

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

January 20, 2020

Contents

0.1	Rewrite Systems and Properties	3
0.1.1	Lifting of Rewrite Rules	3
0.1.2	Consistency Preservation	4
0.1.3	Full Lifting	5
0.2	Transformation testing	5
0.2.1	Definition and first Properties	5
0.2.2	Invariant conservation	7
0.3	Rewrite Rules	9
0.3.1	Elimination of the Equivalences	9
0.3.2	Eliminate Implication	10
0.3.3	Eliminate all the True and False in the formula	11
0.3.4	PushNeg	15
0.3.5	Push Inside	17
0.4	The Full Transformations	22
0.4.1	Abstract Definition	22
0.4.2	Conjunctive Normal Form	24
0.4.3	Disjunctive Normal Form	24
0.5	More aggressive simplifications: Removing true and false at the beginning	25
0.5.1	Transformation	25
0.5.2	More invariants	26
0.5.3	The new CNF and DNF transformation	26
0.6	Link with Multiset Version	27
0.6.1	Transformation to Multiset	27
0.6.2	Equisatisfiability of the two Versions	27

theory *Prop-Abstract-Transformation*

imports *Entailment-Definition.Prop-Logic Weidenbach-Book-Base.Wellfounded-More*

begin

This file is devoted to abstract properties of the transformations, like consistency preservation and lifting from terms to proposition.

0.1 Rewrite Systems and Properties

0.1.1 Lifting of Rewrite Rules

We can lift a rewrite relation r over a full formula: the relation r works on terms, while *propo-rew-step* works on formulas.

inductive *propo-rew-step* :: ('v propo \Rightarrow 'v propo \Rightarrow bool) \Rightarrow 'v propo \Rightarrow 'v propo \Rightarrow bool
for r :: 'v propo \Rightarrow 'v propo \Rightarrow bool **where**

global-rel: $r \varphi \psi \implies \text{propo-rew-step } r \varphi \psi \mid$
propo-rew-one-step-lift: $\text{propo-rew-step } r \varphi \varphi' \implies \text{wf-conn } c (\psi s @ \varphi \# \psi s') \implies \text{propo-rew-step } r (\text{conn } c (\psi s @ \varphi \# \psi s')) (\text{conn } c (\psi s @ \varphi' \# \psi s'))$

Here is a more precise link between the lifting and the subformulas: if a rewriting takes place between φ and φ' , then there are two subformulas ψ in φ and ψ' in φ' , ψ' is the result of the rewriting of r on ψ .

This lemma is only a health condition:

lemma *propo-rew-step-subformula-imp*:
shows $\text{propo-rew-step } r \varphi \varphi' \implies \exists \psi \psi'. \psi \preceq \varphi \wedge \psi' \preceq \varphi' \wedge r \psi \psi'$
 <proof>

The converse is moreover true: if there is a ψ and ψ' , then every formula φ containing ψ , can be rewritten into a formula φ' , such that it contains ψ' .

lemma *propo-rew-step-subformula-rec*:
fixes $\psi \psi' \varphi :: 'v \text{ propo}$
shows $\psi \preceq \varphi \implies r \psi \psi' \implies (\exists \varphi'. \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \varphi \varphi')$
 <proof>

lemma *propo-rew-step-subformula*:
 $(\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi') \longleftrightarrow (\exists \varphi'. \text{propo-rew-step } r \varphi \varphi')$
 <proof>

lemma *consistency-decompose-into-list*:
assumes $\text{wf}: \text{wf-conn } c \ l$ **and** $\text{wf}': \text{wf-conn } c \ l'$
and same: $\forall n. A \models l ! n \longleftrightarrow (A \models l' ! n)$
shows $A \models \text{conn } c \ l \longleftrightarrow A \models \text{conn } c \ l'$
 <proof>

Relation between *propo-rew-step* and the rewriting we have seen before: *propo-rew-step* $r \varphi \varphi'$ means that we rewrite ψ inside φ (ie at a path p) into ψ' .

lemma *propo-rew-step-rewrite*:
fixes $\varphi \varphi' :: 'v \text{ propo}$ **and** $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$
assumes $\text{propo-rew-step } r \varphi \varphi'$
shows $\exists \psi \psi' p. r \psi \psi' \wedge \text{path-to } p \varphi \psi \wedge \text{replace-at } p \varphi \psi' = \varphi'$
 <proof>

0.1.2 Consistency Preservation

We define *preserve-models*: it means that a relation preserves consistency.

definition *preserve-models where*
 $\text{preserve-models } r \longleftrightarrow (\forall \varphi \psi. r \varphi \psi \longrightarrow (\forall A. A \models \varphi \longleftrightarrow A \models \psi))$

lemma *propo-rew-step-preservers-val-explicit*:
 $\text{propo-rew-step } r \varphi \psi \implies \text{preserve-models } r \implies \text{propo-rew-step } r \varphi \psi \implies (\forall A. A \models \varphi \longleftrightarrow A \models \psi)$
 <proof>

lemma *propo-rew-step-preservers-val*:
assumes $\text{preserve-models } r$
shows $\text{preserve-models } (\text{propo-rew-step } r)$
 <proof>

lemma *preserve-models-OO[intro]*:
preserve-models $f \implies \text{preserve-models } g \implies \text{preserve-models } (f \text{ OO } g)$
 ⟨proof⟩

lemma *star-consistency-preservation-explicit*:
assumes $(\text{propo-rew-step } r)^{\wedge**} \varphi \psi$ **and** *preserve-models* r
shows $\forall A. A \models \varphi \longleftrightarrow A \models \psi$
 ⟨proof⟩

lemma *star-consistency-preservation*:
preserve-models $r \implies \text{preserve-models } (\text{propo-rew-step } r)^{\wedge**}$
 ⟨proof⟩

0.1.3 Full Lifting

In the previous a relation was lifted to a formula, now we define the relation such it is applied as long as possible. The definition is thus simply: it can be derived and nothing more can be derived.

lemma *full-ropo-rew-step-preservers-val[simp]*:
preserve-models $r \implies \text{preserve-models } (\text{full } (\text{propo-rew-step } r))$
 ⟨proof⟩

lemma *full-propo-rew-step-subformula*:
 $\text{full } (\text{propo-rew-step } r) \varphi' \varphi \implies \neg(\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi')$
 ⟨proof⟩

0.2 Transformation testing

0.2.1 Definition and first Properties

To prove correctness of our transformation, we create a *all-subformula-st* predicate. It tests recursively all subformulas. At each step, the actual formula is tested. The aim of this *test-symb* function is to test locally some properties of the formulas (i.e. at the level of the connective or at first level). This allows a clause description between the rewrite relation and the *test-symb*

definition *all-subformula-st* :: $(\text{'a propo} \Rightarrow \text{bool}) \Rightarrow \text{'a propo} \Rightarrow \text{bool}$ **where**
all-subformula-st test-symb $\varphi \equiv \forall \psi. \psi \preceq \varphi \longrightarrow \text{test-symb } \psi$

lemma *test-symb-imp-all-subformula-st[simp]*:
 $\text{test-symb } FT \implies \text{all-subformula-st test-symb } FT$
 $\text{test-symb } FF \implies \text{all-subformula-st test-symb } FF$
 $\text{test-symb } (FVar x) \implies \text{all-subformula-st test-symb } (FVar x)$
 ⟨proof⟩

lemma *all-subformula-st-test-symb-true-phi*:
 $\text{all-subformula-st test-symb } \varphi \implies \text{test-symb } \varphi$
 ⟨proof⟩

lemma *all-subformula-st-decomp-imp*:

$wf\text{-conn } c \ l \implies (test\text{-symb } (conn \ c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st } test\text{-symb } \varphi))$
 $\implies \text{all-subformula-st } test\text{-symb } (conn \ c \ l)$
 <proof>

To ease the finding of proofs, we give some explicit theorem about the decomposition.

lemma *all-subformula-st-decomp-rec*:
 $\text{all-subformula-st } test\text{-symb } (conn \ c \ l) \implies wf\text{-conn } c \ l$
 $\implies (test\text{-symb } (conn \ c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st } test\text{-symb } \varphi))$
 <proof>

lemma *all-subformula-st-decomp*:
fixes $c :: 'v \text{ connective}$ **and** $l :: 'v \text{ propo list}$
assumes $wf\text{-conn } c \ l$
shows $\text{all-subformula-st } test\text{-symb } (conn \ c \ l)$
 $\longleftrightarrow (test\text{-symb } (conn \ c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st } test\text{-symb } \varphi))$
 <proof>

lemma *helper-fact*: $c \in \text{binary-connectives} \longleftrightarrow (c = COr \vee c = CAnd \vee c = CEq \vee c = CImp)$
 <proof>

lemma *all-subformula-st-decomp-explicit[simp]*:
fixes $\varphi \ \psi :: 'v \text{ propo}$
shows $\text{all-subformula-st } test\text{-symb } (FAnd \ \varphi \ \psi)$
 $\longleftrightarrow (test\text{-symb } (FAnd \ \varphi \ \psi) \wedge \text{all-subformula-st } test\text{-symb } \varphi \wedge \text{all-subformula-st } test\text{-symb } \psi)$
and $\text{all-subformula-st } test\text{-symb } (FOr \ \varphi \ \psi)$
 $\longleftrightarrow (test\text{-symb } (FOr \ \varphi \ \psi) \wedge \text{all-subformula-st } test\text{-symb } \varphi \wedge \text{all-subformula-st } test\text{-symb } \psi)$
and $\text{all-subformula-st } test\text{-symb } (FNot \ \varphi)$
 $\longleftrightarrow (test\text{-symb } (FNot \ \varphi) \wedge \text{all-subformula-st } test\text{-symb } \varphi)$
and $\text{all-subformula-st } test\text{-symb } (FEq \ \varphi \ \psi)$
 $\longleftrightarrow (test\text{-symb } (FEq \ \varphi \ \psi) \wedge \text{all-subformula-st } test\text{-symb } \varphi \wedge \text{all-subformula-st } test\text{-symb } \psi)$
and $\text{all-subformula-st } test\text{-symb } (FImp \ \varphi \ \psi)$
 $\longleftrightarrow (test\text{-symb } (FImp \ \varphi \ \psi) \wedge \text{all-subformula-st } test\text{-symb } \varphi \wedge \text{all-subformula-st } test\text{-symb } \psi)$
 <proof>

