

# Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019



# Contents

<b>1</b>	<b>Normalisation</b>	<b>5</b>
1.1	Logics . . . . .	5
1.1.1	Definition and Abstraction . . . . .	5
1.1.2	Properties of the Abstraction . . . . .	6
1.1.3	Subformulas and Properties . . . . .	9
1.1.4	Positions . . . . .	12
1.2	Semantics over the Syntax . . . . .	15
1.3	Rewrite Systems and Properties . . . . .	16
1.3.1	Lifting of Rewrite Rules . . . . .	16
1.3.2	Consistency Preservation . . . . .	19
1.3.3	Full Lifting . . . . .	20
1.4	Transformation testing . . . . .	20
1.4.1	Definition and first Properties . . . . .	20
1.4.2	Invariant conservation . . . . .	23
1.5	Rewrite Rules . . . . .	26
1.5.1	Elimination of the Equivalences . . . . .	26
1.5.2	Eliminate Implication . . . . .	28
1.5.3	Eliminate all the True and False in the formula . . . . .	29
1.5.4	PushNeg . . . . .	35
1.5.5	Push Inside . . . . .	40
1.6	The Full Transformations . . . . .	54
1.6.1	Abstract Definition . . . . .	54
1.6.2	Conjunctive Normal Form . . . . .	56
1.6.3	Disjunctive Normal Form . . . . .	57
1.7	More aggressive simplifications: Removing true and false at the beginning . . . . .	58
1.7.1	Transformation . . . . .	58
1.7.2	More invariants . . . . .	60
1.7.3	The new CNF and DNF transformation . . . . .	64
1.8	Link with Multiset Version . . . . .	65
1.8.1	Transformation to Multiset . . . . .	65
1.8.2	Equisatisfiability of the two Versions . . . . .	65
<b>2</b>	<b>Resolution-based techniques</b>	<b>73</b>
2.1	Resolution . . . . .	73
2.1.1	Simplification Rules . . . . .	73
2.1.2	Unconstrained Resolution . . . . .	75
2.1.3	Inference Rule . . . . .	75
2.1.4	Lemma about the Simplified State . . . . .	90
2.1.5	Resolution and Invariants . . . . .	93

2.2	Superposition . . . . .	113
2.2.1	We can now define the rules of the calculus . . . . .	120
<b>theory</b>	<i>Prop-Logic</i>	
<b>imports</b>	<i>Main</i>	
<b>begin</b>		

# Chapter 1

## Normalisation

We define here the normalisation from formula towards conjunctive and disjunctive normal form, including normalisation towards multiset of multisets to represent CNF.

### 1.1 Logics

In this section we define the syntax of the formula and an abstraction over it to have simpler proofs. After that we define some properties like subformula and rewriting.

#### 1.1.1 Definition and Abstraction

The propositional logic is defined inductively. The type parameter is the type of the variables.

**datatype** *'v propo* =  
 *FT* | *FF* | *FVar 'v* | *FNot 'v propo* | *FAnd 'v propo 'v propo* | *FOR 'v propo 'v propo*  
 | *FImp 'v propo 'v propo* | *FEq 'v propo 'v propo*

We do not define any notation for the formula, to distinguish properly between the formulas and Isabelle's logic.

To ease the proofs, we will write the the formula on a homogeneous manner, namely a connecting argument and a list of arguments.

**datatype** *'v connective* = *CT* | *CF* | *CVar 'v* | *CNot* | *CAnd* | *COr* | *CImp* | *CEq*

**abbreviation** *nullary-connective*  $\equiv$   $\{CF\} \cup \{CT\} \cup \{CVar\ x \mid x. True\}$

**definition** *binary-connectives*  $\equiv$   $\{CAnd, COr, CImp, CEq\}$

We define our own induction principal: instead of distinguishing every constructor, we group them by arity.

**lemma** *propo-induct-arity*[*case-names nullary unary binary*]:

**fixes**  $\varphi\ \psi :: 'v\ propo$   
 **assumes** *nullary*:  $\bigwedge \varphi\ x. \varphi = FF \vee \varphi = FT \vee \varphi = FVar\ x \implies P\ \varphi$   
 **and** *unary*:  $\bigwedge \psi. P\ \psi \implies P\ (FNot\ \psi)$   
 **and** *binary*:  $\bigwedge \varphi\ \psi1\ \psi2. P\ \psi1 \implies P\ \psi2 \implies \varphi = FAnd\ \psi1\ \psi2 \vee \varphi = FOR\ \psi1\ \psi2 \vee \varphi = FImp\ \psi1$   
  $\psi2$   
  $\vee \varphi = FEq\ \psi1\ \psi2 \implies P\ \varphi$   
 **shows**  $P\ \psi$   
 **apply** (*induct rule: propo.induct*)  
 **using** *assms* **by** *metis+*

The function *conn* is the interpretation of our representation (connective and list of arguments). We define any thing that has no sense to be false

```
fun conn :: 'v connective  $\Rightarrow$  'v propo list  $\Rightarrow$  'v propo where
conn CT [] = FT |
conn CF [] = FF |
conn (CVar v) [] = FVar v |
conn CNot [ $\varphi$ ] = FNot  $\varphi$  |
conn CAnd ( $\varphi$  # [ $\psi$ ]) = FAnd  $\varphi$   $\psi$  |
conn COr ( $\varphi$  # [ $\psi$ ]) = FOr  $\varphi$   $\psi$  |
conn CImp ( $\varphi$  # [ $\psi$ ]) = FImp  $\varphi$   $\psi$  |
conn CEq ( $\varphi$  # [ $\psi$ ]) = FEq  $\varphi$   $\psi$  |
conn - - = FF
```

We will often use case distinction, based on the arity of the '*v connective*, thus we define our own splitting principle.

```
lemma connective-cases-arity[case-names nullary binary unary]:
assumes nullary:  $\bigwedge x. c = CT \vee c = CF \vee c = CVar x \implies P$ 
and binary:  $c \in \text{binary-connectives} \implies P$ 
and unary:  $c = CNot \implies P$ 
shows P
using assms by (cases c) (auto simp: binary-connectives-def)
```

```
lemma connective-cases-arity-2[case-names nullary unary binary]:
assumes nullary:  $c \in \text{nullary-connective} \implies P$ 
and unary:  $c = CNot \implies P$ 
and binary:  $c \in \text{binary-connectives} \implies P$ 
shows P
using assms by (cases c, auto simp add: binary-connectives-def)
```

Our previous definition is not necessary correct (connective and list of arguments), so we define an inductive predicate.

```
inductive wf-conn :: 'v connective  $\Rightarrow$  'v propo list  $\Rightarrow$  bool for c :: 'v connective where
wf-conn-nullary[simp]:  $(c = CT \vee c = CF \vee c = CVar v) \implies \text{wf-conn } c []$  |
wf-conn-unary[simp]:  $c = CNot \implies \text{wf-conn } c [\psi]$  |
wf-conn-binary[simp]:  $c \in \text{binary-connectives} \implies \text{wf-conn } c (\psi \# \psi' \# [])$ 
thm wf-conn.induct
```

```
lemma wf-conn-induct[consumes 1, case-names CT CF CVar CNot COr CAnd CImp CEq]:
assumes wf-conn c x and
 $\bigwedge v. c = CT \implies P []$  and
 $\bigwedge v. c = CF \implies P []$  and
 $\bigwedge v. c = CVar v \implies P []$  and
 $\bigwedge \psi. c = CNot \implies P [\psi]$  and
 $\bigwedge \psi \psi'. c = COr \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CAnd \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CImp \implies P [\psi, \psi']$  and
 $\bigwedge \psi \psi'. c = CEq \implies P [\psi, \psi']$ 
shows P x
using assms by induction (auto simp: binary-connectives-def)
```

### 1.1.2 Properties of the Abstraction

First we can define simplification rules.

```
lemma wf-conn-conn[simp]:
```

```

wf-conn CT l  $\implies$  conn CT l = FT
wf-conn CF l  $\implies$  conn CF l = FF
wf-conn (CVar x) l  $\implies$  conn (CVar x) l = FVar x
apply (simp-all add: wf-conn.simps)
unfolding binary-connectives-def by simp-all

```

```

lemma wf-conn-list-decomp[simp]:
  wf-conn CT l  $\longleftrightarrow$  l = []
  wf-conn CF l  $\longleftrightarrow$  l = []
  wf-conn (CVar x) l  $\longleftrightarrow$  l = []
  wf-conn CNot ( $\xi$  @  $\varphi$  #  $\xi'$ )  $\longleftrightarrow$   $\xi$  = []  $\wedge$   $\xi'$  = []
apply (simp-all add: wf-conn.simps)
unfolding binary-connectives-def apply simp-all
by (metis append-Nil append-is-Nil-conv list.distinct(1) list.sel(3) tl-append2)

