

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 26, 2016

Contents

0.1	Rewrite systems and properties	3
0.1.1	Lifting of rewrite rules	3
0.1.2	Consistency preservation	6
0.1.3	Full Lifting	7
0.2	Transformation testing	7
0.2.1	Definition and first properties	7
0.2.2	Invariant conservation	10
0.3	Rewrite Rules	13
0.3.1	Elimination of the equivalences	13
0.3.2	Eliminate Implication	15
0.3.3	Eliminate all the True and False in the formula	16
0.3.4	PushNeg	22
0.3.5	Push inside	27
0.4	The full transformations	41
0.4.1	Abstract Property characterizing that only some connective are inside the others	41
0.4.2	Conjunctive Normal Form	43
0.4.3	Disjunctive Normal Form	45
0.5	More aggressive simplifications: Removing true and false at the beginning	45
0.5.1	Transformation	45
0.5.2	More invariants	47
0.5.3	The new CNF and DNF transformation	51
0.6	Link with Multiset Version	52
0.6.1	Transformation to Multiset	52
0.6.2	Equisatisfiability of the two Version	52

theory *Prop-Abstract-Transformation*

imports *Main Prop-Logic 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* :: $(\text{'}v \text{ propo} \Rightarrow \text{'}v \text{ propo} \Rightarrow \text{bool}) \Rightarrow \text{'}v \text{ propo} \Rightarrow \text{'}v \text{ propo} \Rightarrow \text{bool}$

```

for  $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$  where
global-rel:  $r \ \varphi \ \psi \Longrightarrow \text{propo-rew-step } r \ \varphi \ \psi \mid$ 
propo-rew-one-step-lift:  $\text{propo-rew-step } r \ \varphi \ \varphi' \Longrightarrow \text{wf-conn } c \ (\psi s \ @ \ \varphi \ \# \ \psi s') \Longrightarrow \text{propo-rew-step } r \ (\text{conn } c \ (\psi s \ @ \ \varphi \ \# \ \psi s')) \ (\text{conn } c \ (\psi s \ @ \ \varphi' \ \# \ \psi s'))$ 

```

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  $\text{propo-rew-step } r \ \varphi \ \varphi' \Longrightarrow \exists \ \psi \ \psi'. \ \psi \preceq \varphi \wedge \psi' \preceq \varphi' \wedge r \ \psi \ \psi'$ 
apply (induct rule: propo-rew-step.induct)
using subformula.simps subformula-into-subformula apply blast
using wf-conn-no-arity-change subformula-into-subformula wf-conn-no-arity-change-helper
in-set-conv-decomp by metis

```

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  $\psi \preceq \varphi \Longrightarrow r \ \psi \ \psi' \Longrightarrow (\exists \ \varphi'. \ \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \varphi \ \varphi')$ 
proof (induct  $\varphi$  rule: subformula.induct)
case subformula-refl
then have  $\text{propo-rew-step } r \ \psi \ \psi'$  using propo-rew-step.intros by auto
moreover have  $\psi' \preceq \psi'$  using Prop-Logic.subformula-refl by auto
ultimately show  $\exists \ \varphi'. \ \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \psi \ \varphi'$  by fastforce
next
case (subformula-into-subformula  $\psi'' \ l \ c$ )
note  $IH = \text{this}(4)$  and  $r = \text{this}(5)$  and  $\psi'' = \text{this}(1)$  and  $\text{wf} = \text{this}(2)$  and  $\text{incl} = \text{this}(3)$ 
then obtain  $\varphi'$  where  $*$ :  $\psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \psi'' \ \varphi'$  by metis
moreover obtain  $\xi \ \xi' :: 'v \text{ propo list}$  where
 $l: l = \xi \ @ \ \psi'' \ \# \ \xi'$  using List.split-list  $\psi''$  by metis
ultimately have  $\text{propo-rew-step } r \ (\text{conn } c \ l) \ (\text{conn } c \ (\xi \ @ \ \varphi' \ \# \ \xi'))$ 
using propo-rew-step.intros(2)  $\text{wf}$  by metis
moreover have  $\psi' \preceq \text{conn } c \ (\xi \ @ \ \varphi' \ \# \ \xi')$ 
using  $\text{wf} * \text{wf-conn-no-arity-change}$  Prop-Logic.subformula-into-subformula
by (metis (no-types) in-set-conv-decomp  $l$  wf-conn-no-arity-change-helper)
ultimately show  $\exists \ \varphi'. \ \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ (\text{conn } c \ l) \ \varphi'$  by metis
qed

```

```

lemma propo-rew-step-subformula:
 $(\exists \ \psi \ \psi'. \ \psi \preceq \varphi \wedge r \ \psi \ \psi') \longleftrightarrow (\exists \ \varphi'. \ \text{propo-rew-step } r \ \varphi \ \varphi')$ 
using propo-rew-step-subformula-imp propo-rew-step-subformula-rec by metis+

```

```

lemma consistency-decompose-into-list:
assumes  $\text{wf}: \text{wf-conn } c \ l$  and  $\text{wf}': \text{wf-conn } c \ l'$ 
and  $\text{same}: \forall n. \ A \models l \ ! \ n \longleftrightarrow (A \models l' \ ! \ n)$ 
shows  $A \models \text{conn } c \ l \longleftrightarrow A \models \text{conn } c \ l'$ 
proof (cases  $c$  rule: connective-cases-arity-2)
case nullary
then show  $(A \models \text{conn } c \ l) \longleftrightarrow (A \models \text{conn } c \ l')$  using  $\text{wf} \ \text{wf}'$  by auto
next
case unary note  $c = \text{this}$ 
then obtain  $a$  where  $l: l = [a]$  using wf-conn-Not-decomp  $\text{wf}$  by metis

```

```

obtain  $a'$  where  $l': l' = [a']$  using wf-conn-Not-decomp  $wf' c$  by metis
have  $A \models a \longleftrightarrow A \models a'$  using  $l l'$  by (metis nth-Cons-0 same)
then show  $A \models \text{conn } c \ l \longleftrightarrow A \models \text{conn } c \ l'$  using  $l l' c$  by auto
next
case binary note  $c = \text{this}$ 
then obtain  $a \ b$  where  $l: l = [a, b]$ 
  using wf-conn-bin-list-length list-length2-decomp wf by metis
obtain  $a' \ b'$  where  $l': l' = [a', b']$ 
  using wf-conn-bin-list-length list-length2-decomp wf' c by metis

have  $p: A \models a \longleftrightarrow A \models a' \ A \models b \longleftrightarrow A \models b'$ 
  using  $l l'$  same by (metis diff-Suc-1 nth-Cons' nat.distinct(2)) +
show  $A \models \text{conn } c \ l \longleftrightarrow A \models \text{conn } c \ l'$ 
  using wf c p unfolding binary-connectives-def l l' by auto
qed

```

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 propo-rew-step  $r \ \varphi \ \varphi'$ 
shows  $\exists \psi \ \psi' \ p. r \ \psi \ \psi' \wedge \text{path-to } p \ \varphi \ \psi \wedge \text{replace-at } p \ \varphi \ \psi' = \varphi'$ 
using assms

```

proof (*induct rule: propo-rew-step.induct*)

```

case(global-rel  $\varphi \ \psi$ )
moreover have path-to  $\square \ \varphi \ \varphi$  by auto
moreover have replace-at  $\square \ \varphi \ \psi = \psi$  by auto
ultimately show ?case by metis

```

next

```

case (propo-rew-one-step-lift  $\varphi \ \varphi' \ c \ \xi \ \xi'$ ) note  $\text{rel} = \text{this}(1)$  and  $\text{IH0} = \text{this}(2)$  and  $\text{corr} = \text{this}(3)$ 
obtain  $\psi \ \psi' \ p$  where  $\text{IH}: r \ \psi \ \psi' \wedge \text{path-to } p \ \varphi \ \psi \wedge \text{replace-at } p \ \varphi \ \psi' = \varphi'$  using  $\text{IH0}$  by metis

```

```

{
  fix  $x :: 'v$ 
  assume  $c = CT \vee c = CF \vee c = CVar \ x$ 
  then have False using corr by auto
  then have  $\exists \psi \ \psi' \ p. r \ \psi \ \psi' \wedge \text{path-to } p \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi$ 
     $\wedge \text{replace-at } p \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi' = \text{conn } c \ (\xi @ (\varphi' \# \xi'))$ 
    by fast
}
moreover {
  assume  $c: c = CNot$ 
  then have empty:  $\xi = [] \ \xi' = []$  using corr by auto
  have path-to  $(L \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi$ 
    using  $c \ \text{empty} \ \text{IH} \ \text{wf-conn-unary path-to-l}$  by fastforce
  moreover have replace-at  $(L \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi' = \text{conn } c \ (\xi @ (\varphi' \# \xi'))$ 
    using  $c \ \text{empty} \ \text{IH}$  by auto
  ultimately have  $\exists \psi \ \psi' \ p. r \ \psi \ \psi' \wedge \text{path-to } p \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi$ 
     $\wedge \text{replace-at } p \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi' = \text{conn } c \ (\xi @ (\varphi' \# \xi'))$ 
    using  $\text{IH}$  by metis
}
moreover {
  assume  $c: c \in \text{binary-connectives}$ 
  have length  $(\xi @ \varphi \# \xi') = 2$  using wf-conn-bin-list-length corr c by metis
  then have length  $\xi + \text{length } \xi' = 1$  by auto
  then have ld:  $(\text{length } \xi = 1 \wedge \text{length } \xi' = 0) \vee (\text{length } \xi = 0 \wedge \text{length } \xi' = 1)$  by arith
}

```

```

obtain  $a\ b$  where  $ab$ :  $(\xi = [] \wedge \xi' = [b]) \vee (\xi = [a] \wedge \xi' = [])$ 
using  $ld$  by ( $case\text{-}tac\ \xi$ ,  $case\text{-}tac\ \xi'$ ,  $auto$ )
{
  assume  $\varphi$ :  $\xi = [] \wedge \xi' = [b]$ 
  have  $path\text{-}to\ (L\#p)\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi$ 
    using  $\varphi\ c\ IH\ ab\ corr$  by ( $simp\ add$ :  $path\text{-}to\text{-}l$ )
  moreover have  $replace\text{-}at\ (L\#p)\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi' = conn\ c\ (\xi @ (\varphi' \# \xi'))$ 
    using  $c\ IH\ ab\ \varphi$  unfolding  $binary\text{-}connectives\text{-}def$  by  $auto$ 
  ultimately have  $\exists\ \psi\ \psi'\ p.\ r\ \psi\ \psi' \wedge path\text{-}to\ p\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi$ 
     $\wedge\ replace\text{-}at\ p\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi' = conn\ c\ (\xi @ (\varphi' \# \xi'))$ 
    using  $IH$  by  $metis$ 
}
moreover {
  assume  $\varphi$ :  $\xi = [a]\ \xi' = []$ 
  then have  $path\text{-}to\ (R\#p)\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi$ 
    using  $c\ IH\ corr\ path\text{-}to\text{-}r\ corr\ \varphi$  by ( $simp\ add$ :  $path\text{-}to\text{-}r$ )
  moreover have  $replace\text{-}at\ (R\#p)\ (conn\ c\ (\xi @ (\varphi \# \xi')))\ \psi' = conn\ c\ (\xi @ (\varphi' \# \xi'))$ 
    using  $c\ IH\ ab\ \varphi$  unfolding  $binary\text{-}connectives\text{-}def$  by  $auto$ 
  ultimately have  $?case$  using  $IH$  by  $metis$ 
}
ultimately have  $?case$  using  $ab$  by  $blast$ 
}
ultimately show  $?case$  using  $connective\text{-}cases\text{-}arity$  by  $blast$ 
qed

```

0.1.2 Consistency preservation

We define *preserves-un-sat*: it means that a relation preserves consistency.

definition *preserves-un-sat* **where**

$preserves\text{-}un\text{-}sat\ r \longleftrightarrow (\forall\ \varphi\ \psi.\ r\ \varphi\ \psi \longrightarrow (\forall\ A.\ A \models \varphi \longleftrightarrow A \models \psi))$

lemma *propo-rew-step-preservers-val-explicit*:

$propo\text{-}rew\text{-}step\ r\ \varphi\ \psi \implies preserves\text{-}un\text{-}sat\ r \implies propo\text{-}rew\text{-}step\ r\ \varphi\ \psi \implies (\forall\ A.\ A \models \varphi \longleftrightarrow A \models \psi)$

unfolding *preserves-un-sat-def*

proof (*induction rule*: *propo-rew-step.induct*)

case *global-rel*

then **show** $?case$ **by** *simp*

next

case (*propo-rew-one-step-lift* $\varphi\ \varphi'\ c\ \xi\ \xi'$) **note** $rel = this(1)$ **and** $wf = this(2)$

and $IH = this(3)[OF\ this(4)\ this(1)]$ **and** $consistent = this(4)$

{

fix A

from IH **have** $\forall n.\ (A \models (\xi @ \varphi \# \xi') ! n) = (A \models (\xi @ \varphi' \# \xi') ! n)$

by (*metis* (*mono-tags*, *hide-lams*) *list-update-length* *nth-Cons-0* *nth-append-length-plus* *nth-list-update-neq*)

then **have** $(A \models conn\ c\ (\xi @ \varphi \# \xi')) = (A \models conn\ c\ (\xi @ \varphi' \# \xi'))$

by (*meson* *consistency-decompose-into-list* $wf\ wf\text{-}conn\text{-}no\text{-}arity\text{-}change\text{-}helper$ *wf-conn-no-arity-change*)

}

then **show** $\forall A.\ A \models conn\ c\ (\xi @ \varphi \# \xi') \longleftrightarrow A \models conn\ c\ (\xi @ \varphi' \# \xi')$ **by** *auto*

qed

lemma *propo-rew-step-preservers-val'*:

assumes *preserves-un-sat* *r*
shows *preserves-un-sat* (*propo-rew-step* *r*)
using *assms* **by** (*simp* *add*: *preserves-un-sat-def propo-rew-step-preservers-val-explicit*)

lemma *preserves-un-sat-OO[intro]*:
preserves-un-sat *f* \implies *preserves-un-sat* *g* \implies *preserves-un-sat* (*f* *OO* *g*)
unfolding *preserves-un-sat-def* **by** *auto*

lemma *star-consistency-preservation-explicit*:
assumes (*propo-rew-step* *r*)^{**} $\varphi \psi$ **and** *preserves-un-sat* *r*
shows $\forall A. A \models \varphi \longleftrightarrow A \models \psi$
using *assms* **by** (*induct* *rule*: *rtranclp-induct*)
(auto simp add: propo-rew-step-preservers-val-explicit)

lemma *star-consistency-preservation*:
preserves-un-sat *r* \implies *preserves-un-sat* (*propo-rew-step* *r*)^{**}
by (*simp* *add*: *star-consistency-preservation-explicit preserves-un-sat-def*)

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]*:
preserves-un-sat *r* \implies *preserves-un-sat* (*full* (*propo-rew-step* *r*))
by (*metis* *full-def preserves-un-sat-def star-consistency-preservation*)

lemma *full-propo-rew-step-subformula*:
full (*propo-rew-step* *r*) $\varphi' \varphi \implies \neg(\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi')$
unfolding *full-def* **using** *propo-rew-step-subformula-rec* **by** *metis*

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* :: (*'a* *propo* \Rightarrow *bool*) \Rightarrow *'a* *propo* \Rightarrow *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]*:
test-symb *FT* \implies *all-subformula-st* *test-symb* *FT*
test-symb *FF* \implies *all-subformula-st* *test-symb* *FF*
test-symb (*FVar* *x*) \implies *all-subformula-st* *test-symb* (*FVar* *x*)
unfolding *all-subformula-st-def* **using** *subformula-leaf* **by** *metis+*

lemma *all-subformula-st-test-symb-true-phi*:

all-subformula-st test-symb $\varphi \implies \text{test-symb } \varphi$
unfolding *all-subformula-st-def* **by** *auto*

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

wf-conn $c \ l \implies (\text{test-symb } (\text{conn } c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st test-symb } \varphi))$
 $\implies \text{all-subformula-st test-symb } (\text{conn } c \ l)$
unfolding *all-subformula-st-def* **by** *auto*

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

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

all-subformula-st test-symb $(\text{conn } c \ l) \implies \text{wf-conn } c \ l$
 $\implies (\text{test-symb } (\text{conn } c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st test-symb } \varphi))$
unfolding *all-subformula-st-def* **by** *auto*

lemma *all-subformula-st-decomp*:

fixes $c :: 'v \text{ connective}$ **and** $l :: 'v \text{ propo list}$
assumes *wf-conn* $c \ l$
shows *all-subformula-st test-symb* $(\text{conn } c \ l)$
 $\longleftrightarrow (\text{test-symb } (\text{conn } c \ l) \wedge (\forall \varphi \in \text{set } l. \text{all-subformula-st test-symb } \varphi))$
using *assms all-subformula-st-decomp-rec all-subformula-st-decomp-imp* **by** *metis*

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

unfolding *binary-connectives-def* **by** *auto*

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

fixes $\varphi \ \psi :: 'v \text{ propo}$
shows *all-subformula-st test-symb* $(\text{FAnd } \varphi \ \psi)$
 $\longleftrightarrow (\text{test-symb } (\text{FAnd } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$
and *all-subformula-st test-symb* $(\text{FOr } \varphi \ \psi)$
 $\longleftrightarrow (\text{test-symb } (\text{FOr } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$
and *all-subformula-st test-symb* $(\text{FNot } \varphi)$
 $\longleftrightarrow (\text{test-symb } (\text{FNot } \varphi) \wedge \text{all-subformula-st test-symb } \varphi)$
and *all-subformula-st test-symb* $(\text{FEq } \varphi \ \psi)$
 $\longleftrightarrow (\text{test-symb } (\text{FEq } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$
and *all-subformula-st test-symb* $(\text{FImp } \varphi \ \psi)$
 $\longleftrightarrow (\text{test-symb } (\text{FImp } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$

proof –

have *all-subformula-st test-symb* $(\text{FAnd } \varphi \ \psi) \longleftrightarrow \text{all-subformula-st test-symb } (\text{conn } \text{CAnd } [\varphi, \psi])$
by *auto*

moreover have $\dots \longleftrightarrow \text{test-symb } (\text{conn } \text{CAnd } [\varphi, \psi]) \wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi)$

using *all-subformula-st-decomp wf-conn-helper-facts(5)* **by** *metis*

finally show *all-subformula-st test-symb* $(\text{FAnd } \varphi \ \psi)$

$\longleftrightarrow (\text{test-symb } (\text{FAnd } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$
by *simp*

have *all-subformula-st test-symb* $(\text{FOr } \varphi \ \psi) \longleftrightarrow \text{all-subformula-st test-symb } (\text{conn } \text{COr } [\varphi, \psi])$
by *auto*

moreover have $\dots \longleftrightarrow$

$(\text{test-symb } (\text{conn } \text{COr } [\varphi, \psi]) \wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi))$

using *all-subformula-st-decomp wf-conn-helper-facts(6)* **by** *metis*

finally show *all-subformula-st test-symb* $(\text{FOr } \varphi \ \psi)$

$\longleftrightarrow (\text{test-symb } (\text{FOr } \varphi \ \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$
by *simp*

have *all-subformula-st test-symb* $(\text{FEq } \varphi \ \psi) \longleftrightarrow \text{all-subformula-st test-symb } (\text{conn } \text{CEq } [\varphi, \psi])$
by *auto*


```

moreover have ...
   $\longleftrightarrow$  (test-symb (conn CEq  $[\varphi, \psi]$ )  $\wedge$  ( $\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi$ ))
  using all-subformula-st-decomp wf-conn-helper-facts(8) by metis
finally show all-subformula-st test-symb (FEq  $\varphi \psi$ )
   $\longleftrightarrow$  (test-symb (FEq  $\varphi \psi$ )  $\wedge$  all-subformula-st test-symb  $\varphi \wedge$  all-subformula-st test-symb  $\psi$ )
  by simp

have all-subformula-st test-symb (FImp  $\varphi \psi$ )  $\longleftrightarrow$  all-subformula-st test-symb (conn CImp  $[\varphi, \psi]$ )
  by auto
moreover have ...
   $\longleftrightarrow$  (test-symb (conn CImp  $[\varphi, \psi]$ )  $\wedge$  ( $\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi$ ))
  using all-subformula-st-decomp wf-conn-helper-facts(7) by metis
finally show all-subformula-st test-symb (FImp  $\varphi \psi$ )
   $\longleftrightarrow$  (test-symb (FImp  $\varphi \psi$ )  $\wedge$  all-subformula-st test-symb  $\varphi \wedge$  all-subformula-st test-symb  $\psi$ )
  by simp

have all-subformula-st test-symb (FNot  $\varphi$ )  $\longleftrightarrow$  all-subformula-st test-symb (conn CNot  $[\varphi]$ )
  by auto
moreover have ... = (test-symb (conn CNot  $[\varphi]$ )  $\wedge$  ( $\forall \xi \in \text{set } [\varphi]. \text{all-subformula-st test-symb } \xi$ ))
  using all-subformula-st-decomp wf-conn-helper-facts(1) by metis
finally show all-subformula-st test-symb (FNot  $\varphi$ )
   $\longleftrightarrow$  (test-symb (FNot  $\varphi$ )  $\wedge$  all-subformula-st test-symb  $\varphi$ ) by simp
qed

```

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

lemma *subformula-all-subformula-st*:

```

 $\psi \preceq \varphi \implies \text{all-subformula-st test-symb } \varphi \implies \text{all-subformula-st test-symb } \psi$ 
by (induct rule: subformula.induct, auto simp add: all-subformula-st-decomp)

```

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 $\neg \text{all-subformula-st test-symb } \varphi$, then something can be rewritten in φ .

lemma *no-test-symb-step-exists*:

```

fixes r:: 'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  bool and test-symb:: 'v propo  $\Rightarrow$  bool and x:: 'v
and  $\varphi$ :: 'v propo
assumes
  test-symb-false-nullary:  $\forall x. \text{test-symb } FF \wedge \text{test-symb } FT \wedge \text{test-symb } (FVar\ x) \text{ and }$ 
   $\forall \varphi'. \varphi' \preceq \varphi \longrightarrow (\neg \text{test-symb } \varphi') \longrightarrow (\exists \psi. r\ \varphi'\ \psi) \text{ and }$ 
   $\neg \text{all-subformula-st test-symb } \varphi$ 
shows  $\exists \psi\ \psi'. \psi \preceq \varphi \wedge r\ \psi\ \psi'$ 
using assms
proof (induct  $\varphi$  rule: propo-induct-arity)
case (nullary  $\varphi\ x$ )
then show  $\exists \psi\ \psi'. \psi \preceq \varphi \wedge r\ \psi\ \psi'$ 
  using wf-conn-nullary test-symb-false-nullary by fastforce
next
case (unary  $\varphi$ ) note IH = this(1)[OF this(2)] and r = this(2) and nst = this(3) and subf = this(4)
from r IH nst have H:  $\neg \text{all-subformula-st test-symb } \varphi \implies \exists \psi. \psi \preceq \varphi \wedge (\exists \psi'. r\ \psi\ \psi')$ 
by (metis subformula-in-subformula-not subformula-refl subformula-trans)
{
  assume n:  $\neg \text{test-symb } (FNot\ \varphi)$ 
  obtain  $\psi$  where  $r\ (FNot\ \varphi)\ \psi$  using subformula-refl r n nst by blast
  moreover have  $FNot\ \varphi \preceq FNot\ \varphi$  using subformula-refl by auto

```

```

    ultimately have  $\exists \psi \psi'. \psi \preceq FNot \varphi \wedge r \psi \psi'$  by metis
  }
  moreover {
    assume  $n: test-symb (FNot \varphi)$ 
    then have  $\neg all-subformula-st test-symb \varphi$ 
      using  $all-subformula-st-decomp-explicit(3) nst subf$  by blast
    then have  $\exists \psi \psi'. \psi \preceq FNot \varphi \wedge r \psi \psi'$ 
      using  $H subformula-in-subformula-not subformula-refl subformula-trans$  by blast
  }
  ultimately show  $\exists \psi \psi'. \psi \preceq FNot \varphi \wedge r \psi \psi'$  by blast
next
case (binary  $\varphi \varphi1 \varphi2$ )
note  $IH\varphi1-0 = this(1)[OF this(4)]$  and  $IH\varphi2-0 = this(2)[OF this(4)]$  and  $r = this(4)$ 
and  $\varphi = this(3)$  and  $le = this(5)$  and  $nst = this(6)$ 

obtain  $c :: 'v$  connective where
 $c: (c = CAnd \vee c = COr \vee c = CImp \vee c = CEq) \wedge conn\ c\ [\varphi1, \varphi2] = \varphi$ 
using  $\varphi$  by fastforce

then have  $corr: wf-conn\ c\ [\varphi1, \varphi2]$  using  $wf-conn.simps$  unfolding  $binary-connectives-def$  by auto
have  $inc: \varphi1 \preceq \varphi \varphi2 \preceq \varphi$  using  $binary-connectives-def\ c\ subformula-in-binary-conn$  by blast+
from  $r\ IH\varphi1-0$  have  $IH\varphi1: \neg all-subformula-st test-symb \varphi1 \implies \exists \psi \psi'. \psi \preceq \varphi1 \wedge r \psi \psi'$ 
  using  $inc(1)\ subformula-trans\ le$  by blast
from  $r\ IH\varphi2-0$  have  $IH\varphi2: \neg all-subformula-st test-symb \varphi2 \implies \exists \psi. \psi \preceq \varphi2 \wedge (\exists \psi'. r \psi \psi')$ 
  using  $inc(2)\ subformula-trans\ le$  by blast
have cases:  $\neg test-symb \varphi \vee \neg all-subformula-st test-symb \varphi1 \vee \neg all-subformula-st test-symb \varphi2$ 
  using  $c\ nst$  by auto
show  $\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi'$ 
  using  $IH\varphi1\ IH\varphi2\ subformula-trans\ inc\ subformula-refl\ cases\ le$  by blast
qed

```

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 $\forall \varphi' \psi. \varphi' \preceq \Phi \longrightarrow r \varphi' \psi \longrightarrow all-subformula-st test-symb \varphi' \longrightarrow all-subformula-st test-symb \psi$ 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: $\forall c\ \xi\ \varphi\ \xi'\ \varphi'. \varphi \preceq \Phi \longrightarrow propo-rew-step\ r\ \varphi\ \varphi' \longrightarrow wf-conn\ c\ (\xi @ \varphi \# \xi') \longrightarrow test-symb (conn\ c\ (\xi @ \varphi \# \xi')) \longrightarrow test-symb \varphi' \longrightarrow test-symb (conn\ c\ (\xi @ \varphi' \# \xi'))$

Invariant while lifting of the rewriting relation

The condition $\varphi \preceq \Phi$ (that will be 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 $r :: 'v\ propo \Rightarrow 'v\ propo \Rightarrow bool$ **and** $test-symb :: 'v\ propo \Rightarrow bool$ **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  $\text{all-subformula-st test-symb } \psi$ 
using  $\text{assms}(3-5)$ 
proof (induct rule: propo-rew-step.induct)
case global-rel
then show ?case using  $H$  by simp
next
case (propo-rew-one-step-lift  $\varphi \varphi' c \xi \xi'$ )
note  $\text{rel} = \text{this}(1)$  and  $\varphi = \text{this}(2)$  and  $\text{corr} = \text{this}(3)$  and  $\Phi = \text{this}(4)$  and  $\text{nst} = \text{this}(5)$ 
have  $\text{sq}: \varphi \preceq \Phi$ 
using  $\Phi \text{ corr subformula-into-subformula subformula-refl subformula-trans}$ 
by (metis in-set-conv-decomp)
from  $\text{corr}$  have  $\forall \psi. \psi \in \text{set } (\xi @ \varphi \# \xi') \longrightarrow \text{all-subformula-st test-symb } \psi$ 
using  $\text{all-subformula-st-decomp nst}$  by blast
then have  $*$ :  $\forall \psi. \psi \in \text{set } (\xi @ \varphi' \# \xi') \longrightarrow \text{all-subformula-st test-symb } \psi$  using  $\varphi \text{ sq}$  by fastforce
then have  $\text{test-symb } \varphi'$  using  $\text{all-subformula-st-test-symb-true-phi}$  by auto
moreover from  $\text{corr nst}$  have  $\text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi'))$ 
using  $\text{all-subformula-st-decomp}$  by blast
ultimately have  $\text{test-symb: test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$  using  $H' \text{ sq corr rel}$  by blast

have  $\text{wf-conn } c (\xi @ \varphi' \# \xi')$ 
by (metis wf-conn-no-arity-change-helper corr wf-conn-no-arity-change)
then show  $\text{all-subformula-st test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ 
using  $*$   $\text{test-symb}$  by (metis all-subformula-st-decomp)
qed

```

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  $\text{all-subformula-st test-symb } \psi$ 
using  $\text{propo-rew-step-inv-stay}[\text{of } \varphi \text{ } r \text{ test-symb } \varphi \psi] \text{ assms subformula-refl}$  by metis

```

The lemmas can be lifted to *propo-rew-step* r^\perp 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
 $H: \forall \varphi \psi. \text{propo-rew-step } r \varphi \psi \longrightarrow \text{all-subformula-st test-symb } \varphi$ 

```

```

  → all-subformula-st test-symb ψ and
H': ∀ (c:: 'v connective) ξ φ ξ' φ'. φ ≼ Φ → propo-rew-step r φ φ'
  → wf-conn c (ξ @ φ # ξ') → test-symb (conn c (ξ @ φ # ξ')) → test-symb φ'
  → test-symb (conn c (ξ @ φ' # ξ')) and
  φ ≼ Φ and
full: full (propo-rew-step r) φ ψ and
init: all-subformula-st test-symb φ
shows all-subformula-st test-symb ψ
using assms unfolding full-def
proof -
have rel: (propo-rew-step r)** φ ψ
  using full unfolding full-def by auto
then show all-subformula-st test-symb ψ
  using init
proof (induct rule: rtrancplp-induct)
  case base
  then show all-subformula-st test-symb φ by blast
next
  case (step b c) note star = this(1) and IH = this(3) and one = this(2) and all = this(4)
  then have all-subformula-st test-symb b by metis
  then show all-subformula-st test-symb c using propo-rew-step-inv-stay' H H' rel one by auto
qed
qed

lemma full-propo-rew-step-inv-stay':
fixes r:: 'v propo ⇒ 'v propo ⇒ bool and test-symb:: 'v propo ⇒ bool and x:: 'v
and φ ψ:: 'v propo
assumes
  H: ∀ φ ψ. propo-rew-step r φ ψ → all-subformula-st test-symb φ
  → all-subformula-st test-symb ψ and
  H': ∀ (c:: 'v connective) ξ φ ξ' φ'. propo-rew-step r φ φ' → wf-conn c (ξ @ φ # ξ')
  → test-symb (conn c (ξ @ φ # ξ')) → test-symb φ' → test-symb (conn c (ξ @ φ' # ξ')) and
  full: full (propo-rew-step r) φ ψ and
  init: all-subformula-st test-symb φ
shows all-subformula-st test-symb ψ
using full-propo-rew-step-inv-stay-with-inc[of r test-symb φ] assms subformula-refl by metis

lemma full-propo-rew-step-inv-stay:
fixes r:: 'v propo ⇒ 'v propo ⇒ bool and test-symb:: 'v propo ⇒ bool and x:: 'v
and φ ψ:: 'v propo
assumes
  H: ∀ φ ψ. r φ ψ → all-subformula-st test-symb φ → all-subformula-st test-symb ψ and
  H': ∀ (c:: 'v connective) ξ φ ξ' φ'. wf-conn c (ξ @ φ # ξ') → test-symb (conn c (ξ @ φ # ξ'))
  → test-symb φ' → test-symb (conn c (ξ @ φ' # ξ')) and
  full: full (propo-rew-step r) φ ψ and
  init: all-subformula-st test-symb φ
shows all-subformula-st test-symb ψ
unfolding full-def
proof -
have rel: (propo-rew-step r)^** φ ψ
  using full unfolding full-def by auto
then show all-subformula-st test-symb ψ
  using init
proof (induct rule: rtrancplp-induct)
  case base
  then show all-subformula-st test-symb φ by blast

```

```

next
  case (step b c)
  note star = this(1) and IH = this(3) and one = this(2) and all = this(4)
  then have all-subformula-st test-symb b by metis
  then show all-subformula-st test-symb c
    using propo-rew-step-inv-stay subformula-refl H H' rel one by auto
qed
qed

lemma full-propo-rew-step-inv-stay-conn:
  fixes r:: 'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  bool and test-symb:: 'v propo  $\Rightarrow$  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
    full: full (propo-rew-step r)  $\varphi \psi$  and
    init: all-subformula-st test-symb  $\varphi$ 
  shows all-subformula-st test-symb  $\psi$ 
proof -
  have  $\bigwedge (c:: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \text{wf-conn } c (\xi @ \varphi \# \xi') \Longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \Longrightarrow \text{test-symb } \varphi' \Longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ 
    using H' by (metis wf-conn-no-arity-change-helper wf-conn-no-arity-change)
  then show all-subformula-st test-symb  $\psi$ 
    using H full init full-propo-rew-step-inv-stay by blast
qed

end
theory Prop-Normalisation
imports Main Prop-Logic Prop-Abstract-Transformation ../lib/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  $\varphi \psi$ ) (FAnd (FImp  $\varphi \psi$ ) (FImp  $\psi \varphi$ ))

```

```

lemma elim-equiv-transformation-consistent:
  A  $\models$  FEq  $\varphi \psi \longleftrightarrow A \models$  FAnd (FImp  $\varphi \psi$ ) (FImp  $\psi \varphi$ )
  by auto

```

```

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

```

by (*induct rule: elim-equiv.induct, auto*)

lemma *elim-equiv-consistent: preserves-un-sat elim-equiv*
unfolding *preserves-un-sat-def* **by** (*simp add: elim-equiv-explicit*)

lemma *elimEquiv-lifted-consistant:*
preserves-un-sat (full (propo-rew-step elim-equiv))
by (*simp add: elim-equiv-consistent*)

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 CEq*
by (*metis connective.distinct(13,25,35,43) wf no-equiv-symb.elims(3) no-equiv-symb.simps(1)*
wf-conn.cases wf-conn-list(6))

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

lemma *no-equiv-eq[simp]:*
fixes $\varphi \psi$:: 'v propo
shows
 \neg *no-equiv (FEq $\varphi \psi$)*
no-equiv FT
no-equiv FF
using *no-equiv-symb.simps(1) all-subformula-st-test-symb-true-phi* **unfolding** *no-equiv-def* **by** *auto*

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 propo
shows
no-equiv (FNot φ) \longleftrightarrow no-equiv φ
no-equiv (FAnd $\varphi \psi$) \longleftrightarrow (no-equiv $\varphi \wedge$ no-equiv ψ)
no-equiv (FOr $\varphi \psi$) \longleftrightarrow (no-equiv $\varphi \vee$ no-equiv ψ)
no-equiv (FImp $\varphi \psi$) \longleftrightarrow (no-equiv $\varphi \wedge$ no-equiv ψ)
by (*auto simp: no-equiv-def*)

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 φ :: 'v propo
assumes *no-equiv: \neg no-equiv φ*
shows $\exists \psi \psi'. \psi \preceq \varphi \wedge$ *elim-equiv $\psi \psi'$*
proof –
have *test-symb-false-nullary:*
 $\forall x::'v. \text{no-equiv-symb } FF \wedge \text{no-equiv-symb } FT \wedge \text{no-equiv-symb } (FVar\ x)$
unfolding *no-equiv-def* **by** *auto*

```

moreover {
  fix c:: 'v connective and l:: 'v propo list and ψ:: 'v propo
    assume a1: elim-equiv (conn c l) ψ
    have  $\bigwedge p$  pa.  $\neg$  elim-equiv (p::'v propo) pa  $\vee$   $\neg$  no-equiv-symb p
      using elim-equiv.cases no-equiv-symb.simps(1) by blast
    then have elim-equiv (conn c l) ψ  $\implies$   $\neg$ no-equiv-symb (conn c l) using a1 by metis
}
moreover have H':  $\forall \psi$ .  $\neg$ elim-equiv FT ψ  $\forall \psi$ .  $\neg$ elim-equiv FF ψ  $\forall \psi$  x.  $\neg$ elim-equiv (FVar x) ψ
using elim-equiv.cases by auto
moreover have  $\bigwedge \varphi$ .  $\neg$  no-equiv-symb φ  $\implies$   $\exists \psi$ . elim-equiv φ ψ
by (case-tac φ, auto simp: elim-equiv.simps)
then have  $\bigwedge \varphi'$ .  $\varphi' \preceq \varphi \implies \neg$ no-equiv-symb φ'  $\implies$   $\exists \psi$ . elim-equiv φ' ψ by force
ultimately show ?thesis
using no-test-symb-step-exists no-equiv test-symb-false-nullary unfolding no-equiv-def by blast
qed

```

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) φ ψ  $\implies$  no-equiv ψ
using full-propo-rew-step-subformula no-equiv-elim-equiv-step by blast

```

0.3.2 Eliminate Implication

After that, we can eliminate the implication symbols.

```

inductive elim-imp :: 'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  bool where
[simp]: elim-imp (FImp φ ψ) (FOr (FNot φ) ψ)

```

```

lemma elim-imp-transformation-consistent:
  A  $\models$  FImp φ ψ  $\longleftrightarrow$  A  $\models$  FOr (FNot φ) ψ
by auto

```

```

lemma elim-imp-explicit: elim-imp φ ψ  $\implies$   $\forall A$ . A  $\models$  φ  $\longleftrightarrow$  A  $\models$  ψ
by (induct φ ψ rule: elim-imp.induct, auto)

```

```

lemma elim-imp-consistent: preserves-un-sat elim-imp
unfolding preserves-un-sat-def by (simp add: elim-imp-explicit)

```

```

lemma elim-imp-lifted-consistent:
  preserves-un-sat (full (propo-rew-step elim-imp))
by (simp add: elim-imp-consistent)

```

```

fun no-imp-symb where
no-imp-symb (FImp -) = False |
no-imp-symb - = True

```

```

lemma no-imp-symb-conn-characterization:
  wf-conn c l  $\implies$  no-imp-symb (conn c l)  $\longleftrightarrow$  c  $\neq$  CImp
by (induction rule: wf-conn-induct) auto

```

```

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)

```

$no\text{-}imp\ FT$
 $no\text{-}imp\ FF$
unfolding $no\text{-}imp\text{-}def$ **by** $auto$

lemma $all\text{-}subformula\text{-}st\text{-}decomp\text{-}explicit\text{-}imp[simp]$:
fixes $\varphi\ \psi :: 'v\ propo$
shows
 $no\text{-}imp\ (FNot\ \varphi) \longleftrightarrow no\text{-}imp\ \varphi$
 $no\text{-}imp\ (FAnd\ \varphi\ \psi) \longleftrightarrow (no\text{-}imp\ \varphi \wedge no\text{-}imp\ \psi)$
 $no\text{-}imp\ (FOr\ \varphi\ \psi) \longleftrightarrow (no\text{-}imp\ \varphi \wedge no\text{-}imp\ \psi)$
by $auto$

Invariant of the $elim\text{-}imp$ transformation

lemma $elim\text{-}imp\text{-}no\text{-}equiv$:
 $elim\text{-}imp\ \varphi\ \psi \implies no\text{-}equiv\ \varphi \implies no\text{-}equiv\ \psi$
by ($induct\ \varphi\ \psi$ $rule$: $elim\text{-}imp.induct$, $auto$)

lemma $elim\text{-}imp\text{-}inv$:
fixes $\varphi\ \psi :: 'v\ propo$
assumes $full\ (propo\text{-}rew\text{-}step\ elim\text{-}imp)\ \varphi\ \psi$ **and** $no\text{-}equiv\ \varphi$
shows $no\text{-}equiv\ \psi$
using $full\text{-}propo\text{-}rew\text{-}step\text{-}inv\text{-}stay\text{-}conn[of\ elim\text{-}imp\ no\text{-}equiv\text{-}symb\ \varphi\ \psi]$ $assms\ elim\text{-}imp\text{-}no\text{-}equiv$
 $no\text{-}equiv\text{-}symb\text{-}conn\text{-}characterization$ **unfolding** $no\text{-}equiv\text{-}def$ **by** $metis$

lemma $no\text{-}no\text{-}imp\text{-}elim\text{-}imp\text{-}step\text{-}exists$:
fixes $\varphi :: 'v\ propo$
assumes $no\text{-}equiv$: $\neg no\text{-}imp\ \varphi$
shows $\exists\ \psi\ \psi'.\ \psi \preceq \varphi \wedge elim\text{-}imp\ \psi\ \psi'$

proof –

have $test\text{-}symb\text{-}false\text{-}nullary$: $\forall x.\ no\text{-}imp\text{-}symb\ FF \wedge no\text{-}imp\text{-}symb\ FT \wedge no\text{-}imp\text{-}symb\ (FVar\ (x:: 'v))$
by $auto$

moreover {

fix $c:: 'v\ connective$ **and** $l :: 'v\ propo\ list$ **and** $\psi :: 'v\ propo$
have H : $elim\text{-}imp\ (conn\ c\ l)\ \psi \implies \neg no\text{-}imp\text{-}symb\ (conn\ c\ l)$
by ($auto\ elim$: $elim\text{-}imp.cases$)

}

moreover

have H' : $\forall\ \psi.\ \neg elim\text{-}imp\ FT\ \psi\ \forall\ \psi.\ \neg elim\text{-}imp\ FF\ \psi\ \forall\ \psi\ x.\ \neg elim\text{-}imp\ (FVar\ x)\ \psi$
by ($auto\ elim$: $elim\text{-}imp.cases$) $+$

moreover

have $\bigwedge\ \varphi.\ \neg no\text{-}imp\text{-}symb\ \varphi \implies \exists\ \psi.\ elim\text{-}imp\ \varphi\ \psi$
by ($case\text{-}tac\ \varphi$) ($force\ simp$: $elim\text{-}imp.simps$) $+$

then have $\bigwedge\ \varphi'.\ \varphi' \preceq \varphi \implies \neg no\text{-}imp\text{-}symb\ \varphi' \implies \exists\ \psi.\ elim\text{-}imp\ \varphi'\ \psi$ **by** $force$

ultimately show $?thesis$

using $no\text{-}test\text{-}symb\text{-}step\text{-}exists\ no\text{-}equiv\ test\text{-}symb\text{-}false\text{-}nullary$ **unfolding** $no\text{-}imp\text{-}def$ **by** $blast$

qed

lemma $no\text{-}imp\text{-}full\text{-}propo\text{-}rew\text{-}step\text{-}elim\text{-}imp$: $full\ (propo\text{-}rew\text{-}step\ elim\text{-}imp)\ \varphi\ \psi \implies no\text{-}imp\ \psi$
using $full\text{-}propo\text{-}rew\text{-}step\text{-}subformula\ no\text{-}no\text{-}imp\text{-}elim\text{-}imp\text{-}step\text{-}exists$ **by** $blast$

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*: *preserves-un-sat elimTB*

proof –