As *all-subformula-st* tests recursively, the function is true on every subformula.

lemma *subformula-all-subformula-st*:
 $\psi \preceq \varphi \implies \text{all-subformula-st } test\text{-symb } \varphi \implies \text{all-subformula-st } test\text{-symb } \psi$
 <proof>

The following theorem *no-test-symb-step-exists* shows the link between the *test-symb* function and the corresponding rewrite relation r : if we assume that if every time *test-symb* is true, then a r can be applied, finally as long as $\neg \text{all-subformula-st } test\text{-symb } \varphi$, then something can be rewritten in φ .

lemma *no-test-symb-step-exists*:
fixes $r :: 'v \text{ propo} \implies 'v \text{ propo} \implies \text{bool}$ **and** $test\text{-symb} :: 'v \text{ propo} \implies \text{bool}$ **and** $x :: 'v$
and $\varphi :: 'v \text{ propo}$
assumes
test-symb-false-nullary: $\forall x. test\text{-symb } FF \wedge test\text{-symb } FT \wedge test\text{-symb } (FVar \ x)$ **and**
 $\forall \varphi'. \varphi' \preceq \varphi \longrightarrow (\neg test\text{-symb } \varphi') \longrightarrow (\exists \psi. r \ \varphi' \ \psi)$ **and**
 $\neg \text{all-subformula-st } test\text{-symb } \varphi$
shows $\exists \psi \ \psi'. \psi \preceq \varphi \wedge r \ \psi \ \psi'$
 <proof>

0.2.2 Invariant conservation

If two rewrite relation are independant (or at least independant enough), then the property characterizing the first relation *all-subformula-st test-symb* remains true. The next show the same property, with changes in the assumptions.

The assumption $\forall \varphi' \psi. \varphi' \preceq \Phi \longrightarrow r \varphi' \psi \longrightarrow \text{all-subformula-st test-symb } \varphi' \longrightarrow \text{all-subformula-st test-symb } \psi$ means that rewriting with r does not mess up the property we want to preserve locally.

The previous assumption is not enough to go from r to *propo-rew-step* r : we have to add the assumption that rewriting inside does not mess up the term: $\forall c \xi \varphi \xi' \varphi'. \varphi \preceq \Phi \longrightarrow \text{propo-rew-step } r \varphi \varphi' \longrightarrow \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi' \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$

Invariant while lifting of the Rewriting Relation

The condition $\varphi \preceq \Phi$ (that will by used with $\Phi = \varphi$ most of the time) is here to ensure that the recursive conditions on Φ will moreover hold for the subterm we are rewriting. For example if there is no equivalence symbol in Φ , we do not have to care about equivalence symbols in the two previous assumptions.

lemma *propo-rew-step-inv-stay'*:

fixes $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb} :: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$
and $\varphi \psi \Phi :: 'v \text{ propo}$
assumes $H: \forall \varphi' \psi. \varphi' \preceq \Phi \longrightarrow r \varphi' \psi \longrightarrow \text{all-subformula-st test-symb } \varphi' \longrightarrow \text{all-subformula-st test-symb } \psi$
and $H': \forall (c :: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \varphi \preceq \Phi \longrightarrow \text{propo-rew-step } r \varphi \varphi' \longrightarrow \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi' \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ **and**
 $\text{propo-rew-step } r \varphi \psi$ **and**
 $\varphi \preceq \Phi$ **and**
 $\text{all-subformula-st test-symb } \varphi$
shows $\text{all-subformula-st test-symb } \psi$
 $\langle \text{proof} \rangle$

The need for $\varphi \preceq \Phi$ is not always necessary, hence we moreover have a version without inclusion.

lemma *propo-rew-step-inv-stay*:

fixes $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb} :: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$
and $\varphi \psi :: 'v \text{ propo}$
assumes
 $H: \forall \varphi' \psi. r \varphi' \psi \longrightarrow \text{all-subformula-st test-symb } \varphi' \longrightarrow \text{all-subformula-st test-symb } \psi$ **and**
 $H': \forall (c :: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi' \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ **and**
 $\text{propo-rew-step } r \varphi \psi$ **and**
 $\text{all-subformula-st test-symb } \varphi$
shows $\text{all-subformula-st test-symb } \psi$
 $\langle \text{proof} \rangle$

The lemmas can be lifted to *propo-rew-step* r^\downarrow instead of *propo-rew-step*

Invariant after all Rewriting

lemma *full-propo-rew-step-inv-stay-with-inc*:

fixes $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb} :: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$

and $\varphi \psi :: 'v \text{ propo}$

assumes

$H: \forall \varphi \psi. \text{propo-rew-step } r \varphi \psi \longrightarrow \text{all-subformula-st test-symb } \varphi$
 $\longrightarrow \text{all-subformula-st test-symb } \psi$ **and**

$H': \forall (c:: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \varphi \preceq \Phi \longrightarrow \text{propo-rew-step } r \varphi \varphi'$
 $\longrightarrow \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi'$
 $\longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ **and**
 $\varphi \preceq \Phi$ **and**

$\text{full: full (propo-rew-step } r) \varphi \psi$ **and**

$\text{init: all-subformula-st test-symb } \varphi$

shows $\text{all-subformula-st test-symb } \psi$

$\langle \text{proof} \rangle$

lemma $\text{full-propo-rew-step-inv-stay'}$:

fixes $r:: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb}:: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$

and $\varphi \psi :: 'v \text{ propo}$

assumes

$H: \forall \varphi \psi. \text{propo-rew-step } r \varphi \psi \longrightarrow \text{all-subformula-st test-symb } \varphi$
 $\longrightarrow \text{all-subformula-st test-symb } \psi$ **and**

$H': \forall (c:: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \text{propo-rew-step } r \varphi \varphi' \longrightarrow \text{wf-conn } c (\xi @ \varphi \# \xi')$
 $\longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi' \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ **and**

$\text{full: full (propo-rew-step } r) \varphi \psi$ **and**

$\text{init: all-subformula-st test-symb } \varphi$

shows $\text{all-subformula-st test-symb } \psi$

$\langle \text{proof} \rangle$

lemma $\text{full-propo-rew-step-inv-stay}$:

fixes $r:: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb}:: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$

and $\varphi \psi :: 'v \text{ propo}$

assumes

$H: \forall \varphi \psi. r \varphi \psi \longrightarrow \text{all-subformula-st test-symb } \varphi \longrightarrow \text{all-subformula-st test-symb } \psi$ **and**

$H': \forall (c:: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi'))$
 $\longrightarrow \text{test-symb } \varphi' \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ **and**

$\text{full: full (propo-rew-step } r) \varphi \psi$ **and**

$\text{init: all-subformula-st test-symb } \varphi$

shows $\text{all-subformula-st test-symb } \psi$

$\langle \text{proof} \rangle$

lemma $\text{full-propo-rew-step-inv-stay-conn}$:

fixes $r:: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **and** $\text{test-symb}:: 'v \text{ propo} \Rightarrow \text{bool}$ **and** $x :: 'v$

and $\varphi \psi :: 'v \text{ propo}$

assumes

$H: \forall \varphi \psi. r \varphi \psi \longrightarrow \text{all-subformula-st test-symb } \varphi \longrightarrow \text{all-subformula-st test-symb } \psi$ **and**

$H': \forall (c:: 'v \text{ connective}) l l'. \text{wf-conn } c l \longrightarrow \text{wf-conn } c l'$
 $\longrightarrow (\text{test-symb } (\text{conn } c l) \longleftrightarrow \text{test-symb } (\text{conn } c l'))$ **and**

$\text{full: full (propo-rew-step } r) \varphi \psi$ **and**

$\text{init: all-subformula-st test-symb } \varphi$

shows $\text{all-subformula-st test-symb } \psi$

$\langle \text{proof} \rangle$

end

theory $\text{Prop-Normalisation}$

imports $\text{Entailment-Definition.Prop-Logic Prop-Abstract-Transformation Nested-Multisets-Ordinals.Multiset-More}$

begin

Given the previous definition about abstract rewriting and theorem about them, we now have the detailed rule making the transformation into CNF/DNF.

0.3 Rewrite Rules

The idea of Christoph Weidenbach's book is to remove gradually the operators: first equivalencies, then implication, after that the unused true/false and finally the reorganizing the or/and. We will prove each transformation separately.