```

```

lemma wf-conn-list:
  wf-conn c l  $\implies$  conn c l = FT  $\longleftrightarrow$  (c = CT  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FF  $\longleftrightarrow$  (c = CF  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FVar x  $\longleftrightarrow$  (c = CVar x  $\wedge$  l = [])
  wf-conn c l  $\implies$  conn c l = FAnd a b  $\longleftrightarrow$  (c = CAnd  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FOr a b  $\longleftrightarrow$  (c = COr  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FEq a b  $\longleftrightarrow$  (c = CEq  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FImp a b  $\longleftrightarrow$  (c = CImp  $\wedge$  l = a # b # [])
  wf-conn c l  $\implies$  conn c l = FNot a  $\longleftrightarrow$  (c = CNot  $\wedge$  l = a # [])
apply (induct l rule: wf-conn.induct)
unfolding binary-connectives-def by auto

```

In the binary connective cases, we will often decompose the list of arguments (of length 2) into two elements.

```

lemma list-length2-decomp: length l = 2  $\implies$  ( $\exists$  a b. l = a # b # [])
apply (induct l, auto)
by (rename-tac l, case-tac l, auto)

```

wf-conn for binary operators means that there are two arguments.

```

lemma wf-conn-bin-list-length:
  fixes l :: 'v propo list
  assumes conn: c  $\in$  binary-connectives
  shows length l = 2  $\longleftrightarrow$  wf-conn c l
proof
  assume length l = 2
  then show wf-conn c l using wf-conn-binary list-length2-decomp using conn by metis
next
  assume wf-conn c l
  then show length l = 2 (is ?P l)
  proof (cases rule: wf-conn.induct)
  case wf-conn-nullary
  then show ?P [] using conn binary-connectives-def
  using connective.distinct(11) connective.distinct(13) connective.distinct(9) by blast
next
  fix  $\psi$  :: 'v propo
  case wf-conn-unary
  then show ?P [ $\psi$ ] using conn binary-connectives-def
  using connective.distinct by blast

```

```

next
  fix  $\psi \psi' :: 'v \text{ propo}$ 
  show  $?P [\psi, \psi']$  by auto
qed
qed

```

```

lemma wf-conn-not-list-length[iff]:
  fixes  $l :: 'v \text{ propo list}$ 
  shows  $\text{wf-conn } CNot\ l \longleftrightarrow \text{length } l = 1$ 
  apply auto
  apply (metis append-Nil connective.distinct(5,17,27) length-Cons list.size(3) wf-conn.simps
    wf-conn-list-decomp(4))
  by (simp add: length-Suc-conv wf-conn.simps)

```

Decomposing the Not into an element is moreover very useful.

```

lemma wf-conn-Not-decomp:
  fixes  $l :: 'v \text{ propo list}$  and  $a :: 'v$ 
  assumes  $\text{corr}: \text{wf-conn } CNot\ l$ 
  shows  $\exists a. l = [a]$ 
  by (metis (no-types, lifting) One-nat-def Suc-length-conv corr length-0-conv
    wf-conn-not-list-length)

```

The *wf-conn* remains correct if the length of list does not change. This lemma is very useful when we do one rewriting step

```

lemma wf-conn-no-arity-change:
   $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l \longleftrightarrow \text{wf-conn } c\ l'$ 
proof -
  {
    fix  $l\ l'$ 
    have  $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l \implies \text{wf-conn } c\ l'$ 
      apply (cases  $c\ l$  rule: wf-conn.induct, auto)
      by (metis wf-conn-bin-list-length)
  }
  then show  $\text{length } l = \text{length } l' \implies \text{wf-conn } c\ l = \text{wf-conn } c\ l'$  by metis
qed

```

```

lemma wf-conn-no-arity-change-helper:
   $\text{length } (\xi @ \varphi \# \xi') = \text{length } (\xi @ \varphi' \# \xi')$ 
  by auto

```

The injectivity of *conn* is useful to prove equality of the connectives and the lists.

```

lemma conn-inj-not:
  assumes  $\text{correct}: \text{wf-conn } c\ l$ 
  and  $\text{conn}: \text{conn } c\ l = FNot\ \psi$ 
  shows  $c = CNot$  and  $l = [\psi]$ 
  apply (cases  $c\ l$  rule: wf-conn.cases)
  using correct conn unfolding binary-connectives-def apply auto
  apply (cases  $c\ l$  rule: wf-conn.cases)
  using correct conn unfolding binary-connectives-def by auto

```

```

lemma conn-inj:
  fixes  $c\ ca :: 'v \text{ connective}$  and  $l\ \psi s :: 'v \text{ propo list}$ 
  assumes  $\text{corr}: \text{wf-conn } ca\ l$ 
  and  $\text{corr}': \text{wf-conn } c\ \psi s$ 

```



```

and eq: conn ca l = conn c  $\psi$ s
shows ca = c  $\wedge$   $\psi$ s = l
using corr
proof (cases ca l rule: wf-conn.cases)
case (wf-conn-nullary v)
then show ca = c  $\wedge$   $\psi$ s = l using assms
by (metis conn.simps(1) conn.simps(2) conn.simps(3) wf-conn-list(1-3))
next
case (wf-conn-unary  $\psi'$ )
then have *: FNot  $\psi'$  = conn c  $\psi$ s using conn-inj-not eq assms by auto
then have c = ca by (metis conn-inj-not(1) corr' wf-conn-unary(2))
moreover have  $\psi$ s = l using * conn-inj-not(2) corr' wf-conn-unary(1) by force
ultimately show ca = c  $\wedge$   $\psi$ s = l by auto
next
case (wf-conn-binary  $\psi'$   $\psi''$ )
then show ca = c  $\wedge$   $\psi$ s = l
using eq corr' unfolding binary-connectives-def apply (cases ca, auto simp add: wf-conn-list)
using wf-conn-list(4-7) corr' by metis+
qed

```

### 1.1.3 Subformulas and Properties

A characterization using sub-formulas is interesting for rewriting: we will define our relation on the sub-term level, and then lift the rewriting on the term-level. So the rewriting takes place on a subformula.

**inductive** *subformula* :: '*v* *propo*  $\Rightarrow$  '*v* *propo*  $\Rightarrow$  bool (infix  $\preceq$  45) **for**  $\varphi$  **where**  
*subformula-refl*[simp]:  $\varphi \preceq \varphi$  |  
*subformula-into-subformula*:  $\psi \in \text{set } l \Rightarrow \text{wf-conn } c \ l \Rightarrow \varphi \preceq \psi \Rightarrow \varphi \preceq \text{conn } c \ l$

On the *subformula-into-subformula*, we can see why we use our *conn* representation: one case is enough to express the subformulas property instead of listing all the cases.

This is an example of a property related to subformulas.

```

lemma subformula-in-subformula-not:
shows b: FNot  $\varphi \preceq \psi \Rightarrow \varphi \preceq \psi$ 
apply (induct rule: subformula.induct)
using subformula-into-subformula wf-conn-unary subformula-refl list.set-intros(1) subformula-refl
by (fastforce intro: subformula-into-subformula)+

```

```

lemma subformula-in-binary-conn:
assumes conn:  $c \in \text{binary-connectives}$ 
shows  $f \preceq \text{conn } c \ [f, g]$ 
and  $g \preceq \text{conn } c \ [f, g]$ 
proof -
have a: wf-conn c (f# [g]) using conn wf-conn-binary binary-connectives-def by auto
moreover have b:  $f \preceq f$  using subformula-refl by auto
ultimately show  $f \preceq \text{conn } c \ [f, g]$ 
by (metis append-Nil in-set-conv-decomp subformula-into-subformula)
next
have a: wf-conn c ([f] @ [g]) using conn wf-conn-binary binary-connectives-def by auto
moreover have b:  $g \preceq g$  using subformula-refl by auto
ultimately show  $g \preceq \text{conn } c \ [f, g]$  using subformula-into-subformula by force
qed

```

**lemma** *subformula-trans*:

$\psi \preceq \psi' \implies \varphi \preceq \psi \implies \varphi \preceq \psi'$   
**apply** (induct  $\psi'$  rule: subformula.inducts)  
**by** (auto simp: subformula-into-subformula)

**lemma** subformula-leaf:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**assumes** incl:  $\varphi \preceq \psi$   
**and** simple:  $\psi = FT \vee \psi = FF \vee \psi = FVar x$   
**shows**  $\varphi = \psi$   
**using** incl simple  
**by** (induct rule: subformula.induct, auto simp: wf-conn-list)

**lemma** subformula-not-incl-eq:  
**assumes**  $\varphi \preceq \text{conn } c \ l$   
**and** wf-conn  $c \ l$   
**and**  $\forall \psi. \psi \in \text{set } l \longrightarrow \neg \varphi \preceq \psi$   
**shows**  $\varphi = \text{conn } c \ l$   
**using** assms **apply** (induction conn  $c \ l$  rule: subformula.induct, auto)  
**using** conn-inj **by** blast

