*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-restart*** ([], *CS'*, {#}, *None*) *T*›
using $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-stgy-*$cdcl_W$*-restart*[*OF st*] **by** *simp*
have *ent*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init T*›
using $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-learned-clauses-entailed*[*OF res*] *H*[*OF dist*]
unfolding ‹*CS'* = *CS*› $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *simp*
have [*simp*]: ‹*init-clss T* = *CS*›
using $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-restart-init-clss*[*OF res*] **by** *simp*
show *?thesis*
using $cdcl_W$*-restart-mset.full-*$cdcl_W$*-stgy-inv-normal-form*[*OF full-T invs ent*] *s*
by (*auto simp*: *true-annots-true-cls lits-of-def*)
next
case *2*
moreover have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*init-state CS*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
by *auto*
ultimately show *?thesis*
using *H*[*OF dist*] $cdcl_W$*-restart-mset.full-*$cdcl_W$*-stgy-inv-normal-form*[*of* ‹*init-state CS*›
‹*init-state CS*›] *s*
by *auto*
qed
qed
have *H*: ‹
∃ *s'*∈{(*b*, *M*).
b ⟶
(*if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS*
else M = *None*)}.
(*s*, *s'*) ∈ {((*b*, *S*), *b'*, *T*). *b* = *b'* ∧ (*b* ⟶ *S* = *T*)}›
if ‹*Multiset.Ball CS' distinct-mset*›
‹*CS* = *CS'*› **and**
‹*s* ∈ *uncurry*
(λ*b T*. (*b*, *if conflicting T* = *None then Some* (*map lit-of* (*trail T*))
else None)) ‘
(*if* ¬ *xb then* {(*xb*, *xa*)}
else {(*b*, *U*). *b* ⟶ *conclusive-CDCL-run CS' xa U*})› **and**
‹*xa* ∈ {*T*. *T* = *init-state CS'*}›
for *CS CS'* :: ‹*nat literal multiset multiset*› **and** *s* **and** *xa* **and** *xb* :: *bool*
proof −
obtain *b T* **where**
s: ‹*s* = (*b*, *T*)› **by** (*cases s*)
have
‹¬*xb* ⟶ ¬*b*› **and**

```
    b: ‹b ⟶ T ∈ (λT. if conflicting T = None then Some (map lit-of (trail T)) else None) `
  Collect (conclusive-CDCL-run CS (init-state CS))›
    using that(3,4)
    by (force simp add: image-iff s that split: if-splits)+

   show ?thesis
   proof (cases b)
    case True
    then have T: ‹T ∈ (λT. if conflicting T = None then Some (map lit-of (trail T)) else None) `
      Collect (conclusive-CDCL-run CS (init-state CS))›
      using b by fast
    show ?thesis
      using H[OF that(1,2) T]
      by (rule-tac x = ‹s› in bexI)
        (auto simp add: s True that)
   qed (auto simp: s)
  qed

  have if-RES: ‹(if xb then RETURN x
      else RES P) = (RES (if xb then {x} else P))› for x xb P
    by (auto simp: RETURN-def)
  show ?thesis
    unfolding SAT-bounded'-def model-if-satisfiable-bounded-def SAT-bounded-def Let-def
      nres-monad3
    apply (intro frefI nres-relI)
    apply refine-vcg
    subgoal for CS' CS
      unfolding RES-RETURN-RES RES-RES-RETURN-RES2 if-RES
      apply (rule RES-refine)
      unfolding pair-in-Id-conv bex-triv-one-point1 bex-triv-one-point2
      using H by presburger
    done
qed

lemma SAT-bounded-model-if-satisfiable':
  ‹(uncurry (λ-. SAT-bounded'), uncurry (λ-. model-if-satisfiable-bounded)) ∈
    [λ(-, CS). (∀C ∈# CS. distinct-mset C)]f Id ×r Id→ ⟨{((b, S), (b', T)). b = b' ∧ (b ⟶ S =
T)}⟩nres-rel›
  using SAT-bounded-model-if-satisfiable unfolding fref-def
  by auto

definition SAT-l-bounded' where
  ‹SAT-l-bounded' CS = do{
    (b, S) ← SAT-l-bounded CS;
    RETURN (b, if b ∧ get-conflict-l S = None then Some (map lit-of (get-trail-l S)) else None)
  }›


definition SAT0-bounded' where
  ‹SAT0-bounded' CS = do{
    (b, S) ← SAT0-bounded CS;
    RETURN (b, if b ∧ get-conflict S = None then Some (map lit-of (get-trail S)) else None)
  }›

lemma SAT-l-bounded'-SAT0-bounded':
  assumes ‹Multiset.Ball (mset `# mset CS) distinct-mset›
```

shows ‹$SAT\text{-}l\text{-}bounded'$ $CS \le \Downarrow \{((b, S), (b', T)).\ b = b' \wedge (b \longrightarrow S = T)\}$ ($SAT0\text{-}bounded'$ CS)›
unfolding *SAT-l-bounded′-def SAT0-bounded′-def*
apply *refine-vcg*
apply (*rule SAT-l-bounded-SAT0-bounded*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

lemma *SAT0-bounded′-SAT-bounded′*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*›
shows ‹$SAT0\text{-}bounded'$ $CS \le \Downarrow \{((b, S), (b', T)).\ b = b' \wedge (b \longrightarrow S = T)\}$ ($SAT\text{-}bounded'$ (*mset* ‘# *mset CS*))›
unfolding *SAT-bounded′-def SAT0-bounded′-def*
apply *refine-vcg*
apply (*rule SAT0-bounded-SAT-bounded*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def twl-st-l twl-st*)
done

definition *IsaSAT-bounded* :: ‹*nat clause-l list* ⇒ (*bool* × *nat literal list option*) *nres*› **where**
‹*IsaSAT-bounded CS* = *do*{
  (*b*, *S*) ← *SAT-wl-bounded CS*;
  RETURN (*b*, *if* $b \wedge$ *get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}›