0.3.1 Elimination of the Equivalences

The first transformation consists in removing every equivalence symbol.

inductive *elim-equiv* :: 'v propo \Rightarrow 'v propo \Rightarrow bool **where**
elim-equiv[simp]: *elim-equiv* (FEq φ ψ) (FAnd (FImp φ ψ) (FImp ψ φ))

lemma *elim-equiv-transformation-consistent*:
 $A \models \text{FEq } \varphi \ \psi \longleftrightarrow A \models \text{FAnd } (\text{FImp } \varphi \ \psi) \ (\text{FImp } \psi \ \varphi)$
 <proof>

lemma *elim-equiv-explicit*: *elim-equiv* $\varphi \ \psi \Longrightarrow \forall A. A \models \varphi \longleftrightarrow A \models \psi$
 <proof>

lemma *elim-equiv-consistent*: *preserve-models* *elim-equiv*
 <proof>

lemma *elimEqv-lifted-consistent*:
preserve-models (full (propo-rew-step *elim-equiv*))
 <proof>

This function ensures that there is no equivalencies left in the formula tested by *no-equiv-symb*.

fun *no-equiv-symb* :: 'v propo \Rightarrow bool **where**
no-equiv-symb (FEq -) = False |
no-equiv-symb - = True

Given the definition of *no-equiv-symb*, it does not depend on the formula, but only on the connective used.

lemma *no-equiv-symb-conn-characterization*[simp]:
fixes *c* :: 'v connective **and** *l* :: 'v propo list
assumes *wf*: *wf-conn* *c* *l*
shows *no-equiv-symb* (conn *c* *l*) $\longleftrightarrow c \neq \text{CEq}$
 <proof>

definition *no-equiv* **where** *no-equiv* = *all-subformula-st no-equiv-symb*

lemma *no-equiv-eq*[simp]:
fixes $\varphi \ \psi$:: 'v propo
shows
 $\neg \text{no-equiv } (\text{FEq } \varphi \ \psi)$
no-equiv FT
no-equiv FF
 <proof>

The following lemma helps to reconstruct *no-equiv* expressions: this representation is easier to use than the set definition.

lemma *all-subformula-st-decomp-explicit-no-equiv*[iff]:

fixes $\varphi \psi :: 'v \text{ propo}$

shows

$\text{no-equiv } (FNot \ \varphi) \longleftrightarrow \text{no-equiv } \varphi$
 $\text{no-equiv } (FAnd \ \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$
 $\text{no-equiv } (FOr \ \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$
 $\text{no-equiv } (FImp \ \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$
 $\langle \text{proof} \rangle$

A theorem to show the link between the rewrite relation *elim-equiv* and the function *no-equiv-symb*. This theorem is one of the assumption we need to characterize the transformation.

lemma *no-equiv-elim-equiv-step*:

fixes $\varphi :: 'v \text{ propo}$

assumes *no-equiv*: $\neg \text{no-equiv } \varphi$

shows $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{elim-equiv } \psi \ \psi'$

$\langle \text{proof} \rangle$

Given all the previous theorem and the characterization, once we have rewritten everything, there is no equivalence symbol any more.

lemma *no-equiv-full-propo-rew-step-elim-equiv*:

full (*propo-rew-step elim-equiv*) $\varphi \ \psi \implies \text{no-equiv } \psi$

$\langle \text{proof} \rangle$

0.3.2 Eliminate Implication

After that, we can eliminate the implication symbols.

inductive *elim-imp* :: $'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$ **where**

[*simp*]: *elim-imp* (*FImp* $\varphi \ \psi$) (*FOr* (*FNot* φ) ψ)

lemma *elim-imp-transformation-consistent*:

$A \models FImp \ \varphi \ \psi \longleftrightarrow A \models FOr \ (FNot \ \varphi) \ \psi$

$\langle \text{proof} \rangle$

lemma *elim-imp-explicit*: *elim-imp* $\varphi \ \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$

$\langle \text{proof} \rangle$

lemma *elim-imp-consistent*: *preserve-models elim-imp*

$\langle \text{proof} \rangle$

lemma *elim-imp-lifted-consistent*:

preserve-models (*full* (*propo-rew-step elim-imp*))

$\langle \text{proof} \rangle$

fun *no-imp-symb* **where**

no-imp-symb (*FImp* -) = *False* |

no-imp-symb - = *True*

lemma *no-imp-symb-conn-characterization*:

wf-conn $c \ l \implies \text{no-imp-symb } (\text{conn } c \ l) \longleftrightarrow c \neq CImp$

$\langle \text{proof} \rangle$

definition *no-imp* **where** *no-imp* \equiv *all-subformula-st no-imp-symb*

declare *no-imp-def*[*simp*]

lemma *no-imp-Imp*[*simp*]:

\neg *no-imp* (*FImp* φ ψ)
no-imp *FT*
no-imp *FF*
<proof>

lemma *all-subformula-st-decomp-explicit-imp*[*simp*]:

fixes φ $\psi :: 'v$ *propo*
shows
no-imp (*FNot* φ) \longleftrightarrow *no-imp* φ
no-imp (*FAnd* φ ψ) \longleftrightarrow (*no-imp* φ \wedge *no-imp* ψ)
no-imp (*FOr* φ ψ) \longleftrightarrow (*no-imp* φ \wedge *no-imp* ψ)
<proof>

Invariant of the *elim-imp* transformation

lemma *elim-imp-no-equiv*:

elim-imp φ $\psi \implies$ *no-equiv* $\varphi \implies$ *no-equiv* ψ
<proof>

lemma *elim-imp-inv*:

fixes φ $\psi :: 'v$ *propo*
assumes *full* (*propo-rew-step* *elim-imp*) φ ψ **and** *no-equiv* φ
shows *no-equiv* ψ
<proof>

lemma *no-no-imp-elim-imp-step-exists*:

fixes $\varphi :: 'v$ *propo*
assumes *no-equiv*: \neg *no-imp* φ
shows $\exists \psi \psi'. \psi \preceq \varphi \wedge$ *elim-imp* $\psi \psi'$
<proof>

lemma *no-imp-full-propo-rew-step-elim-imp*: *full* (*propo-rew-step* *elim-imp*) φ $\psi \implies$ *no-imp* ψ

<proof>

0.3.3 Eliminate all the True and False in the formula

Contrary to the book, we have to give the transformation and the “commutative” transformation. The latter is implicit in the book.

inductive *elimTB* **where**

ElimTB1: *elimTB* (*FAnd* φ *FT*) φ |
ElimTB1': *elimTB* (*FAnd* *FT* φ) φ |

ElimTB2: *elimTB* (*FAnd* φ *FF*) *FF* |
ElimTB2': *elimTB* (*FAnd* *FF* φ) *FF* |

ElimTB3: *elimTB* (*FOr* φ *FT*) *FT* |
ElimTB3': *elimTB* (*FOr* *FT* φ) *FT* |

ElimTB4: *elimTB* (*FOr* φ *FF*) φ |
ElimTB4': *elimTB* (*FOr* *FF* φ) φ |

ElimTB5: *elimTB* (*FNot* *FT*) *FF* |
ElimTB6: *elimTB* (*FNot* *FF*) *FT*

lemma *elimTB-consistent: preserve-models elimTB*

<proof>

inductive *no-T-F-symb* :: 'v propo \Rightarrow bool **where**

no-T-F-symb-comp: $c \neq CF \Longrightarrow c \neq CT \Longrightarrow wf\text{-conn } c \ l \Longrightarrow (\forall \varphi \in set \ l. \varphi \neq FT \wedge \varphi \neq FF)$
 $\Longrightarrow no\text{-T-F-symb } (conn \ c \ l)$

lemma *wf-conn-no-T-F-symb-iff[simp]*:

wf-conn c ψ s \Longrightarrow

no-T-F-symb (conn c ψ s) $\longleftrightarrow (c \neq CF \wedge c \neq CT \wedge (\forall \psi \in set \ \psi.s. \psi \neq FF \wedge \psi \neq FT))$

<proof>

lemma *wf-conn-no-T-F-symb-iff-explicit[simp]*:

no-T-F-symb (FAnd $\varphi \ \psi$) $\longleftrightarrow (\forall \chi \in set \ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$

no-T-F-symb (FOr $\varphi \ \psi$) $\longleftrightarrow (\forall \chi \in set \ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$

no-T-F-symb (FEq $\varphi \ \psi$) $\longleftrightarrow (\forall \chi \in set \ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$

no-T-F-symb (FImp $\varphi \ \psi$) $\longleftrightarrow (\forall \chi \in set \ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$

<proof>

lemma *no-T-F-symb-false[simp]*:

fixes *c* :: 'v *connective*

shows

$\neg no\text{-T-F-symb } (FT :: 'v \ propo)$

$\neg no\text{-T-F-symb } (FF :: 'v \ propo)$

<proof>

lemma *no-T-F-symb-bool[simp]*:

fixes *x* :: 'v

shows *no-T-F-symb (FVar x)*

<proof>

lemma *no-T-F-symb-fnot-imp*:

$\neg no\text{-T-F-symb } (FNot \ \varphi) \Longrightarrow \varphi = FT \vee \varphi = FF$

<proof>

lemma *no-T-F-symb-fnot[simp]*:

no-T-F-symb (FNot φ) $\longleftrightarrow \neg(\varphi = FT \vee \varphi = FF)$

<proof>

Actually it is not possible to remove every *FT* and *FF*: if the formula is equal to true or false, we can not remove it.

inductive *no-T-F-symb-except-toplevel* **where**

no-T-F-symb-except-toplevel-true[simp]: *no-T-F-symb-except-toplevel FT* |

no-T-F-symb-except-toplevel-false[simp]: *no-T-F-symb-except-toplevel FF* |

noTrue-no-T-F-symb-except-toplevel[simp]: *no-T-F-symb $\varphi \Longrightarrow no\text{-T-F-symb-except-toplevel } \varphi$*

lemma *no-T-F-symb-except-toplevel-bool*:

fixes *x* :: 'v

shows *no-T-F-symb-except-toplevel (FVar x)*

<proof>

lemma *no-T-F-symb-except-toplevel-not-decom*:
 $\varphi \neq FT \implies \varphi \neq FF \implies \text{no-T-F-symb-except-toplevel } (F\text{Not } \varphi)$
 $\langle \text{proof} \rangle$

lemma *no-T-F-symb-except-toplevel-bin-decom*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes $\varphi \neq FT$ **and** $\varphi \neq FF$ **and** $\psi \neq FT$ **and** $\psi \neq FF$
and $c \in \text{binary-connectives}$
shows $\text{no-T-F-symb-except-toplevel } (\text{conn } c [\varphi, \psi])$
 $\langle \text{proof} \rangle$

lemma *no-T-F-symb-except-toplevel-if-is-a-true-false*:
fixes $l :: 'v \text{ propo list}$ **and** $c :: 'v \text{ connective}$
assumes $\text{corr: wf-conn } c \ l$
and $FT \in \text{set } l \vee FF \in \text{set } l$
shows $\neg \text{no-T-F-symb-except-toplevel } (\text{conn } c \ l)$
 $\langle \text{proof} \rangle$

lemma *no-T-F-symb-except-top-level-false-example[simp]*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$
shows
 $\neg \text{no-T-F-symb-except-toplevel } (F\text{And } \varphi \ \psi)$
 $\neg \text{no-T-F-symb-except-toplevel } (F\text{Or } \varphi \ \psi)$
 $\neg \text{no-T-F-symb-except-toplevel } (F\text{Imp } \varphi \ \psi)$
 $\neg \text{no-T-F-symb-except-toplevel } (F\text{Eq } \varphi \ \psi)$
 $\langle \text{proof} \rangle$

lemma *no-T-F-symb-except-top-level-false-not[simp]*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes $\varphi = FT \vee \varphi = FF$
shows
 $\neg \text{no-T-F-symb-except-toplevel } (F\text{Not } \varphi)$
 $\langle \text{proof} \rangle$