**lemma** wf-subformula-conn-cases:  
 $\text{wf-conn } c \ l \implies \varphi \preceq \text{conn } c \ l \longleftrightarrow (\varphi = \text{conn } c \ l \vee (\exists \psi. \psi \in \text{set } l \wedge \varphi \preceq \psi))$   
**apply** standard  
**using** subformula-not-incl-eq **apply** metis  
**by** (auto simp add: subformula-into-subformula)

**lemma** subformula-decomp-explicit[simp]:  
 $\varphi \preceq FAnd \ \psi \ \psi' \longleftrightarrow (\varphi = FAnd \ \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$  (is ?P FAnd)  
 $\varphi \preceq FOr \ \psi \ \psi' \longleftrightarrow (\varphi = FOr \ \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$   
 $\varphi \preceq FEq \ \psi \ \psi' \longleftrightarrow (\varphi = FEq \ \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$   
 $\varphi \preceq FImp \ \psi \ \psi' \longleftrightarrow (\varphi = FImp \ \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$

**proof** –

**have** wf-conn CAnd  $[\psi, \psi']$  **by** (simp add: binary-connectives-def)  
**then have**  $\varphi \preceq \text{conn } CAnd \ [\psi, \psi'] \longleftrightarrow$   
 $(\varphi = \text{conn } CAnd \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$   
**using** wf-subformula-conn-cases **by** metis  
**then show** ?P FAnd **by** auto

**next**

**have** wf-conn COr  $[\psi, \psi']$  **by** (simp add: binary-connectives-def)  
**then have**  $\varphi \preceq \text{conn } COr \ [\psi, \psi'] \longleftrightarrow$   
 $(\varphi = \text{conn } COr \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$   
**using** wf-subformula-conn-cases **by** metis  
**then show** ?P FOr **by** auto

**next**

**have** wf-conn CEq  $[\psi, \psi']$  **by** (simp add: binary-connectives-def)  
**then have**  $\varphi \preceq \text{conn } CEq \ [\psi, \psi'] \longleftrightarrow$   
 $(\varphi = \text{conn } CEq \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$   
**using** wf-subformula-conn-cases **by** metis  
**then show** ?P FEq **by** auto

**next**

**have** wf-conn CImp  $[\psi, \psi']$  **by** (simp add: binary-connectives-def)  
**then have**  $\varphi \preceq \text{conn } CImp \ [\psi, \psi'] \longleftrightarrow$   
 $(\varphi = \text{conn } CImp \ [\psi, \psi'] \vee (\exists \psi''. \psi'' \in \text{set } [\psi, \psi'] \wedge \varphi \preceq \psi''))$   
**using** wf-subformula-conn-cases **by** metis  
**then show** ?P FImp **by** auto

**qed**

**lemma** *wf-conn-helper-facts*[*iff*]:

*wf-conn* *CNot* [ $\varphi$ ]  
*wf-conn* *CT* []  
*wf-conn* *CF* []  
*wf-conn* (*CVar*  $x$ ) []  
*wf-conn* *CAnd* [ $\varphi$ ,  $\psi$ ]  
*wf-conn* *COr* [ $\varphi$ ,  $\psi$ ]  
*wf-conn* *CImp* [ $\varphi$ ,  $\psi$ ]  
*wf-conn* *CEq* [ $\varphi$ ,  $\psi$ ]  
**using** *wf-conn.intros* **unfolding** *binary-connectives-def* **by** *fastforce+*

**lemma** *exists-c-conn*:  $\exists c l. \varphi = \text{conn } c l \wedge \text{wf-conn } c l$

**by** (*cases*  $\varphi$ ) *force+*

**lemma** *subformula-conn-decomp*[*simp*]:

**assumes** *wf*: *wf-conn*  $c l$

**shows**  $\varphi \preceq \text{conn } c l \longleftrightarrow (\varphi = \text{conn } c l \vee (\exists \psi \in \text{set } l. \varphi \preceq \psi))$  (**is**  $?A \longleftrightarrow ?B$ )

**proof** (*rule iffI*)

**{**  
**fix**  $\xi$   
**have**  $\varphi \preceq \xi \implies \xi = \text{conn } c l \implies \text{wf-conn } c l \implies \forall x::'a \text{ propo} \in \text{set } l. \neg \varphi \preceq x \implies \varphi = \text{conn } c l$   
**apply** (*induct rule: subformula.induct*)  
**apply** *simp*  
**using** *conn-inj* **by** *blast*  
**}**  
**moreover assume**  $?A$   
**ultimately show**  $?B$  **using** *wf* **by** *metis*

**next**

**assume**  $?B$

**then show**  $\varphi \preceq \text{conn } c l$  **using** *wf* *wf-subformula-conn-cases* **by** *blast*

**qed**

**lemma** *subformula-leaf-explicit*[*simp*]:

$\varphi \preceq FT \longleftrightarrow \varphi = FT$   
 $\varphi \preceq FF \longleftrightarrow \varphi = FF$   
 $\varphi \preceq FVar x \longleftrightarrow \varphi = FVar x$   
**apply** *auto*  
**using** *subformula-leaf* **by** *metis* +

The variables inside the formula gives precisely the variables that are needed for the formula.

**primrec** *vars-of-prop*::  $'v \text{ propo} \Rightarrow 'v \text{ set}$  **where**

*vars-of-prop* *FT* = {} |  
*vars-of-prop* *FF* = {} |  
*vars-of-prop* (*FVar*  $x$ ) = { $x$ } |  
*vars-of-prop* (*FNot*  $\varphi$ ) = *vars-of-prop*  $\varphi$  |  
*vars-of-prop* (*FAnd*  $\varphi \psi$ ) = *vars-of-prop*  $\varphi \cup \text{vars-of-prop } \psi$  |  
*vars-of-prop* (*FOr*  $\varphi \psi$ ) = *vars-of-prop*  $\varphi \cup \text{vars-of-prop } \psi$  |  
*vars-of-prop* (*FImp*  $\varphi \psi$ ) = *vars-of-prop*  $\varphi \cup \text{vars-of-prop } \psi$  |  
*vars-of-prop* (*FEq*  $\varphi \psi$ ) = *vars-of-prop*  $\varphi \cup \text{vars-of-prop } \psi$

**lemma** *vars-of-prop-incl-conn*:

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

**assumes** *corr*: *wf-conn*  $c l$  **and** *incl*:  $\psi \in \text{set } l$

**shows** *vars-of-prop*  $\psi \subseteq \text{vars-of-prop } (\text{conn } c l)$

**proof** (*cases c rule: connective-cases-arity-2*)

```

case nullary
then have False using corr incl by auto
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop (conn } c \text{ } l)$  by blast
next
case binary note c = this
then obtain a b where ab:  $l = [a, b]$ 
  using wf-conn-bin-list-length list-length2-decomp corr by metis
then have  $\psi = a \vee \psi = b$  using incl by auto
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop (conn } c \text{ } l)$ 
  using ab c unfolding binary-connectives-def by auto
next
case unary note c = this
fix  $\varphi :: 'v \text{ propo}$ 
have  $l = [\psi]$  using corr c incl split-list by force
then show vars-of-prop  $\psi \subseteq \text{vars-of-prop (conn } c \text{ } l)$  using c by auto
qed

```

The set of variables is compatible with the subformula order.

**lemma** *subformula-vars-of-prop*:

```

 $\varphi \preceq \psi \implies \text{vars-of-prop } \varphi \subseteq \text{vars-of-prop } \psi$ 
apply (induct rule: subformula.induct)
apply simp
using vars-of-prop-incl-conn by blast

```

#### 1.1.4 Positions

Instead of 1 or 2 we use  $L$  or  $R$

**datatype** *sign* =  $L \mid R$

We use *nil* instead of  $\varepsilon$ .

**fun** *pos* ::  $'v \text{ propo} \Rightarrow \text{sign list set}$  **where**

```

pos FF =  $\{\emptyset\} \mid$ 
pos FT =  $\{\emptyset\} \mid$ 
pos (FVar x) =  $\{\emptyset\} \mid$ 
pos (FAnd  $\varphi \psi$ ) =  $\{\emptyset\} \cup \{L \# p \mid p. p \in \text{pos } \varphi\} \cup \{R \# p \mid p. p \in \text{pos } \psi\} \mid$ 
pos (FOr  $\varphi \psi$ ) =  $\{\emptyset\} \cup \{L \# p \mid p. p \in \text{pos } \varphi\} \cup \{R \# p \mid p. p \in \text{pos } \psi\} \mid$ 
pos (FEq  $\varphi \psi$ ) =  $\{\emptyset\} \cup \{L \# p \mid p. p \in \text{pos } \varphi\} \cup \{R \# p \mid p. p \in \text{pos } \psi\} \mid$ 
pos (FImp  $\varphi \psi$ ) =  $\{\emptyset\} \cup \{L \# p \mid p. p \in \text{pos } \varphi\} \cup \{R \# p \mid p. p \in \text{pos } \psi\} \mid$ 
pos (FNot  $\varphi$ ) =  $\{\emptyset\} \cup \{L \# p \mid p. p \in \text{pos } \varphi\}$ 