lemma *IsaSAT-bounded-alt-def*:
‹*IsaSAT-bounded CS* = *do*{
  ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
  ASSERT(*distinct-mset-set* (*mset* ‘ *set CS*));
  let $\mathcal{A}_{in}'$ = *extract-atms-clss CS* {};
  S ← *RETURN init-state-wl*;
  T ← *init-dt-wl′ CS* (*to-init-state S*);
  failed ← *SPEC* (λ- :: *bool*. *True*);
  if ¬*failed then do* {
    RETURN (*False*, *extract-stats init-state-wl*)
  } *else do* {
    let T = *from-init-state T*;
    if get-conflict-wl T ≠ *None*
    then RETURN (*True*, *extract-stats T*)
    else if CS = [] *then RETURN* (*True*, *Some* [])
    else do {
      ASSERT (*extract-atms-clss CS* {} ≠ {});
      ASSERT(*isasat-input-bounded-nempty* (*mset-set* $\mathcal{A}_{in}'$));
      ASSERT(*mset* ‘# *ran-mf* (*get-clauses-wl T*) + *get-unit-clauses-wl T* + *get-subsumed-clauses-wl T* = *mset* ‘# *mset CS*);
      ASSERT(*learned-clss-l* (*get-clauses-wl T*) = {#});
      T ← *rewatch-st T*;
      T ← *RETURN* (*finalise-init T*);
      (*b*, *S*) ← *cdcl-twl-stgy-restart-prog-bounded-wl T*;
      RETURN (*b*, *if* $b \wedge$ *get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
    }
  }
}› (**is** ‹*?A* = *?B*›) **for** *CS opts*
proof −
have *H*: ‹$A = B \Longrightarrow A \le \Downarrow Id\ B$› **for** *A B*

by *auto*
have *1*: ‹*?A* ≤ ⇓ *Id ?B*›
unfolding *IsaSAT-bounded-def SAT-wl-bounded-def nres-bind-let-law If-bind-distrib nres-monad-laws Let-def finalise-init-def*
apply (*refine-vcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
apply (*rule H*; *solves auto*)
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

have *2*: ‹*?B* ≤ ⇓ *Id ?A*›
unfolding *IsaSAT-bounded-def SAT-wl-bounded-def nres-bind-let-law If-bind-distrib nres-monad-laws Let-def finalise-init-def*
apply (*refine-vcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
apply (*rule H*; *solves auto*)
subgoal by *auto*
done

show *?thesis*
using *1 2* **by** *simp*
qed

definition *IsaSAT-bounded-heur* :: ‹*opts* ⇒ *nat clause-l list* ⇒ (*bool* × (*bool* × *nat literal list* × *stats*)) *nres*› **where**
‹*IsaSAT-bounded-heur opts CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*);
let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
let $\mathcal{A}_{in}''$ = *virtual-copy* $\mathcal{A}_{in}'$;
let b = *opts-unbounded-mode opts*;
S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur False CS S*;
let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
if isasat-fast-init T ∧ ¬*is-failed-heur-init T*
then do {

if ¬*get-conflict-wl-is-None-heur-init T*
then RETURN (*True*, *empty-init-code*)
else if CS = [] *then do* {*stat* ← *empty-conflict-code*; *RETURN* (*True*, *stat*)}
else do {
ASSERT($\mathcal{A}_{in}''$ ≠ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
- ← *isasat-information-banner T*;
ASSERT((λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*). *fst-As* ≠ *None* ∧
lst-As ≠ *None*) *T*);
ASSERT(*rewatch-heur-st-fast-pre T*);
T ← *rewatch-heur-st-fast T*;
ASSERT(*isasat-fast-init T*);
T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
ASSERT(*isasat-fast T*);
(*b*, *U*) ← *cdcl-twl-stgy-restart-prog-bounded-wl-heur T*;
RETURN (*b*, *if b* ∧ *get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else RETURN (*False*, *empty-init-code*)
}›

definition *empty-conflict-code'* :: ‹(*bool* × - *list* × *stats*) *nres*› **where**
‹*empty-conflict-code'* = *do*{
let M0 = [];
RETURN (*False*, *M0*, (*0*, *0*, *0*, *0*, *0*, *0*, *0*,
0))}›

lemma *IsaSAT-bounded-heur-alt-def*:
‹*IsaSAT-bounded-heur opts CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*);
let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
S ← *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur False CS S*;
failed ← *RETURN* ((*isasat-fast-init T* ∧ ¬*is-failed-heur-init T*));
if ¬*failed*
then do {
RETURN (*False*, *empty-init-code*)
} *else do* {
let T = *convert-state* $\mathcal{A}_{in}'$ *T*;
if ¬*get-conflict-wl-is-None-heur-init T*
then RETURN (*True*, *empty-init-code*)
else if CS = [] *then do* {*stat* ← *empty-conflict-code*; *RETURN* (*True*, *stat*)}
else do {
ASSERT($\mathcal{A}_{in}'$ ≠ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}'$);
ASSERT((λ(*M'*, *N'*, *D'*, *Q'*, *W'*, ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*), *to-remove*), φ, *clvls*). *fst-As* ≠ *None* ∧
lst-As ≠ *None*) *T*);
ASSERT(*rewatch-heur-st-fast-pre T*);

$T \leftarrow$ *rewatch-heur-st-fast* T;
ASSERT(*isasat-fast-init* T);
$T \leftarrow$ *finalise-init-code opts* (T::*twl-st-wl-heur-init*);
ASSERT(*isasat-fast* T);
(b, U) $\leftarrow$ *cdcl-twl-stgy-restart-prog-bounded-wl-heur* T;
RETURN (b, *if* $b \wedge$ *get-conflict-wl-is-None-heur* U *then extract-model-of-state-stat* U
  else extract-state-stat U)
}
}
}›
unfolding *Let-def IsaSAT-bounded-heur-def init-state-wl-heur-fast-def*
bind-to-let-conv isasat-information-banner-def virtual-copy-def
id-apply
unfolding
convert-state-def de-Morgan-disj not-not if-not-swap
by (*intro bind-cong*[*OF refl*] *if-cong*[*OF refl*] *refl*)

lemma *IsaSAT-heur-bounded-IsaSAT-bounded*:
‹*IsaSAT-bounded-heur b CS* $\leq \Downarrow$(*bool-rel* $\times_f$ *model-stat-rel*) (*IsaSAT-bounded CS*)›
proof −
have *init-dt-wl-heur*: ‹*init-dt-wl-heur True CS S* $\leq$
$\Downarrow$(*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True O* {(S, T). S = *remove-watched* $T \wedge$
get-watched-wl (*fst* T) = (λ-. [])})
(*init-dt-wl*′ *CS* T)›
if
‹*case* (*CS*, T) *of*
(*CS*, S) $\Rightarrow$
($\forall$ C∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* C)) $\wedge$
distinct-mset-set (*mset* ‘ *set CS*)› **and**
‹((*CS*, S), *CS*, T) ∈ ⟨*Id*⟩*list-rel* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS S*,
THEN order-trans])
apply (*rule that*(*1*))
apply (*rule that*(*2*))
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl*′*-def RES-RETURN-RES conc-fun-RES*
Image-iff)+
done
qed
have *init-dt-wl-heur-b*: ‹*init-dt-wl-heur False CS S* $\leq$
$\Downarrow$(*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *False O* {(S, T). S = *remove-watched* $T \wedge$
get-watched-wl (*fst* T) = (λ-. [])})
(*init-dt-wl*′ *CS* T)›
if
‹*case* (*CS*, T) *of*
(*CS*, S) $\Rightarrow$
($\forall$ C∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* C)) $\wedge$
distinct-mset-set (*mset* ‘ *set CS*)› **and**
‹((*CS*, S), *CS*, T) ∈ ⟨*Id*⟩*list-rel* $\times_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS S*,