```
{
  fix  $\varphi \psi$ :: 'b propo
  have elimTB  $\varphi \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$  by (induction rule: elimTB.inducts) auto
}
then show ?thesis using preserves-un-sat-def by auto
qed
```

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

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

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

```
wf-conn  $c \ \psi s \implies$ 
  no-T-F-symb (conn  $c \ \psi s$ )  $\longleftrightarrow (c \neq CF \wedge c \neq CT \wedge (\forall \psi \in \text{set } \psi s. \psi \neq FF \wedge \psi \neq FT))$ 
unfolding no-T-F-symb.simps apply (cases c)
  using wf-conn-list(1) apply fastforce
  using wf-conn-list(2) apply fastforce
  using wf-conn-list(3) apply fastforce
  apply (metis (no-types, hide-lams) conn-inj connective.distinct(5,17))
  using conn-inj apply blast+
done
```

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

```
no-T-F-symb (FAnd  $\varphi \ \psi$ )  $\longleftrightarrow (\forall \chi \in \text{set } [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$ 
no-T-F-symb (FOr  $\varphi \ \psi$ )  $\longleftrightarrow (\forall \chi \in \text{set } [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$ 
no-T-F-symb (FEq  $\varphi \ \psi$ )  $\longleftrightarrow (\forall \chi \in \text{set } [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$ 
no-T-F-symb (FImp  $\varphi \ \psi$ )  $\longleftrightarrow (\forall \chi \in \text{set } [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$ 
apply (metis conn.simps(36) conn.simps(37) conn.simps(5) propo.distinct(19)
  wf-conn-helper-facts(5) wf-conn-no-T-F-symb-iff)
apply (metis conn.simps(36) conn.simps(37) conn.simps(6) propo.distinct(22)
  wf-conn-helper-facts(6) wf-conn-no-T-F-symb-iff)
using wf-conn-no-T-F-symb-iff apply fastforce
by (metis conn.simps(36) conn.simps(37) conn.simps(7) propo.distinct(23) wf-conn-helper-facts(7)
  wf-conn-no-T-F-symb-iff)
```

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

```

fixes  $c :: 'v$  connective
shows
   $\neg$ no-T-F-symb ( $FT :: 'v$  propo)
   $\neg$ no-T-F-symb ( $FF :: 'v$  propo)
  by (metis (no-types) conn.simps(1,2) wf-conn-no-T-F-symb-iff wf-conn-nullary)+

lemma no-T-F-symb-bool[simp]:
  fixes  $x :: 'v$ 
  shows no-T-F-symb ( $FVar\ x$ )
  using no-T-F-symb-comp wf-conn-nullary by (metis connective.distinct(3, 15) conn.simps(3)
    empty-iff list.set(1))

```

```

lemma no-T-F-symb-fnot-imp:
   $\neg$ no-T-F-symb ( $FNot\ \varphi$ )  $\implies \varphi = FT \vee \varphi = FF$ 
proof (rule ccontr)
  assume  $n: \neg$  no-T-F-symb ( $FNot\ \varphi$ )
  assume  $\neg (\varphi = FT \vee \varphi = FF)$ 
  then have  $\forall \varphi' \in \text{set } [\varphi]. \varphi' \neq FT \wedge \varphi' \neq FF$  by auto
  moreover have wf-conn  $CNot\ [\varphi]$  by simp
  ultimately have no-T-F-symb ( $FNot\ \varphi$ )
    using no-T-F-symb.intros by (metis conn.simps(4) connective.distinct(5,17))
  then show False using  $n$  by blast
qed

```

```

lemma no-T-F-symb-fnot[simp]:
  no-T-F-symb ( $FNot\ \varphi$ )  $\longleftrightarrow \neg (\varphi = FT \vee \varphi = FF)$ 
  using no-T-F-symb.simps no-T-F-symb-fnot-imp by (metis conn-inj-not(2) list.set-intros(1))

```

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  $\varphi \implies$  no-T-F-symb-except-toplevel  $\varphi$ 

```

```

lemma no-T-F-symb-except-toplevel-bool:
  fixes  $x :: 'v$ 
  shows no-T-F-symb-except-toplevel ( $FVar\ x$ )
  by simp

```

```

lemma no-T-F-symb-except-toplevel-not-decom:
   $\varphi \neq FT \implies \varphi \neq FF \implies$  no-T-F-symb-except-toplevel ( $FNot\ \varphi$ )
  by simp

```

```

lemma no-T-F-symb-except-toplevel-bin-decom:
  fixes  $\varphi\ \psi :: 'v$  propo
  assumes  $\varphi \neq FT$  and  $\varphi \neq FF$  and  $\psi \neq FT$  and  $\psi \neq FF$ 
  and  $c: c \in \text{binary-connectives}$ 
  shows no-T-F-symb-except-toplevel (conn  $c\ [\varphi, \psi]$ )
  by (metis (no-types, lifting) assms  $c$  conn.simps(4) list.discI noTrue-no-T-F-symb-except-toplevel
    wf-conn-no-T-F-symb-iff no-T-F-symb-fnot set.ConsD wf-conn-binary wf-conn-helper-facts(1)
    wf-conn-list-decomp(1,2))

```

```

lemma no-T-F-symb-except-toplevel-if-is-a-true-false:
  fixes  $l :: 'v$  propo list and  $c :: 'v$  connective

```

assumes *corr*: *wf-conn c l*
and *FT* ∈ *set l* ∨ *FF* ∈ *set l*
shows $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (conn } c \text{ l)}$
by (*metis* *assms empty-iff no-}T\text{-}F\text{-symb-except-toplevel.simps wf-conn-no-}T\text{-}F\text{-symb-iff set-empty wf-conn-list(1,2))*

lemma *no-}T\text{-}F\text{-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
 $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (FAnd } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (FOr } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (FImp } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (FEq } \varphi \psi)$
using *assms no-}T\text{-}F\text{-symb-except-toplevel-if-is-a-true-false unfolding binary-connectives-def*
by (*metis* (*no-types*) *conn.simps(5-8) insert-iff list.simps(14-15) wf-conn-helper-facts(5-8))*+)

lemma *no-}T\text{-}F\text{-symb-except-top-level-false-not[simp]*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes $\varphi = FT \vee \varphi = FF$
shows
 $\neg \text{no-}T\text{-}F\text{-symb-except-toplevel} \text{ (FNot } \varphi)$
by (*simp add: assms no-}T\text{-}F\text{-symb-except-toplevel.simps*)

This is the local extension of *no-}T\text{-}F\text{-symb-except-toplevel*.

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

lemma *no-}T\text{-}F\text{-except-top-level-false*:
fixes *l* :: '*v propo list* **and** *c* :: '*v connective*
assumes *wf-conn c l*
and *FT* ∈ *set l* ∨ *FF* ∈ *set l*
shows $\neg \text{no-}T\text{-}F\text{-except-top-level} \text{ (conn } c \text{ l)}$
by (*simp add: all-subformula-st-decomp assms no-}T\text{-}F\text{-except-top-level-def no-}T\text{-}F\text{-symb-except-toplevel-if-is-a-true-false*)

lemma *no-}T\text{-}F\text{-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
 $\neg \text{no-}T\text{-}F\text{-except-top-level} \text{ (FAnd } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-except-top-level} \text{ (FOr } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-except-top-level} \text{ (FEq } \varphi \psi)$
 $\neg \text{no-}T\text{-}F\text{-except-top-level} \text{ (FImp } \varphi \psi)$
by (*metis all-subformula-st-test-symb-true-phi assms no-}T\text{-}F\text{-except-top-level-def no-}T\text{-}F\text{-symb-except-top-level-false-example*)+

lemma *no-}T\text{-}F\text{-symb-except-toplevel-no-}T\text{-}F\text{-symb*:

no-T-F-symb-except-toplevel $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies \text{no-T-F-symb } \varphi$
by (*induct rule*: *no-T-F-symb-except-toplevel.induct*, *auto*)

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 $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies \text{no-T-F } \varphi$
unfolding *no-T-F-except-top-level-def* *no-T-F-def* **apply** (*induct* φ)
using *no-T-F-symb-fnot* **by** *fastforce*+

lemma *no-T-F-no-T-F-except-top-level*:
no-T-F $\varphi \implies \text{no-T-F-except-top-level } \varphi$
unfolding *no-T-F-except-top-level-def* *no-T-F-def*
unfolding *all-subformula-st-def* **by** *auto*

lemma *no-T-F-except-top-level-simp*[*simp*]: *no-T-F-except-top-level* FF *no-T-F-except-top-level* FT
unfolding *no-T-F-except-top-level-def* **by** *auto*

lemma *no-T-F-no-T-F-except-top-level'*[*simp*]:
no-T-F-except-top-level $\varphi \longleftrightarrow (\varphi = FF \vee \varphi = FT \vee \text{no-T-F } \varphi)$
using *no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb* *no-T-F-no-T-F-except-top-level*
by *auto*

lemma *no-T-F-bin-decomp*[*simp*]:
assumes *c*: $c \in \text{binary-connectives}$
shows *no-T-F* (*conn* *c* [φ , ψ]) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$

proof –

have *wf*: *wf-conn* *c* [φ , ψ] **using** *c* **by** *auto*
then have *no-T-F* (*conn* *c* [φ , ψ]) $\longleftrightarrow (\text{no-T-F-symb} (\text{conn } c [\varphi, \psi]) \wedge \text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
by (*simp add*: *all-subformula-st-decomp* *no-T-F-def*)
then show *no-T-F* (*conn* *c* [φ , ψ]) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
using *c* *wf* *all-subformula-st-decomp* *list.discI* *no-T-F-def* *no-T-F-symb-except-toplevel-bin-decom*
no-T-F-symb-except-toplevel-no-T-F-symb *no-T-F-symb-false*(1,2) *wf-conn-helper-facts*(2,3)
wf-conn-list(1,2) **by** *metis*

qed

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]) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
using *no-T-F-bin-decomp* *assms* **unfolding** *binary-connectives-def* **by** *blast*

lemma *no-T-F-comp-expanded-explicit*[*simp*]:
fixes $\varphi \psi :: 'v \text{ propo}$
shows
no-T-F (*FAnd* $\varphi \psi$) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
no-T-F (*FOr* $\varphi \psi$) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
no-T-F (*FEq* $\varphi \psi$) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
no-T-F (*FImp* $\varphi \psi$) $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$
using *assms* *conn.simps*(5–8) *no-T-F-bin-decomp-expanded* **by** (*metis* (*no-types*))+

lemma *no-T-F-comp-not*[*simp*]:
fixes $\varphi \psi :: 'v \text{ propo}$
shows *no-T-F* (*FNot* φ) $\longleftrightarrow \text{no-T-F } \varphi$
by (*metis* *all-subformula-st-decomp-explicit*(3) *all-subformula-st-test-symb-true-phi* *no-T-F-def*
no-T-F-symb-false(1,2) *no-T-F-symb-fnot-imp*)

lemma *no-T-F-decomp*:

```

fixes  $\varphi \ \psi :: 'v \text{ propo}$ 
assumes  $\varphi$ :  $\text{no-T-F } (F\text{And } \varphi \ \psi) \vee \text{no-T-F } (F\text{Or } \varphi \ \psi) \vee \text{no-T-F } (F\text{Eq } \varphi \ \psi) \vee \text{no-T-F } (F\text{Imp } \varphi \ \psi)$ 
shows  $\text{no-T-F } \psi$  and  $\text{no-T-F } \varphi$ 
using assms by auto

lemma no-T-F-decomp-not:
  fixes  $\varphi :: 'v \text{ propo}$ 
  assumes  $\varphi$ :  $\text{no-T-F } (F\text{Not } \varphi)$ 
  shows  $\text{no-T-F } \varphi$ 
  using assms by auto

lemma no-T-F-symb-except-toplevel-step-exists:
  fixes  $\varphi \ \psi :: 'v \text{ propo}$ 
  assumes  $\text{no-equiv } \varphi$  and  $\text{no-imp } \varphi$ 
  shows  $\psi \preceq \varphi \implies \neg \text{no-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTB } \psi \ \psi'$ 
proof (induct  $\psi$  rule: propo-induct-arity)
  case (nullary  $\varphi' \ x$ )
  then have False using no-T-F-symb-except-toplevel-true no-T-F-symb-except-toplevel-false by auto
  then show ?case by blast
next
  case (unary  $\psi$ )
  then have  $\psi = FF \vee \psi = FT$  using no-T-F-symb-except-toplevel-not-decom by blast
  then show ?case using ElimTB5 ElimTB6 by blast
next
  case (binary  $\varphi' \ \psi1 \ \psi2$ )
  note  $IH1 = \text{this}(1)$  and  $IH2 = \text{this}(2)$  and  $\varphi' = \text{this}(3)$  and  $F\varphi = \text{this}(4)$  and  $n = \text{this}(5)$ 
  {
    assume  $\varphi' = F\text{Imp } \psi1 \ \psi2 \vee \varphi' = F\text{Eq } \psi1 \ \psi2$ 
    then have False using  $n \ F\varphi$  subformula-all-subformula-st assms
      by (metis (no-types) no-equiv-eq(1) no-equiv-def no-imp-Imp(1) no-imp-def)
    then have ?case by blast
  }
  moreover {
    assume  $\varphi'$ :  $\varphi' = F\text{And } \psi1 \ \psi2 \vee \varphi' = F\text{Or } \psi1 \ \psi2$ 
    then have  $\psi1 = FT \vee \psi2 = FT \vee \psi1 = FF \vee \psi2 = FF$ 
      using no-T-F-symb-except-toplevel-bin-decom conn.simps(5,6)  $n$  unfolding binary-connectives-def
      by fastforce+
    then have ?case using elimTB.intros  $\varphi'$  by blast
  }
  ultimately show ?case using  $\varphi'$  by blast
qed

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:  $\text{no-equiv } \varphi$  and no-imp:  $\text{no-imp } \varphi$ 
  shows  $\exists \psi \ \psi'. \psi \preceq \varphi \wedge \text{elimTB } \psi \ \psi'$ 
proof –
  have test-symb-false-nullary:  $\forall x. \text{no-T-F-symb-except-toplevel } (FF:: 'v \text{ propo})$ 
     $\wedge \text{no-T-F-symb-except-toplevel } FT \wedge \text{no-T-F-symb-except-toplevel } (F\text{Var } (x:: 'v))$  by auto
  moreover {
    fix  $c:: 'v \text{ connective}$  and  $l:: 'v \text{ propo list}$  and  $\psi:: 'v \text{ propo}$ 
    have  $H$ :  $\text{elimTB } (\text{conn } c \ l) \ \psi \implies \neg \text{no-T-F-symb-except-toplevel } (\text{conn } c \ l)$ 
      by (cases conn  $c \ l$  rule: elimTB.cases, auto)
  }
  moreover {
    fix  $x:: 'v$ 

```

```

  have H': no-T-F-except-top-level FT no-T-F-except-top-level FF
    no-T-F-except-top-level (FVar x)
  by (auto simp: no-T-F-except-top-level-def test-symb-false-nullary)
}
moreover {
  fix  $\psi$ 
  have  $\psi \preceq \varphi \implies \neg \text{no-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTB } \psi \psi'$ 
    using no-T-F-symb-except-toplevel-step-exists no-equiv no-imp by auto
}
ultimately show ?thesis
  using no-test-symb-step-exists noTB unfolding no-T-F-except-top-level-def by blast
qed

```

lemma *elimTB-inv*:

```

  fixes  $\varphi \psi :: 'v \text{ propo}$ 
  assumes full (propo-rew-step elimTB)  $\varphi \psi$ 
  and no-equiv  $\varphi$  and no-imp  $\varphi$ 
  shows no-equiv  $\psi$  and no-imp  $\psi$ 
proof -
  {
    fix  $\varphi \psi :: 'v \text{ propo}$ 
    have H: elimTB  $\varphi \psi \implies \text{no-equiv } \varphi \implies \text{no-equiv } \psi$ 
      by (induct  $\varphi \psi$  rule: elimTB.induct, auto)
  }
  then show no-equiv  $\psi$ 
    using full-propo-rew-step-inv-stay-conn[of elimTB no-equiv-symb  $\varphi \psi$ ]
      no-equiv-symb-conn-characterization assms unfolding no-equiv-def by metis
next
  {
    fix  $\varphi \psi :: 'v \text{ propo}$ 
    have H: elimTB  $\varphi \psi \implies \text{no-imp } \varphi \implies \text{no-imp } \psi$ 
      by (induct  $\varphi \psi$  rule: elimTB.induct, auto)
  }
  then show no-imp  $\psi$ 
    using full-propo-rew-step-inv-stay-conn[of elimTB no-imp-symb  $\varphi \psi$ ] assms
      no-imp-symb-conn-characterization unfolding no-imp-def by metis
qed

```

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

```

  fixes  $\varphi \psi :: 'v \text{ propo}$ 
  assumes no-equiv  $\varphi$  and no-imp  $\varphi$  and full (propo-rew-step elimTB)  $\varphi \psi$ 
  shows no-T-F-except-top-level  $\psi$ 
  using full-propo-rew-step-subformula no-T-F-except-top-level-rew assms elimTB-inv by fastforce

```

0.3.4 PushNeg

Push the negation inside the formula, until the litteral.

inductive *pushNeg* **where**

```

PushNeg1[simp]: pushNeg (FNot (FAnd  $\varphi \psi$ )) (FOr (FNot  $\varphi$ ) (FNot  $\psi$ )) |
PushNeg2[simp]: pushNeg (FNot (FOr  $\varphi \psi$ )) (FAnd (FNot  $\varphi$ ) (FNot  $\psi$ )) |
PushNeg3[simp]: pushNeg (FNot (FNot  $\varphi$ ))  $\varphi$ 

```

lemma *pushNeg-transformation-consistent*:

```

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