```

**lemma** *finite-pos*: *finite* (*pos*  $\varphi$ )

**by** (*induct*  $\varphi$ , *auto*)

**lemma** *finite-inj-comp-set*:

```

fixes s :: 'v set
assumes finite: finite s
and inj: inj f
shows card ( $\{f \text{ } p \mid p. p \in s\}$ ) = card s
using finite

```

**proof** (*induct* s *rule*: *finite-induct*)

**show** *card*  $\{f \text{ } p \mid p. p \in \{\}\} = \text{card } \{\}$  **by** *auto*

**next**

```

fix x :: 'v and s :: 'v set
assume f: finite s and notin:  $x \notin s$ 
and IH: card  $\{f \text{ } p \mid p. p \in s\} = \text{card } s$ 

```

**have**  $f'$ : *finite*  $\{f\ p \mid p. p \in \text{insert } x\ s\}$  **using**  $f$  **by** *auto*  
**have** *notin'*:  $f\ x \notin \{f\ p \mid p. p \in s\}$  **using** *notin inj injD* **by** *fastforce*  
**have**  $\{f\ p \mid p. p \in \text{insert } x\ s\} = \text{insert } (f\ x) \{f\ p \mid p. p \in s\}$  **by** *auto*  
**then have**  $\text{card } \{f\ p \mid p. p \in \text{insert } x\ s\} = 1 + \text{card } \{f\ p \mid p. p \in s\}$   
**using** *finite card-insert-disjoint f' notin'* **by** *auto*  
**moreover have**  $\dots = \text{card } (\text{insert } x\ s)$  **using** *notin f IH* **by** *auto*  
**finally show**  $\text{card } \{f\ p \mid p. p \in \text{insert } x\ s\} = \text{card } (\text{insert } x\ s)$  .  
**qed**

**lemma** *cons-inject*:

*inj ((#) s)*  
**by** (*meson injI list.inject*)

**lemma** *finite-insert-nil-cons*:

*finite s  $\implies$  card (insert [] {L # p | p. p  $\in$  s}) = 1 + card {L # p | p. p  $\in$  s}*  
**using** *card-insert-disjoint* **by** *auto*

**lemma** *cord-not[simp]*:

*card (pos (FNot  $\varphi$ )) = 1 + card (pos  $\varphi$ )*  
**by** (*simp add: cons-inject finite-inj-comp-set finite-pos*)

**lemma** *card-seperate*:

**assumes** *finite s1 and finite s2*  
**shows**  $\text{card } (\{L \# p \mid p. p \in s1\} \cup \{R \# p \mid p. p \in s2\}) = \text{card } (\{L \# p \mid p. p \in s1\})$   
 $+ \text{card } (\{R \# p \mid p. p \in s2\})$  (**is**  $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$ )

**proof** –

**have** *finite ?L* **using** *assms* **by** *auto*  
**moreover have** *finite ?R* **using** *assms* **by** *auto*  
**moreover have**  $?L \cap ?R = \{\}$  **by** *blast*  
**ultimately show** *?thesis* **using** *assms card-Un-disjoint* **by** *blast*

**qed**

**definition** *prop-size* **where** *prop-size  $\varphi = \text{card } (\text{pos } \varphi)$*

**lemma** *prop-size-vars-of-prop*:

**fixes**  $\varphi :: 'v\ \text{propo}$   
**shows**  $\text{card } (\text{vars-of-prop } \varphi) \leq \text{prop-size } \varphi$

**unfolding** *prop-size-def* **apply** (*induct  $\varphi$ , auto simp add: cons-inject finite-inj-comp-set finite-pos*)

**proof** –

**fix**  $\varphi1\ \varphi2 :: 'v\ \text{propo}$   
**assume** *IH1: card (vars-of-prop  $\varphi1$ )  $\leq$  card (pos  $\varphi1$ )*  
**and** *IH2: card (vars-of-prop  $\varphi2$ )  $\leq$  card (pos  $\varphi2$ )*  
**let**  $?L = \{L \# p \mid p. p \in \text{pos } \varphi1\}$   
**let**  $?R = \{R \# p \mid p. p \in \text{pos } \varphi2\}$   
**have**  $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$   
**using** *card-seperate finite-pos* **by** *blast*  
**moreover have**  $\dots = \text{card } (\text{pos } \varphi1) + \text{card } (\text{pos } \varphi2)$   
**by** (*simp add: cons-inject finite-inj-comp-set finite-pos*)  
**moreover have**  $\dots \geq \text{card } (\text{vars-of-prop } \varphi1) + \text{card } (\text{vars-of-prop } \varphi2)$  **using** *IH1 IH2* **by** *arith*  
**then have**  $\dots \geq \text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2)$  **using** *card-Un-le le-trans* **by** *blast*  
**ultimately**  
**show**  $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$   
 $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$   
 $\text{card } (\text{vars-of-prop } \varphi1 \cup \text{vars-of-prop } \varphi2) \leq \text{Suc } (\text{card } (?L \cup ?R))$

```

      card (vars-of-prop  $\varphi 1 \cup$  vars-of-prop  $\varphi 2$ )  $\leq$  Suc (card (?L  $\cup$  ?R))
    by auto
  qed

```

```

value pos (FImp (FAnd (FVar P) (FVar Q)) (FOr (FVar P) (FVar Q)))

```

```

inductive path-to :: sign list  $\Rightarrow$  'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  bool where
  path-to-refl[intro]: path-to []  $\varphi$   $\varphi$  |
  path-to-l:  $c \in$  binary-connectives  $\vee$   $c =$  CNot  $\implies$  wf-conn  $c$  ( $\varphi \# l$ )  $\implies$  path-to  $p$   $\varphi$   $\varphi' \implies$ 
    path-to ( $L \# p$ ) (conn  $c$  ( $\varphi \# l$ ))  $\varphi'$  |
  path-to-r:  $c \in$  binary-connectives  $\implies$  wf-conn  $c$  ( $\psi \# \varphi \# []$ )  $\implies$  path-to  $p$   $\varphi$   $\varphi' \implies$ 
    path-to ( $R \# p$ ) (conn  $c$  ( $\psi \# \varphi \# []$ ))  $\varphi'$ 

```

There is a deep link between subformulas and pathes: a (correct) path leads to a subformula and a subformula is associated to a given path.

**lemma** path-to-subformula:

```

  path-to  $p$   $\varphi$   $\varphi' \implies \varphi' \preceq \varphi$ 
apply (induct rule: path-to.induct)
apply simp
apply (metis list.set-intros(1) subformula-into-subformula)
using subformula-trans subformula-in-binary-conn(2) by metis

```

**lemma** subformula-path-exists:

```

  fixes  $\varphi$   $\varphi'::$  'v propo
  shows  $\varphi' \preceq \varphi \implies \exists p. \text{path-to } p \varphi \varphi'$ 

```

**proof** (induct rule: subformula.induct)

```

  case subformula-refl
  have path-to []  $\varphi'$   $\varphi'$  by auto
  then show  $\exists p. \text{path-to } p \varphi' \varphi'$  by metis

```

**next**

```

  case (subformula-into-subformula  $\psi$   $l$   $c$ )
  note wf = this(2) and IH = this(4) and  $\psi =$  this(1)
  then obtain  $p$  where  $p: \text{path-to } p \psi \varphi'$  by metis

```

```

  {
    fix  $x::$  'v
    assume  $c =$  CT  $\vee$   $c =$  CF  $\vee$   $c =$  CVar  $x$ 
    then have False using subformula-into-subformula by auto
    then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast
  }

```

**moreover** {

```

  assume  $c: c =$  CNot
  then have  $l = [\psi]$  using wf  $\psi$  wf-conn-Not-decomp by fastforce
  then have path-to ( $L \# p$ ) (conn  $c$   $l$ )  $\varphi'$  by (metis  $c$  wf-conn-unary  $p$  path-to- $l$ )
  then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast

```

}

**moreover** {

```

  assume  $c: c \in$  binary-connectives
  obtain  $a$   $b$  where  $ab: [a, b] = l$  using subformula-into-subformula  $c$  wf-conn-bin-list-length
    list-length2-decomp by metis
  then have  $a = \psi \vee b = \psi$  using  $\psi$  by auto
  then have path-to ( $L \# p$ ) (conn  $c$   $l$ )  $\varphi' \vee$  path-to ( $R \# p$ ) (conn  $c$   $l$ )  $\varphi'$  using  $c$  path-to- $l$ 
    path-to-r  $p$   $ab$  by (metis wf-conn-binary)
  then have  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  by blast

```

}

**ultimately show**  $\exists p. \text{path-to } p (\text{conn } c \ l) \varphi'$  **using** connective-cases-arity **by** metis