THEN order-trans])
apply (*rule that*(*1*))
using *that*(*2*) **apply** (*force simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl′-def RES-RETURN-RES conc-fun-RES Image-iff*)+
done
qed
have *virtual-copy*: ‹(*virtual-copy* $\mathcal{A}$, ()) ∈ {($\mathcal{B}$, *c*). *c* = () ∧ $\mathcal{B}$ = $\mathcal{A}$}› **for** $\mathcal{B}$ $\mathcal{A}$
by (*auto simp*: *virtual-copy-def*)
have *input-le*: ‹∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *uint32-max*›
if ‹*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {}))›
proof (*intro ballI*)
fix *C L*
assume ‹*C* ∈ *set CS*› **and** ‹*L* ∈ *set C*›
then have ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (*extract-atms-clss CS* {}))›
by (*auto simp*: *extract-atms-clss-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit L* ≤ *uint32-max*›
using *that* **by** *auto*
qed
have *lits-C*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*mset-set* (*extract-atms-clss CS* {})) (*mset C*)›
if ‹*C* ∈ *set CS*› **for** *C CS*
using *that*
by (*force simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
in-all-lits-of-m-ain-atms-of-iff extract-atms-clss-alt-def
atm-of-eq-atm-of)
have *init-state-wl-heur*: ‹*isasat-input-bounded* $\mathcal{A}$ ⟹
init-state-wl-heur $\mathcal{A}$ ≤ *SPEC* (λ*c*. (*c*, *init-state-wl*) ∈
{(*S*, *S′*). (*S*, *S′*) ∈ *twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *True* ∧
inres (*init-state-wl-heur* $\mathcal{A}$) *S*})› **for** $\mathcal{A}$
by (*rule init-state-wl-heur-init-state-wl*[*THEN fref-to-Down-unRET-uncurry0-SPEC*,
of $\mathcal{A}$, *THEN strengthen-SPEC*, *THEN order-trans*])
auto

have *get-conflict-wl-is-None-heur-init*: ‹ (*Tb*, *Tc*)
∈ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* ∧
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})}) ⟹
(¬ *get-conflict-wl-is-None-heur-init Tb*) = (*get-conflict-wl Tc* ≠ *None*)› **for** *Tb Tc U V*
by (*cases V*) (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp′*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *get-conflict-wl-is-None-heur-init3*: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init T*) = (*get-conflict-wl* (*fst Ta*) ≠ *None*)› **for** *T Ta failed faileda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-heur-parsing-no-WL-def*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *finalise-init-nempty*: ‹*x1i* ≠ *None*› ‹*x1j* ≠ *None*›
if
T: ‹(*Tb*, *Tc*)
∈ ({(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* ∧

get-clauses-wl-heur-init S = get-clauses-wl-heur-init U ∧
get-conflict-wl-heur-init S = get-conflict-wl-heur-init U ∧
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) ∧
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) ∧
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})}› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb* =
(*x1*, *x2*)›
for *ba S T Ta Tb Tc uu x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f*
x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k U V
proof −
show ‹*x1i* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
show ‹*x1j* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def*
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff)
qed

have *banner*: ‹*isasat-information-banner*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(-, -). *True*})› **for** *Tb*
by (*auto simp*: *isasat-information-banner-def*)

let *?TT* = ‹*rewatch-heur-st-rewatch-st-rel CS*›
have *finalise-init-code*: ‹*finalise-init-code b*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?A*) **and**
finalise-init-code3: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?B*)
if
T: ‹(*Tb*, *Tc*) ∈ *?TT U V*› **and**
confl: ‹¬ *get-conflict-wl Tc* ≠ *None*› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
clss-CS: ‹*mset* ‘# *ran-mf* (*get-clauses-wl Tc*) + *get-unit-clauses-wl Tc* + *get-subsumed-clauses-wl Tc* =
mset ‘# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl Tc*) = {#}›

for *ba S T Ta Tb Tc u v U V*
proof −
have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl* **by** *auto*
have *2*: ‹*all-atms-st Tc* ≠ {#}›
using *nempty* **unfolding** *all-atms-def all-lits-alt-def clss-CS*[*unfolded add.assoc*]
by (*auto simp*: *extract-atms-clss-alt-def*
all-lits-of-mm-empty-iff)
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *nempty* **unfolding** *all-atms-def all-lits-alt-def clss-CS*[*unfolded add.assoc*]
apply (*auto simp*: *extract-atms-clss-alt-def*
all-lits-of-mm-empty-iff in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def)
by (*metis* (*no-types*, *lifting*) *UN-iff atm-of-all-lits-of-mm*(*2*) *atm-of-lit-in-atms-of*
atms-of-mmltiset atms-of-ms-mset-unfold in-set-mset-eq-in set-image-mset)
have *H*: ‹$A = B \Longrightarrow x \in A \Longrightarrow x \in B$› **for** *A B x*
by *auto*
have *H'*: ‹$A = B \Longrightarrow A\ x \Longrightarrow B\ x$› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-init-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$-*cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$-*all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)
show *?A*
by (*rule finalise-init-finalise-init*[*THEN fref-to-Down-unRET-curry-SPEC*, *of b*])
(*use 1 2 learned 4* **in** *auto*)
then show *?B* **unfolding** *convert-state-def* **by** *auto*
qed

have *get-conflict-wl-is-None-heur-init2*: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)) =
(*get-conflict-wl* (*from-init-state V*) ≠ *None*)› **for** *U V*
by (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def twl-st-heur-parsing-no-WL-def
option-lookup-clause-rel-def convert-state-def from-init-state-def)

have *finalise-init2*: ‹*x1i* ≠ *None*› ‹*x1j* ≠ *None*›
if
T: ‹(*T*, *Ta*)