```

$A \models \text{FNot } (\text{FOr } \varphi \ \psi) \iff A \models (\text{FAnd } (\text{FNot } \varphi) \ (\text{FNot } \psi))$
 $A \models \text{FNot } (\text{FNot } \varphi) \iff A \models \varphi$
by *auto*

lemma *pushNeg-explicit*: $\text{pushNeg } \varphi \ \psi \implies \forall A. A \models \varphi \iff A \models \psi$
by (*induct* $\varphi \ \psi$ *rule*: *pushNeg.induct*, *auto*)

lemma *pushNeg-consistent*: *preserves-un-sat pushNeg*
unfolding *preserves-un-sat-def* **by** (*simp add*: *pushNeg-explicit*)

lemma *pushNeg-lifted-consistant*:
preserves-un-sat (full (propo-rew-step pushNeg))
by (*simp add*: *pushNeg-consistent*)

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

lemma *simple-decomp*:
simple $\varphi \iff (\varphi = \text{FT} \vee \varphi = \text{FF} \vee (\exists x. \varphi = \text{FVar } x))$
by (*cases* φ) *auto*

lemma *subformula-conn-decomp-simple*:
fixes $\varphi \ \psi :: 'v \text{ propo}$
assumes *s*: *simple* ψ
shows $\varphi \preceq \text{FNot } \psi \iff (\varphi = \text{FNot } \psi \vee \varphi = \psi)$
proof –
have $\varphi \preceq \text{conn } \text{CNot } [\psi] \iff (\varphi = \text{conn } \text{CNot } [\psi] \vee (\exists \psi \in \text{set } [\psi]. \varphi \preceq \psi))$
using *subformula-conn-decomp wf-conn-helper-facts(1)* **by** *metis*
then show $\varphi \preceq \text{FNot } \psi \iff (\varphi = \text{FNot } \psi \vee \varphi = \psi)$ **using** *s* **by** (*auto simp*: *simple-decomp*)
qed

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

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]*:
 $\neg \text{simple-not } (\text{FNot } (\text{FAnd } \varphi \ \psi))$
 $\neg \text{simple-not } (\text{FNot } (\text{FOr } \varphi \ \psi))$

by auto

lemma *simple-not-step-exists*:

fixes $\varphi \ \psi :: 'v \text{ propo}$
 assumes *no-equiv* φ **and** *no-imp* φ
 shows $\psi \preceq \varphi \implies \neg \text{simple-not-symb } \psi \implies \exists \psi'. \text{pushNeg } \psi \ \psi'$
 apply (induct ψ , auto)
 apply (rename-tac ψ , case-tac ψ , auto intro: *pushNeg.intros*)
 by (metis *assms*(1,2) *no-imp-Imp*(1) *no-equiv-eq*(1) *no-imp-def* *no-equiv-def*
subformula-in-subformula-not *subformula-all-subformula-st*)+

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 –

have $\forall x. \text{simple-not-symb } (FF :: 'v \text{ propo}) \wedge \text{simple-not-symb } FT \wedge \text{simple-not-symb } (FVar \ (x :: 'v))$
 by auto
 moreover {
 fix $c :: 'v \text{ connective}$ **and** $l :: 'v \text{ propo list}$ **and** $\psi :: 'v \text{ propo}$
 have $H: \text{pushNeg } (\text{conn } c \ l) \ \psi \implies \neg \text{simple-not-symb } (\text{conn } c \ l)$
 by (cases *conn* $c \ l$ rule: *pushNeg.cases*) auto
 }
 moreover {
 fix $x :: 'v$
 have $H': \text{simple-not } FT \ \text{simple-not } FF \ \text{simple-not } (FVar \ x)$
 by *simp-all*
 }
 moreover {
 fix $\psi :: 'v \text{ propo}$
 have $\psi \preceq \varphi \implies \neg \text{simple-not-symb } \psi \implies \exists \psi'. \text{pushNeg } \psi \ \psi'$
 using *simple-not-step-exists* *no-equiv* *no-imp* **by** *blast*
 }
 ultimately show *?thesis* **using** *no-test-symb-step-exists* *noTB* **unfolding** *simple-not-def* **by** *blast*
qed

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

no-T-F-except-top-level (*FNot* (*FAnd* $\varphi \ \psi$)) \implies *no-T-F-except-top-level* (*FOr* (*FNot* φ) (*FNot* ψ))
using *no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb* *no-T-F-comp-not* *no-T-F-decomp*(1)
no-T-F-decomp(2) *no-T-F-no-T-F-except-top-level* **by** (metis *no-T-F-comp-expanded-explicit*(2)
propo.distinct(5,17))

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

no-T-F-except-top-level (*FNot* (*FOr* $\varphi \ \psi$)) \implies *no-T-F-except-top-level* (*FAnd* (*FNot* φ) (*FNot* ψ))
by *auto*

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'))
by *auto*

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

propo-rew-step *pushNeg* $\varphi \ \psi \implies$ *no-T-F-except-top-level* $\varphi \implies$ *no-T-F-symb* $\varphi \implies$ *no-T-F-symb* ψ
apply (induct rule: *propo-rew-step.induct*)
apply (cases rule: *pushNeg.cases*)


```

apply simp-all
apply (metis no-T-F-symb-pushNeg(1))
apply (metis no-T-F-symb-pushNeg(2))
apply (simp, metis all-subformula-st-test-symb-true-phi no-T-F-def)
proof –
  fix  $\varphi \varphi'$ : 'a propo and  $c$ :: 'a connective and  $\xi \xi'$ :: 'a propo list
  assume rel: propo-rew-step pushNeg  $\varphi \varphi'$ 
  and IH: no-T-F  $\varphi \implies$  no-T-F-symb  $\varphi \implies$  no-T-F-symb  $\varphi'$ 
  and wf: wf-conn  $c (\xi @ \varphi \# \xi')$ 
  and  $n$ : conn  $c (\xi @ \varphi \# \xi') = FF \vee$  conn  $c (\xi @ \varphi \# \xi') = FT \vee$  no-T-F (conn  $c (\xi @ \varphi \# \xi')$ )
  and  $x$ :  $c \neq CF \wedge c \neq CT \wedge \varphi \neq FF \wedge \varphi \neq FT \wedge (\forall \psi \in \text{set } \xi \cup \text{set } \xi'. \psi \neq FF \wedge \psi \neq FT)$ 
  then have  $c \neq CF \wedge c \neq CT \wedge$  wf-conn  $c (\xi @ \varphi' \# \xi')$ 
    using wf-conn-no-arity-change-helper wf-conn-no-arity-change by metis
  moreover have  $n'$ : no-T-F (conn  $c (\xi @ \varphi \# \xi')$ ) using  $n$  by (simp add: wf wf-conn-list(1,2))
  moreover
  {
    have no-T-F  $\varphi$ 
      by (metis Un-iff all-subformula-st-decomp list.set-intros(1)  $n'$  wf no-T-F-def set-append)
    moreover then have no-T-F-symb  $\varphi$ 
      by (simp add: all-subformula-st-test-symb-true-phi no-T-F-def)
    ultimately have  $\varphi' \neq FF \wedge \varphi' \neq FT$ 
      using IH no-T-F-symb-false(1) no-T-F-symb-false(2) by blast
    then have  $\forall \psi \in \text{set } (\xi @ \varphi' \# \xi'). \psi \neq FF \wedge \psi \neq FT$  using  $x$  by auto
  }
  ultimately show no-T-F-symb (conn  $c (\xi @ \varphi' \# \xi')$ ) by (simp add:  $x$ )
qed

```

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

propo-rew-step pushNeg $\varphi \psi \implies$ *no-T-F* $\varphi \implies$ *no-T-F* ψ

proof (*induct rule*: *propo-rew-step.induct*)

case *global-rel*

then show ?*case*

by (*metis (no-types, lifting) no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb*
no-T-F-def no-T-F-except-top-level-pushNeg1 no-T-F-except-top-level-pushNeg2
no-T-F-no-T-F-except-top-level all-subformula-st-decomp-explicit(3) pushNeg.simps
simple.simps(1,2,5,6))

next

case (*propo-rew-one-step-lift* $\varphi \varphi' c \xi \xi'$)

note *rel* = *this(1)* **and** *IH* = *this(2)* **and** *wf* = *this(3)* **and** *no-T-F* = *this(4)*

moreover have *wf'*: *wf-conn* $c (\xi @ \varphi' \# \xi')$

using *wf-conn-no-arity-change wf-conn-no-arity-change-helper wf* **by** *metis*

ultimately show *no-T-F* (*conn* $c (\xi @ \varphi' \# \xi')$)

using *all-subformula-st-test-symb-true-phi*

by (*fastforce simp*: *no-T-F-def all-subformula-st-decomp wf wf'*)

qed

lemma *pushNeg-inv*:

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

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

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 –

{

fix $\varphi \psi$:: 'v *propo*

assume *rel*: *propo-rew-step pushNeg* $\varphi \psi$

```

and no: no-T-F-except-top-level  $\varphi$ 
then have no-T-F-except-top-level  $\psi$ 
proof -
{
  assume  $\varphi = FT \vee \varphi = FF$ 
  from rel this have False
  apply (induct rule: propo-rew-step.induct)
  using pushNeg.cases apply blast
  using wf-conn-list(1) wf-conn-list(2) by auto
  then have no-T-F-except-top-level  $\psi$  by blast
}
moreover {
  assume  $\varphi \neq FT \wedge \varphi \neq FF$ 
  then have no-T-F  $\varphi$ 
  by (metis no no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb)
  then have no-T-F  $\psi$ 
  using propo-rew-step-pushNeg-no-T-F rel by auto
  then have no-T-F-except-top-level  $\psi$  by (simp add: no-T-F-no-T-F-except-top-level)
}
ultimately show no-T-F-except-top-level  $\psi$  by metis
qed
}
moreover {
  fix c :: 'v connective and  $\xi \xi' :: 'v \text{ propo list}$  and  $\zeta \zeta' :: 'v \text{ propo}$ 
  assume rel: propo-rew-step pushNeg  $\zeta \zeta'$ 
  and incl:  $\zeta \preceq \varphi$ 
  and corr: wf-conn c ( $\xi @ \zeta \# \xi'$ )
  and no-T-F: no-T-F-symb-except-toplevel (conn c ( $\xi @ \zeta \# \xi'$ ))
  and n: no-T-F-symb-except-toplevel  $\zeta'$ 
  have no-T-F-symb-except-toplevel (conn c ( $\xi @ \zeta' \# \xi'$ ))
  proof
    have p: no-T-F-symb (conn c ( $\xi @ \zeta \# \xi'$ ))
    using corr wf-conn-list(1) wf-conn-list(2) no-T-F-symb-except-toplevel-no-T-F-symb no-T-F
    by blast
    have l:  $\forall \varphi \in \text{set } (\xi @ \zeta \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$ 
    using corr wf-conn-no-T-F-symb-iff p by blast
    from rel incl have  $\zeta' \neq FT \wedge \zeta' \neq FF$ 
    apply (induction  $\zeta \zeta'$  rule: propo-rew-step.induct)
    apply (cases rule: pushNeg.cases, auto)
    by (metis assms(4) no-T-F-symb-except-top-level-false-not no-T-F-except-top-level-def
        all-subformula-st-test-symb-true-phi subformula-in-subformula-not
        subformula-all-subformula-st append-is-Nil-conv list.distinct(1)
        wf-conn-no-arity-change-helper wf-conn-list(1,2) wf-conn-no-arity-change)+
    then have  $\forall \varphi \in \text{set } (\xi @ \zeta' \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$  using l by auto
    moreover have  $c \neq CT \wedge c \neq CF$  using corr by auto
    ultimately show no-T-F-symb (conn c ( $\xi @ \zeta' \# \xi'$ ))
    by (metis corr no-T-F-symb-comp wf-conn-no-arity-change wf-conn-no-arity-change-helper)
  qed
}
ultimately show no-T-F-except-top-level  $\psi$ 
using full-propo-rew-step-inv-stay-with-inc[of pushNeg no-T-F-symb-except-toplevel  $\varphi$ ] assms
subformula-refl unfolding no-T-F-except-top-level-def full-unfold by metis
next
{
  fix  $\varphi \psi :: 'v \text{ propo}$ 
  have H: pushNeg  $\varphi \psi \implies \text{no-equiv } \varphi \implies \text{no-equiv } \psi$ 

```

```

    by (induct  $\varphi \psi$  rule: pushNeg.induct, auto)
  }
then show no-equiv  $\psi$ 
  using full-propo-rew-step-inv-stay-conn[of pushNeg no-equiv-symb  $\varphi \psi$ ]
  no-equiv-symb-conn-characterization assms unfolding no-equiv-def full-unfold by metis
next
{
  fix  $\varphi \psi :: 'v$  propo
  have  $H: \text{pushNeg } \varphi \psi \implies \text{no-imp } \varphi \implies \text{no-imp } \psi$ 
    by (induct  $\varphi \psi$  rule: pushNeg.induct, auto)
}
then show no-imp  $\psi$ 
  using full-propo-rew-step-inv-stay-conn[of pushNeg no-imp-symb  $\varphi \psi$ ] assms
  no-imp-symb-conn-characterization unfolding no-imp-def full-unfold by metis
qed

```

lemma *pushNeg-full-propo-rew-step*:
fixes $\varphi \psi :: 'v$ propo
assumes
 no-equiv φ **and**
 no-imp φ **and**
 full (propo-rew-step pushNeg) $\varphi \psi$ **and**
 no-T-F-except-top-level φ
shows simple-not ψ
using assms full-propo-rew-step-subformula pushNeg-inv(1,2) simple-not-rew **by** blast

0.3.5 Push inside

inductive *push-conn-inside* :: $'v$ connective $\Rightarrow 'v$ connective $\Rightarrow 'v$ propo $\Rightarrow 'v$ propo $\Rightarrow \text{bool}$
for $c c' :: 'v$ connective **where**
push-conn-inside-l[simp]: $c = CAnd \vee c = COr \implies c' = CAnd \vee c' = COr$
 $\implies \text{push-conn-inside } c c' (\text{conn } c [\text{conn } c' [\varphi 1, \varphi 2], \psi])$
 $(\text{conn } c' [\text{conn } c [\varphi 1, \psi], \text{conn } c [\varphi 2, \psi]]) \mid$
push-conn-inside-r[simp]: $c = CAnd \vee c = COr \implies c' = CAnd \vee c' = COr$
 $\implies \text{push-conn-inside } c c' (\text{conn } c [\psi, \text{conn } c' [\varphi 1, \varphi 2]])$
 $(\text{conn } c' [\text{conn } c [\psi, \varphi 1], \text{conn } c [\psi, \varphi 2]])$

lemma *push-conn-inside-explicit*: $\text{push-conn-inside } c c' \varphi \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$
by (induct $\varphi \psi$ rule: push-conn-inside.induct, auto)

lemma *push-conn-inside-consistent*: preserves-un-sat (push-conn-inside $c c'$)
unfolding preserves-un-sat-def **by** (simp add: push-conn-inside-explicit)

lemma *propo-rew-step-push-conn-inside[simp]*:
 $\neg \text{propo-rew-step } (\text{push-conn-inside } c c') FT \psi \neg \text{propo-rew-step } (\text{push-conn-inside } c c') FF \psi$
proof –
{
 {
 fix $\varphi \psi$
 have $\text{push-conn-inside } c c' \varphi \psi \implies \varphi = FT \vee \varphi = FF \implies \text{False}$
 by (induct rule: push-conn-inside.induct, auto)
 } **note** $H = \text{this}$
 fix φ
 have $\text{propo-rew-step } (\text{push-conn-inside } c c') \varphi \psi \implies \varphi = FT \vee \varphi = FF \implies \text{False}$

```

apply (induct rule: propo-rew-step.induct, auto simp: wf-conn-list(1) wf-conn-list(2))
using  $H$  by blast+
}
then show
   $\neg$ propo-rew-step (push-conn-inside  $c$   $c'$ )  $FT$   $\psi$ 
   $\neg$ propo-rew-step (push-conn-inside  $c$   $c'$ )  $FF$   $\psi$  by blast+
qed

```

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

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

lemma *c-in-c'-symb-simp*:
 $not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ \xi \Longrightarrow \xi = FF \vee \xi = FT \vee \xi = FVar\ x \vee \xi = FNot\ FF \vee \xi = FNot\ FT$
 $\vee \xi = FNot\ (FVar\ x) \Longrightarrow False$
apply (induct rule: *not-c-in-c'-symb*.induct, auto simp: wf-conn.simps wf-conn-list(1-3))
using conn-inj-not(2) wf-conn-binary **unfolding** binary-connectives-def **by** fastforce+

lemma *c-in-c'-symb-simp'*[simp]:
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ FF$
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ FT$
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (FVar\ x)$
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (FNot\ FF)$
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (FNot\ FT)$
 $\neg not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (FNot\ (FVar\ x))$
using *c-in-c'-symb-simp* **by** metis+

definition *c-in-c'-only* **where**
 $c\text{-}in\text{-}c'\text{-}only\ c\ c' \equiv all\text{-}subformula\text{-}st\ (c\text{-}in\text{-}c'\text{-}symb\ c\ c')$

lemma *c-in-c'-only-simp*[simp]:
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ FF$
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ FT$
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ (FVar\ x)$
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ (FNot\ FF)$
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ (FNot\ FT)$
 $c\text{-}in\text{-}c'\text{-}only\ c\ c'\ (FNot\ (FVar\ x))$
unfolding *c-in-c'-only-def* **by** auto

lemma *not-c-in-c'-symb-commute*:
 $not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ \xi \Longrightarrow wf\text{-}conn\ c\ [\varphi, \psi] \Longrightarrow \xi = conn\ c\ [\varphi, \psi]$
 $\Longrightarrow not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (conn\ c\ [\psi, \varphi])$
proof (induct rule: *not-c-in-c'-symb*.induct)
case (*not-c-in-c'-symb-r* $\varphi'\ \varphi''\ \psi'$) **note** $H = this$
then have $\psi: \psi = conn\ c'\ [\varphi'', \psi']$ **using** conn-inj **by** auto
have $wf\text{-}conn\ c\ [conn\ c'\ [\varphi'', \psi'], \varphi]$
using $H(1)$ wf-conn-no-arity-change length-Cons **by** metis
then show $not\text{-}c\text{-}in\text{-}c'\text{-}symb\ c\ c'\ (conn\ c\ [\psi, \varphi])$
unfolding ψ **using** *not-c-in-c'-symb.intros*(1) H **by** auto

next

case (not-c-in-c'-symb-l φ' φ'' ψ') **note** $H = \text{this}$
 then have $\varphi = \text{conn } c' [\varphi', \varphi'']$ **using** *conn-inj* **by** *auto*
 moreover have $\text{wf-conn } c [\psi', \text{conn } c' [\varphi', \varphi'']]$
 using $H(1)$ *wf-conn-no-arity-change length-Cons* **by** *metis*
 ultimately show *not-c-in-c'-symb* $c c' (\text{conn } c [\psi, \varphi])$
 using *not-c-in-c'-symb.intros(2)* *conn-inj not-c-in-c'-symb-l.hyps*
not-c-in-c'-symb-l.prem(1,2) **by** *blast*

qed

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

$\text{wf-conn } c [\varphi, \psi] \implies \text{c-in-c'-symb } c c' (\text{conn } c [\varphi, \psi]) \longleftrightarrow \text{c-in-c'-symb } c c' (\text{conn } c [\psi, \varphi])$
using *not-c-in-c'-symb-commute wf-conn-no-arity-change* **by** (*metis length-Cons*)

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

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

proof –

have $?A \longleftrightarrow (\text{c-in-c'-symb } c c' (\text{conn } c [\varphi, \psi])$
 $\wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st } (\text{c-in-c'-symb } c c') \xi))$
using *all-subformula-st-decomp wf unfolding c-in-c'-only-def* **by** *fastforce*
 also have $\dots \longleftrightarrow (\text{c-in-c'-symb } c c' (\text{conn } c [\psi, \varphi])$
 $\wedge (\forall \xi \in \text{set } [\psi, \varphi]. \text{all-subformula-st } (\text{c-in-c'-symb } c c') \xi))$
using *not-c-in-c'-symb-commute' wf* **by** *auto*

also

have $\text{wf-conn } c [\psi, \varphi]$ **using** *wf-conn-no-arity-change wf* **by** (*metis length-Cons*)
 then have $(\text{c-in-c'-symb } c c' (\text{conn } c [\psi, \varphi])$
 $\wedge (\forall \xi \in \text{set } [\psi, \varphi]. \text{all-subformula-st } (\text{c-in-c'-symb } c c') \xi))$
 $\longleftrightarrow ?B$

using *all-subformula-st-decomp unfolding c-in-c'-only-def* **by** *fastforce*

finally show *?thesis* .