**qed**

```

fun replace-at :: sign list ⇒ 'v propo ⇒ 'v propo ⇒ 'v propo where
replace-at [] - ψ = ψ |
replace-at (L # l) (FAnd φ φ') ψ = FAnd (replace-at l φ ψ) φ' |
replace-at (R # l) (FAnd φ φ') ψ = FAnd φ (replace-at l φ' ψ) |
replace-at (L # l) (FOr φ φ') ψ = FOr (replace-at l φ ψ) φ' |
replace-at (R # l) (FOr φ φ') ψ = FOr φ (replace-at l φ' ψ) |
replace-at (L # l) (FEq φ φ') ψ = FEq (replace-at l φ ψ) φ' |
replace-at (R # l) (FEq φ φ') ψ = FEq φ (replace-at l φ' ψ) |
replace-at (L # l) (FImp φ φ') ψ = FImp (replace-at l φ ψ) φ' |
replace-at (R # l) (FImp φ φ') ψ = FImp φ (replace-at l φ' ψ) |
replace-at (L # l) (FNot φ) ψ = FNot (replace-at l φ ψ)

```

## 1.2 Semantics over the Syntax

Given the syntax defined above, we define a semantics, by defining an evaluation function *eval*. This function is the bridge between the logic as we define it here and the built-in logic of Isabelle.

```

fun eval :: ('v ⇒ bool) ⇒ 'v propo ⇒ bool (infix | = 50) where
 $\mathcal{A} \models FT = True$  |
 $\mathcal{A} \models FF = False$  |
 $\mathcal{A} \models FVar\ v = (\mathcal{A}\ v)$  |
 $\mathcal{A} \models FNot\ \varphi = (\neg(\mathcal{A} \models \varphi))$  |
 $\mathcal{A} \models FAnd\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \wedge \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FOr\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \vee \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FImp\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longrightarrow \mathcal{A} \models \varphi_2)$  |
 $\mathcal{A} \models FEq\ \varphi_1\ \varphi_2 = (\mathcal{A} \models \varphi_1 \longleftrightarrow \mathcal{A} \models \varphi_2)$ 

```

```

definition evalf (infix | =f 50) where
evalf φ ψ = (∀ A. A ⊨ φ ⟶ A ⊨ ψ)

```

The deduction rule is in the book. And the proof looks like to the one of the book.

**theorem** *deduction-theorem*:

$\varphi \models_f \psi \longleftrightarrow (\forall A. A \models FImp\ \varphi\ \psi)$

**proof**

```

assume H: φ ⊨f ψ
{
  fix A
  have A ⊨ FImp φ ψ
  proof (cases A ⊨ φ)
    case True
    then have A ⊨ ψ using H unfolding evalf-def by metis
    then show A ⊨ FImp φ ψ by auto
  next
    case False
    then show A ⊨ FImp φ ψ by auto
  qed
}
then show ∀ A. A ⊨ FImp φ ψ by blast
next
assume A: ∀ A. A ⊨ FImp φ ψ
show φ ⊨f ψ
proof (rule ccontr)
  assume ¬ φ ⊨f ψ
  then obtain A where A ⊨ φ and ¬ A ⊨ ψ using evalf-def by metis

```

```

    then have  $\neg A \models \text{FImp } \varphi \ \psi$  by auto
    then show False using A by blast
qed

```

A shorter proof:

```

lemma  $\varphi \models_f \psi \iff (\forall A. A \models \text{FImp } \varphi \ \psi)$ 
  by (simp add: evalf-def)

```

```

definition same-over-set:: ( $'v \Rightarrow \text{bool}$ )  $\Rightarrow$  ( $'v \Rightarrow \text{bool}$ )  $\Rightarrow$   $'v \text{ set} \Rightarrow \text{bool}$  where
  same-over-set A B S = ( $\forall c \in S. A \ c = B \ c$ )

```

If two mapping  $A$  and  $B$  have the same value over the variables, then the same formula are satisfiable.

```

lemma same-over-set-eval:
  assumes same-over-set A B (vars-of-prop  $\varphi$ )
  shows  $A \models \varphi \iff B \models \varphi$ 
  using assms unfolding same-over-set-def by (induct  $\varphi$ , auto)

```

```

end
theory Prop-Abstract-Transformation
imports Prop-Logic Weidenbach-Book-Base.Wellfounded-More

```

```

begin

```

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

## 1.3 Rewrite Systems and Properties

### 1.3.1 Lifting of Rewrite Rules

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

```

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

```

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

This lemma is only a health condition:

```

lemma propo-rew-step-subformula-imp:
shows  $\text{propo-rew-step } r \ \varphi \ \varphi' \implies \exists \psi \ \psi'. \ \psi \preceq \varphi \wedge \psi' \preceq \varphi' \wedge r \ \psi \ \psi'$ 
  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  $\psi$  and  $\psi'$ , then every formula  $\varphi$  containing  $\psi$ , can be rewritten into a formula  $\varphi'$ , such that it contains  $\psi'$ .



```

lemma propo-rew-step-subformula-rec:
  fixes  $\psi \ \psi' \ \varphi :: 'v \text{ propo}$ 
  shows  $\psi \preceq \varphi \implies r \ \psi \ \psi' \implies (\exists \varphi'. \ \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \psi \ \varphi')$ 
proof (induct  $\varphi$  rule: subformula.induct)
  case subformula-refl
  then have 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 = this(4) and  $r = \text{this}(5)$  and  $\psi'' = \text{this}(1)$  and  $wf = \text{this}(2)$  and  $\text{incl} = \text{this}(3)$ 
  then obtain  $\varphi'$  where  $\ast: \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \psi'' \ \varphi'$  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 propo-rew-step  $r \ (\text{conn } c \ l) \ (\text{conn } c \ (\xi @ \varphi' \# \xi'))$ 
    using propo-rew-step.intros(2)  $wf$  by metis
  moreover have  $\psi' \preceq \text{conn } c \ (\xi @ \varphi' \# \xi')$ 
    using  $wf \ast wf\text{-conn-no-arity-change}$  Prop-Logic.subformula-into-subformula
    by (metis (no-types) in-set-conv-decomp  $l \ wf\text{-conn-no-arity-change-helper}$ )
  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  $wf: wf\text{-conn } c \ l$  and  $wf': wf\text{-conn } c \ l'$ 
  and same:  $\forall n. \ A \models l \ ! \ n \longleftrightarrow (A \models l' \ ! \ n)$ 
  shows  $A \models \text{conn } c \ l \longleftrightarrow A \models \text{conn } c \ l'$ 
proof (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  $wf \ wf'$  by auto
next
  case unary note  $c = \text{this}$ 
  then obtain  $a$  where  $l: l = [a]$  using wf-conn-Not-decomp  $wf$  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  $p: A \models a \longleftrightarrow A \models a' \wedge A \models b \longleftrightarrow A \models b'$ 
    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  $\psi$  inside  $\varphi$  (ie at a path  $p$ ) into  $\psi'$ .

```

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 rel = this(1) and IH0 = this(2) and corr = this(3)
  obtain  $\psi \ \psi' \ p$  where IH:  $r \ \psi \ \psi' \wedge \text{path-to } p \ \varphi \ \psi \wedge \text{replace-at } p \ \varphi \ \psi' = \varphi'$  using 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 empty IH 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 empty 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 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-tac  $\xi$ , case-tac  $\xi'$ , auto)
  {
    assume  $\varphi: \xi = [] \wedge \xi' = [b]$ 
    have path-to  $(L \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi$ 
      using  $\varphi \ c \ IH \ ab \ corr$  by (simp add: path-to-l)
    moreover have replace-at  $(L \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi' = \text{conn } c \ (\xi @ (\varphi' \# \xi'))$ 
      using  $c \ IH \ ab \ \varphi$  unfolding binary-connectives-def 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 IH by metis
  }
}
moreover {
  assume  $\varphi: \xi = [a] \ \xi' = []$ 
  then have path-to  $(R \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi$ 
    using  $c \ IH \ corr \text{path-to-r corr } \varphi$  by (simp add: path-to-r)
  moreover have replace-at  $(R \# p) \ (\text{conn } c \ (\xi @ (\varphi \# \xi'))) \ \psi' = \text{conn } c \ (\xi @ (\varphi' \# \xi'))$ 
    using  $c \ IH \ ab \ \varphi$  unfolding binary-connectives-def by auto
  ultimately have ?case using IH by metis
}
}

```

```

    ultimately have ?case using ab by blast
  }
  ultimately show ?case using connective-cases-arity by blast
qed

```

### 1.3.2 Consistency Preservation

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

**definition** *preserve-models* **where**

*preserve-models*  $r \longleftrightarrow (\forall \varphi \psi. r \varphi \psi \longrightarrow (\forall A. A \models \varphi \longleftrightarrow A \models \psi))$

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

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

**unfolding** *preserve-models-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**  $\text{rel} = \text{this}(1)$  **and**  $\text{wf} = \text{this}(2)$

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

{

**fix**  $A$

**from**  $\text{IH}$  **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-neg*)

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

**by** (*meson* *consistency-decompose-into-list* *wf* *wf-conn-no-arity-change-helper* *wf-conn-no-arity-change*)

}

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

**qed**

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

**assumes** *preserve-models*  $r$

**shows** *preserve-models* (*propo-rew-step*  $r$ )

**using** *assms* **by** (*simp* *add*: *preserve-models-def* *propo-rew-step-preservers-val-explicit*)

**lemma** *preserve-models-OO[intro]*:

*preserve-models*  $f \implies \text{preserve-models } g \implies \text{preserve-models } (f \text{ OO } g)$