$\in$ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *b O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
st:
‹*x2g* = (*x1j*, *x2h*)›
‹*x2f* = (*x1i*, *x2g*)›
‹*x2e* = (*x1h*, *x2f*)›
‹*x1f* = (*x1g*, *x2e*)›
‹*x1e* = (*x1f*, *x2i*)›
‹*x2j* = (*x1k*, *x2k*)›
‹*x2d* = (*x1e*, *x2j*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)› **and**
conv: ‹*convert-state* ((*mset-set* (*extract-atms-clss CS* {}))) *T* =
(*x1*, *x2*)›
for *uu ba S T Ta baa uua uub x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x1f
x1g x2e x1h x2f x1i x2g x1j x2h x2i x2j x1k x2k b*
proof −
show ‹*x1i* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff*)
show ‹*x1j* ≠ *None*›
using *T conv nempty*
unfolding *st*
by (*cases x1i*)
(*auto simp*: *convert-state-def twl-st-heur-parsing-def
twl-st-heur-parsing-no-WL-def
isa-vmtf-init-def vmtf-init-def mset-set-empty-iff*)
qed

have *rewatch-heur-st-fast-pre*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
$\in$ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
for *uu ba S T Ta baa uua uub*
proof −
have ‹*valid-arena* (*get-clauses-wl-heur-init T*) (*get-clauses-wl* (*fst Ta*))
(*set* (*get-vdom-heur-init T*))›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
using *length-le* **unfolding** *rewatch-heur-st-fast-pre-def convert-state-def
isasat-fast-init-def uint64-max-def uint32-max-def*
by (*auto dest*: *valid-arena-in-vdom-le-arena*)
qed
have *rewatch-heur-st-fast-pre2*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *T*)›
if

T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
and
failed: ‹¬*is-failed-heur-init T*›
for *uu ba S T Ta baa uua uub*
proof −
have
Ta: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
using *failed T* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
from *rewatch-heur-st-fast-pre*[*OF this length-le*]
show *?thesis* **by** *simp*
qed
have *finalise-init-code2*: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ {(*S'*, *T'*).
(*S'*, *T'*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S'*})›
if
Ta: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
confl: ‹¬ *get-conflict-wl* (*from-init-state Ta*) ≠ *None*› **and**
‹*CS* ≠ []› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))› **and**
clss-CS: ‹*mset* ‘# *ran-mf* (*get-clauses-wl* (*from-init-state Ta*)) +
get-unit-clauses-wl (*from-init-state Ta*) + *get-subsumed-clauses-wl* (*from-init-state Ta*) =
mset ‘# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl* (*from-init-state Ta*)) = {#}› **and**
‹*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})) ≠ {#}› **and**
‹*isasat-input-bounded-nempty*
(*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))› **and**
‹*case convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T of*
(*M'*, *N'*, *D'*, *Q'*, *W'*, *xa*, *xb*) ⇒
(*case xa of*
(*x*, *xa*) ⇒
(*case x of*
(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ⇒
λ*to-remove* (φ, *clvls*). *fst-As* ≠ *None* ∧ *lst-As* ≠ *None*)
xa)
xb› **and**
T: ‹(*Tb*, *Tc*) ∈ *?TT T Ta*› **and**
failed: ‹¬*is-failed-heur-init T*›
for *uu ba S T Ta baa uua uub V W b Tb Tc*
proof −
have
Ta: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}›
using *failed Ta* **by** (*cases T*; *cases Ta*) (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)

have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl T* **by** (*auto simp*: *from-init-state-def*)

have *Ta-Tc*: ‹*all-atms-st Tc* = *all-atms-st* (*from-init-state Ta*)›
using *T Ta*
unfolding *all-lits-alt-def mem-Collect-eq prod.case relcomp.simps*
all-atms-def add.assoc **apply** −
apply *normalize-goal*+
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def*
twl-st-heur-parsing-no-WL-def twl-st-heur-parsing-def
from-init-state-def)
moreover have *3*: ‹*set-mset* (*all-atms-st* (*from-init-state Ta*)) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
unfolding *all-lits-alt-def mem-Collect-eq prod.case relcomp.simps*
all-atms-def clss-CS[*unfolded add.assoc*] **apply** −
by (*auto simp*: *extract-atms-clss-alt-def*
atm-of-all-lits-of-mm atms-of-ms-def)
ultimately have *2*: ‹*all-atms-st Tc* ≠ {#}›
using *nempty*
by *auto*

have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-init-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›] *from-init-state-def*
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$-*cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$-*all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)

show *?thesis*
apply (*rule finalise-init-finalise-init-full*[*unfolded conc-fun-RETURN*,
THEN order-trans])
by (*use 1 2 learned 4 T* **in** ‹*auto simp*: *from-init-state-def convert-state-def*›)
qed
have *isasat-fast*: ‹*isasat-fast Td*›
if
fast: ‹¬ ¬ *isasat-fast-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {})))
T)› **and**
Tb: ‹(*Tb*, *Tc*) ∈ *?TT T Ta*› **and**
Td: ‹(*Td*, *Te*)
∈ {(*S'*, *T'*).
(*S'*, *T'*) ∈ *twl-st-heur* ∧

get-clauses-wl-heur-init Tb = *get-clauses-wl-heur S′*}›
for *uu ba S T Ta baa uua uub Tb Tc Td Te*
proof −
show *?thesis*
using *fast Td Tb*
by (*auto simp*: *convert-state-def isasat-fast-init-def sint64-max-def*
uint32-max-def uint64-max-def isasat-fast-def)
qed
define *init-succesfull* **where** ‹*init-succesfull T* = *RETURN* ((*isasat-fast-init T* ∧ ¬ *is-failed-heur-init T*))› **for** *T*
define *init-succesfull2* **where** ‹*init-succesfull2* = *SPEC* (λ- :: *bool*. *True*)›
have [*refine*]: ‹*init-succesfull T* ≤ ⇓ {(*b*, *b′*). (*b* = *b′*) ∧ (*b* ⟷ (*isasat-fast-init T* ∧ ¬ *is-failed-heur-init T*))}
init-succesfull2› **for** *T*
by (*auto simp*: *init-succesfull-def init-succesfull2-def intro*!: *RETURN-RES-refine*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *IsaSAT-bounded-heur-alt-def IsaSAT-bounded-alt-def init-succesfull-def*[*symmetric*]
apply (*rewrite at* ‹*do* {- ← *init-dt-wl′* - -; - ← (¤ :: *bool nres*); *If* - - - }› *init-succesfull2-def*[*symmetric*])
apply (*refine-vcg virtual-copy init-state-wl-heur banner*)
subgoal by (*rule input-le*)
subgoal by (*rule distinct-mset-mset-set*)

apply (*rule init-dt-wl-heur-b*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal by(*auto simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
subgoal by *auto*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*
empty-init-code-def)
subgoal unfolding *from-init-state-def convert-state-def*
by (*rule get-conflict-wl-is-None-heur-init3*)
subgoal by (*simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*rule finalise-init2*)
subgoal by (*rule finalise-init2*)
subgoal for *uu ba S T Ta baa*
by (*rule rewatch-heur-st-fast-pre2*; *assumption?*)
(*clarsimp-all simp add*: *convert-state-def*)
apply (*rule rewatch-heur-st-rewatch-st3*[*unfolded virtual-copy-def id-apply*]; *assumption?*)
subgoal by *auto*
subgoal by (*clarsimp simp add*: *isasat-fast-init-def convert-state-def*)
apply (*rule finalise-init-code2*; *assumption?*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
apply (*rule-tac r1* = ‹*length* (*get-clauses-wl-heur Td*)› **in**
cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D[*THEN fref-to-Down*])
subgoal by (*simp add*: *isasat-fast-def sint64-max-def uint32-max-def*