This is the local extension of *no-T-F-symb-except-toplevel*.

definition *no-T-F-except-top-level* **where**
 $\text{no-T-F-except-top-level} \equiv \text{all-subformula-st no-T-F-symb-except-toplevel}$

This is another property we will use. While this version might seem to be the one we want to prove, it is not since *FT* can not be reduced.

definition *no-T-F* **where**
 $\text{no-T-F} \equiv \text{all-subformula-st no-T-F-symb}$

lemma *no-T-F-except-top-level-false*:
fixes $l :: 'v \text{ propo list}$ **and** $c :: 'v \text{ connective}$
assumes $\text{wf-conn } c \ l$
and $FT \in \text{set } l \vee FF \in \text{set } l$
shows $\neg \text{no-T-F-except-top-level } (\text{conn } c \ l)$
 $\langle \text{proof} \rangle$

lemma *no-T-F-except-top-level-false-example[simp]*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$

shows

\neg no-*T-F-except-top-level* (*FAnd* φ ψ)
 \neg no-*T-F-except-top-level* (*FOr* φ ψ)
 \neg no-*T-F-except-top-level* (*FEq* φ ψ)
 \neg no-*T-F-except-top-level* (*FImp* φ ψ)
(*proof*)

lemma *no-T-F-symb-except-toplevel-no-T-F-symb*:

no-*T-F-symb-except-toplevel* $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies$ no-*T-F-symb* φ
(*proof*)

The two following lemmas give the precise link between the two definitions.

lemma *no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb*:

no-*T-F-except-top-level* $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies$ no-*T-F* φ
(*proof*)

lemma *no-T-F-no-T-F-except-top-level*:

no-*T-F* $\varphi \implies$ no-*T-F-except-top-level* φ
(*proof*)

lemma *no-T-F-except-top-level-simp[simp]*: no-*T-F-except-top-level* *FF* no-*T-F-except-top-level* *FT*

(*proof*)

lemma *no-T-F-no-T-F-except-top-level'[simp]*:

no-*T-F-except-top-level* $\varphi \longleftrightarrow (\varphi = FF \vee \varphi = FT \vee$ no-*T-F* $\varphi)$
(*proof*)

lemma *no-T-F-bin-decomp[simp]*:

assumes *c*: *c* \in *binary-connectives*

shows no-*T-F* (*conn* *c* [φ , ψ]) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

(*proof*)

lemma *no-T-F-bin-decomp-expanded[simp]*:

assumes *c*: *c* = *CAnd* \vee *c* = *COr* \vee *c* = *CEq* \vee *c* = *CImp*

shows no-*T-F* (*conn* *c* [φ , ψ]) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

(*proof*)

lemma *no-T-F-comp-expanded-explicit[simp]*:

fixes φ ψ :: '*v propo*

shows

no-*T-F* (*FAnd* φ ψ) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

no-*T-F* (*FOr* φ ψ) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

no-*T-F* (*FEq* φ ψ) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

no-*T-F* (*FImp* φ ψ) \longleftrightarrow (no-*T-F* $\varphi \wedge$ no-*T-F* ψ)

(*proof*)

lemma *no-T-F-comp-not[simp]*:

fixes φ ψ :: '*v propo*

shows no-*T-F* (*FNot* φ) \longleftrightarrow no-*T-F* φ

(*proof*)

lemma *no-T-F-decomp*:

fixes φ ψ :: '*v propo*

assumes φ : no-*T-F* (*FAnd* φ ψ) \vee no-*T-F* (*FOr* φ ψ) \vee no-*T-F* (*FEq* φ ψ) \vee no-*T-F* (*FImp* φ ψ)

shows no-*T-F* ψ **and** no-*T-F* φ

$\langle proof \rangle$

lemma *no-T-F-decomp-not*:

fixes $\varphi :: 'v \text{ propo}$

assumes $\varphi: \text{no-T-F } (FNot \ \varphi)$

shows *no-T-F* φ

$\langle proof \rangle$

lemma *no-T-F-symb-except-toplevel-step-exists*:

fixes $\varphi \ \psi :: 'v \text{ propo}$

assumes *no-equiv* φ **and** *no-imp* φ

shows $\psi \preceq \varphi \implies \neg \text{no-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTB } \psi \ \psi'$

$\langle proof \rangle$

lemma *no-T-F-except-top-level-rew*:

fixes $\varphi :: 'v \text{ propo}$

assumes *noTB*: $\neg \text{no-T-F-except-top-level } \varphi$ **and** *no-equiv*: *no-equiv* φ **and** *no-imp*: *no-imp* φ

shows $\exists \psi \ \psi'. \psi \preceq \varphi \wedge \text{elimTB } \psi \ \psi'$

$\langle proof \rangle$

lemma *elimTB-inv*:

fixes $\varphi \ \psi :: 'v \text{ propo}$

assumes *full* (*propo-rew-step* *elimTB*) $\varphi \ \psi$

and *no-equiv* φ **and** *no-imp* φ

shows *no-equiv* ψ **and** *no-imp* ψ

$\langle proof \rangle$

lemma *elimTB-full-propo-rew-step*:

fixes $\varphi \ \psi :: 'v \text{ propo}$

assumes *no-equiv* φ **and** *no-imp* φ **and** *full* (*propo-rew-step* *elimTB*) $\varphi \ \psi$

shows *no-T-F-except-top-level* ψ

$\langle proof \rangle$

0.3.4 PushNeg

Push the negation inside the formula, until the literal.

inductive *pushNeg* **where**

PushNeg1[*simp*]: *pushNeg* (*FNot* (*FAnd* $\varphi \ \psi$)) (*FOr* (*FNot* φ) (*FNot* ψ)) |

PushNeg2[*simp*]: *pushNeg* (*FNot* (*FOr* $\varphi \ \psi$)) (*FAnd* (*FNot* φ) (*FNot* ψ)) |

PushNeg3[*simp*]: *pushNeg* (*FNot* (*FNot* φ)) φ

lemma *pushNeg-transformation-consistent*:

$A \models FNot \ (FAnd \ \varphi \ \psi) \longleftrightarrow A \models (FOr \ (FNot \ \varphi) \ (FNot \ \psi))$

$A \models FNot \ (FOr \ \varphi \ \psi) \longleftrightarrow A \models (FAnd \ (FNot \ \varphi) \ (FNot \ \psi))$

$A \models FNot \ (FNot \ \varphi) \longleftrightarrow A \models \varphi$

$\langle proof \rangle$

lemma *pushNeg-explicit*: *pushNeg* $\varphi \ \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$

$\langle proof \rangle$

lemma *pushNeg-consistent*: *preserve-models* *pushNeg*

$\langle proof \rangle$

lemma *pushNeg-lifted-consistant*:
preserve-models (full (propo-rew-step pushNeg))
 ⟨proof⟩

fun *simple* **where**
simple FT = True |
simple FF = True |
simple (FVar -) = True |
simple - = False

lemma *simple-decomp*:
simple $\varphi \longleftrightarrow (\varphi = FT \vee \varphi = FF \vee (\exists x. \varphi = FVar x))$
 ⟨proof⟩

lemma *subformula-conn-decomp-simple*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *s: simple ψ*
shows $\varphi \preceq FNot \psi \longleftrightarrow (\varphi = FNot \psi \vee \varphi = \psi)$
 ⟨proof⟩

lemma *subformula-conn-decomp-explicit[simp]*:
fixes $\varphi :: 'v \text{ propo}$ **and** $x :: 'v$
shows
 $\varphi \preceq FNot FT \longleftrightarrow (\varphi = FNot FT \vee \varphi = FT)$
 $\varphi \preceq FNot FF \longleftrightarrow (\varphi = FNot FF \vee \varphi = FF)$
 $\varphi \preceq FNot (FVar x) \longleftrightarrow (\varphi = FNot (FVar x) \vee \varphi = FVar x)$
 ⟨proof⟩

fun *simple-not-symb* **where**
simple-not-symb (FNot φ) = (simple φ) |
simple-not-symb - = True

definition *simple-not* **where**
simple-not = all-subformula-st simple-not-symb
declare *simple-not-def[simp]*

lemma *simple-not-Not[simp]*:
 $\neg \text{simple-not } (FNot (FAnd \varphi \psi))$
 $\neg \text{simple-not } (FNot (FOr \varphi \psi))$
 ⟨proof⟩

lemma *simple-not-step-exists*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *no-equiv φ* **and** *no-imp φ*
shows $\psi \preceq \varphi \implies \neg \text{simple-not-symb } \psi \implies \exists \psi'. \text{pushNeg } \psi \psi'$
 ⟨proof⟩

lemma *simple-not-rew*:
fixes $\varphi :: 'v \text{ propo}$
assumes *noTB: $\neg \text{simple-not } \varphi$* **and** *no-equiv: no-equiv φ* **and** *no-imp: no-imp φ*
shows $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{pushNeg } \psi \psi'$
 ⟨proof⟩

lemma *no-T-F-except-top-level-pushNeg1*:

no-T-F-except-top-level (*FNot* (*FAnd* φ ψ)) \implies *no-T-F-except-top-level* (*FOr* (*FNot* φ) (*FNot* ψ))
 ⟨*proof*⟩

lemma *no-T-F-except-top-level-pushNeg2*:

no-T-F-except-top-level (*FNot* (*FOr* φ ψ)) \implies *no-T-F-except-top-level* (*FAnd* (*FNot* φ) (*FNot* ψ))
 ⟨*proof*⟩

lemma *no-T-F-symb-pushNeg*:

no-T-F-symb (*FOr* (*FNot* φ') (*FNot* ψ'))
no-T-F-symb (*FAnd* (*FNot* φ') (*FNot* ψ'))
no-T-F-symb (*FNot* (*FNot* φ'))
 ⟨*proof*⟩

lemma *propo-rew-step-pushNeg-no-T-F-symb*:

propo-rew-step pushNeg φ $\psi \implies$ *no-T-F-except-top-level* $\varphi \implies$ *no-T-F-symb* $\varphi \implies$ *no-T-F-symb* ψ
 ⟨*proof*⟩

lemma *propo-rew-step-pushNeg-no-T-F*:

propo-rew-step pushNeg φ $\psi \implies$ *no-T-F* $\varphi \implies$ *no-T-F* ψ
 ⟨*proof*⟩

lemma *pushNeg-inv*:

fixes φ $\psi :: 'v$ *propo*
assumes *full* (*propo-rew-step pushNeg*) φ ψ
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ
 ⟨*proof*⟩

lemma *pushNeg-full-propo-rew-step*:

fixes φ $\psi :: 'v$ *propo*
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step pushNeg*) φ ψ **and**
no-T-F-except-top-level φ
shows *simple-not* ψ
 ⟨*proof*⟩