**unfolding** *preserve-models-def* **by** *auto*

**lemma** *star-consistency-preservation-explicit*:

**assumes**  $(\text{propo-rew-step } r)^{**} \varphi \psi$  **and** *preserve-models*  $r$

**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*:

*preserve-models*  $r \implies \text{preserve-models } (\text{propo-rew-step } r)^{**}$

**by** (*simp* *add*: *star-consistency-preservation-explicit* *preserve-models-def*)

### 1.3.3 Full Lifting

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

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

## 1.4 Transformation testing

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

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

**lemma** *helper-fact*:  $c \in \text{binary-connectives} \longleftrightarrow (c = COr \vee c = CAnd \vee c = CEq \vee c = CImp)$   
**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* (*FAnd*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FAnd \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$   
**and** *all-subformula-st test-symb* (*FOr*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FOr \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$   
**and** *all-subformula-st test-symb* (*FNot*  $\varphi$ )  
 $\longleftrightarrow (\text{test-symb } (FNot \varphi) \wedge \text{all-subformula-st test-symb } \varphi)$   
**and** *all-subformula-st test-symb* (*FEq*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FEq \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$   
**and** *all-subformula-st test-symb* (*FImp*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FImp \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$

**proof** –  
**have** *all-subformula-st test-symb* (*FAnd*  $\varphi \psi$ )  $\longleftrightarrow$  *all-subformula-st test-symb* (*conn CAnd*  $[\varphi, \psi]$ )  
**by** *auto*  
**moreover have**  $\dots \longleftrightarrow \text{test-symb } (\text{conn } CAnd [\varphi, \psi]) \wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi)$   
**using** *all-subformula-st-decomp wf-conn-helper-facts(5)* **by** *metis*  
**finally show** *all-subformula-st test-symb* (*FAnd*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FAnd \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$   
**by** *simp*

**have** *all-subformula-st test-symb* (*FOr*  $\varphi \psi$ )  $\longleftrightarrow$  *all-subformula-st test-symb* (*conn COr*  $[\varphi, \psi]$ )  
**by** *auto*  
**moreover have**  $\dots \longleftrightarrow$   
 $(\text{test-symb } (\text{conn } COr [\varphi, \psi]) \wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st test-symb } \xi))$   
**using** *all-subformula-st-decomp wf-conn-helper-facts(6)* **by** *metis*  
**finally show** *all-subformula-st test-symb* (*FOr*  $\varphi \psi$ )  
 $\longleftrightarrow (\text{test-symb } (FOr \varphi \psi) \wedge \text{all-subformula-st test-symb } \varphi \wedge \text{all-subformula-st test-symb } \psi)$   
**by** *simp*

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

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

**have** *all-subformula-st test-symb* (*FNot*  $\varphi$ )  $\longleftrightarrow$  *all-subformula-st test-symb* (*conn CNot*  $[\varphi]$ )  
**by** *auto*  
**moreover have**  $\dots = (\text{test-symb } (\text{conn } CNot [\varphi]) \wedge (\forall \xi \in \text{set } [\varphi]. \text{all-subformula-st test-symb } \xi))$   
**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  $\varphi$ .

**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*:  $\forall x. \text{test-symb } FF \wedge \text{test-symb } FT \wedge \text{test-symb } (FVar\ x)$  **and**  
 $\forall \varphi'. \varphi' \preceq \varphi \longrightarrow (\neg \text{test-symb } \varphi') \longrightarrow (\exists \psi. r\ \varphi'\ \psi)$  **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**  $\neg \text{all-subformula-st test-symb } \varphi$

**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  $IH\varphi1: \neg \text{all-subformula-st test-symb } \varphi1 \implies \exists \psi \psi'. \psi \preceq \varphi1 \wedge r \psi \psi'$ 
  using  $\text{inc}(1)$   $\text{subformula-trans le}$  by  $\text{blast}$ 
from  $r$   $IH\varphi2-0$  have  $IH\varphi2: \neg \text{all-subformula-st test-symb } \varphi2 \implies \exists \psi. \psi \preceq \varphi2 \wedge (\exists \psi'. r \psi \psi')$ 
  using  $\text{inc}(2)$   $\text{subformula-trans le}$  by  $\text{blast}$ 
have cases:  $\neg \text{test-symb } \varphi \vee \neg \text{all-subformula-st test-symb } \varphi1 \vee \neg \text{all-subformula-st test-symb } \varphi2$ 
  using  $c \text{ nst}$  by  $\text{auto}$ 
show  $\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi'$ 
  using  $IH\varphi1$   $IH\varphi2$   $\text{subformula-trans inc subformula-refl cases le}$  by  $\text{blast}$ 
qed

```

### 1.4.2 Invariant conservation

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

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

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

### Invariant while lifting of the Rewriting Relation

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

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

```

fixes  $r:: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$  and  $\text{test-symb}:: 'v \text{ propo} \Rightarrow \text{bool}$  and  $x:: 'v$ 
and  $\varphi \psi \Phi:: 'v \text{ propo}$ 
assumes  $H: \forall \varphi' \psi. \varphi' \preceq \Phi \longrightarrow r \varphi' \psi \longrightarrow \text{all-subformula-st test-symb } \varphi'$ 
   $\longrightarrow \text{all-subformula-st test-symb } \psi$ 
and  $H': \forall (c:: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \varphi \preceq \Phi \longrightarrow \text{propo-rew-step } r \varphi \varphi'$ 
   $\longrightarrow \text{wf-conn } c (\xi @ \varphi \# \xi') \longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \longrightarrow \text{test-symb } \varphi'$ 
   $\longrightarrow \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$  and
   $\text{propo-rew-step } r \varphi \psi$  and
   $\varphi \preceq \Phi$  and
   $\text{all-subformula-st test-symb } \varphi$ 
shows  $\text{all-subformula-st test-symb } \psi$ 
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 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  $\varphi$  sq by fastforce
  then have test-symb  $\varphi'$  using all-subformula-st-test-symb-true-phi by auto
  moreover from corr nst have test-symb (conn c ( $\xi @ \varphi \# \xi'$ ))
    using all-subformula-st-decomp by blast
  ultimately have test-symb: test-symb (conn c ( $\xi @ \varphi' \# \xi'$ )) using  $H'$  sq corr rel by blast

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

propo-rew-step  $r \varphi \psi$  **and**

all-subformula-st test-symb  $\varphi$

**shows** all-subformula-st test-symb  $\psi$

**using** propo-rew-step-inv-stay'[of  $\varphi$   $r$  test-symb  $\varphi \psi$ ] *assms* subformula-refl **by** metis

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

## Invariant after all Rewriting

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

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

**assumes**

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

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

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

init: all-subformula-st test-symb  $\varphi$

**shows** all-subformula-st test-symb  $\psi$

**using** *assms* unfolding full-def

**proof** –

**have** rel: (propo-rew-step  $r$ )\*\*  $\varphi \psi$

**using** full unfolding full-def **by** auto

**then show** all-subformula-st test-symb  $\psi$

**using** init

**proof** (induct rule: rtranclp-induct)

**case** base

**then show** all-subformula-st test-symb  $\varphi$  **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**  $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$  **and**  $\text{test-symb} :: 'v \text{ propo} \Rightarrow \text{bool}$  **and**  $x :: 'v$   
**and**  $\varphi \psi :: 'v \text{ propo}$

**assumes**

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

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

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

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

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

**using** *full-propo-rew-step-inv-stay-with-inc*[of  $r$   $\text{test-symb } \varphi$ ] *assms subformula-refl* **by** *metis*

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

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

**assumes**

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

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

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

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

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

**unfolding** *full-def*

**proof** –

**have**  $\text{rel}: (\text{propo-rew-step } r)^{**} \varphi \psi$

**using** *full unfolding full-def* **by** *auto*

**then show**  $\text{all-subformula-st test-symb } \psi$

**using** *init*

**proof** (*induct rule: rtrancpl-induct*)

**case** *base*

**then show**  $\text{all-subformula-st test-symb } \varphi$  **by** *blast*

**next**

**case** (*step b c*)

**note**  $\text{star} = \text{this}(1)$  **and**  $\text{IH} = \text{this}(3)$  **and**  $\text{one} = \text{this}(2)$  **and**  $\text{all} = \text{this}(4)$

**then have**  $\text{all-subformula-st test-symb } b$  **by** *metis*

**then show**  $\text{all-subformula-st test-symb } c$

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

**assumes**

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

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

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

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

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

**proof** –

```

have  $\bigwedge(c :: 'v \text{ connective}) \xi \varphi \xi' \varphi'. \text{wf-conn } c (\xi @ \varphi \# \xi')$ 
   $\implies \text{test-symb } (\text{conn } c (\xi @ \varphi \# \xi')) \implies \text{test-symb } \varphi' \implies \text{test-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$ 
  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 Prop-Logic Prop-Abstract-Transformation Nested-Multisets-Ordinals.Multiset-More
begin