uint64-max-def)
subgoal by *fast*
subgoal by *simp*
subgoal premises *p*
using *p*(*28*−)
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done
qed

lemma *ISASAT-bounded-SAT-l-bounded′*:
assumes ‹*Multiset.Ball* (*mset* ‘# *mset CS*) *distinct-mset*› **and**
‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-bounded CS* ≤ ⇓ {((*b*, *S*), (*b*′, *S*′)). *b* = *b*′ ∧ (*b* ⟶ *S* = *S*′)} (*SAT-l-bounded′ CS*)›
unfolding *IsaSAT-bounded-def SAT-l-bounded′-def*
apply *refine-vcg*
apply (*rule SAT-wl-bounded-SAT-l-bounded*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

lemma *IsaSAT-bounded-heur-model-if-sat*:
assumes ‹∀ *C* ∈# *mset* ‘# *mset CS*. *distinct-mset C*› **and**
‹*isasat-input-bounded* (*mset-set* ($\bigcup$ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-bounded-heur opts CS* ≤ ⇓ {((*b*, *m*), (*b*′, *m*′)). *b*=*b*′ ∧ (*b* ⟶ (*m*,*m*′) ∈ *model-stat-rel*)}
(*model-if-satisfiable-bounded* (*mset* ‘# *mset CS*))›
apply (*rule IsaSAT-heur-bounded-IsaSAT-bounded*[*THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule ISASAT-bounded-SAT-l-bounded′*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-l-bounded′-SAT0-bounded′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT0-bounded′-SAT-bounded′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-bounded-model-if-satisfiable*[*THEN fref-to-Down*, *of* ‹*mset* ‘# *mset CS*›])
subgoal using *assms* **by** *auto*

subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule conc-fun-R-mono*)
apply *standard*
apply (*clarsimp simp*: *model-stat-rel-def*)
done

lemma *IsaSAT-bounded-heur-model-if-sat'*:
‹(*uncurry IsaSAT-bounded-heur*, *uncurry* (λ-. *model-if-satisfiable-bounded*)) ∈
[λ(-, *CS*). (∀ *C* ∈# *CS*. *distinct-mset C*) ∧
(∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*)]$_f$
Id $\times_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨{((*b*, *m*), (*b'*, *m'*)). *b*=*b'* ∧ (*b* ⟶ (*m*,*m'*) ∈ *model-stat-rel*)}⟩*nres-rel*›
proof −
have *H*: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
if *CS-p*: ‹∀ *C*∈#*CS'*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*› **and**
‹(*CS*, *CS'*) ∈ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*›
for *CS CS'*
unfolding *isasat-input-bounded-def*
proof
fix *L*
assume *L*: ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
then obtain *C* **where**
L: ‹*C*∈*set CS* ∧ (*L* ∈*set C* ∨ − *L* ∈ *set C*)›
apply (*cases L*)
apply (*auto simp*: *extract-atms-clss-alt-def uint32-max-def*
$\mathcal{L}_{all}$-*def*)+
apply (*metis literal.exhaust-sel*)+
done
have ‹*nat-of-lit L* ≤ *uint32-max* ∨ *nat-of-lit* (−*L*) ≤ *uint32-max*›
using *L CS-p that* **by** (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
then show ‹*nat-of-lit L* ≤ *uint32-max*›
using *L*
by (*cases L*) (*auto simp*: *extract-atms-clss-alt-def uint32-max-def*)
qed
show *?thesis*
apply (*intro frefI nres-relI*)
unfolding *uncurry-def*
apply *clarify*
subgoal for *o1 o2 o3 CS o1' o2' o3' CS'*
apply (*rule IsaSAT-bounded-heur-model-if-sat*[*THEN order-trans*, *of CS* - ‹(*o1*, *o2*, *o3*)›])
subgoal by (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
subgoal by (*rule H*) *auto*
apply (*auto simp*: *list-mset-rel-def mset-rel-def br-def*
br-def mset-rel-def p2rel-def rel-mset-def
rel2p-def[*abs-def*] *list-all2-op-eq-map-right-iff'*)
done
done
qed

end

theory *IsaSAT-LLVM*
imports *Version IsaSAT-CDCL-LLVM*
IsaSAT-Initialisation-LLVM Version IsaSAT
IsaSAT-Restart-LLVM
begin

Chapter 22

Code of Full IsaSAT

abbreviation *model-stat-assn* **where**
‹*model-stat-assn* ≡ *bool1-assn* $\times_a$ (*arl64-assn unat-lit-assn*) $\times_a$ *stats-assn*›

abbreviation *model-stat-assn*$_0$ ::
bool ×
nat literal list ×
64 word ×
64 word × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*
⇒ *1 word* ×
(*64 word* × *64 word* × *32 word ptr*) ×
64 word ×
64 word × *64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*
⇒ *llvm-amemory* ⇒ *bool*
where
‹*model-stat-assn*$_0$ ≡ *bool1-assn* $\times_a$ (*al-assn unat-lit-assn*) $\times_a$ *stats-assn*›

abbreviation *lits-with-max-assn* :: ‹*nat multiset*
⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* $\times_a$ *uint32-nat-assn*) *lits-with-max-rel*›

abbreviation *lits-with-max-assn*$_0$ :: ‹*nat multiset*
⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*lits-with-max-assn*$_0$ ≡ *hr-comp* (*al-assn atom-assn* $\times_a$ *unat32-assn*) *lits-with-max-rel*›

lemma *lits-with-max-assn-alt-def*: ‹*lits-with-max-assn* = *hr-comp* (*arl64-assn atom-assn* $\times_a$ *uint32-nat-assn*)
(*lits-with-max-rel O* ⟨*nat-rel*⟩*IsaSAT-Initialisation.mset-rel*)›
proof −

have *1*: ‹(*lits-with-max-rel O* ⟨*nat-rel*⟩*IsaSAT-Initialisation.mset-rel*) = *lits-with-max-rel*›
by (*auto simp*: *mset-rel-def p2rel-def rel2p-def*[*abs-def*] *br-def*
rel-mset-def lits-with-max-rel-def list-rel-def list-all2-op-eq-map-right-iff′ list.rel-eq)
show *?thesis*
unfolding *1*
by *auto*
qed

lemma *init-state-wl-D′-code-isasat*: ‹(*hr-comp isasat-init-assn*
(*Id* $\times_f$
(*Id* $\times_f$
(*Id* $\times_f$
(*nat-rel* $\times_f$