0.3.5 Push Inside

inductive *push-conn-inside* :: $'v$ *connective* \Rightarrow $'v$ *connective* \Rightarrow $'v$ *propo* \Rightarrow $'v$ *propo* \Rightarrow *bool*

for c $c' :: 'v$ *connective* **where**

push-conn-inside-l[simp]: $c = CAnd \vee c = COr \implies c' = CAnd \vee c' = COr$

\implies *push-conn-inside* c c' (*conn* c [*conn* c' [$\varphi 1$, $\varphi 2$], ψ])

(*conn* c' [*conn* c [$\varphi 1$, ψ], *conn* c [$\varphi 2$, ψ]]) |

push-conn-inside-r[simp]: $c = CAnd \vee c = COr \implies c' = CAnd \vee c' = COr$

\implies *push-conn-inside* c c' (*conn* c [ψ , *conn* c' [$\varphi 1$, $\varphi 2$]])

(*conn* c' [*conn* c [ψ , $\varphi 1$], *conn* c [ψ , $\varphi 2$]])

lemma *push-conn-inside-explicit*: *push-conn-inside* c c' φ $\psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$

⟨*proof*⟩

lemma *push-conn-inside-consistent*: *preserve-models* (*push-conn-inside* c c')

$\langle \text{proof} \rangle$

lemma *propo-rew-step-push-conn-inside*[simp]:

$\neg \text{propo-rew-step} (\text{push-conn-inside } c \ c') \text{ FT } \psi \neg \text{propo-rew-step} (\text{push-conn-inside } c \ c') \text{ FF } \psi$
 $\langle \text{proof} \rangle$

inductive *not-c-in-c'-symb*:: 'v connective \Rightarrow 'v connective \Rightarrow 'v propo \Rightarrow bool for c c' where

not-c-in-c'-symb-l[simp]: $\text{wf-conn } c \ [\text{conn } c' \ [\varphi, \varphi'], \psi] \Longrightarrow \text{wf-conn } c' \ [\varphi, \varphi']$

$\Longrightarrow \text{not-c-in-c'-symb } c \ c' (\text{conn } c \ [\text{conn } c' \ [\varphi, \varphi'], \psi]) \mid$

not-c-in-c'-symb-r[simp]: $\text{wf-conn } c \ [\psi, \text{conn } c' \ [\varphi, \varphi']] \Longrightarrow \text{wf-conn } c' \ [\varphi, \varphi']$

$\Longrightarrow \text{not-c-in-c'-symb } c \ c' (\text{conn } c \ [\psi, \text{conn } c' \ [\varphi, \varphi']])$

abbreviation *c-in-c'-symb* c c' $\varphi \equiv \neg \text{not-c-in-c'-symb } c \ c' \ \varphi$

lemma *c-in-c'-symb-simp*:

$\text{not-c-in-c'-symb } c \ c' \ \xi \Longrightarrow \xi = \text{FF} \vee \xi = \text{FT} \vee \xi = \text{FVar } x \vee \xi = \text{FNot } \text{FF} \vee \xi = \text{FNot } \text{FT}$
 $\vee \xi = \text{FNot } (\text{FVar } x) \Longrightarrow \text{False}$

$\langle \text{proof} \rangle$

lemma *c-in-c'-symb-simp'*[simp]:

$\neg \text{not-c-in-c'-symb } c \ c' \ \text{FF}$

$\neg \text{not-c-in-c'-symb } c \ c' \ \text{FT}$

$\neg \text{not-c-in-c'-symb } c \ c' \ (\text{FVar } x)$

$\neg \text{not-c-in-c'-symb } c \ c' \ (\text{FNot } \text{FF})$

$\neg \text{not-c-in-c'-symb } c \ c' \ (\text{FNot } \text{FT})$

$\neg \text{not-c-in-c'-symb } c \ c' \ (\text{FNot } (\text{FVar } x))$

$\langle \text{proof} \rangle$

definition *c-in-c'-only* where

c-in-c'-only c c' \equiv all-subformula-st (*c-in-c'-symb* c c')

lemma *c-in-c'-only-simp*[simp]:

c-in-c'-only c c' FF

c-in-c'-only c c' FT

c-in-c'-only c c' ($\text{FVar } x$)

c-in-c'-only c c' ($\text{FNot } \text{FF}$)

c-in-c'-only c c' ($\text{FNot } \text{FT}$)

c-in-c'-only c c' ($\text{FNot } (\text{FVar } x)$)

$\langle \text{proof} \rangle$

lemma *not-c-in-c'-symb-commute*:

$\text{not-c-in-c'-symb } c \ c' \ \xi \Longrightarrow \text{wf-conn } c \ [\varphi, \psi] \Longrightarrow \xi = \text{conn } c \ [\varphi, \psi]$

$\Longrightarrow \text{not-c-in-c'-symb } c \ c' (\text{conn } c \ [\psi, \varphi])$

$\langle \text{proof} \rangle$

lemma *not-c-in-c'-symb-commute'*:

$\text{wf-conn } c \ [\varphi, \psi] \Longrightarrow \text{c-in-c'-symb } c \ c' (\text{conn } c \ [\varphi, \psi]) \longleftrightarrow \text{c-in-c'-symb } c \ c' (\text{conn } c \ [\psi, \varphi])$

$\langle \text{proof} \rangle$

lemma *not-c-in-c'-comm*:

assumes *wf*: $\text{wf-conn } c \ [\varphi, \psi]$

shows *c-in-c'-only* c c' ($\text{conn } c \ [\varphi, \psi]$) \longleftrightarrow *c-in-c'-only* c c' ($\text{conn } c \ [\psi, \varphi]$) (is ?A \longleftrightarrow ?B)

$\langle \text{proof} \rangle$

lemma *not-c-in-c'-simp[simp]*:

fixes $\varphi1 \ \varphi2 \ \psi :: 'v \text{ propo}$ **and** $x :: 'v$

shows

$c\text{-in-}c'\text{-symb } c \ c' \ FT$

$c\text{-in-}c'\text{-symb } c \ c' \ FF$

$c\text{-in-}c'\text{-symb } c \ c' \ (FVar \ x)$

$wf\text{-conn } c \ [conn \ c' \ [\varphi1, \ \varphi2], \ \psi] \implies wf\text{-conn } c' \ [\varphi1, \ \varphi2]$

$\implies \neg \ c\text{-in-}c'\text{-only } c \ c' \ (conn \ c \ [conn \ c' \ [\varphi1, \ \varphi2], \ \psi])$

$\langle proof \rangle$

lemma *c-in-c'-symb-not[simp]*:

fixes $c \ c' :: 'v \text{ connective}$ **and** $\psi :: 'v \text{ propo}$

shows $c\text{-in-}c'\text{-symb } c \ c' \ (FNot \ \psi)$

$\langle proof \rangle$

lemma *c-in-c'-symb-step-exists*:

fixes $\varphi :: 'v \text{ propo}$

assumes $c: c = CAnd \vee c = COr$ **and** $c': c' = CAnd \vee c' = COr$

shows $\psi \preceq \varphi \implies \neg \ c\text{-in-}c'\text{-symb } c \ c' \ \psi \implies \exists \psi'. \text{push-conn-inside } c \ c' \ \psi \ \psi'$

$\langle proof \rangle$

lemma *c-in-c'-symb-rew*:

fixes $\varphi :: 'v \text{ propo}$

assumes $noTB: \neg \ c\text{-in-}c'\text{-only } c \ c' \ \varphi$

and $c: c = CAnd \vee c = COr$ **and** $c': c' = CAnd \vee c' = COr$

shows $\exists \psi \ \psi'. \psi \preceq \varphi \wedge \text{push-conn-inside } c \ c' \ \psi \ \psi'$

$\langle proof \rangle$

lemma *push-conn-insidec-in-c'-symb-no-T-F*:

fixes $\varphi \ \psi :: 'v \text{ propo}$

shows $\text{propo-rew-step } (\text{push-conn-inside } c \ c') \ \varphi \ \psi \implies no\text{-T-F } \varphi \implies no\text{-T-F } \psi$

$\langle proof \rangle$

lemma *simple-propo-rew-step-push-conn-inside-inv*:

$\text{propo-rew-step } (\text{push-conn-inside } c \ c') \ \varphi \ \psi \implies \text{simple } \varphi \implies \text{simple } \psi$

$\langle proof \rangle$

lemma *simple-propo-rew-step-inv-push-conn-inside-simple-not*:

fixes $c \ c' :: 'v \text{ connective}$ **and** $\varphi \ \psi :: 'v \text{ propo}$

shows $\text{propo-rew-step } (\text{push-conn-inside } c \ c') \ \varphi \ \psi \implies \text{simple-not } \varphi \implies \text{simple-not } \psi$