qed

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

fixes $\varphi 1 \ \varphi 2 \ \psi :: 'v \text{ propo}$ **and** $x :: 'v$
shows
 $\text{c-in-c'-symb } c c' FT$
 $\text{c-in-c'-symb } c c' FF$
 $\text{c-in-c'-symb } c c' (FVar \ x)$
 $\text{wf-conn } c [\text{conn } c' [\varphi 1, \varphi 2], \psi] \implies \text{wf-conn } c' [\varphi 1, \varphi 2]$
 $\implies \neg \text{c-in-c'-only } c c' (\text{conn } c [\text{conn } c' [\varphi 1, \varphi 2], \psi])$
apply (*simp-all add: c-in-c'-only-def*)
using *all-subformula-st-test-symb-true-phi not-c-in-c'-symb-l* **by** *blast*

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

fixes $c \ c' :: 'v \text{ connective}$ **and** $\psi :: 'v \text{ propo}$
shows $\text{c-in-c'-symb } c c' (FNot \ \psi)$

proof –

{
fix $\xi :: 'v \text{ propo}$
 have $\text{not-c-in-c'-symb } c c' (FNot \ \psi) \implies \text{False}$
apply (*induct FNot ψ rule: not-c-in-c'-symb.induct*)
using *conn-inj-not(2)* **by** *blast+*
 }

then show *?thesis* **by** *auto*

qed

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

apply (*induct* ψ *rule*: *propo-induct-arity*)

apply *auto*[2]

proof –

fix $\psi1 \ \psi2 \ \varphi':: 'v \text{ propo}$

assume $IH\psi1: \psi1 \preceq \varphi \implies \neg c\text{-in-}c'\text{-symb } c \ c' \ \psi1 \implies Ex \ (\text{push-conn-inside } c \ c' \ \psi1)$

and $IH\psi2: \psi2 \preceq \varphi \implies \neg c\text{-in-}c'\text{-symb } c \ c' \ \psi2 \implies Ex \ (\text{push-conn-inside } c \ c' \ \psi2)$

and $\varphi': \varphi' = FAnd \ \psi1 \ \psi2 \vee \varphi' = FOr \ \psi1 \ \psi2 \vee \varphi' = FImp \ \psi1 \ \psi2 \vee \varphi' = FEq \ \psi1 \ \psi2$

and $in\varphi: \varphi' \preceq \varphi$ **and** $n0: \neg c\text{-in-}c'\text{-symb } c \ c' \ \varphi'$

then have $n: \text{not-}c\text{-in-}c'\text{-symb } c \ c' \ \varphi'$ **by** *auto*

{

assume $\varphi': \varphi' = \text{conn } c \ [\psi1, \psi2]$

obtain $a \ b$ **where** $\psi1 = \text{conn } c' \ [a, b] \vee \psi2 = \text{conn } c' \ [a, b]$

using $n \ \varphi'$ **apply** (*induct* *rule*: *not-c-in-c'-symb.induct*)

using c **by** *force+*

then have $Ex \ (\text{push-conn-inside } c \ c' \ \varphi')$

unfolding φ' **apply** *auto*

using *push-conn-inside.intros*(1) $c \ c'$ **apply** *blast*

using *push-conn-inside.intros*(2) $c \ c'$ **by** *blast*

}

moreover {

assume $\varphi': \varphi' \neq \text{conn } c \ [\psi1, \psi2]$

have $\forall \varphi \ c \ ca. \exists \varphi1 \ \psi1 \ \psi2 \ \psi1' \ \psi2' \ \varphi2'. \text{conn } (c::'v \text{ connective}) \ [\varphi1, \text{conn } ca \ [\psi1, \psi2]] = \varphi$
 $\vee \text{conn } c \ [\text{conn } ca \ [\psi1', \psi2'], \varphi2'] = \varphi \vee c\text{-in-}c'\text{-symb } c \ ca \ \varphi$

by (*metis not-c-in-c'-symb.cases*)

then have $Ex \ (\text{push-conn-inside } c \ c' \ \varphi')$

by (*metis (no-types) c \ c' n push-conn-inside-l push-conn-inside-r*)

}

ultimately show $Ex \ (\text{push-conn-inside } c \ c' \ \varphi')$ **by** *blast*

qed

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

proof –

have *test-symb-false-nullary*:

$\forall x. c\text{-in-}c'\text{-symb } c \ c' \ (FF:: 'v \text{ propo}) \wedge c\text{-in-}c'\text{-symb } c \ c' \ FT$

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

by *auto*

moreover {

fix $x :: 'v$

have $H': c\text{-in-}c'\text{-symb } c \ c' \ FT \ c\text{-in-}c'\text{-symb } c \ c' \ FF \ c\text{-in-}c'\text{-symb } c \ c' \ (FVar \ x)$

by *simp+*

}

moreover {

fix $\psi :: 'v \text{ propo}$

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

by (*auto simp: asms*(2) $c' \ c\text{-in-}c'\text{-symb-step-exists}$)

}

```

ultimately show ?thesis using noTB no-test-symb-step-exists[of c-in-c'-symb c c']
  unfolding c-in-c'-only-def by metis
qed

lemma push-conn-insidec-in-c'-symb-no-T-F:
  fixes  $\varphi \ \psi :: 'v \text{ propo}$ 
  shows propo-rew-step (push-conn-inside c c')  $\varphi \ \psi \implies \text{no-T-F } \varphi \implies \text{no-T-F } \psi$ 
proof (induct rule: propo-rew-step.induct)
  case (global-rel  $\varphi \ \psi$ )
  then show no-T-F  $\psi$ 
    by (cases rule: push-conn-inside.cases, auto)
next
  case (propo-rew-one-step-lift  $\varphi \ \varphi' \ c \ \xi \ \xi'$ )
  note rel = this(1) and IH = this(2) and wf = this(3) and no-T-F = this(4)
  have no-T-F  $\varphi$ 
    using wf no-T-F no-T-F-def subformula-into-subformula subformula-all-subformula-st
    subformula-refl by (metis (no-types) in-set-conv-decomp)
  then have  $\varphi'$ : no-T-F  $\varphi'$  using IH by blast

  have  $\forall \zeta \in \text{set } (\xi @ \varphi \# \xi'). \text{no-T-F } \zeta$  by (metis wf no-T-F no-T-F-def all-subformula-st-decomp)
  then have n:  $\forall \zeta \in \text{set } (\xi @ \varphi' \# \xi'). \text{no-T-F } \zeta$  using  $\varphi'$  by auto
  then have n':  $\forall \zeta \in \text{set } (\xi @ \varphi' \# \xi'). \zeta \neq FF \wedge \zeta \neq FT$ 
    using  $\varphi'$  by (metis no-T-F-symb-false(1) no-T-F-symb-false(2) no-T-F-def
    all-subformula-st-test-symb-true-phi)

  have wf': wf-conn c ( $\xi @ \varphi' \# \xi'$ )
    using wf wf-conn-no-arity-change by (metis wf-conn-no-arity-change-helper)
  {
    fix x :: 'v
    assume c = CT  $\vee$  c = CF  $\vee$  c = CVar x
    then have False using wf by auto
    then have no-T-F (conn c ( $\xi @ \varphi' \# \xi'$ )) by blast
  }
  moreover {
    assume c: c = CNot
    then have  $\xi = [] \ \xi' = []$  using wf by auto
    then have no-T-F (conn c ( $\xi @ \varphi' \# \xi'$ ))
      using c by (metis  $\varphi'$  conn.simps(4) no-T-F-symb-false(1,2) no-T-F-symb-fnot no-T-F-def
      all-subformula-st-decomp-explicit(3) all-subformula-st-test-symb-true-phi self-append-conv2)
  }
  moreover {
    assume c: c  $\in$  binary-connectives
    then have no-T-F-symb (conn c ( $\xi @ \varphi' \# \xi'$ )) using wf' n' no-T-F-symb.simps by fastforce
    then have no-T-F (conn c ( $\xi @ \varphi' \# \xi'$ ))
      by (metis all-subformula-st-decomp-imp wf' n no-T-F-def)
  }
  ultimately show no-T-F (conn c ( $\xi @ \varphi' \# \xi'$ )) using connective-cases-arity by auto
qed

lemma simple-propo-rew-step-push-conn-inside-inv:
  propo-rew-step (push-conn-inside c c')  $\varphi \ \psi \implies \text{simple } \varphi \implies \text{simple } \psi$ 
  apply (induct rule: propo-rew-step.induct)
  apply (rename-tac  $\varphi$ , case-tac  $\varphi$ , auto simp: push-conn-inside.simps)[]
  by (metis append-is-Nil-conv list.distinct(1) simple.elims(2) wf-conn-list(1-3))

```

lemma *simple-propo-rew-step-inv-push-conn-inside-simple-not*:
fixes $c\ c' :: 'v\ \text{connective}$ **and** $\varphi\ \psi :: 'v\ \text{propo}$
shows $\text{propo-rew-step}\ (\text{push-conn-inside}\ c\ c')\ \varphi\ \psi \implies \text{simple-not}\ \varphi \implies \text{simple-not}\ \psi$
proof (*induct rule: propo-rew-step.induct*)
case (*global-rel* $\varphi\ \psi$)
then show ?case **by** (*cases* φ , *auto simp: push-conn-inside.simps*)
next
case (*propo-rew-one-step-lift* $\varphi\ \varphi'\ ca\ \xi\ \xi'$) **note** $\text{rew} = \text{this}(1)$ **and** $\text{IH} = \text{this}(2)$ **and** $\text{wf} = \text{this}(3)$
and $\text{simple} = \text{this}(4)$
show ?case
proof (*cases ca rule: connective-cases-arity*)
case *nullary*
then show ?thesis **using** *propo-rew-one-step-lift* **by** *auto*
next
case *binary* **note** $ca = \text{this}$
obtain $a\ b$ **where** $ab: \xi @ \varphi' \# \xi' = [a, b]$
using $\text{wf}\ ca\ \text{list-length2-decomp}\ \text{wf-conn-bin-list-length}$
by (*metis (no-types) wf-conn-no-arity-change-helper*)
have $\forall \zeta \in \text{set}\ (\xi @ \varphi \# \xi').\ \text{simple-not}\ \zeta$
by (*metis wf all-subformula-st-decomp simple simple-not-def*)
then have $\forall \zeta \in \text{set}\ (\xi @ \varphi' \# \xi').\ \text{simple-not}\ \zeta$ **using** IH **by** *simp*
moreover have $\text{simple-not-symb}\ (\text{conn}\ ca\ (\xi @ \varphi' \# \xi'))$ **using** ca
by (*metis ab conn.simps(5-8) helper-fact simple-not-symb.simps(5) simple-not-symb.simps(6)*
simple-not-symb.simps(7) simple-not-symb.simps(8))
ultimately show ?thesis
by (*simp add: ab all-subformula-st-decomp ca*)
next
case *unary*
then show ?thesis
using $\text{rew}\ \text{simple-propo-rew-step-push-conn-inside-inv}[OF\ \text{rew}]\ \text{IH}\ \text{local.wf}\ \text{simple}$ **by** *auto*
qed
qed

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**
 $\text{wf-conn}\ c\ (\xi @ \varphi \# \xi')$ **and**
 $\text{simple-not-symb}\ (\text{conn}\ c\ (\xi @ \varphi \# \xi'))$ **and**
 $\text{simple-not-symb}\ \varphi'$
shows $\text{simple-not-symb}\ (\text{conn}\ c\ (\xi @ \varphi' \# \xi'))$
using *assms*
proof (*induction rule: propo-rew-step.induct*)
print-cases
case (*global-rel*)
then show ?case
by (*metis conn.simps(12,17) list.discI push-conn-inside.cases simple-not-symb.elims(3)*
wf-conn-helper-facts(5) wf-conn-list(2) wf-conn-list(8) wf-conn-no-arity-change
wf-conn-no-arity-change-helper)
next
case (*propo-rew-one-step-lift* $\varphi\ \varphi'\ c'\ \chi s\ \chi s'$) **note** $\text{tel} = \text{this}(1)$ **and** $\text{wf} = \text{this}(2)$ **and**
 $\text{IH} = \text{this}(3)$ **and** $\text{wf}' = \text{this}(4)$ **and** $\text{simple}' = \text{this}(5)$ **and** $\text{simple} = \text{this}(6)$
then show ?case
proof (*cases c' rule: connective-cases-arity*)
case *nullary*


```

    then show ?thesis using wf simple simple' by auto
next
case binary note c = this(1)
have corr': wf-conn c (ξ @ conn c' (χs @ φ' # χs') # ξ')
  using wf wf-conn-no-arity-change
  by (metis wf' wf-conn-no-arity-change-helper)
then show ?thesis
  using c propo-rew-one-step-lift wf
  by (metis conn.simps(17) connective.distinct(37) propo-rew-step-subformula-imp
      push-conn-inside.cases simple-not-symb.elims(3) wf-conn.simps wf-conn-list(2,8))
next
case unary
then have empty: χs = [] χs' = [] using wf by auto
then show ?thesis using simple unary simple' wf wf'
  by (metis connective.distinct(37) connective.distinct(39) propo-rew-step-subformula-imp
      push-conn-inside.cases simple-not-symb.elims(3) tel wf-conn-list(8)
      wf-conn-no-arity-change wf-conn-no-arity-change-helper)
qed
qed

```

lemma *push-conn-inside-not-true-false:*
push-conn-inside c c' φ ψ ⇒ ψ ≠ FT ∧ ψ ≠ FF
by (induct rule: *push-conn-inside.induct*, auto)

lemma *push-conn-inside-inv:*

```

fixes φ ψ :: '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 -
{
  {
    fix φ ψ :: 'v propo
    have H: push-conn-inside c c' φ ψ ⇒ all-subformula-st simple-not-symb φ
      ⇒ all-subformula-st simple-not-symb ψ
      by (induct φ ψ rule: push-conn-inside.induct, auto)
  } note H = this
}

```

```

fix φ ψ :: 'v propo
have H: propo-rew-step (push-conn-inside c c') φ ψ ⇒ all-subformula-st simple-not-symb φ
  ⇒ all-subformula-st simple-not-symb ψ
  apply (induct φ ψ rule: propo-rew-step.induct)
  using H apply simp
proof (rename-tac φ φ' ca ψs ψs', case-tac ca rule: connective-cases-arity)
  fix φ φ' :: 'v propo and c:: 'v connective and ξ ξ':: 'v propo list
  and x:: 'v
  assume wf-conn c (ξ @ φ # ξ')
  and c = CT ∨ c = CF ∨ c = CVar x
  then have ξ @ φ # ξ' = [] by auto
  then have False by auto
  then show all-subformula-st simple-not-symb (conn c (ξ @ φ' # ξ')) by blast
next
fix φ φ' :: 'v propo and ca:: 'v connective and ξ ξ':: 'v propo list
and x :: 'v
assume rel: propo-rew-step (push-conn-inside c c') φ φ'
and φ-φ': all-subformula-st simple-not-symb φ ⇒ all-subformula-st simple-not-symb φ'

```

```

and corr: wf-conn ca ( $\xi @ \varphi \# \xi'$ )
and n: all-subformula-st simple-not-symb (conn ca ( $\xi @ \varphi \# \xi'$ ))
and c: ca = CNot

have empty:  $\xi = [] \ \xi' = []$  using c corr by auto
then have simple-not:all-subformula-st simple-not-symb (FNot  $\varphi$ ) using corr c n by auto
then have simple  $\varphi$ 
  using all-subformula-st-test-symb-true-phi simple-not-symb.simps(1) by blast
then have simple  $\varphi'$ 
  using rel simple-propo-rew-step-push-conn-inside-inv by blast
then show all-subformula-st simple-not-symb (conn ca ( $\xi @ \varphi' \# \xi'$ )) using c empty
  by (metis simple-not  $\varphi$ - $\varphi'$  append-Nil conn.simps(4) all-subformula-st-decomp-explicit(3)
    simple-not-symb.simps(1))
next
fix  $\varphi \ \varphi' :: 'v$  propo and ca :: 'v connective and  $\xi \ \xi' :: 'v$  propo list
and x :: 'v
assume rel: propo-rew-step (push-conn-inside c c')  $\varphi \ \varphi'$ 
and n $\varphi$ : all-subformula-st simple-not-symb  $\varphi \implies$  all-subformula-st simple-not-symb  $\varphi'$ 
and corr: wf-conn ca ( $\xi @ \varphi \# \xi'$ )
and n: all-subformula-st simple-not-symb (conn ca ( $\xi @ \varphi \# \xi'$ ))
and c: ca  $\in$  binary-connectives

have all-subformula-st simple-not-symb  $\varphi$ 
  using n c corr all-subformula-st-decomp by fastforce
then have  $\varphi'$ : all-subformula-st simple-not-symb  $\varphi'$  using n $\varphi$  by blast
obtain a b where ab: [a, b] = ( $\xi @ \varphi \# \xi'$ )
  using corr c list-length2-decomp wf-conn-bin-list-length by metis
then have  $\xi @ \varphi' \# \xi' = [a, \varphi'] \vee (\xi @ \varphi' \# \xi') = [\varphi', b]$ 
  using ab by (metis (no-types, hide-lams) append-Cons append-Nil append-Nil2
    append-is-Nil-conv butlast.simps(2) butlast-append list.sel(3) tl-append2)
moreover
{
  fix  $\chi :: 'v$  propo
  have wf': wf-conn ca [a, b]
    using ab corr by presburger
  have all-subformula-st simple-not-symb (conn ca [a, b])
    using ab n by presburger
  then have all-subformula-st simple-not-symb  $\chi \vee \chi \notin$  set ( $\xi @ \varphi' \# \xi'$ )
    using wf' by (metis (no-types)  $\varphi'$  all-subformula-st-decomp calculation insert-iff
      list.set(2))
}
then have  $\forall \varphi. \varphi \in$  set ( $\xi @ \varphi' \# \xi'$ )  $\longrightarrow$  all-subformula-st simple-not-symb  $\varphi$ 
  by (metis (no-types))

moreover have simple-not-symb (conn ca ( $\xi @ \varphi' \# \xi'$ ))
  using ab conn-inj-not(1) corr wf-conn-list-decomp(4) wf-conn-no-arity-change
    not-Cons-self2 self-append-conv2 simple-not-symb.elims(3) by (metis (no-types) c
    calculation(1) wf-conn-binary)
moreover have wf-conn ca ( $\xi @ \varphi' \# \xi'$ ) using c calculation(1) by auto
ultimately show all-subformula-st simple-not-symb (conn ca ( $\xi @ \varphi' \# \xi'$ ))
  by (metis all-subformula-st-decomp-imp)
qed
}
moreover {
  fix ca :: 'v connective and  $\xi \ \xi' :: 'v$  propo list and  $\varphi \ \varphi' :: 'v$  propo
  have propo-rew-step (push-conn-inside c c')  $\varphi \ \varphi' \implies$  wf-conn ca ( $\xi @ \varphi \# \xi'$ )

```

```

    => simple-not-symb (conn ca (ξ @ φ # ξ')) => simple-not-symb φ'
    => simple-not-symb (conn ca (ξ @ φ' # ξ'))
  by (metis append-self-conv2 conn.simps(4) conn-inj-not(1) simple-not-symb.elims(3)
    simple-not-symb.simps(1) simple-propo-rew-step-push-conn-inside-inv
    wf-conn-no-arity-change-helper wf-conn-list-decomp(4) wf-conn-no-arity-change)
}
ultimately show simple-not ψ
  using full-propo-rew-step-inv-stay'[of push-conn-inside c c' simple-not-symb] assms
  unfolding no-T-F-except-top-level-def simple-not-def full-unfold by metis
next
{
  fix φ ψ :: 'v propo
  have H: propo-rew-step (push-conn-inside c c') φ ψ => no-T-F-except-top-level φ
    => no-T-F-except-top-level ψ
  proof -
    assume rel: propo-rew-step (push-conn-inside c c') φ ψ
    and no-T-F-except-top-level φ
    then have no-T-F φ ∨ φ = FF ∨ φ = FT
      by (metis no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb)
    moreover {
      assume φ = FF ∨ φ = FT
      then have False using rel propo-rew-step-push-conn-inside by blast
      then have no-T-F-except-top-level ψ by blast
    }
    moreover {
      assume no-T-F φ ∧ φ ≠ FF ∧ φ ≠ FT
      then have no-T-F ψ using rel push-conn-inside-in-c'-symb-no-T-F by blast
      then have no-T-F-except-top-level ψ using no-T-F-no-T-F-except-top-level by blast
    }
    ultimately show no-T-F-except-top-level ψ by blast
  qed
}
moreover {
  fix ca :: 'v connective and ξ ξ' :: 'v propo list and φ φ' :: 'v propo
  assume rel: propo-rew-step (push-conn-inside c c') φ φ'
  assume corr: wf-conn ca (ξ @ φ # ξ')
  then have c: ca ≠ CT ∧ ca ≠ CF by auto
  assume no-T-F: no-T-F-symb-except-toplevel (conn ca (ξ @ φ # ξ'))
  have no-T-F-symb-except-toplevel (conn ca (ξ @ φ' # ξ'))
  proof
    have c: ca ≠ CT ∧ ca ≠ CF using corr by auto
    have ζ: ∀ ζ ∈ set (ξ @ φ # ξ'). ζ ≠ FT ∧ ζ ≠ FF
      using corr no-T-F no-T-F-symb-except-toplevel-if-is-a-true-false by blast
    then have φ ≠ FT ∧ φ ≠ FF by auto
    from rel this have φ' ≠ FT ∧ φ' ≠ FF
    apply (induct rule: propo-rew-step.induct)
    by (metis append-is-Nil-conv conn.simps(2) conn-inj list.distinct(1)
      wf-conn-helper-facts(3) wf-conn-list(1) wf-conn-no-arity-change
      wf-conn-no-arity-change-helper push-conn-inside-not-true-false)+
    then have ∀ ζ ∈ set (ξ @ φ' # ξ'). ζ ≠ FT ∧ ζ ≠ FF using ζ by auto
    moreover have wf-conn ca (ξ @ φ' # ξ')
      using corr wf-conn-no-arity-change by (metis wf-conn-no-arity-change-helper)
    ultimately show no-T-F-symb (conn ca (ξ @ φ' # ξ')) using no-T-F-symb.intros c by metis
  qed
}
ultimately show no-T-F-except-top-level ψ