$(\langle\langle Id\rangle list\text{-}rel\rangle list\text{-}rel \times_f$
$(Id \times_f (\langle bool\text{-}rel\rangle list\text{-}rel \times_f (nat\text{-}rel \times_f (Id \times_f (Id \times_f Id)))))))))))) = isasat\text{-}init\text{-}assn\rangle$
by *auto*

definition *model-assn* **where**
‹*model-assn* = *hr-comp model-stat-assn model-stat-rel*›

lemma *extract-model-of-state-stat-alt-def*:
‹*RETURN o extract-model-of-state-stat* = (λ((*M*, *M'*), *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*,
outl, *stats*,
heur, *vdom*, *avdom*, *lcount*, *opts*, *old-arena*).
do {*mop-free M'*; *mop-free N'*; *mop-free D'*; *mop-free j*; *mop-free W'*; *mop-free vm*;
mop-free clvls;
mop-free cach; *mop-free lbd*; *mop-free outl*; *mop-free heur*;
mop-free vdom; *mop-free avdom*; *mop-free opts*;
mop-free old-arena;
RETURN (*False*, *M*, *stats*)
})›
by (*auto simp*: *extract-model-of-state-stat-def mop-free-def intro*!: *ext*)

schematic-goal *mk-free-lookup-clause-rel-assn*[*sepref-frame-free-rules*]: *MK-FREE lookup-clause-rel-assn ?fr*
unfolding *conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: *MK-FREE conflict-option-rel-assn ?fr*
unfolding *conflict-option-rel-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-vmtf-remove-assn*[*sepref-frame-free-rules*]: *MK-FREE vmtf-remove-assn ?fr*
unfolding *vmtf-remove-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-cach-refinement-l-assn*[*sepref-frame-free-rules*]: *MK-FREE cach-refinement-l-assn ?fr*
unfolding *cach-refinement-l-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: *MK-FREE lbd-assn ?fr*
unfolding *lbd-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-opts-assn*[*sepref-frame-free-rules*]: *MK-FREE opts-assn ?fr*
unfolding *opts-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

schematic-goal *mk-free-heuristic-assn*[*sepref-frame-free-rules*]: *MK-FREE heuristic-assn ?fr*
unfolding *heuristic-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

thm *array-mk-free*

context
  fixes *l-dummy* :: *'l*::*len2 itself*
  fixes *ll-dummy* :: *'ll*::*len2 itself*
  fixes *L LL AA*
  defines [*simp*]: *L* ≡ (*LENGTH* (*'l*))
  defines [*simp*]: *LL* ≡ (*LENGTH* (*'ll*))
  defines [*simp*]: *AA* ≡ *raw-aal-assn TYPE*(*'l*::*len2*) *TYPE*(*'ll*::*len2*)
begin
 private lemma *n-unf*: *hr-comp AA* (⟨⟨*the-pure A*⟩*list-rel*⟩*list-rel*) = *aal-assn A* **unfolding** *aal-assn-def AA-def* **..**

  context
    notes [*fcomp-norm-unfold*] = *n-unf*
  begin

lemma *aal-assn-free*[*sepref-frame-free-rules*]: *MK-FREE AA aal-free*
  apply *rule* **by** *vcg*
  sepref-decl-op *list-list-free*: λ-::- *list list*. () :: ⟨⟨*A*⟩*list-rel*⟩*list-rel* → *unit-rel* **.**

lemma *hn-aal-free-raw*: (*aal-free*,*RETURN o op-list-list-free*) ∈ $AA^d \rightarrow_a$ *unit-assn*
  by *sepref-to-hoare vcg*

  sepref-decl-impl *aal-free*: *hn-aal-free-raw*
   .

  lemmas *array-mk-free*[*sepref-frame-free-rules*] = *hn-MK-FREEI*[*OF aal-free-hnr*]
end
end

schematic-goal *mk-free-isasat-init-assn*[*sepref-frame-free-rules*]: *MK-FREE isasat-init-assn ?fr*
  unfolding *isasat-init-assn-def*
  by (*rule free-thms sepref-frame-free-rules*)+

sepref-def *extract-model-of-state-stat*
  is ‹*RETURN o extract-model-of-state-stat*›
  :: ‹$isasat\text{-}bounded\text{-}assn^d \rightarrow_a$ *model-stat-assn*›
  supply [[*goals-limit=1*]]
  unfolding *extract-model-of-state-stat-alt-def isasat-bounded-assn-def*
   trail-pol-fast-assn-def
  by *sepref*

lemmas [*sepref-fr-rules*] = *extract-model-of-state-stat.refine*

lemma *extract-state-stat-alt-def*:
  ‹*RETURN o extract-state-stat* = (λ(*M*, *N'*, *D'*, *j*, *W'*, *vm*, *clvls*, *cach*, *lbd*, *outl*, *stats*,
     heur,
     vdom, *avdom*, *lcount*, *opts*, *old-arena*).
    do {*mop-free M*; *mop-free N'*; *mop-free D'*; *mop-free j*; *mop-free W'*; *mop-free vm*;
      mop-free clvls;
      mop-free cach; *mop-free lbd*; *mop-free outl*; *mop-free heur*;
      mop-free vdom; *mop-free avdom*; *mop-free opts*;
      mop-free old-arena;
     RETURN (*True*, [], *stats*)})›
  by (*auto simp*: *extract-state-stat-def mop-free-def intro*!: *ext*)

sepref-def *extract-state-stat*

is ‹*RETURN o extract-state-stat*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *model-stat-assn*›
supply [[*goals-limit=1*]]
unfolding *extract-state-stat-alt-def isasat-bounded-assn-def*
al-fold-custom-empty[**where** *'l=64*]
by *sepref*

lemma *convert-state-hnr*:
‹(*uncurry* (*return oo* (λ- *S. S*)), *uncurry* (*RETURN oo convert-state*))
∈ *ghost-assn*^{k} $*_a$ (*isasat-init-assn*)^d $\rightarrow_a$
isasat-init-assn›
unfolding *convert-state-def*
by *sepref-to-hoare vcg*

sepref-def *IsaSAT-use-fast-mode-impl*
is ‹*uncurry0* (*RETURN IsaSAT-use-fast-mode*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *IsaSAT-use-fast-mode-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *IsaSAT-use-fast-mode-impl.refine extract-state-stat.refine*

sepref-def *empty-conflict-code'*
is ‹*uncurry0* (*empty-conflict-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-assn*›
unfolding *empty-conflict-code-def*
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *annotate-assn*[**where** *A*=‹*arl64-assn unat-lit-assn*›])
by *sepref*

declare *empty-conflict-code'.refine*[*sepref-fr-rules*]