$\langle proof \rangle$

lemma *propo-rew-step-push-conn-inside-simple-not*:

fixes $\varphi \ \varphi' :: 'v \text{ propo}$ **and** $\xi \ \xi' :: 'v \text{ propo list}$ **and** $c :: 'v \text{ connective}$

assumes

$\text{propo-rew-step } (\text{push-conn-inside } c \ c') \ \varphi \ \varphi'$ **and**

$wf\text{-conn } c \ (\xi \ @ \ \varphi \ \# \ \xi')$ **and**

$\text{simple-not-symb } (conn \ c \ (\xi \ @ \ \varphi \ \# \ \xi'))$ **and**

$\text{simple-not-symb } \varphi'$

shows $\text{simple-not-symb } (conn \ c \ (\xi \ @ \ \varphi' \ \# \ \xi'))$

$\langle proof \rangle$

lemma *push-conn-inside-not-true-false*:
push-conn-inside c c' φ $\psi \implies \psi \neq FT \wedge \psi \neq FF$
 ⟨proof⟩

lemma *push-conn-inside-inv*:
fixes φ $\psi :: 'v$ *propo*
assumes *full* (*propo-rew-step* (*push-conn-inside* c c')) φ ψ
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
 ⟨proof⟩

lemma *push-conn-inside-full-propo-rew-step*:
fixes φ $\psi :: 'v$ *propo*
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step* (*push-conn-inside* c c')) φ ψ **and**
no-T-F-except-top-level φ **and**
simple-not φ **and**
 $c = CAnd \vee c = COr$ **and**
 $c' = CAnd \vee c' = COr$
shows *c-in-c'-only* c c' ψ
 ⟨proof⟩

Only one type of connective in the formula (+ not)

inductive *only-c-inside-symb* :: $'v$ *connective* $\Rightarrow 'v$ *propo* \Rightarrow *bool* **for** $c :: 'v$ *connective* **where**
simple-only-c-inside[*simp*]: *simple* $\varphi \implies$ *only-c-inside-symb* c φ |
simple-cnot-only-c-inside[*simp*]: *simple* $\varphi \implies$ *only-c-inside-symb* c (*FNot* φ) |
only-c-inside-into-only-c-inside: *wf-conn* c $l \implies$ *only-c-inside-symb* c (*conn* c l)

lemma *only-c-inside-symb-simp*[*simp*]:
only-c-inside-symb c *FF* *only-c-inside-symb* c *FT* *only-c-inside-symb* c (*FVar* x) ⟨proof⟩

definition *only-c-inside* **where** *only-c-inside* $c =$ *all-subformula-st* (*only-c-inside-symb* c)

lemma *only-c-inside-symb-decomp*:
only-c-inside-symb c $\psi \longleftrightarrow$ (*simple* ψ
 $\vee (\exists \varphi'. \psi = FNot \varphi' \wedge$ *simple* $\varphi')$
 $\vee (\exists l. \psi =$ *conn* c $l \wedge$ *wf-conn* c $l))$
 ⟨proof⟩

lemma *only-c-inside-symb-decomp-not*[*simp*]:
fixes $c :: 'v$ *connective*
assumes $c: c \neq CNot$
shows *only-c-inside-symb* c (*FNot* ψ) \longleftrightarrow *simple* ψ
 ⟨proof⟩

lemma *only-c-inside-decomp-not*[*simp*]:
assumes $c: c \neq CNot$
shows *only-c-inside* c (*FNot* ψ) \longleftrightarrow *simple* ψ
 ⟨proof⟩

lemma *only-c-inside-decomp*:

only-c-inside $c \varphi \longleftrightarrow$
 $(\forall \psi. \psi \preceq \varphi \longrightarrow (simple \psi \vee (\exists \varphi'. \psi = FNot \varphi' \wedge simple \varphi') \vee (\exists l. \psi = conn \ c \ l \wedge wf\text{-}conn \ c \ l)))$
 $\langle proof \rangle$

lemma *only-c-inside-c-c'-false*:

fixes $c \ c' :: 'v \text{ connective}$ **and** $l :: 'v \text{ propo list}$ **and** $\varphi :: 'v \text{ propo}$
assumes $cc': c \neq c'$ **and** $c: c = CAnd \vee c = COr$ **and** $c': c' = CAnd \vee c' = COr$
and *only*: *only-c-inside* $c \ \varphi$ **and** *incl*: $conn \ c' \ l \preceq \varphi$ **and** *wf*: $wf\text{-}conn \ c' \ l$
shows *False*
 $\langle proof \rangle$

lemma *only-c-inside-implies-c-in-c'-symb*:

assumes $\delta: c \neq c'$ **and** $c: c = CAnd \vee c = COr$ **and** $c': c' = CAnd \vee c' = COr$
shows *only-c-inside* $c \ \varphi \implies c\text{-in-}c'\text{-symb} \ c \ c' \ \varphi$
 $\langle proof \rangle$

lemma *c-in-c'-symb-decomp-level1*:

fixes $l :: 'v \text{ propo list}$ **and** $c \ c' \ ca :: 'v \text{ connective}$
shows $wf\text{-}conn \ ca \ l \implies ca \neq c \implies c\text{-in-}c'\text{-symb} \ c \ c' \ (conn \ ca \ l)$
 $\langle proof \rangle$

lemma *only-c-inside-implies-c-in-c'-only*:

assumes $\delta: c \neq c'$ **and** $c: c = CAnd \vee c = COr$ **and** $c': c' = CAnd \vee c' = COr$
shows *only-c-inside* $c \ \varphi \implies c\text{-in-}c'\text{-only} \ c \ c' \ \varphi$
 $\langle proof \rangle$

lemma *c-in-c'-symb-c-implies-only-c-inside*:

assumes $\delta: c = CAnd \vee c = COr \ c' = CAnd \vee c' = COr \ c \neq c'$ **and** *wf*: $wf\text{-}conn \ c \ [\varphi, \psi]$
and *inv*: *no-equiv* $(conn \ c \ l)$ *no-imp* $(conn \ c \ l)$ *simple-not* $(conn \ c \ l)$
shows $wf\text{-}conn \ c \ l \implies c\text{-in-}c'\text{-only} \ c \ c' \ (conn \ c \ l) \implies (\forall \psi \in \text{set } l. \text{only-c-inside} \ c \ \psi)$
 $\langle proof \rangle$

Push Conjunction

definition *pushConj* **where** $pushConj = \text{push-conn-inside} \ CAnd \ COr$

lemma *pushConj-consistent*: *preserve-models* $pushConj$

$\langle proof \rangle$

definition *and-in-or-symb* **where** $and\text{-in-}or\text{-symb} = c\text{-in-}c'\text{-symb} \ CAnd \ COr$

definition *and-in-or-only* **where**

$and\text{-in-}or\text{-only} = \text{all-subformula-st} \ (c\text{-in-}c'\text{-symb} \ CAnd \ COr)$

lemma *pushConj-inv*:

fixes $\varphi \ \psi :: 'v \text{ propo}$
assumes *full* $(\text{propo-rew-step} \ pushConj) \ \varphi \ \psi$
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
 $\langle proof \rangle$

lemma *pushConj-full-propo-rew-step*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step pushConj*) $\varphi \psi$ **and**
no-T-F-except-top-level φ **and**
simple-not φ
shows *and-in-or-only* ψ
 $\langle \text{proof} \rangle$

Push Disjunction

definition *pushDisj* **where** *pushDisj* = *push-conn-inside COr CAnd*

lemma *pushDisj-consistent: preserve-models pushDisj*
 $\langle \text{proof} \rangle$

definition *or-in-and-symb* **where** *or-in-and-symb* = *c-in-c'-symb COr CAnd*

definition *or-in-and-only* **where**
or-in-and-only = *all-subformula-st (c-in-c'-symb COr CAnd)*

lemma *not-or-in-and-only-or-and[simp]*:
 $\sim \text{or-in-and-only } (FOr (FAnd \psi1 \psi2) \varphi')$
 $\langle \text{proof} \rangle$

lemma *pushDisj-inv*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *full* (*propo-rew-step pushDisj*) $\varphi \psi$
and *no-equiv* φ **and** *no-imp* φ **and** *no-T-F-except-top-level* φ **and** *simple-not* φ
shows *no-equiv* ψ **and** *no-imp* ψ **and** *no-T-F-except-top-level* ψ **and** *simple-not* ψ
 $\langle \text{proof} \rangle$

lemma *pushDisj-full-propo-rew-step*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes
no-equiv φ **and**
no-imp φ **and**
full (*propo-rew-step pushDisj*) $\varphi \psi$ **and**
no-T-F-except-top-level φ **and**
simple-not φ
shows *or-in-and-only* ψ
 $\langle \text{proof} \rangle$

0.4 The Full Transformations

0.4.1 Abstract Definition

The normal form is a super group of groups

inductive *grouped-by* :: *'a connective* \Rightarrow *'a propo* \Rightarrow *bool* **for** *c* **where**
simple-is-grouped[simp]: *simple* $\varphi \Longrightarrow$ *grouped-by* *c* φ |
simple-not-is-grouped[simp]: *simple* $\varphi \Longrightarrow$ *grouped-by* *c* (*FNot* φ) |

*connected-is-group[simp]: grouped-by c $\varphi \implies$ grouped-by c $\psi \implies$ wf-conn c $[\varphi, \psi]$
 \implies grouped-by c (conn c $[\varphi, \psi]$)*

lemma *simple-clause[simp]:*

grouped-by c FT
grouped-by c FF
grouped-by c (FVar x)
grouped-by c (FNot FT)
grouped-by c (FNot FF)
grouped-by c (FNot (FVar x))
 \langle proof \rangle

lemma *only-c-inside-symb-c-eq-c':*

*only-c-inside-symb c (conn c' $[\varphi1, \varphi2]$) \implies c' = CAnd \vee c' = COr \implies wf-conn c' $[\varphi1, \varphi2]$
 \implies c' = c*
 \langle proof \rangle

lemma *only-c-inside-c-eq-c':*

only-c-inside c (conn c' $[\varphi1, \varphi2]$) \implies c' = CAnd \vee c' = COr \implies wf-conn c' $[\varphi1, \varphi2] \implies$ c = c'
 \langle proof \rangle

lemma *only-c-inside-imp-grouped-by:*

assumes c: c \neq CNot **and** c': c' = CAnd \vee c' = COr
shows *only-c-inside c $\varphi \implies$ grouped-by c φ (is ?O $\varphi \implies$?G φ)*
 \langle proof \rangle

lemma *grouped-by-false:*

grouped-by c (conn c' $[\varphi, \psi]$) \implies c \neq c' \implies wf-conn c' $[\varphi, \psi] \implies$ False
 \langle proof \rangle