```

```

using full-propo-rew-step-inv-stay'[of push-conn-inside c c' no-T-F-symb-except-toplevel]
assms unfolding no-T-F-except-top-level-def full-unfold by metis

next
{
  fix  $\varphi \psi :: 'v \text{ propo}$ 
  have  $H: \text{push-conn-inside } c \ c' \ \varphi \ \psi \implies \text{no-equiv } \varphi \implies \text{no-equiv } \psi$ 
    by (induct  $\varphi \ \psi$  rule: push-conn-inside.induct, auto)
}
then show no-equiv  $\psi$ 
using full-propo-rew-step-inv-stay-conn[of push-conn-inside c c' no-equiv-symb] assms
no-equiv-symb-conn-characterization unfolding no-equiv-def by metis

next
{
  fix  $\varphi \psi :: 'v \text{ propo}$ 
  have  $H: \text{push-conn-inside } c \ c' \ \varphi \ \psi \implies \text{no-imp } \varphi \implies \text{no-imp } \psi$ 
    by (induct  $\varphi \ \psi$  rule: push-conn-inside.induct, auto)
}
then show no-imp  $\psi$ 
using full-propo-rew-step-inv-stay-conn[of push-conn-inside c c' no-imp-symb] assms
no-imp-symb-conn-characterization unfolding no-imp-def by metis
qed

```

lemma push-conn-inside-full-propo-rew-step:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes
 no-equiv φ **and**
 no-imp φ **and**
 full (propo-rew-step (push-conn-inside c c')) $\varphi \ \psi$ **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
using c-in-c'-symb-rew assms full-propo-rew-step-subformula **by** blast

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

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

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) **by** auto

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 $\psi \iff$ (simple ψ
 $\vee (\exists \varphi'. \psi = FNot \ \varphi' \wedge \text{simple } \varphi')$
 $\vee (\exists l. \psi = \text{conn } c \ l \wedge \text{wf-conn } c \ l))$

by (auto simp: only-c-inside-symb.intros(3)) (induct rule: only-c-inside-symb.induct, auto)

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

apply (auto simp: only-c-inside-symb.intros(3))

by (induct FNot ψ rule: only-c-inside-symb.induct, auto simp: wf-conn-list(8) c)

lemma only-c-inside-decomp-not[simp]:

assumes $c: c \neq CNot$

shows only-c-inside c (FNot ψ) \longleftrightarrow simple ψ

by (metis (no-types, hide-lams) all-subformula-st-def all-subformula-st-test-symb-true-phi c
only-c-inside-def only-c-inside-symb-decomp-not simple-only-c-inside
subformula-conn-decomp-simple)

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-conn $c\ l)))$)

unfolding only-c-inside-def by (auto simp: all-subformula-st-def only-c-inside-symb-decomp)

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

fixes $c\ c' :: 'v$ connective and $l :: 'v$ propo list and $\varphi :: 'v$ 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 φ and incl: conn $c'\ l \preceq \varphi$ and wf: wf-conn $c'\ l$

shows False

proof –

let $? \psi = conn\ c'\ l$

have simple $? \psi \vee (\exists \varphi'. ? \psi = FNot \varphi' \wedge$ simple $\varphi') \vee (\exists l. ? \psi = conn\ c\ l \wedge$ wf-conn $c\ l)$

using only-c-inside-decomp only incl by blast

moreover have \neg simple $? \psi$

using wf simple-decomp by (metis c' connective.distinct(19) connective.distinct(7,9,21,29,31)
wf-conn-list(1-3))

moreover

{
 fix φ'
 have $? \psi \neq FNot \varphi'$ using c' conn-inj-not(1) wf by blast
}

ultimately obtain $l :: 'v$ propo list where $? \psi = conn\ c\ l \wedge$ wf-conn $c\ l$ by metis

then have $c = c'$ using conn-inj wf by metis

then show False using cc' by auto

qed

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 $\varphi \implies c\text{-in-}c'\text{-symb}\ c\ c'\ \varphi$

apply (rule ccontr)

apply (cases rule: not-c-in-c'-symb.cases, auto)

by (metis $\delta\ c\ c'$ connective.distinct(37,39) list.distinct(1) only-c-inside-c-c'-false
subformula-in-binary-conn(1,2) wf-conn.simps)+

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

fixes $l :: 'v$ propo list and $c\ c' ca :: 'v$ connective

shows wf-conn $ca\ l \implies ca \neq c \implies c\text{-in-}c'\text{-symb}\ c\ c'\ (conn\ ca\ l)$

proof –

have *not-c-in-c'-symb* $c\ c'$ (*conn* $ca\ l$) \implies *wf-conn* $ca\ l \implies ca = c$
by (*induct conn* $ca\ l$ *rule: not-c-in-c'-symb.induct, auto simp: conn-inj*)
then show *wf-conn* $ca\ l \implies ca \neq c \implies c\text{-in-}c'\text{-symb } c\ c'$ (*conn* $ca\ l$) **by** *blast*
qed

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

assumes δ : $c \neq c'$ **and** c : $c = CAnd \vee c = COr$ **and** c' : $c' = CAnd \vee c' = COr$
shows *only-c-inside* $c\ \varphi \implies c\text{-in-}c'\text{-only } c\ c'\ \varphi$
unfolding *c-in-c'-only-def all-subformula-st-def*
using *only-c-inside-implies-c-in-c'-symb*
by (*metis all-subformula-st-def assms(1) c c' only-c-inside-def subformula-trans*)

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

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

using *inv*

proof (*induct conn* $c\ l$ *arbitrary: l rule: propo-induct-arity*)

case (*nullary* x)

then show *?case* **by** (*auto simp: wf-conn-list assms*)

next

case (*unary* $\varphi\ la$)

then have $c = CNot \wedge la = [\varphi]$ **by** (*metis (no-types) wf-conn-list(8)*)

then show *?case* **using** *assms(2) assms(1)* **by** *blast*

next

case (*binary* $\varphi1\ \varphi2$)

note $IH\varphi1 = \text{this}(1)$ **and** $IH\varphi2 = \text{this}(2)$ **and** $\varphi = \text{this}(3)$ **and** *only* $= \text{this}(5)$ **and** *wf* $= \text{this}(4)$
and *no-equiv* $= \text{this}(6)$ **and** *no-imp* $= \text{this}(7)$ **and** *simple-not* $= \text{this}(8)$

then have l : $l = [\varphi1, \varphi2]$ **by** (*meson wf-conn-list(4-7)*)

let $?\varphi = \text{conn } c\ l$

obtain $c1\ l1\ c2\ l2$ **where** $\varphi1$: $\varphi1 = \text{conn } c1\ l1$ **and** $\text{wf}\varphi1$: *wf-conn* $c1\ l1$

and $\varphi2$: $\varphi2 = \text{conn } c2\ l2$ **and** $\text{wf}\varphi2$: *wf-conn* $c2\ l2$ **using** *exists-c-conn* **by** *metis*

then have *c-in-only* $\varphi1$: *c-in-c'-only* $c\ c'$ (*conn* $c1\ l1$) **and** *c-in-c'-only* $c\ c'$ (*conn* $c2\ l2$)

using *only l unfolding c-in-c'-only-def* **using** *assms(1)* **by** *auto*

have *inc* $\varphi1$: $\varphi1 \preceq ?\varphi$ **and** *inc* $\varphi2$: $\varphi2 \preceq ?\varphi$

using $\varphi1\ \varphi2\ \varphi$ *local.wf* **by** (*metis conn.simps(5-8) helper-fact subformula-in-binary-conn(1,2))+*

have $c1\text{-eq}$: $c1 \neq CEq$ **and** $c2\text{-eq}$: $c2 \neq CEq$

unfolding *no-equiv-def* **using** *inc* $\varphi1$ *inc* $\varphi2$ **by** (*metis* $\varphi1\ \varphi2\ \text{wf}\varphi1\ \text{wf}\varphi2\ \text{assms}(1)\ \text{no-equiv}\ \text{no-equiv-eq}(1)\ \text{no-equiv-symb.elims}(3)\ \text{no-equiv-symb-conn-characterization}\ \text{wf-conn-list}(4,5)\ \text{no-equiv-def subformula-all-subformula-st})+$

have $c1\text{-imp}$: $c1 \neq CImp$ **and** $c2\text{-imp}$: $c2 \neq CImp$

using *no-imp* **by** (*metis* $\varphi1\ \varphi2\ \text{all-subformula-st-decomp-explicit-imp}(2,3)\ \text{assms}(1)\ \text{conn.simps}(5,6)\ l\ \text{no-imp-imp}(1)\ \text{no-imp-symb.elims}(3)\ \text{no-imp-symb-conn-characterization}\ \text{wf}\varphi1\ \text{wf}\varphi2\ \text{all-subformula-st-decomp no-imp-symb-conn-characterization})+$

have $c1c$: $c1 \neq c'$

proof

assume $c1c$: $c1 = c'$

then obtain $\xi1\ \xi2$ **where** $l1$: $l1 = [\xi1, \xi2]$

by (*metis* *assms(2) connective.distinct(37,39) helper-fact wf* $\varphi1$ *wf-conn.simps wf-conn-list-decomp(1-3)*)

have *c-in-c'-only* $c\ c'$ (*conn* $c\ [\text{conn } c'\ l1, \varphi2]$) **using** $c1c\ l$ *only* $\varphi1$ **by** *auto*

```

    moreover have not-c-in-c'-symb c c' (conn c [conn c' l1,  $\varphi 2$ ])
    using l1  $\varphi 1$  c1c l local.wf not-c-in-c'-symb-l wf  $\varphi 1$  by blast
    ultimately show False using  $\varphi 1$  c1c l l1 local.wf not-c-in-c'-simp(4) wf  $\varphi 1$  by blast
  qed
then have ( $\varphi 1 = \text{conn } c \text{ l1} \wedge \text{wf-conn } c \text{ l1}$ )  $\vee$  ( $\exists \psi 1. \varphi 1 = \text{FNot } \psi 1$ )  $\vee$  simple  $\varphi 1$ 
  by (metis  $\varphi 1$  assms(1-3) c1-eq c1-imp simple.elims(3) wf  $\varphi 1$  wf-conn-list(4) wf-conn-list(5-7))
moreover {
  assume  $\varphi 1 = \text{conn } c \text{ l1} \wedge \text{wf-conn } c \text{ l1}$ 
  then have only-c-inside c  $\varphi 1$ 
    by (metis IH  $\varphi 1$   $\varphi 1$  all-subformula-st-decomp-imp inc  $\varphi 1$  no-equiv no-equiv-def no-imp no-imp-def
      c-in-only  $\varphi 1$  only-c-inside-def only-c-inside-into-only-c-inside simple-not simple-not-def
      subformula-all-subformula-st)
}
moreover {
  assume  $\exists \psi 1. \varphi 1 = \text{FNot } \psi 1$ 
  then obtain  $\psi 1$  where  $\varphi 1 = \text{FNot } \psi 1$  by metis
  then have only-c-inside c  $\varphi 1$ 
    by (metis all-subformula-st-def assms(1) connective.distinct(37,39) inc  $\varphi 1$ 
      only-c-inside-decomp-not simple-not simple-not-def simple-not-symb.simps(1))
}
moreover {
  assume simple  $\varphi 1$ 
  then have only-c-inside c  $\varphi 1$ 
    by (metis all-subformula-st-decomp-explicit(3) assms(1) connective.distinct(37,39)
      only-c-inside-decomp-not only-c-inside-def)
}
ultimately have only-c-inside  $\varphi 1$ : only-c-inside c  $\varphi 1$  by metis

have c-in-only  $\varphi 2$ : c-in-c'-only c c' (conn c2 l2)
  using only l  $\varphi 2$  wf  $\varphi 2$  assms unfolding c-in-c'-only-def by auto
have c2c: c2  $\neq$  c'
proof
  assume c2c: c2 = c'
  then obtain  $\xi 1$   $\xi 2$  where l2: l2 = [ $\xi 1$ ,  $\xi 2$ ]
    by (metis assms(2) wf  $\varphi 2$  wf-conn.simps connective.distinct(7,9,19,21,29,31,37,39))
  then have c-in-c'-symb c c' (conn c [ $\varphi 1$ , conn c' l2])
    using c2c l only  $\varphi 2$  all-subformula-st-test-symb-true-phi unfolding c-in-c'-only-def by auto
  moreover have not-c-in-c'-symb c c' (conn c [ $\varphi 1$ , conn c' l2])
    using assms(1) c2c l2 not-c-in-c'-symb-r wf  $\varphi 2$  wf-conn-helper-facts(5,6) by metis
  ultimately show False by auto
qed
then have ( $\varphi 2 = \text{conn } c \text{ l2} \wedge \text{wf-conn } c \text{ l2}$ )  $\vee$  ( $\exists \psi 2. \varphi 2 = \text{FNot } \psi 2$ )  $\vee$  simple  $\varphi 2$ 
  using c2-eq by (metis  $\varphi 2$  assms(1-3) c2-eq c2-imp simple.elims(3) wf  $\varphi 2$  wf-conn-list(4-7))
moreover {
  assume  $\varphi 2 = \text{conn } c \text{ l2} \wedge \text{wf-conn } c \text{ l2}$ 
  then have only-c-inside c  $\varphi 2$ 
    by (metis IH  $\varphi 2$   $\varphi 2$  all-subformula-st-decomp inc  $\varphi 2$  no-equiv no-equiv-def no-imp no-imp-def
      c-in-only  $\varphi 2$  only-c-inside-def only-c-inside-into-only-c-inside simple-not simple-not-def
      subformula-all-subformula-st)
}
moreover {
  assume  $\exists \psi 2. \varphi 2 = \text{FNot } \psi 2$ 
  then obtain  $\psi 2$  where  $\varphi 2 = \text{FNot } \psi 2$  by metis
  then have only-c-inside c  $\varphi 2$ 
    by (metis all-subformula-st-def assms(1-3) connective.distinct(38,40) inc  $\varphi 2$ 
      only-c-inside-decomp-not simple-not simple-not-def simple-not-symb.simps(1))
}

```

```

}
moreover {
  assume simple  $\varphi 2$ 
  then have only-c-inside  $c \varphi 2$ 
    by (metis all-subformula-st-decomp-explicit(3) assms(1) connective.distinct(37,39)
      only-c-inside-decomp-not only-c-inside-def)
}
ultimately have only-c-inside $\varphi 2$ : only-c-inside  $c \varphi 2$  by metis
show ?case using l only-c-inside $\varphi 1$  only-c-inside $\varphi 2$  by auto
qed

```

Push Conjunction

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

lemma *pushConj-consistent*: *preserves-un-sat pushConj*
unfolding *pushConj-def* **by** (*simp add: push-conn-inside-consistent*)

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

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

lemma *pushConj-inv*:
fixes $\varphi \psi :: 'v \text{ propo}$
assumes *full (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*
using *push-conn-inside-inv assms unfolding pushConj-def by metis+*

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*
using *assms push-conn-inside-full-propo-rew-step*
unfolding *pushConj-def and-in-or-only-def c-in-c'-only-def by (metis (no-types))*

Push Disjunction

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

lemma *pushDisj-consistent*: *preserves-un-sat pushDisj*
unfolding *pushDisj-def* **by** (*simp add: push-conn-inside-consistent*)

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]*:
 $\sim \text{or-in-and-only } (FOr \ (FAnd \ \psi1 \ \psi2) \ \varphi')$
unfolding *or-in-and-only-def*
by (*metis all-subformula-st-test-symb-true-phi conn.simps(5-6) not-c-in-c'-symb-l*
wf-conn-helper-facts(5) wf-conn-helper-facts(6))

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*
using *push-conn-inside-inv assms unfolding pushDisj-def by metis+*

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*
using *assms push-conn-inside-full-propo-rew-step*
unfolding *pushDisj-def or-in-and-only-def c-in-c'-only-def by (metis (no-types))*

0.4 The full transformations

0.4.1 Abstract Property characterizing that only some connective are inside the others

Definition

The normal is a super group of groups

inductive *grouped-by* :: *'a connective \Rightarrow 'a propo \Rightarrow bool for c where*
simple-is-grouped[simp]: simple $\varphi \Rightarrow$ grouped-by c φ |
simple-not-is-grouped[simp]: simple $\varphi \Rightarrow$ grouped-by c (FNot φ) |
connected-is-group[simp]: grouped-by c $\varphi \Rightarrow$ grouped-by c $\psi \Rightarrow$ wf-conn c $[\varphi, \psi]$
 \Rightarrow 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))
by *simp+*

lemma *only-c-inside-symb-c-eq-c'*:
 $\text{only-c-inside-symb } c \ (\text{conn } c' \ [\varphi1, \varphi2]) \Rightarrow c' = CAnd \vee c' = COr \Rightarrow \text{wf-conn } c' \ [\varphi1, \varphi2]$
 $\Rightarrow c' = c$
by (*induct conn c' $[\varphi1, \varphi2]$ rule: only-c-inside-symb.induct, auto simp: conn-inj*)

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

only-c-inside c (*conn* c' [$\varphi 1$, $\varphi 2$]) $\implies c' = CAnd \vee c' = COr \implies wf\text{-}conn\ c' [\varphi 1, \varphi 2] \implies c = c'$
unfolding *only-c-inside-def* *all-subformula-st-def* **using** *only-c-inside-symb-c-eq-c'* *subformula-refl*
by *blast*

lemma *only-c-inside-imp-grouped-by*:
assumes $c: c \neq CNot$ **and** $c': c' = CAnd \vee c' = COr$
shows *only-c-inside* $c\ \varphi \implies grouped\text{-}by\ c\ \varphi$ (**is** $?O\ \varphi \implies ?G\ \varphi$)
proof (*induct* φ *rule: propo-induct-arity*)
case (*nullary* $\varphi\ x$)
then show $?G\ \varphi$ **by** *auto*
next
case (*unary* ψ)
then show $?G\ (FNot\ \psi)$ **by** (*auto simp: c*)
next
case (*binary* $\varphi\ \varphi 1\ \varphi 2$)
note $IH\ \varphi 1 = this(1)$ **and** $IH\ \varphi 2 = this(2)$ **and** $\varphi = this(3)$ **and** $only = this(4)$
have $\varphi\text{-}conn: \varphi = conn\ c\ [\varphi 1, \varphi 2]$ **and** $wf: wf\text{-}conn\ c\ [\varphi 1, \varphi 2]$
proof –
obtain $c''\ l''$ **where** $\varphi\text{-}c'': \varphi = conn\ c''\ l''$ **and** $wf: wf\text{-}conn\ c''\ l''$
using *exists-c-conn* **by** *metis*
then have $l'': l'' = [\varphi 1, \varphi 2]$ **using** φ **by** (*metis wf-conn-list(4-7)*)
have *only-c-inside-symb* $c\ (conn\ c''\ [\varphi 1, \varphi 2])$
using *only all-subformula-st-test-symb-true-phi*
unfolding *only-c-inside-def* $\varphi\text{-}c''\ l''$ **by** *metis*
then have $c = c''$
by (*metis* $\varphi\ \varphi\text{-}c''\ conn\text{-}inj\ conn\text{-}inj\text{-}not(2)\ l''\ list.distinct(1)\ list.inject\ wf$
only-c-inside-symb.cases simple.simps(5-8))
then show $\varphi = conn\ c\ [\varphi 1, \varphi 2]$ **and** $wf\text{-}conn\ c\ [\varphi 1, \varphi 2]$ **using** $\varphi\text{-}c''\ wf\ l''$ **by** *auto*
qed
have *grouped-by* $c\ \varphi 1$ **using** $wf\ IH\ \varphi 1\ IH\ \varphi 2\ \varphi\text{-}conn\ only\ \varphi$ **unfolding** *only-c-inside-def* **by** *auto*
moreover have *grouped-by* $c\ \varphi 2$
using $wf\ \varphi\ IH\ \varphi 1\ IH\ \varphi 2\ \varphi\text{-}conn\ only$ **unfolding** *only-c-inside-def* **by** *auto*
ultimately show $?G\ \varphi$ **using** $\varphi\text{-}conn\ connected\text{-}is\text{-}group\ local.wf$ **by** *blast*
qed

lemma *grouped-by-false*:
grouped-by $c\ (conn\ c'\ [\varphi, \psi]) \implies c \neq c' \implies wf\text{-}conn\ c'\ [\varphi, \psi] \implies False$
apply (*induct* $conn\ c'\ [\varphi, \psi]$ *rule: grouped-by.induct*)
apply (*auto simp: simple-decomp wf-conn-list, auto simp: conn-inj*)
by (*metis list.distinct(1) list.sel(3) wf-conn-list(8)+*)