sepref-def *empty-init-code'*
is ‹*uncurry0* (*RETURN empty-init-code*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *model-stat-assn*›
unfolding *empty-init-code-def al-fold-custom-empty*[**where** *'l=64*]
apply (*rewrite* **in** ‹*RETURN* (-, ⌑,-)› *annotate-assn*[**where** *A*=‹*arl64-assn unat-lit-assn*›])
by *sepref*

declare *empty-init-code'.refine*[*sepref-fr-rules*]

sepref-register *init-dt-wl-heur-full*

sepref-register *to-init-state from-init-state get-conflict-wl-is-None-init extract-stats*
init-dt-wl-heur

definition *isasat-fast-bound* :: ‹*nat*› **where**
‹*isasat-fast-bound* = *sint64-max* − (*uint32-max div 2* + *6*)›

lemma *isasat-fast-bound-alt-def*: ‹*isasat-fast-bound* = *9223372034707292154*›
unfolding *isasat-fast-bound-def sint64-max-def uint32-max-def*
by *simp*

sepref-def *isasat-fast-bound-impl*
is ‹*uncurry0* (*RETURN isasat-fast-bound*)›

:: ‹*unit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
unfolding *isasat-fast-bound-alt-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

lemmas [*sepref-fr-rules*] = *isasat-fast-bound-impl.refine*

lemma *isasat-fast-init-alt-def*:
‹*RETURN o isasat-fast-init* = (λ(*M*, *N*, -). *RETURN* (*length N* $\leq$ *isasat-fast-bound*))›
by (*auto simp*: *isasat-fast-init-def uint64-max-def uint32-max-def isasat-fast-bound-def intro*!: *ext*)

sepref-def *isasat-fast-init-code*
is ‹*RETURN o isasat-fast-init*›
:: ‹*isasat-init-assn*^{k} $\rightarrow_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-fast-init-alt-def isasat-init-assn-def isasat-fast-bound-def*[*symmetric*]
by *sepref*

declare *isasat-fast-init-code.refine*[*sepref-fr-rules*]

declare *convert-state-hnr*[*sepref-fr-rules*]

sepref-register
cdcl-twl-stgy-restart-prog-wl-heur

declare *init-state-wl-D'-code.refine*[*FCOMP init-state-wl-D'*[*unfolded convert-fref*],
unfolded lits-with-max-assn-alt-def[*symmetric*] *init-state-wl-heur-fast-def*[*symmetric*],
unfolded init-state-wl-D'-code-isasat, *sepref-fr-rules*]

thm *init-state-wl-D'-code.refine*[*FCOMP init-state-wl-D'*[*unfolded convert-fref*],
unfolded lits-with-max-assn-alt-def[*symmetric*]]

lemma [*sepref-fr-rules*]: ‹(*init-state-wl-D'-code*, *init-state-wl-heur-fast*)
$\in$ [λx. *distinct-mset x* $\wedge$
($\forall$ *L*$\in$#$\mathcal{L}_{all}$ *x*.
nat-of-lit L
$\leq$ *uint32-max*)]$_a$ *lits-with-max-assn*^{k} $\rightarrow$ *isasat-init-assn*›
using *init-state-wl-D'-code.refine*[*FCOMP init-state-wl-D'*[*unfolded convert-fref*]]
unfolding *lits-with-max-assn-alt-def*[*symmetric*] *init-state-wl-D'-code-isasat*
init-state-wl-heur-fast-def
by *auto*

lemma *is-failed-heur-init-alt-def*:
‹*is-failed-heur-init* = (λ(-, -, -, -, -, -, -, -, -, -, -, *failed*). *failed*)›
by (*auto*)

sepref-def *is-failed-heur-init-impl*
is ‹*RETURN o is-failed-heur-init*›
:: ‹*isasat-init-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *isasat-init-assn-def is-failed-heur-init-alt-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *is-failed-heur-init-impl.refine*

definition *ghost-assn* **where** ‹*ghost-assn* = *hr-comp unit-assn virtual-copy-rel*›

lemma [*sepref-fr-rules*]: ‹(*return o* (λ-. ()), *RETURN o virtual-copy*) ∈ *lits-with-max-assn*^{k} $\rightarrow_a$ *ghost-assn*›
proof −
  have [*simp*]: ‹(λ*s*. (∃ *xa*. (↑(*xa* = *x*)) *s*)) = (↑ *True*)› **for** *s* :: ‹′*b*::*sep-algebra*› **and** *x* :: ′*a*
    by (*auto simp*: *pred-lift-extract-simps*)

  show *?thesis*
    unfolding *virtual-copy-def ghost-assn-def virtual-copy-rel-def*
    apply *sepref-to-hoare*
    apply *vcg′*
    apply (*auto simp*: *ENTAILS-def hr-comp-def snat-rel-def pure-true-conv*)
    apply (*rule Defer-Slot.remove-slot*)
    done
qed

sepref-register *virtual-copy empty-conflict-code empty-init-code*
  isasat-fast-init is-failed-heur-init
  extract-model-of-state-stat extract-state-stat
  isasat-information-banner
  finalise-init-code
  IsaSAT-Initialisation.rewatch-heur-st-fast
  get-conflict-wl-is-None-heur
  cdcl-twl-stgy-prog-bounded-wl-heur
  get-conflict-wl-is-None-heur-init
  convert-state

lemma *isasat-information-banner-alt-def*:
  ‹*isasat-information-banner S* =
    RETURN (())›
  by (*auto simp*: *isasat-information-banner-def*)

schematic-goal *mk-free-ghost-assn*[*sepref-frame-free-rules*]: *MK-FREE ghost-assn ?fr*
  unfolding *ghost-assn-def*
  by (*rule free-thms sepref-frame-free-rules*)+

sepref-def *IsaSAT-code*
  is ‹*uncurry IsaSAT-bounded-heur*›
  :: ‹*opts-assn*^{d} $*_a$ (*clauses-ll-assn*)^k $\rightarrow_a$ *bool1-assn* $\times_a$ *model-stat-assn*›
  supply [[*goals-limit=1*]] *isasat-fast-init-def*[*simp*]
  unfolding *IsaSAT-bounded-heur-def empty-conflict-def*[*symmetric*]
    get-conflict-wl-is-None extract-model-of-state-def[*symmetric*]
    extract-stats-def[*symmetric*] *init-dt-wl-heur-b-def*[*symmetric*]
    length-get-clauses-wl-heur-init-def[*symmetric*]
    init-dt-step-wl-heur-unb-def[*symmetric*] *init-dt-wl-heur-unb-def*[*symmetric*]
    length-0-conv[*symmetric*] *op-list-list-len-def*[*symmetric*]
    isasat-information-banner-alt-def
  supply *get-conflict-wl-is-None-heur-init-def*[*simp*]
  supply *get-conflict-wl-is-None-def*[*simp*]
   option.splits[*split*]
   extract-stats-def[*simp del*]
  apply (*rewrite at* ‹*extract-atms-clss* - ¤› *op-extract-list-empty-def*[*symmetric*])
  apply (*rewrite at* ‹*extract-atms-clss* - ¤› *op-extract-list-empty-def*[*symmetric*])
  apply (*annot-snat-const TYPE*(*64*))
  by *sepref*