```

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

## 1.5 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.

### 1.5.1 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  $\varphi \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$ 
by (induct rule: elim-equiv.induct, auto)

```

```

lemma elim-equiv-consistent: preserve-models elim-equiv
unfolding preserve-models-def by (simp add: elim-equiv-explicit)

```

```

lemma elimEquiv-lifted-consistant:
preserve-models (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  $\varphi \psi :: 'v \text{ propo}$

shows

$\neg \text{no-equiv } (FEq \varphi \psi)$

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

shows

no-equiv (FNot  $\varphi$ )  $\longleftrightarrow$  no-equiv  $\varphi$

no-equiv (FAnd  $\varphi \psi$ )  $\longleftrightarrow$  (no-equiv  $\varphi \wedge$  no-equiv  $\psi$ )

no-equiv (FOr  $\varphi \psi$ )  $\longleftrightarrow$  (no-equiv  $\varphi \wedge$  no-equiv  $\psi$ )

no-equiv (FImp  $\varphi \psi$ )  $\longleftrightarrow$  (no-equiv  $\varphi \wedge$  no-equiv  $\psi$ )

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

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

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

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

assume  $a1: \text{elim-equiv } (\text{conn } c \ l) \ \psi$

have  $\bigwedge p \text{ pa}. \neg \text{elim-equiv } (p::'v \text{ propo}) \text{ pa} \vee \neg \text{no-equiv-symb } p$

using elim-equiv.cases no-equiv-symb.simps(1) by blast

then have  $\text{elim-equiv } (\text{conn } c \ l) \ \psi \implies \neg \text{no-equiv-symb } (\text{conn } c \ l) \text{ using } a1 \text{ by metis}$

}

moreover have  $H': \forall \psi. \neg \text{elim-equiv } FT \ \psi \vee \psi. \neg \text{elim-equiv } FF \ \psi \vee \psi \ x. \neg \text{elim-equiv } (FVar x) \ \psi$

using elim-equiv.cases by auto

moreover have  $\bigwedge \varphi. \neg \text{no-equiv-symb } \varphi \implies \exists \psi. \text{elim-equiv } \varphi \ \psi$

by (case-tac  $\varphi$ , auto simp: elim-equiv.simps)

then have  $\bigwedge \varphi'. \varphi' \preceq \varphi \implies \neg \text{no-equiv-symb } \varphi' \implies \exists \psi. \text{elim-equiv } \varphi' \ \psi$  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)  $\varphi \psi \implies \text{no-equiv } \psi$

using full-propo-rew-step-subformula no-equiv-elim-equiv-step by blast

### 1.5.2 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  $\varphi$   $\psi$ ) (FOr (FNot  $\varphi$ )  $\psi$ )*

**lemma** *elim-imp-transformation-consistent*:  
 $A \models \text{FImp } \varphi \ \psi \longleftrightarrow A \models \text{FOr } (\text{FNot } \varphi) \ \psi$   
**by** *auto*

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

**lemma** *elim-imp-consistent*: *preserve-models elim-imp*  
**unfolding** *preserve-models-def* **by** (*simp add: elim-imp-explicit*)

**lemma** *elim-imp-lifted-consistent*:  
*preserve-models (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*:  
 $\text{wf-conn } c \ l \implies \text{no-imp-symb } (\text{conn } c \ l) \longleftrightarrow c \neq \text{CImp}$   
**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*]:  
 $\neg \text{no-imp } (\text{FImp } \varphi \ \psi)$   
 $\text{no-imp } \text{FT}$   
 $\text{no-imp } \text{FF}$   
**unfolding** *no-imp-def* **by** *auto*

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

Invariant of the *elim-imp* transformation

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

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

*no-equiv-symb-conn-characterization* **unfolding** *no-equiv-def* **by** *metis*

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

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

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

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

**proof** –

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

**moreover** {

**fix**  $c::'v \text{ connective}$  **and**  $l::'v \text{ propo list}$  **and**  $\psi::'v \text{ propo}$

**have**  $H: \text{elim-imp } (\text{conn } c \ l) \ \psi \implies \neg \text{no-imp-symb } (\text{conn } c \ l)$

**by** (*auto elim: elim-imp.cases*)

}

**moreover**

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

**by** (*auto elim: elim-imp.cases*) +

**moreover**

**have**  $\bigwedge \varphi. \neg \text{no-imp-symb } \varphi \implies \exists \psi. \text{elim-imp } \varphi \ \psi$

**by** (*case-tac*  $\varphi$ ) (*force simp: elim-imp.simps*) +

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

**ultimately show** *?thesis*

**using** *no-test-symb-step-exists no-equiv test-symb-false-nullary* **unfolding** *no-imp-def* **by** *blast*

**qed**

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

**using** *full-propo-rew-step-subformula no-no-imp-elim-imp-step-exists* **by** *blast*

### 1.5.3 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*  $\varphi$  *FT*)  $\varphi$  |

*ElimTB1'*: *elimTB* (*FAnd* *FT*  $\varphi$ )  $\varphi$  |

*ElimTB2*: *elimTB* (*FAnd*  $\varphi$  *FF*) *FF* |

*ElimTB2'*: *elimTB* (*FAnd* *FF*  $\varphi$ ) *FF* |

*ElimTB3*: *elimTB* (*FOr*  $\varphi$  *FT*) *FT* |

*ElimTB3'*: *elimTB* (*FOr* *FT*  $\varphi$ ) *FT* |

*ElimTB4*: *elimTB* (*FOr*  $\varphi$  *FF*)  $\varphi$  |

*ElimTB4'*: *elimTB* (*FOr* *FF*  $\varphi$ )  $\varphi$  |

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

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

**lemma** *elimTB-consistent*: *preserve-models elimTB*

**proof** –

{

**fix**  $\varphi \psi::'b \text{ propo}$

**have**  $\text{elimTB } \varphi \ \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$  **by** (*induction rule: elimTB.inducts*) *auto*

}

**then show** *?thesis using preserve-models-def by auto*  
**qed**

**inductive** *no-T-F-symb* :: '*v* *propo*  $\Rightarrow$  *bool* **where**  
*no-T-F-symb-comp*:  $c \neq CF \Rightarrow c \neq CT \Rightarrow wf\_conn\ c\ l \Rightarrow (\forall \varphi \in set\ l. \varphi \neq FT \wedge \varphi \neq FF)$   
 $\Rightarrow no-T-F-symb\ (conn\ c\ l)$

**lemma** *wf-conn-no-T-F-symb-iff[simp]*:  
 $wf\_conn\ c\ \psi s \Rightarrow$   
 $no-T-F-symb\ (conn\ c\ \psi s) \longleftrightarrow (c \neq CF \wedge c \neq CT \wedge (\forall \psi \in set\ \psi s. \psi \neq FF \wedge \psi \neq FT))$   
**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]*:  
 $no-T-F-symb\ (FAnd\ \varphi\ \psi) \longleftrightarrow (\forall \chi \in set\ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$   
 $no-T-F-symb\ (FOr\ \varphi\ \psi) \longleftrightarrow (\forall \chi \in set\ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$   
 $no-T-F-symb\ (FEq\ \varphi\ \psi) \longleftrightarrow (\forall \chi \in set\ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$   
 $no-T-F-symb\ (FImp\ \varphi\ \psi) \longleftrightarrow (\forall \chi \in set\ [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$   
**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*:  
 $\neg no-T-F-symb\ (FNot\ \varphi) \Rightarrow \varphi = FT \vee \varphi = FF$   
**proof** (*rule ccontr*)  
**assume** *n*:  $\neg no-T-F-symb\ (FNot\ \varphi)$   
**assume**  $\neg (\varphi = FT \vee \varphi = FF)$   
**then have**  $\forall \varphi' \in set\ [\varphi]. \varphi' \neq FT \wedge \varphi' \neq FF$  **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* ∈ *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**  $FT \in set\ l \vee FF \in set\ l$   
**shows**  $\neg no-T-F-symb-except-toplevel (conn\ c\ l)$   
**by** (*metis assms empty-iff no-T-F-symb-except-toplevel.simps wf-conn-no-T-F-symb-iff set-empty wf-conn-list(1,2)*)

**lemma** *no-T-F-symb-except-top-level-false-example[simp]*:  
**fixes**  $\varphi \psi$  :: '*v* *propo*  
**assumes**  $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$   
**shows**  
 $\neg no-T-F-symb-except-toplevel (FAnd\ \varphi\ \psi)$   
 $\neg no-T-F-symb-except-toplevel (FOr\ \varphi\ \psi)$   
 $\neg no-T-F-symb-except-toplevel (FImp\ \varphi\ \psi)$   
 $\neg no-T-F-symb-except-toplevel (FEq\ \varphi\ \psi)$   
**using** *assms no-T-F-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-F-symb-except-top-level-false-not[simp]*:  
**fixes**  $\varphi \psi$  :: '*v* *propo*  
**assumes**  $\varphi = FT \vee \varphi = FF$   
**shows**