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* \Rightarrow '*a* *connective* \Rightarrow '*a* *propo* \Rightarrow *bool* **for** $c\ c'$ **where**
grouped-is-super-grouped[*simp*]: *grouped-by* $c\ \varphi \implies super\text{-}grouped\text{-}by\ c\ c'\ \varphi$ |
connected-is-super-group: *super-grouped-by* $c\ c'\ \varphi \implies super\text{-}grouped\text{-}by\ c\ c'\ \psi \implies wf\text{-}conn\ c\ [\varphi, \psi]$
 $\implies super\text{-}grouped\text{-}by\ c\ c'\ (conn\ c'\ [\varphi, \psi])$

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 $c\ c'\ (FNot\ (FVar\ x))$

by auto

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* $c \ c' \ \varphi$

\implies *super-grouped-by* $c \ c' \ \varphi$

(**is** ?NE $\varphi \implies$?NI $\varphi \implies$?SN $\varphi \implies$?C $\varphi \implies$?S φ)

proof (*induct* φ *rule: propo-induct-arity*)

case (*nullary* $\varphi \ x$)

then show ?S φ **by** auto

next

case (*unary* φ)

then have *simple-not-symb* (*FNot* φ)

using *all-subformula-st-test-symb-true-phi* **unfolding** *simple-not-def* **by** blast

then have $\varphi = FT \vee \varphi = FF \vee (\exists \ x. \ \varphi = FVar \ x)$ **by** (*cases* φ , auto)

then show ?S (*FNot* φ) **by** auto

next

case (*binary* $\varphi \ \varphi1 \ \varphi2$)

note *IH* $\varphi1 = this(1)$ **and** *IH* $\varphi2 = this(2)$ **and** *no-equiv* $= this(4)$ **and** *no-imp* $= this(5)$

and *simpleN* $= this(6)$ **and** *c-in-c'-only* $= this(7)$ **and** $\varphi' = this(3)$

{

assume $\varphi = FImp \ \varphi1 \ \varphi2 \vee \varphi = FEq \ \varphi1 \ \varphi2$

then have *False* **using** *no-equiv no-imp* **by** auto

then have ?S φ **by** auto

}

moreover {

assume $\varphi: \varphi = conn \ c' \ [\varphi1, \varphi2] \wedge wf_conn \ c' \ [\varphi1, \varphi2]$

have *c-in-c'-only*: *c-in-c'-only* $c \ c' \ \varphi1 \wedge c-in-c'-only \ c \ c' \ \varphi2 \wedge c-in-c'-symb \ c \ c' \ \varphi$

using *c-in-c'-only* φ' **unfolding** *c-in-c'-only-def* **by** auto

have *super-grouped-by* $c \ c' \ \varphi1$ **using** $\varphi \ c'$ *no-equiv no-imp simpleN IH* $\varphi1$ *c-in-c'-only* **by** auto

moreover have *super-grouped-by* $c \ c' \ \varphi2$

using $\varphi \ c'$ *no-equiv no-imp simpleN IH* $\varphi2$ *c-in-c'-only* **by** auto

ultimately have ?S φ

using *super-grouped-by.intros*(2) φ **by** (*metis* $c \ wf_conn_helper_facts(5,6)$)

}

moreover {

assume $\varphi: \varphi = conn \ c \ [\varphi1, \varphi2] \wedge wf_conn \ c \ [\varphi1, \varphi2]$

then have *only-c-inside* $c \ \varphi1 \wedge only-c-inside \ c \ \varphi2$

using *c-in-c'-symb-c-implies-only-c-inside* $c \ c' \ c-in-c'-only \ list.set-intros(1)$

wf_conn_helper_facts(5,6) *no-equiv no-imp simpleN last-ConsL last-ConsR last-in-set*

list.distinct(1) **by** (*metis* (*no-types*, *hide-lams*) *cc'*)

then have *only-c-inside* $c \ (conn \ c \ [\varphi1, \varphi2])$

unfolding *only-c-inside-def* **using** φ

by (*simp add: only-c-inside-into-only-c-inside all-subformula-st-decomp*)

then have *grouped-by* $c \ \varphi$ **using** φ *only-c-inside-imp-grouped-by* c **by** blast

then have ?S φ **using** *super-grouped-by.intros*(1) **by** *metis*

}

ultimately show ?S φ **by** (*metis* $\varphi' \ c \ c' \ cc' \ conn.simps(5,6) \ wf_conn_helper_facts(5,6)$)

qed

0.4.2 Conjunctive Normal Form

definition *is-conj-with-TF* **where** *is-conj-with-TF* $=$ *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* φ

using *c-in-c'-only-super-grouped-by*
unfolding *is-conj-with-TF-def or-in-and-only-def c-in-c'-only-def*
by (*simp add: c-in-c'-only-def c-in-c'-only-super-grouped-by*)

definition *is-cnf* **where**

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

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-consistent: preserves-un-sat cnf-rew*

by (*simp add: cnf-rew-def elimEquiv-lifted-consistant elim-imp-lifted-consistant elimTB-consistent preserves-un-sat-OO pushDisj-consistent pushNeg-lifted-consistant*)

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

apply (*unfold cnf-rew-def OO-def*)

apply *auto*

proof –

fix φ φEq φImp φTB φNeg $\varphiDisj :: 'v$ *propo*

assume *Eq: full (propo-rew-step elim-equiv) φ φEq*

then have *no-equiv: no-equiv φEq using no-equiv-full-propo-rew-step-elim-equiv by blast*

assume *Imp: full (propo-rew-step elim-imp) φEq φImp*

then have *no-imp: no-imp φImp using no-imp-full-propo-rew-step-elim-imp by blast*

have *no-imp-inv: no-equiv φImp using no-equiv Imp elim-imp-inv by blast*

assume *TB: full (propo-rew-step elimTB) φImp φTB*

then have *noTB: no-T-F-except-top-level φTB*

using *no-imp-inv no-imp elimTB-full-propo-rew-step by blast*

have *noTB-inv: no-equiv φTB no-imp φTB using elimTB-inv TB no-imp no-imp-inv by blast+*

assume *Neg: full (propo-rew-step pushNeg) φTB φNeg*

then have *noNeg: simple-not φNeg*

using *noTB-inv noTB pushNeg-full-propo-rew-step by blast*

have *noNeg-inv: no-equiv φNeg no-imp φNeg no-T-F-except-top-level φNeg*

using *pushNeg-inv Neg noTB noTB-inv by blast+*

assume *Disj: full (propo-rew-step pushDisj) φNeg \varphiDisj*

then have *no-Disj: or-in-and-only \varphiDisj*

using *noNeg-inv noNeg pushDisj-full-propo-rew-step by blast*

have *noDisj-inv: no-equiv \varphiDisj no-imp \varphiDisj no-T-F-except-top-level \varphiDisj simple-not \varphiDisj*

using *pushDisj-inv Disj noNeg noNeg-inv by blast+*

moreover have *is-conj-with-TF \varphiDisj*

using *or-in-and-only-conjunction-in-disj noDisj-inv no-Disj by blast*

ultimately show *is-cnf \varphiDisj unfolding is-cnf-def by blast*

qed

0.4.3 Disjunctive Normal Form

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* φ

using *c-in-c'-only-super-grouped-by*

unfolding *is-disj-with-TF-def* *and-in-or-only-def* *c-in-c'-only-def*

by (*simp add: c-in-c'-only-def c-in-c'-only-super-grouped-by*)

definition *is-dnf* $:: 'a \text{ propo} \Rightarrow \text{bool}$ **where**

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

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: preserves-un-sat dnf-rew*

by (*simp add: dnf-rew-def elimEquiv-lifted-consistant elim-imp-lifted-consistant elimTB-consistent preserves-un-sat-OO pushConj-consistent pushNeg-lifted-consistant*)

theorem *dnf-transformation-correction*:

dnf-rew $\varphi \varphi' \implies$ *is-dnf* φ'

apply (*unfold dnf-rew-def OO-def*)

by (*meson and-in-or-only-conjunction-in-disj elimTB-full-propo-rew-step elimTB-inv(1,2)*

elim-imp-inv is-dnf-def no-equiv-full-propo-rew-step-elim-equiv

no-imp-full-propo-rew-step-elim-imp pushConj-full-propo-rew-step pushConj-inv(1-4)

pushNeg-full-propo-rew-step pushNeg-inv(1-3))

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 |

$ElimTBFull4[simp]: elimTBFull (FOr \varphi FF) \varphi \mid$
 $ElimTBFull4'[simp]: elimTBFull (FOr FF \varphi) \varphi \mid$

 $ElimTBFull5[simp]: elimTBFull (FNot FT) FF \mid$
 $ElimTBFull5'[simp]: elimTBFull (FNot FF) FT \mid$

 $ElimTBFull6-l[simp]: elimTBFull (FImp FT \varphi) \varphi \mid$
 $ElimTBFull6-l'[simp]: elimTBFull (FImp FF \varphi) FT \mid$
 $ElimTBFull6-r[simp]: elimTBFull (FImp \varphi FT) FT \mid$
 $ElimTBFull6-r'[simp]: elimTBFull (FImp \varphi FF) (FNot \varphi) \mid$

 $ElimTBFull7-l[simp]: elimTBFull (FEq FT \varphi) \varphi \mid$
 $ElimTBFull7-l'[simp]: elimTBFull (FEq FF \varphi) (FNot \varphi) \mid$
 $ElimTBFull7-r[simp]: elimTBFull (FEq \varphi FT) \varphi \mid$
 $ElimTBFull7-r'[simp]: elimTBFull (FEq \varphi FF) (FNot \varphi) \mid$

The transformation is still consistent.

lemma *elimTBFull-consistent: preserves-un-sat elimTBFull*

proof –

```

{
  fix  $\varphi \psi :: 'b \text{ propo}$ 
  have  $elimTBFull \varphi \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$ 
    by (induct-tac rule: elimTBFull.inducts, auto)
}
then show ?thesis using preserves-un-sat-def by auto
qed

```

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  $\psi \preceq \varphi \implies \neg no-T-F-symb-except-toplevel \psi \implies \exists \psi'. elimTBFull \psi \psi'$ 
proof (induct  $\psi$  rule: propo-induct-arity)
  case (nullary  $\varphi'$ )
  then have False using no-T-F-symb-except-toplevel-true no-T-F-symb-except-toplevel-false by auto
  then show  $Ex (elimTBFull \varphi')$  by blast
next
  case (unary  $\psi$ )
  then have  $\psi = FF \vee \psi = FT$  using no-T-F-symb-except-toplevel-not-decom by blast
  then show  $Ex (elimTBFull (FNot \psi))$  using ElimTBFull5 ElimTBFull5' by blast
next
  case (binary  $\varphi' \psi1 \psi2$ )
  then have  $\psi1 = FT \vee \psi2 = FT \vee \psi1 = FF \vee \psi2 = FF$ 
    by (metis binary-connectives-def conn.simps(5-8) insertI1 insert-commute
      no-T-F-symb-except-toplevel-bin-decom binary.hyps(3))
  then show  $Ex (elimTBFull \varphi')$  using elimTBFull.intros binary.hyps(3) by blast
qed

```

The same applies here. We do not need the assumption, but the deep link between $\neg no-T-F-except-top-level$ 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:  $\neg no-T-F-except-top-level \varphi$ 
shows  $\exists \psi \psi'. \psi \preceq \varphi \wedge elimTBFull \psi \psi'$ 
proof –

```

```

have test-symb-false-nullary:
   $\forall x. \text{no-T-F-symb-except-toplevel } (FF :: 'v \text{ propo}) \wedge \text{no-T-F-symb-except-toplevel } FT$ 
   $\wedge \text{no-T-F-symb-except-toplevel } (FVar (x :: 'v))$ 
  by auto
moreover {
  fix  $c :: 'v \text{ connective}$  and  $l :: 'v \text{ propo list}$  and  $\psi :: 'v \text{ propo}$ 
  have  $H: \text{elimTBFull } (\text{conn } c \ l) \ \psi \implies \neg \text{no-T-F-symb-except-toplevel } (\text{conn } c \ l)$ 
  by (cases conn c l rule: elimTBFull.cases) auto
}
ultimately show ?thesis
using no-test-symb-step-exists[of no-T-F-symb-except-toplevel  $\varphi$  elimTBFull] noTB
no-T-F-symb-except-toplevel-step-exists' unfolding no-T-F-except-top-level-def by metis
qed

```

```

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  $\psi$ 
  using full-propo-rew-step-subformula no-T-F-except-top-level-rew' assms by fastforce

```

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: $\text{propo-rew-step elim-equiv } \varphi \ \psi \implies \text{no-T-F } \varphi \implies \text{no-T-F } \psi$

proof (induct rule: propo-rew-step.induct)

```

  fix  $\varphi' :: 'v \text{ propo}$  and  $\psi' :: 'v \text{ propo}$ 
  assume a1: no-T-F  $\varphi'$ 
  assume a2: elim-equiv  $\varphi' \ \psi'$ 
  have  $\forall x0 \ x1. (\neg \text{elim-equiv } (x1 :: 'v \text{ propo}) \ x0 \vee (\exists v2 \ v3 \ v4 \ v5 \ v6 \ v7. x1 = FEq \ v2 \ v3$ 
     $\wedge x0 = FAnd (FImp \ v4 \ v5) (FImp \ v6 \ v7) \wedge v2 = v4 \wedge v4 = v7 \wedge v3 = v5 \wedge v3 = v6))$ 
  =  $(\neg \text{elim-equiv } x1 \ x0 \vee (\exists v2 \ v3 \ v4 \ v5 \ v6 \ v7. x1 = FEq \ v2 \ v3$ 
     $\wedge x0 = FAnd (FImp \ v4 \ v5) (FImp \ v6 \ v7) \wedge v2 = v4 \wedge v4 = v7 \wedge v3 = v5 \wedge v3 = v6))$ 
  by meson
  then have  $\forall p \ pa. \neg \text{elim-equiv } (p :: 'v \text{ propo}) \ pa \vee (\exists pb \ pc \ pd \ pe \ pf \ pg. p = FEq \ pb \ pc$ 
     $\wedge pa = FAnd (FImp \ pd \ pe) (FImp \ pf \ pg) \wedge pb = pd \wedge pd = pg \wedge pc = pe \wedge pc = pf)$ 
  using elim-equiv.cases by force
  then show no-T-F  $\psi'$  using a1 a2 by fastforce

```

next

```

  fix  $\varphi \ \varphi' :: 'v \text{ propo}$  and  $\xi \ \xi' :: 'v \text{ propo list}$  and  $c :: 'v \text{ connective}$ 
  assume rel: propo-rew-step elim-equiv  $\varphi \ \varphi'$ 
  and IH: no-T-F  $\varphi \implies \text{no-T-F } \varphi'$ 
  and corr: wf-conn c ( $\xi @ \varphi \ \# \ \xi'$ )
  and no-T-F: no-T-F ( $\text{conn } c \ (\xi @ \varphi \ \# \ \xi')$ )
  {
    assume c: c = CNot
    then have empty:  $\xi = [] \ \xi' = []$  using corr by auto
    then have no-T-F  $\varphi$  using no-T-F c no-T-F-decomp-not by auto
    then have no-T-F ( $\text{conn } c \ (\xi @ \varphi' \ \# \ \xi')$ ) using c empty no-T-F-comp-not IH by auto
  }
  moreover {
    assume c: c  $\in$  binary-connectives

```

```

obtain  $a\ b$  where  $ab: \xi @ \varphi \# \xi' = [a, b]$ 
  using  $corr\ c\ list-length2-decomp\ wf-conn-bin-list-length$  by  $metis$ 
then have  $\varphi: \varphi = a \vee \varphi = b$ 
  by ( $metis\ append.simps(1)\ append-is-Nil-conv\ list.distinct(1)\ list.sel(3)\ nth-Cons-0$ 
     $tl-append2$ )
have  $\zeta: \forall \zeta \in set\ (\xi @ \varphi \# \xi').\ no-T-F\ \zeta$ 
  using  $no-T-F\ unfolding\ no-T-F-def$  using  $corr\ all-subformula-st-decomp$  by  $blast$ 

then have  $\varphi': no-T-F\ \varphi'$  using  $ab\ IH\ \varphi$  by  $auto$ 
have  $l': \xi @ \varphi' \# \xi' = [\varphi', b] \vee \xi @ \varphi' \# \xi' = [a, \varphi']$ 
  by ( $metis\ (no-types,\ hide-lams)\ ab\ append-Cons\ append-Nil\ append-Nil2\ butlast.simps(2)$ 
     $butlast-append\ list.distinct(1)\ list.sel(3)$ )
then have  $\forall \zeta \in set\ (\xi @ \varphi' \# \xi').\ no-T-F\ \zeta$  using  $\zeta\ \varphi'\ ab$  by  $fastforce$ 
moreover
  have  $\forall \zeta \in set\ (\xi @ \varphi \# \xi').\ \zeta \neq FT \wedge \zeta \neq FF$ 
    using  $\zeta\ corr\ no-T-F\ no-T-F-except-top-level-false\ no-T-F-no-T-F-except-top-level$  by  $blast$ 
  then have  $no-T-F-symb\ (conn\ c\ (\xi @ \varphi' \# \xi'))$ 
    by ( $metis\ \varphi'\ l'\ ab\ all-subformula-st-test-symb-true-phi\ c\ list.distinct(1)$ 
       $list.set-intros(1,2)\ no-T-F-symb-except-toplevel-bin-decom$ 
       $no-T-F-symb-except-toplevel-no-T-F-symb\ no-T-F-symb-false(1,2)\ no-T-F-def\ wf-conn-binary$ 
       $wf-conn-list(1,2)$ )
  ultimately have  $no-T-F\ (conn\ c\ (\xi @ \varphi' \# \xi'))$ 
    by ( $metis\ l'\ all-subformula-st-decomp-imp\ c\ no-T-F-def\ wf-conn-binary$ )
}
moreover {
  fix  $x$ 
  assume  $c = CVar\ x \vee c = CF \vee c = CT$ 
  then have  $False$  using  $corr$  by  $auto$ 
  then have  $no-T-F\ (conn\ c\ (\xi @ \varphi' \# \xi'))$  by  $auto$ 
}
ultimately show  $no-T-F\ (conn\ c\ (\xi @ \varphi' \# \xi'))$  using  $corr\ wf-conn.cases$  by  $metis$ 
qed