Then the CNF form is a conjunction of clauses: every clause is in CNF form and two formulas in CNF form can be related by an and.

inductive *super-grouped-by:: 'a connective \implies 'a connective \implies 'a propo \implies bool for c c' where*

grouped-is-super-grouped[simp]: grouped-by c $\varphi \implies$ super-grouped-by c c' φ |
*connected-is-super-group: super-grouped-by c c' $\varphi \implies$ super-grouped-by c c' $\psi \implies$ wf-conn c $[\varphi, \psi]$
 \implies super-grouped-by c c' (conn c' $[\varphi, \psi]$)*

lemma *simple-cnf[simp]:*

super-grouped-by c c' FT
super-grouped-by c c' FF
super-grouped-by c c' (FVar x)
super-grouped-by c c' (FNot FT)
super-grouped-by c c' (FNot FF)
super-grouped-by c c' (FNot (FVar x))
 \langle proof \rangle

lemma *c-in-c'-only-super-grouped-by:*

assumes c: c = CAnd \vee c = COr **and** c': c' = CAnd \vee c' = COr **and** cc': c \neq c'
shows *no-equiv $\varphi \implies$ no-imp $\varphi \implies$ simple-not $\varphi \implies$ c-in-c'-only c c' φ
 \implies super-grouped-by c c' φ*
(is ?NE $\varphi \implies$?NI $\varphi \implies$?SN $\varphi \implies$?C $\varphi \implies$?S φ)
 \langle proof \rangle

0.4.2 Conjunctive Normal Form

Definition

definition *is-conj-with-TF* **where** $is-conj-with-TF \equiv super-grouped-by\ COr\ CAnd$

lemma *or-in-and-only-conjunction-in-disj*:

shows $no-equiv\ \varphi \implies no-imp\ \varphi \implies simple-not\ \varphi \implies or-in-and-only\ \varphi \implies is-conj-with-TF\ \varphi$
<proof>

definition *is-cnf* **where**

$is-cnf\ \varphi \equiv is-conj-with-TF\ \varphi \wedge no-T-F-except-top-level\ \varphi$

Full CNF transformation

The full CNF transformation consists simply in chaining all the transformation defined before.

definition *cnf-rew* **where** $cnf-rew =$

$(full\ (propo-rew-step\ elim-equiv))\ OO$
 $(full\ (propo-rew-step\ elim-imp))\ OO$
 $(full\ (propo-rew-step\ elimTB))\ OO$
 $(full\ (propo-rew-step\ pushNeg))\ OO$
 $(full\ (propo-rew-step\ pushDisj))$

lemma *cnf-rew-equivalent: preserve-models cnf-rew*

<proof>

lemma *cnf-rew-is-cnf: cnf-rew $\varphi\ \varphi' \implies is-cnf\ \varphi'$*

<proof>

0.4.3 Disjunctive Normal Form

Definition

definition *is-disj-with-TF* **where** $is-disj-with-TF \equiv super-grouped-by\ CAnd\ COr$

lemma *and-in-or-only-conjunction-in-disj*:

shows $no-equiv\ \varphi \implies no-imp\ \varphi \implies simple-not\ \varphi \implies and-in-or-only\ \varphi \implies is-disj-with-TF\ \varphi$
<proof>

definition *is-dnf* **:: 'a propo \Rightarrow bool** **where**

$is-dnf\ \varphi \longleftrightarrow is-disj-with-TF\ \varphi \wedge no-T-F-except-top-level\ \varphi$

Full DNF transform

The full DNF transformation consists simply in chaining all the transformation defined before.

definition *dnf-rew* **where** $dnf-rew \equiv$

$(full\ (propo-rew-step\ elim-equiv))\ OO$
 $(full\ (propo-rew-step\ elim-imp))\ OO$
 $(full\ (propo-rew-step\ elimTB))\ OO$
 $(full\ (propo-rew-step\ pushNeg))\ OO$
 $(full\ (propo-rew-step\ pushConj))$

lemma *dnf-rew-consistent: preserve-models dnf-rew*

<proof>

theorem *dnf-transformation-correction*:

$dnf\text{-rew } \varphi \varphi' \implies is\text{-dnf } \varphi'$
 ⟨proof⟩

0.5 More aggressive simplifications: Removing true and false at the beginning

0.5.1 Transformation

We should remove FT and FF at the beginning and not in the middle of the algorithm. To do this, we have to use more rules (one for each connective):

inductive *elimTBFull* **where**

ElimTBFull1[simp]: *elimTBFull* (*FAnd* φ *FT*) φ |
ElimTBFull1'[simp]: *elimTBFull* (*FAnd* *FT* φ) φ |

ElimTBFull2[simp]: *elimTBFull* (*FAnd* φ *FF*) *FF* |
ElimTBFull2'[simp]: *elimTBFull* (*FAnd* *FF* φ) *FF* |

ElimTBFull3[simp]: *elimTBFull* (*FOr* φ *FT*) *FT* |
ElimTBFull3'[simp]: *elimTBFull* (*FOr* *FT* φ) *FT* |

ElimTBFull4[simp]: *elimTBFull* (*FOr* φ *FF*) φ |
ElimTBFull4'[simp]: *elimTBFull* (*FOr* *FF* φ) φ |

ElimTBFull5[simp]: *elimTBFull* (*FNot* *FT*) *FF* |
ElimTBFull5'[simp]: *elimTBFull* (*FNot* *FF*) *FT* |

ElimTBFull6-l[simp]: *elimTBFull* (*FImp* *FT* φ) φ |
ElimTBFull6-l'[simp]: *elimTBFull* (*FImp* *FF* φ) *FT* |
ElimTBFull6-r[simp]: *elimTBFull* (*FImp* φ *FT*) *FT* |
ElimTBFull6-r'[simp]: *elimTBFull* (*FImp* φ *FF*) (*FNot* φ) |

ElimTBFull7-l[simp]: *elimTBFull* (*FEq* *FT* φ) φ |
ElimTBFull7-l'[simp]: *elimTBFull* (*FEq* *FF* φ) (*FNot* φ) |
ElimTBFull7-r[simp]: *elimTBFull* (*FEq* φ *FT*) φ |
ElimTBFull7-r'[simp]: *elimTBFull* (*FEq* φ *FF*) (*FNot* φ)

The transformation is still consistent.

lemma *elimTBFull-consistent*: *preserve-models* *elimTBFull*
 ⟨proof⟩

Contrary to the theorem *no-T-F-symb-except-toplevel-step-exists*, we do not need the assumption *no-equiv* φ and *no-imp* φ , since our transformation is more general.

lemma *no-T-F-symb-except-toplevel-step-exists'*:

fixes $\varphi :: 'v \text{ propo}$

shows $\psi \preceq \varphi \implies \neg no\text{-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTBFull } \psi \psi'$

⟨proof⟩

The same applies here. We do not need the assumption, but the deep link between $\neg no\text{-T-F-except-top-level}$ φ and the existence of a rewriting step, still exists.

lemma *no-T-F-except-top-level-rew'*:

fixes $\varphi :: 'v \text{ propo}$

assumes *noTB*: $\neg no\text{-T-F-except-top-level } \varphi$

shows $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{elimTBFull } \psi \psi'$
 ⟨proof⟩

lemma *elimTBFull-full-propo-rew-step*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *full (propo-rew-step elimTBFull)* $\varphi \psi$
shows *no-T-F-except-top-level* ψ
 ⟨proof⟩

0.5.2 More invariants

As the aim is to use the transformation as the first transformation, we have to show some more invariants for *elim-equiv* and *elim-imp*. For the other transformation, we have already proven it.

lemma *propo-rew-step-ElimEquiv-no-T-F*: *propo-rew-step elim-equiv* $\varphi \psi \implies \text{no-T-F } \varphi \implies \text{no-T-F } \psi$
 ⟨proof⟩

lemma *elim-equiv-inv'*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *full (propo-rew-step elim-equiv)* $\varphi \psi$ **and** *no-T-F-except-top-level* φ
shows *no-T-F-except-top-level* ψ
 ⟨proof⟩

lemma *propo-rew-step-ElimImp-no-T-F*: *propo-rew-step elim-imp* $\varphi \psi \implies \text{no-T-F } \varphi \implies \text{no-T-F } \psi$
 ⟨proof⟩

lemma *elim-imp-inv'*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *full (propo-rew-step elim-imp)* $\varphi \psi$ **and** *no-T-F-except-top-level* φ
shows *no-T-F-except-top-level* ψ
 ⟨proof⟩

0.5.3 The new CNF and DNF transformation

The transformation is the same as before, but the order is not the same.

definition *dnf-rew'* :: *'a propo* \Rightarrow *'a propo* \Rightarrow *bool* **where**
dnf-rew' =

(*full (propo-rew-step elimTBFull)*) *OO*
 (*full (propo-rew-step elim-equiv)*) *OO*
 (*full (propo-rew-step elim-imp)*) *OO*
 (*full (propo-rew-step pushNeg)*) *OO*
 (*full (propo-rew-step pushConj)*)

lemma *dnf-rew'-consistent: preserve-models dnf-rew'*
 ⟨proof⟩

theorem *cnf-transformation-correction*:
dnf-rew' $\varphi \varphi' \implies \text{is-dnf } \varphi'$
 ⟨proof⟩