definition *default-opts* **where**

⟨*default-opts* = (*True*, *True*, *True*)⟩

sepref-def *default-opts-impl*
is ⟨*uncurry0* (*RETURN default-opts*)⟩
:: ⟨$\textit{unit-assn}^k \rightarrow_a$ *opts-assn*⟩
unfolding *opts-assn-def default-opts-def*
by *sepref*

definition *IsaSAT-bounded-heur-wrapper* :: ⟨- ⇒ (*nat*) *nres*⟩**where**
⟨*IsaSAT-bounded-heur-wrapper C* = *do* {
(*b*, (*b*′, -)) ← *IsaSAT-bounded-heur default-opts C*;
RETURN ((*if b then 2 else 0*) + (*if b*′ *then 1 else 0*))
}⟩

The calling convention of LLVM and clang is not the same, so returning the model is currently unsupported. We return only the flags (as ints, not as bools) and the statistics.

sepref-register *IsaSAT-bounded-heur default-opts*
sepref-def *IsaSAT-code-wrapped*
is ⟨*IsaSAT-bounded-heur-wrapper*⟩
:: ⟨$(\textit{clauses-ll-assn})^k \rightarrow_a$ *sint64-nat-assn*⟩
supply [[*goals-limit=1*]] *if-splits*[*split*]
unfolding *IsaSAT-bounded-heur-wrapper-def*
apply (*annot-snat-const TYPE*(*64*))
by *sepref*

The setup to transmit the version is a bit complicated, because it LLVM does not support direct export of string literals. Therefore, we actually convert the version to an array chars (more precisely, of machine words – ended with 0) that can be read and printed in isasat.

function *array-of-version* **where**
⟨*array-of-version i str arr* =
(*if i* ≥ *length str then arr*
else array-of-version (*i+1*) *str* (*arr*[*i* := *str* ! *i*]))⟩
by *pat-completeness auto*
termination
apply (*relation* ⟨*measure* (λ(*i*,*str*, *arr*). *length str* − *i*)⟩)
apply *auto*
done

sepref-definition *llvm-version*
is ⟨*uncurry0* (*RETURN* (
let str = *map* (*nat-of-integer o* (*of-char* :: - ⇒ *integer*)) (*String.explode Version.version*) @ [*0*] *in*
array-of-version 0 str (*replicate* (*length str*) *0*)))⟩
:: ⟨$\textit{unit-assn}^k \rightarrow_a$ *array-assn sint32-nat-assn*⟩
supply[[*goals-limit=1*]]
unfolding *Version.version-def String.explode-code*
String.asciis-of-Literal
apply (*auto simp*: *String.asciis-of-Literal of-char-of char-of-char nat-of-integer-def*
simp del: *list-update.simps replicate.simps*)
apply (*annot-snat-const TYPE*(*32*))
unfolding *array-fold-custom-replicate*
unfolding *hf-pres.simps*[*symmetric*]
by *sepref*

experiment
begin

lemmas [*llvm-code*] = *llvm-version-def*

lemmas [*llvm-inline*] =
unit-propagation-inner-loop-body-wl-fast-heur-code-def
NORMAL-PHASE-def DEFAULT-INIT-PHASE-def QUIET-PHASE-def
find-unwatched-wl-st-heur-fast-code-def
update-clause-wl-fast-code-def

export-llvm
IsaSAT-code-wrapped **is** ‹*int64-t IsaSAT-code-wrapped*(*CLAUSES*)›
llvm-version **is** ‹*STRING-VERSION llvm-version*›
default-opts-impl
IsaSAT-code
opts-restart-impl
count-decided-pol-impl **is** ‹*uint32-t count-decided-st-heur-pol-fast*(*TRAIL*)›
arena-lit-impl **is** ‹*uint32-t arena-lit-impl*(*ARENA*, *int64-t*)›
defines ‹
typedef struct {*int64-t size*; *struct* {*int64-t used*; *uint32-t* ∗*clause*;};} *CLAUSE*;
typedef struct {*int64-t num-clauses*; *CLAUSE* ∗*clauses*;} *CLAUSES*;

typedef struct {*int64-t size*; *struct* {*int64-t capacity*; *int32-t* ∗*data*;};} *ARENA*;
typedef int32-t∗ *STRING-VERSION*;

typedef struct {*int64-t size*; *struct* {*int64-t capacity*; *uint32-t* ∗*data*;};} *RAW-TRAIL*;
typedef struct {*int64-t size*; *int8-t* ∗*polarity*;} *POLARITY*;
typedef struct {*int64-t size*; *int32-t* ∗*level*;} *LEVEL*;
typedef struct {*int64-t size*; *int64-t* ∗*reasons*;} *REASONS*;
typedef struct {*int64-t size*; *struct* {*int64-t capacity*; *int32-t* ∗*data*;};} *CONTROL-STACK*;
typedef struct {*RAW-TRAIL raw-trail*;
struct {*POLARITY pol*;
struct {*LEVEL lev*;
struct {*REASONS resasons*;
struct {*int32-t dec-lev*;
CONTROL-STACK cs;};};};};} *TRAIL*;
›
file *code/isasat-restart.ll*

end

definition *model-bounded-assn* **where**
‹*model-bounded-assn* =
hr-comp (*bool1-assn* $\times_a$ *model-stat-assn*$_0$)
{((*b*, *m*), (*b'*, *m'*)). *b*=*b'* ∧ (*b* ⟶ (*m*,*m'*) ∈ *model-stat-rel*)}›

definition *clauses-l-assn* **where**
‹*clauses-l-assn* = *hr-comp* (*IICF-Array-of-Array-List.aal-assn*
unat-lit-assn)
(*list-mset-rel O*
⟨*list-mset-rel*⟩*IsaSAT-Initialisation.mset-rel*)›

theorem *IsaSAT-full-correctness*:
‹(*uncurry IsaSAT-code*, *uncurry* (λ-. *model-if-satisfiable-bounded*))
∈ [λ(-, *a*). *Multiset.Ball a distinct-mset* ∧
(∀ *C*∈#*a*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *uint32-max*)]$_a$ *opts-assn*^{d} $*_a$ *clauses-l-assn*^{k} → *model-bounded-assn*›
using *IsaSAT-code.refine*[*FCOMP IsaSAT-bounded-heur-model-if-sat'*[*unfolded convert-fref*]]
unfolding *model-bounded-assn-def clauses-l-assn-def*

apply *auto*
done

end