```

lemma $elim-equiv-inv'$:

```

fixes  $\varphi\ \psi :: 'v\ propo$ 
assumes  $full\ (propo-rew-step\ elim-equiv)\ \varphi\ \psi$  and  $no-T-F-except-top-level\ \varphi$ 
shows  $no-T-F-except-top-level\ \psi$ 
proof –
{
  fix  $\varphi\ \psi :: 'v\ propo$ 
  have  $propo-rew-step\ elim-equiv\ \varphi\ \psi \implies no-T-F-except-top-level\ \varphi$ 
     $\implies no-T-F-except-top-level\ \psi$ 
  proof –
    assume  $rel: propo-rew-step\ elim-equiv\ \varphi\ \psi$ 
    and  $no: no-T-F-except-top-level\ \varphi$ 
    {
      assume  $\varphi = FT \vee \varphi = FF$ 
      from  $rel$  this have  $False$ 
      apply ( $induct\ rule: propo-rew-step.induct,\ auto\ simp: wf-conn-list(1,2)$ )
      using  $elim-equiv.simps$  by  $blast+$ 
      then have  $no-T-F-except-top-level\ \psi$  by  $blast$ 
    }
  moreover {
    assume  $\varphi \neq FT \wedge \varphi \neq FF$ 
    then have  $no-T-F\ \varphi$ 
    by ( $metis\ no\ no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb$ )
  }
}

```



```

    then have no-T-F  $\psi$  using propo-rew-step-ElimEquiv-no-T-F rel by blast
    then have no-T-F-except-top-level  $\psi$  by (simp add: no-T-F-no-T-F-except-top-level)
  }
  ultimately show no-T-F-except-top-level  $\psi$  by metis
qed
}
moreover {
  fix c :: 'v connective and  $\xi \xi' :: 'v$  propo list and  $\zeta \zeta' :: 'v$  propo
  assume rel: propo-rew-step elim-equiv  $\zeta \zeta'$ 
  and incl:  $\zeta \preceq \varphi$ 
  and corr: wf-conn c ( $\xi @ \zeta \# \xi'$ )
  and no-T-F: no-T-F-symb-except-toplevel (conn c ( $\xi @ \zeta \# \xi'$ ))
  and n: no-T-F-symb-except-toplevel  $\zeta'$ 
  have no-T-F-symb-except-toplevel (conn c ( $\xi @ \zeta' \# \xi'$ ))
  proof
    have p: no-T-F-symb (conn c ( $\xi @ \zeta \# \xi'$ ))
      using corr wf-conn-list(1) wf-conn-list(2) no-T-F-symb-except-toplevel-no-T-F-symb no-T-F
      by blast
    have l:  $\forall \varphi \in \text{set } (\xi @ \zeta \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$ 
      using corr wf-conn-no-T-F-symb-iff p by blast
    from rel incl have  $\zeta' \neq FT \wedge \zeta' \neq FF$ 
      apply (induction  $\zeta \zeta'$  rule: propo-rew-step.induct)
      apply (cases rule: elim-equiv.cases, auto simp: elim-equiv.simps)
      by (metis append-is-Nil-conv list.distinct wf-conn-list(1,2) wf-conn-no-arity-change
        wf-conn-no-arity-change-helper)
    then have  $\forall \varphi \in \text{set } (\xi @ \zeta' \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$  using l by auto
    moreover have  $c \neq CT \wedge c \neq CF$  using corr by auto
    ultimately show no-T-F-symb (conn c ( $\xi @ \zeta' \# \xi'$ ))
      by (metis corr wf-conn-no-arity-change wf-conn-no-arity-change-helper no-T-F-symb-comp)
  qed
}
ultimately show no-T-F-except-top-level  $\psi$ 
  using full-propo-rew-step-inv-stay-with-inc[of elim-equiv no-T-F-symb-except-toplevel  $\varphi$ ]
  assms subformula-refl unfolding no-T-F-except-top-level-def by metis
qed

```

```

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 (induct rule: propo-rew-step.induct)
  case (global-rel  $\varphi' \psi'$ )
  then show no-T-F  $\psi'$ 
    using elim-imp.cases no-T-F-comp-not no-T-F-decomp(1,2)
    by (metis no-T-F-comp-expanded-explicit(2))
next
  case (propo-rew-one-step-lift  $\varphi \varphi' c \xi \xi'$ )
  note rel = this(1) and IH = this(2) and corr = this(3) and no-T-F = this(4)
  {
    assume c: c = CNot
    then have empty:  $\xi = [] \xi' = []$  using corr by auto
    then have no-T-F  $\varphi$  using no-T-F c no-T-F-decomp-not by auto
    then have no-T-F (conn c ( $\xi @ \varphi' \# \xi'$ )) using c empty no-T-F-comp-not IH by auto
  }
  moreover {
    assume c: c  $\in$  binary-connectives
    then obtain a b where ab:  $\xi @ \varphi \# \xi' = [a, b]$ 
      using corr list-length2-decomp wf-conn-bin-list-length by metis
  }

```

```

then have  $\varphi$ :  $\varphi = a \vee \varphi = b$ 
  by (metis append-self-conv2 wf-conn-list-decomp(4) wf-conn-unary list.discI list.sel(3)
      nth-Cons-0 tl-append2)
have  $\zeta$ :  $\forall \zeta \in \text{set } (\xi @ \varphi \# \xi')$ .  $\text{no-T-F } \zeta$  using ab c propo-rew-one-step-lift.premis by auto

then have  $\varphi'$ :  $\text{no-T-F } \varphi'$ 
  using ab IH  $\varphi$  corr no-T-F no-T-F-def all-subformula-st-decomp-explicit by auto
have  $\chi$ :  $\xi @ \varphi' \# \xi' = [\varphi', b] \vee \xi @ \varphi' \# \xi' = [a, \varphi']$ 
  by (metis (no-types, hide-lams) ab append-Cons append-Nil append-Nil2 butlast.simps(2)
      butlast-append list.distinct(1) list.sel(3))
then have  $\forall \zeta \in \text{set } (\xi @ \varphi' \# \xi')$ .  $\text{no-T-F } \zeta$  using  $\zeta$   $\varphi'$  ab by fastforce
moreover
  have no-T-F (last  $(\xi @ \varphi' \# \xi')$ ) by (simp add: calculation)
  then have no-T-F-symb (conn c  $(\xi @ \varphi' \# \xi')$ )
    by (metis  $\chi$   $\varphi'$   $\zeta$  ab all-subformula-st-test-symb-true-phi c last.simps list.distinct(1)
        list.set-intros(1) no-T-F-bin-decomp no-T-F-def)
  ultimately have no-T-F (conn c  $(\xi @ \varphi' \# \xi')$ ) using c  $\chi$  by fastforce
}
moreover {
  fix x
  assume  $c = CVar\ x \vee c = CF \vee c = CT$ 
  then have False using corr by auto
  then have no-T-F (conn c  $(\xi @ \varphi' \# \xi')$ ) by auto
}
ultimately show no-T-F (conn c  $(\xi @ \varphi' \# \xi')$ ) using corr wf-conn.cases by blast
qed

```

```

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  $\varphi$ 
  shows no-T-F-except-top-level  $\psi$ 
proof -
  {
    {
      fix  $\varphi \psi :: 'v \text{ propo}$ 
      have  $H$ :  $\text{elim-imp } \varphi \psi \implies \text{no-T-F-except-top-level } \varphi \implies \text{no-T-F-except-top-level } \psi$ 
        by (induct  $\varphi \psi$  rule: elim-imp.induct, auto)
    } note H = this
    fix  $\varphi \psi :: 'v \text{ propo}$ 
    have propo-rew-step elim-imp  $\varphi \psi \implies \text{no-T-F-except-top-level } \varphi \implies \text{no-T-F-except-top-level } \psi$ 
    proof -
      assume rel: propo-rew-step elim-imp  $\varphi \psi$ 
      and no: no-T-F-except-top-level  $\varphi$ 
      {
        assume  $\varphi = FT \vee \varphi = FF$ 
        from rel this have False
        apply (induct rule: propo-rew-step.induct)
        by (cases rule: elim-imp.cases, auto simp: wf-conn-list(1,2))
        then have no-T-F-except-top-level  $\psi$  by blast
      }
    moreover {
      assume  $\varphi \neq FT \wedge \varphi \neq FF$ 
      then have no-T-F  $\varphi$ 
        by (metis no no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb)
      then have no-T-F  $\psi$ 

```

```

      using rel propo-rew-step-ElimImp-no-T-F by blast
      then have no-T-F-except-top-level  $\psi$  by (simp add: no-T-F-no-T-F-except-top-level)
    }
    ultimately show no-T-F-except-top-level  $\psi$  by metis
  qed
}
moreover {
  fix  $c :: 'v$  connective and  $\xi \xi' :: 'v$  propo list and  $\zeta \zeta' :: 'v$  propo
  assume rel: propo-rew-step elim-imp  $\zeta \zeta'$ 
  and incl:  $\zeta \preceq \varphi$ 
  and corr: wf-conn  $c (\xi @ \zeta \# \xi')$ 
  and no-T-F: no-T-F-symb-except-toplevel (conn  $c (\xi @ \zeta \# \xi')$ )
  and n: no-T-F-symb-except-toplevel  $\zeta'$ 
  have no-T-F-symb-except-toplevel (conn  $c (\xi @ \zeta' \# \xi')$ )
  proof
    have  $p$ : no-T-F-symb (conn  $c (\xi @ \zeta \# \xi')$ )
    by (simp add: corr no-T-F no-T-F-symb-except-toplevel-no-T-F-symb wf-conn-list(1,2))

    have  $l$ :  $\forall \varphi \in \text{set } (\xi @ \zeta \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$ 
    using corr wf-conn-no-T-F-symb-iff  $p$  by blast
    from rel incl have  $\zeta' \neq FT \wedge \zeta' \neq FF$ 
    apply (induction  $\zeta \zeta'$  rule: propo-rew-step.induct)
    apply (cases rule: elim-imp.cases, auto)
    using wf-conn-list(1,2) wf-conn-no-arity-change wf-conn-no-arity-change-helper
    by (metis append-is-Nil-conv list.distinct(1))+
    then have  $\forall \varphi \in \text{set } (\xi @ \zeta' \# \xi'). \varphi \neq FT \wedge \varphi \neq FF$  using  $l$  by auto
    moreover have  $c \neq CT \wedge c \neq CF$  using corr by auto
    ultimately show no-T-F-symb (conn  $c (\xi @ \zeta' \# \xi')$ )
    using corr wf-conn-no-arity-change no-T-F-symb-comp
    by (metis wf-conn-no-arity-change-helper)
  qed
}
ultimately show no-T-F-except-top-level  $\psi$ 
using full-propo-rew-step-inv-stay-with-inc[of elim-imp no-T-F-symb-except-toplevel  $\varphi$ ]
assms subformula-refl unfolding no-T-F-except-top-level-def by metis
qed

```

0.5.3 The new CNF and DNF transformation

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

definition $\text{dnf-rew}' :: 'a \text{ propo} \Rightarrow 'a \text{ propo} \Rightarrow \text{bool}$ where

$\text{dnf-rew}' =$
 (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 $\text{dnf-rew}'$ -consistent: preserves-un-sat $\text{dnf-rew}'$

by (simp add: $\text{dnf-rew}'$ -def elimEquiv-lifted-consistant elim-imp-lifted-consistant
 elimTBFULL-consistent preserves-un-sat-OO pushConj-consistent pushNeg-lifted-consistant)

theorem $\text{cnf-transformation-correction}$:

$\text{dnf-rew}' \varphi \varphi' \Longrightarrow \text{is-dnf } \varphi'$

unfolding $\text{dnf-rew}'$ -def OO-def

```

by (meson and-in-or-only-conjunction-in-disj elimTBFull-full-propo-rew-step elim-equiv-inv'
    elim-imp-inv elim-imp-inv' is-dnf-def no-equiv-full-propo-rew-step-elim-equiv
    no-imp-full-propo-rew-step-elim-imp pushConj-full-propo-rew-step pushConj-inv(1-4)
    pushNeg-full-propo-rew-step pushNeg-inv(1-3))

```

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: preserves-un-sat cnf-rew'*

```

by (simp add: cnf-rew'-def elimEquiv-lifted-consistant elim-imp-lifted-consistant
    elimTBFull-consistent preserves-un-sat-OO pushDisj-consistent pushNeg-lifted-consistant)

```

theorem *cnf'-transformation-correction:*

```

cnf-rew'  $\varphi$   $\varphi'$   $\Longrightarrow$  is-cnf  $\varphi'$ 
unfolding cnf-rew'-def OO-def
by (meson elimTBFull-full-propo-rew-step elim-equiv-inv' elim-imp-inv elim-imp-inv' is-cnf-def
    no-equiv-full-propo-rew-step-elim-equiv no-imp-full-propo-rew-step-elim-imp
    or-in-and-only-conjunction-in-disj pushDisj-full-propo-rew-step pushDisj-inv(1-4)
    pushNeg-full-propo-rew-step pushNeg-inv(1) pushNeg-inv(2) pushNeg-inv(3))

```

end

theory Prop-Logic-Multiset

imports ../lib/Multiset-More Prop-Normalisation Partial-Clausal-Logic

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  $\varphi$   $\psi$ ) = mset-of-conj  $\varphi$  + mset-of-conj  $\psi$  |
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  $\varphi$   $\psi$ ) = mset-of-formula  $\varphi$   $\cup$  mset-of-formula  $\psi$  |
mset-of-formula (FOr  $\varphi$   $\psi$ ) = {mset-of-conj (FOr  $\varphi$   $\psi$ )} |
mset-of-formula (FVar  $\psi$ ) = {mset-of-conj (FVar  $\psi$ )} |
mset-of-formula (FNot  $\psi$ ) = {mset-of-conj (FNot  $\psi$ )} |
mset-of-formula FF = {{#}} |
mset-of-formula FT = {}

```

0.6.2 Equisatisfiability of the two Version

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

```

is-conj-with-TF (FNot  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = \text{FVar } v \vee \varphi = \text{FF} \vee \varphi = \text{FT}$ )
unfolding is-conj-with-TF-def apply (rule iffI)
apply (induction FNot  $\varphi$  rule: super-grouped-by.induct)

```

```

apply (induction FNot  $\varphi$  rule: grouped-by.induct)
  apply simp
  apply (cases  $\varphi$ ; simp)
apply auto
done

```

```

lemma grouped-by-COr-FNot:
  grouped-by COr (FNot  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = FVar\ v \vee \varphi = FF \vee \varphi = FT$ )
  unfolding is-conj-with-TF-def apply (rule iffI)
  apply (induction FNot  $\varphi$  rule: grouped-by.induct)
    apply simp
    apply (cases  $\varphi$ ; simp)
  apply auto
done

```

```

lemma
  shows no-T-F-FF[simp]:  $\neg$ no-T-F FF and
    no-T-F-FT[simp]:  $\neg$ no-T-F FT
  unfolding no-T-F-def all-subformula-st-def by auto

```

```

lemma grouped-by-CAnd-FAnd:
  grouped-by CAnd (FAnd  $\varphi_1$   $\varphi_2$ )  $\longleftrightarrow$  grouped-by CAnd  $\varphi_1 \wedge$  grouped-by CAnd  $\varphi_2$ 
  apply (rule iffI)
  apply (induction FAnd  $\varphi_1$   $\varphi_2$  rule: grouped-by.induct)
  using connected-is-group[of CAnd  $\varphi_1$   $\varphi_2$ ] by auto

```

```

lemma grouped-by-COr-FOr:
  grouped-by COr (FOr  $\varphi_1$   $\varphi_2$ )  $\longleftrightarrow$  grouped-by COr  $\varphi_1 \wedge$  grouped-by COr  $\varphi_2$ 
  apply (rule iffI)
  apply (induction FOr  $\varphi_1$   $\varphi_2$  rule: grouped-by.induct)
  using connected-is-group[of COr  $\varphi_1$   $\varphi_2$ ] by auto

```

```

lemma grouped-by-COr-FAnd[simp]:  $\neg$  grouped-by COr (FAnd  $\varphi_1$   $\varphi_2$ )
  apply clarify
  apply (induction FAnd  $\varphi_1$   $\varphi_2$  rule: grouped-by.induct)
  apply auto
done

```

```

lemma grouped-by-COr-FEq[simp]:  $\neg$  grouped-by COr (FEq  $\varphi_1$   $\varphi_2$ )
  apply clarify
  apply (induction FEq  $\varphi_1$   $\varphi_2$  rule: grouped-by.induct)
  apply auto
done

```

```

lemma [simp]:  $\neg$ grouped-by COr (FImp  $\varphi$   $\psi$ )
  apply clarify
  by (induction FImp  $\varphi$   $\psi$  rule: grouped-by.induct) simp-all

```

```

lemma [simp]:  $\neg$  is-conj-with-TF (FImp  $\varphi$   $\psi$ )
  unfolding is-conj-with-TF-def apply clarify
  by (induction FImp  $\varphi$   $\psi$  rule: super-grouped-by.induct) simp-all

```

```

lemma [simp]:  $\neg$ grouped-by COr (FEq  $\varphi$   $\psi$ )
  apply clarify
  by (induction FEq  $\varphi$   $\psi$  rule: grouped-by.induct) simp-all

```

```

lemma [simp]:  $\neg$  is-conj-with-TF (FEq  $\varphi$   $\psi$ )
  unfolding is-conj-with-TF-def apply clarify
  by (induction FEq  $\varphi$   $\psi$  rule: super-grouped-by.induct) simp-all

```

```

lemma is-conj-with-TF-Fand:
  is-conj-with-TF (FAnd  $\varphi$ 1  $\varphi$ 2)  $\implies$  is-conj-with-TF  $\varphi$ 1  $\wedge$  is-conj-with-TF  $\varphi$ 2
  unfolding is-conj-with-TF-def
  apply (induction FAnd  $\varphi$ 1  $\varphi$ 2 rule: super-grouped-by.induct)
  apply (auto simp: grouped-by-CAnd-FAnd intro: grouped-is-super-grouped)[]
  apply auto[]
  done

```

```

lemma is-conj-with-TF-FOr:
  is-conj-with-TF (FOr  $\varphi$ 1  $\varphi$ 2)  $\implies$  grouped-by COr  $\varphi$ 1  $\wedge$  grouped-by COr  $\varphi$ 2
  unfolding is-conj-with-TF-def
  apply (induction FOr  $\varphi$ 1  $\varphi$ 2 rule: super-grouped-by.induct)
  apply (auto simp: grouped-by-COr-FOr)[]
  apply auto[]
  done

```

```

lemma grouped-by-COr-mset-of-formula:
  grouped-by COr  $\varphi \implies$  mset-of-formula  $\varphi =$  (if  $\varphi = FT$  then {} else {mset-of-conj  $\varphi$ })
  by (induction  $\varphi$ ) (auto simp add: grouped-by-COr-FNot)

```

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: $op \models$ uses a function assigning *True* or *False*, while $op \models s$ uses a set where being in the list means entailment of a literal.

```

theorem
  fixes  $\varphi :: 'v$  propo
  assumes is-cnf  $\varphi$ 
  shows eval A  $\varphi \longleftrightarrow$  Partial-Clausal-Logic.true-cls ( $\{Pos\ v|v. A\ v\} \cup \{Neg\ v|v. \neg A\ v\}$ )
    (mset-of-formula  $\varphi$ )
  using assms
proof (induction  $\varphi$ )
  case FF
  then show ?case by auto
next
  case FT
  then show ?case by auto
next
  case (FVar  $v$ )
  then show ?case by auto
next
  case (FAnd  $\varphi$   $\psi$ )
  then show ?case
  unfolding is-cnf-def by (auto simp: is-conj-with-TF-FNot dest: is-conj-with-TF-Fand
    dest!:is-conj-with-TF-FOr)
next
  case (FOr  $\varphi$   $\psi$ )
  then have [simp]: mset-of-formula  $\varphi = \{mset-of-conj\ \varphi\}$  mset-of-formula  $\psi = \{mset-of-conj\ \psi\}$ 
    unfolding is-cnf-def by (auto dest!:is-conj-with-TF-FOr simp: grouped-by-COr-mset-of-formula
      split: if-splits)
  have is-conj-with-TF  $\varphi$  is-conj-with-TF  $\psi$ 

```

```

    using FOr(3) unfolding is-cnf-def no-T-F-def
    by (metis grouped-is-super-grouped is-conj-with-TF-FOr is-conj-with-TF-def)+
then show ?case using FOr
    unfolding is-cnf-def by simp
next
case (FImp  $\varphi$   $\psi$ )
then show ?case
    unfolding is-cnf-def by auto
next
case (FEq  $\varphi$   $\psi$ )
then show ?case
    unfolding is-cnf-def by auto
next
case (FNot  $\varphi$ )
then show ?case
    unfolding is-cnf-def by (auto simp: is-conj-with-TF-FNot)
qed
end

```