Given all the lemmas before the CNF transformation is easy to prove:

definition *cnf-rew'* :: 'a propo \Rightarrow 'a propo \Rightarrow bool **where**

cnf-rew' =
 (full (propo-rew-step elimTBFull)) OO
 (full (propo-rew-step elim-equiv)) OO
 (full (propo-rew-step elim-imp)) OO
 (full (propo-rew-step pushNeg)) OO
 (full (propo-rew-step pushDisj))

lemma *cnf-rew'-consistent: preserve-models cnf-rew'*
 <proof>

theorem *cnf'-transformation-correction:*
cnf-rew' φ φ' \Longrightarrow is-cnff φ'
 <proof>

end

theory *Prop-Logic-Multiset*

imports *Nested-Multisets-Ordinals.Multiset-More Prop-Normalisation*
Entailment-Definition.Partial-Herbrand-Interpretation

begin

0.6 Link with Multiset Version

0.6.1 Transformation to Multiset

fun *mset-of-conj* :: 'a propo \Rightarrow 'a literal multiset **where**
mset-of-conj (FOr φ ψ) = *mset-of-conj* φ + *mset-of-conj* ψ |
mset-of-conj (FVar v) = {# Pos v #} |
mset-of-conj (FNot (FVar v)) = {# Neg v #} |
mset-of-conj FF = {#}

fun *mset-of-formula* :: 'a propo \Rightarrow 'a literal multiset set **where**
mset-of-formula (FAnd φ ψ) = *mset-of-formula* φ \cup *mset-of-formula* ψ |
mset-of-formula (FOr φ ψ) = {*mset-of-conj* (FOr φ ψ)} |
mset-of-formula (FVar ψ) = {*mset-of-conj* (FVar ψ)} |
mset-of-formula (FNot ψ) = {*mset-of-conj* (FNot ψ)} |
mset-of-formula FF = {{#}} |
mset-of-formula FT = {}

0.6.2 Equisatisfiability of the two Versions

lemma *is-conj-with-TF-FNot:*
is-conj-with-TF (FNot φ) \longleftrightarrow ($\exists v. \varphi =$ FVar $v \vee \varphi =$ FF $\vee \varphi =$ FT)
 <proof>

lemma *grouped-by-COr-FNot:*
grouped-by COr (FNot φ) \longleftrightarrow ($\exists v. \varphi =$ FVar $v \vee \varphi =$ FF $\vee \varphi =$ FT)
 <proof>

lemma
shows *no-T-F-FF[simp]: \neg no-T-F FF* **and**
no-T-F-FT[simp]: \neg no-T-F FT
 <proof>

lemma *grouped-by-CAnd-FAnd:*
grouped-by CAnd (FAnd φ_1 φ_2) \longleftrightarrow *grouped-by CAnd* $\varphi_1 \wedge$ *grouped-by CAnd* φ_2

<proof>

lemma *grouped-by-COr-FOr:*

grouped-by COr (FOr φ_1 φ_2) \longleftrightarrow grouped-by COr $\varphi_1 \wedge$ grouped-by COr φ_2

<proof>

lemma *grouped-by-COr-FAnd[simp]:* \neg *grouped-by COr (FAnd φ_1 φ_2)*

<proof>

lemma *grouped-by-COr-FEq[simp]:* \neg *grouped-by COr (FEq φ_1 φ_2)*

<proof>

lemma *[simp]:* \neg *grouped-by COr (FImp φ ψ)*

<proof>

lemma *[simp]:* \neg *is-conj-with-TF (FImp φ ψ)*

<proof>

lemma *[simp]:* \neg *is-conj-with-TF (FEq φ ψ)*

<proof>

lemma *is-conj-with-TF-Fand:*

is-conj-with-TF (FAnd φ_1 φ_2) \implies is-conj-with-TF $\varphi_1 \wedge$ is-conj-with-TF φ_2

<proof>

lemma *is-conj-with-TF-FOr:*

is-conj-with-TF (FOr φ_1 φ_2) \implies grouped-by COr $\varphi_1 \wedge$ grouped-by COr φ_2

<proof>

lemma *grouped-by-COr-mset-of-formula:*

grouped-by COr $\varphi \implies$ mset-of-formula $\varphi =$ (if $\varphi = FT$ then $\{\}$ else $\{mset-of-conj \varphi\}$)

<proof>

When a formula is in CNF form, then there is equisatisfiability between the multiset version and the CNF form. Remark that the definition for the entailment are slightly different: (\models) uses a function assigning *True* or *False*, while (\models_s) uses a set where being in the list means entailment of a literal.

theorem *cnf-eval-true-cls:*

fixes $\varphi :: 'v$ *propo*

assumes *is-cnf φ*

shows *eval A $\varphi \longleftrightarrow$ Partial-Herbrand-Interpretation.true-cls ($\{Pos\ v|v. A\ v\} \cup \{Neg\ v|v. \neg A\ v\}$)*

(mset-of-formula φ)

<proof>

function *formula-of-mset* $:: 'a$ *clause* $\Rightarrow 'a$ *propo* **where**

formula-of-mset $\varphi =$

(if $\varphi = \{\#\}$ then FF

else

let $v = (SOME\ v. v \in \#\ \varphi);$

$v' = (if\ is-pos\ v\ then\ FVar\ (atm-of\ v)\ else\ FNot\ (FVar\ (atm-of\ v)))$ in

if remove1-mset $v\ \varphi = \{\#\}$ then v'

else FOr v' (formula-of-mset (remove1-mset $v\ \varphi)))$)

<proof>

termination

⟨proof⟩

lemma *formula-of-mset-empty[simp]*: ⟨formula-of-mset {#} = FF⟩

⟨proof⟩

lemma *formula-of-mset-empty-iff[iff]*: ⟨formula-of-mset $\varphi = FF \longleftrightarrow \varphi = \{\#\}$ ⟩

⟨proof⟩

declare *formula-of-mset.simps[simp del]*

function *formula-of-msets* :: 'a literal multiset set \Rightarrow 'a propo **where**

⟨formula-of-msets $\varphi s =$

(if $\varphi s = \{\}$ \vee infinite φs then FT

else

let $v = (\text{SOME } v. v \in \varphi s);$

$v' = \text{formula-of-mset } v$ in

if $\varphi s - \{v\} = \{\}$ then v'

else FAnd $v' (\text{formula-of-msets } (\varphi s - \{v\}))$)⟩

⟨proof⟩

termination

⟨proof⟩

declare *formula-of-msets.simps[simp del]*

lemma *remove1-mset-empty-iff*:

⟨remove1-mset $v \varphi = \{\#\} \longleftrightarrow (\varphi = \{\#\} \vee \varphi = \{\#v\#\})$ ⟩

⟨proof⟩

definition *fun-of-set* **where**

⟨fun-of-set $A x = (\text{if } \text{Pos } x \in A \text{ then True else if } \text{Neg } x \in A \text{ then False else undefined})$ ⟩

lemma *grouped-by-COr-formula-of-mset*: ⟨grouped-by COr (formula-of-mset φ)⟩

⟨proof⟩

lemma *no-T-F-formula-of-mset*: ⟨no-T-F (formula-of-mset φ) **if** ⟨formula-of-mset $\varphi \neq FF$ ⟩ **for** φ ⟩

⟨proof⟩

lemma *mset-of-conj-formula-of-mset[simp]*: ⟨mset-of-conj(formula-of-mset φ) = φ ⟩ **for** φ

⟨proof⟩

lemma *mset-of-formula-formula-of-mset [simp]*: ⟨mset-of-formula (formula-of-mset φ) = $\{\varphi\}$ ⟩ **for** φ

⟨proof⟩

lemma *formula-of-mset-is-cnf*: ⟨is-cnf (formula-of-mset φ)⟩

⟨proof⟩

lemma *eval-cls-iff*:

assumes ⟨consistent-interp A ⟩ **and** ⟨total-over-set A UNIV⟩

shows ⟨eval (fun-of-set A) (formula-of-mset φ) \longleftrightarrow Partial-Herbrand-Interpretation.true-cls $A \{\varphi\}$ ⟩

⟨proof⟩

lemma *is-conj-with-TF-Fand-iff*:

⟨is-conj-with-TF (FAnd $\varphi 1 \varphi 2$) \longleftrightarrow is-conj-with-TF $\varphi 1 \wedge$ is-conj-with-TF $\varphi 2$ ⟩

⟨proof⟩

lemma *is-CNF-Fand*:

⟨is-cnf (FAnd $\varphi \psi$) \longleftrightarrow (is-cnf $\varphi \wedge$ no-T-F φ) \wedge is-cnf $\psi \wedge$ no-T-F ψ ⟩

⟨proof⟩

lemma *no-T-F-formula-of-mset-iff*: ⟨no-T-F (formula-of-mset φ) \longleftrightarrow $\varphi \neq \{\#\}$ ⟩

⟨proof⟩

lemma *no-T-F-formula-of-msets*:

assumes ⟨finite φ ⟩ **and** ⟨ $\{\#\} \notin \varphi$ ⟩ **and** ⟨ $\varphi \neq \{\}$ ⟩

shows ⟨no-T-F (formula-of-msets (φ))⟩

⟨proof⟩

lemma *is-cnf-formula-of-msets*:

assumes ⟨finite φ ⟩ **and** ⟨ $\{\#\} \notin \varphi$ ⟩

shows ⟨is-cnf (formula-of-msets φ)⟩

⟨proof⟩

lemma *mset-of-formula-formula-of-msets*:

assumes ⟨finite φ ⟩

shows ⟨mset-of-formula (formula-of-msets φ) = φ ⟩

⟨proof⟩

lemma

assumes ⟨consistent-interp A ⟩ **and** ⟨total-over-set A UNIV⟩ **and** ⟨finite φ ⟩ **and** ⟨ $\{\#\} \notin \varphi$ ⟩

shows ⟨eval (fun-of-set A) (formula-of-msets φ) \longleftrightarrow Partial-Herbrand-Interpretation.true-class A φ ⟩

⟨proof⟩

end