$\neg$  *no-T-F-symb-except-toplevel* (*FNot*  $\varphi$ )  
**by** (*simp add: assms no-T-F-symb-except-toplevel.simps*)

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

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

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

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

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

**lemma** *no-T-F-except-top-level-false-example*[*simp*]:  
**fixes**  $\varphi \psi :: 'v$  *propo*  
**assumes**  $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$   
**shows**  
 $\neg$ *no-T-F-except-top-level* (*FAnd*  $\varphi \psi$ )  
 $\neg$ *no-T-F-except-top-level* (*FOr*  $\varphi \psi$ )  
 $\neg$ *no-T-F-except-top-level* (*FEq*  $\varphi \psi$ )  
 $\neg$ *no-T-F-except-top-level* (*FImp*  $\varphi \psi$ )  
**by** (*metis all-subformula-st-test-symb-true-phi assms no-T-F-except-top-level-def*  
*no-T-F-symb-except-top-level-false-example*)+

**lemma** *no-T-F-symb-except-toplevel-no-T-F-symb*:  
*no-T-F-symb-except-toplevel*  $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies$  *no-T-F-symb*  $\varphi$   
**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*  $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies$  *no-T-F*  $\varphi$   
**unfolding** *no-T-F-except-top-level-def no-T-F-def* **apply** (*induct*  $\varphi$ )  
**using** *no-T-F-symb-fnot* **by** *fastforce*+

**lemma** *no-T-F-no-T-F-except-top-level*:  
*no-T-F*  $\varphi \implies$  *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*  $\varphi \longleftrightarrow (\varphi = FF \vee \varphi = FT \vee$  *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* ∈ *binary-connectives*  
**shows** *no-T-F* (*conn c* [*φ*, *ψ*])  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
**proof** –  
**have** *wf*: *wf-conn c* [*φ*, *ψ*] **using** *c* **by** *auto*  
**then have** *no-T-F* (*conn c* [*φ*, *ψ*])  $\longleftrightarrow$  (*no-T-F-symb* (*conn c* [*φ*, *ψ*]) ∧ *no-T-F* *φ* ∧ *no-T-F* *ψ*)  
**by** (*simp add: all-subformula-st-decomp no-T-F-def*)  
**then show** *no-T-F* (*conn c* [*φ*, *ψ*])  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
**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* ∨ *c* = *COr* ∨ *c* = *CEq* ∨ *c* = *CImp*  
**shows** *no-T-F* (*conn c* [*φ*, *ψ*])  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
**using** *no-T-F-bin-decomp assms unfolding binary-connectives-def* **by** *blast*

**lemma** *no-T-F-comp-expanded-explicit*[simp]:  
**fixes** *φ ψ* :: '*v* *propo*  
**shows**  
*no-T-F* (*FAnd* *φ ψ*)  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
*no-T-F* (*FOr* *φ ψ*)  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
*no-T-F* (*FEq* *φ ψ*)  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
*no-T-F* (*FImp* *φ ψ*)  $\longleftrightarrow$  (*no-T-F* *φ* ∧ *no-T-F* *ψ*)  
**using** *conn.simps(5–8) no-T-F-bin-decomp-expanded* **by** (*metis (no-types)*)<sup>+</sup>

**lemma** *no-T-F-comp-not*[simp]:  
**fixes** *φ ψ* :: '*v* *propo*  
**shows** *no-T-F* (*FNot* *φ*)  $\longleftrightarrow$  *no-T-F* *φ*  
**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** *φ ψ* :: '*v* *propo*  
**assumes** *φ*: *no-T-F* (*FAnd* *φ ψ*) ∨ *no-T-F* (*FOr* *φ ψ*) ∨ *no-T-F* (*FEq* *φ ψ*) ∨ *no-T-F* (*FImp* *φ ψ*)  
**shows** *no-T-F* *ψ* **and** *no-T-F* *φ*  
**using** *assms* **by** *auto*

**lemma** *no-T-F-decomp-not*:  
**fixes** *φ* :: '*v* *propo*  
**assumes** *φ*: *no-T-F* (*FNot* *φ*)  
**shows** *no-T-F* *φ*  
**using** *assms* **by** *auto*

**lemma** *no-T-F-symb-except-toplevel-step-exists*:  
**fixes** *φ ψ* :: '*v* *propo*  
**assumes** *no-equiv φ* **and** *no-imp φ*  
**shows** *ψ* ≤ *φ*  $\implies$   $\neg$  *no-T-F-symb-except-toplevel* *ψ*  $\implies$   $\exists \psi'. \text{elimTB } \psi \ \psi'$   
**proof** (*induct ψ rule: propo-induct-arity*)  
**case** (*nullary φ' 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 ψ*)  
**then have** *ψ* = *FF* ∨ *ψ* = *FT* **using** *no-T-F-symb-except-toplevel-not-decom* **by** *blast*

```

then show ?case using ElimTB5 ElimTB6 by blast
next
case (binary  $\varphi'$   $\psi_1$   $\psi_2$ )
note IH1 = this(1) and IH2 = this(2) and  $\varphi' = \text{this}(3)$  and  $F\varphi = \text{this}(4)$  and  $n = \text{this}(5)$ 
{
  assume  $\varphi' = FImp \psi_1 \psi_2 \vee \varphi' = FEq \psi_1 \psi_2$ 
  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  $\varphi'$ :  $\varphi' = FAnd \psi_1 \psi_2 \vee \varphi' = FOr \psi_1 \psi_2$ 
  then have  $\psi_1 = FT \vee \psi_2 = FT \vee \psi_1 = FF \vee \psi_2 = FF$ 
    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  $\varphi :: 'v$  propo
assumes noTB:  $\neg$  no-T-F-except-top-level  $\varphi$  and no-equiv: no-equiv  $\varphi$  and no-imp: no-imp  $\varphi$ 
shows  $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{elimTB } \psi \psi'$ 

```

**proof** –

```

have test-symb-false-nullary:  $\forall x. \text{no-T-F-symb-except-toplevel } (FF :: 'v \text{ propo})$ 
   $\wedge \text{no-T-F-symb-except-toplevel } FT \wedge \text{no-T-F-symb-except-toplevel } (FVar (x :: 'v))$  by auto

```

**moreover** {

```

  fix  $c :: 'v$  connective and  $l :: 'v$  propo list and  $\psi :: 'v$  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  $H': \text{no-T-F-except-top-level } FT \ \text{no-T-F-except-top-level } FF$ 
     $\text{no-T-F-except-top-level } (FVar \ x)$ 
    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  $\varphi \ \psi :: 'v$  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  $\varphi \ \psi :: 'v$  propo
  have  $H: \text{elimTB } \varphi \ \psi \implies \text{no-equiv } \varphi \implies \text{no-equiv } \psi$ 

```

```

    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  $\varphi \psi :: 'v$  propo
  have  $H: \text{elimTB } \varphi \psi \implies \text{no-imp } \varphi \implies \text{no-imp } \psi$ 
    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  $\varphi \psi :: 'v$  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

#### 1.5.4 PushNeg

Push the negation inside the formula, until the literal.

**inductive** *pushNeg* **where**  
 $\text{PushNeg1[simp]: } \text{pushNeg } (\text{FNot } (\text{FAnd } \varphi \psi)) (\text{FOr } (\text{FNot } \varphi) (\text{FNot } \psi)) \mid$   
 $\text{PushNeg2[simp]: } \text{pushNeg } (\text{FNot } (\text{FOr } \varphi \psi)) (\text{FAnd } (\text{FNot } \varphi) (\text{FNot } \psi)) \mid$   
 $\text{PushNeg3[simp]: } \text{pushNeg } (\text{FNot } (\text{FNot } \varphi)) \varphi$

**lemma** *pushNeg-transformation-consistent*:  
 $A \models \text{FNot } (\text{FAnd } \varphi \psi) \longleftrightarrow A \models (\text{FOr } (\text{FNot } \varphi) (\text{FNot } \psi))$   
 $A \models \text{FNot } (\text{FOr } \varphi \psi) \longleftrightarrow A \models (\text{FAnd } (\text{FNot } \varphi) (\text{FNot } \psi))$   
 $A \models \text{FNot } (\text{FNot } \varphi) \longleftrightarrow A \models \varphi$   
 by auto

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

**lemma** *pushNeg-consistent*: *preserve-models pushNeg*  
 unfolding preserve-models-def by (simp add: pushNeg-explicit)

**lemma** *pushNeg-lifted-consistent*:  
*preserve-models (full (propo-rew-step pushNeg))*  
 by (simp add: pushNeg-consistent)

**fun** *simple* **where**  
 $\text{simple } FT = \text{True} \mid$   
 $\text{simple } FF = \text{True} \mid$   
 $\text{simple } (\text{FVar } -) = \text{True} \mid$   
 $\text{simple } - = \text{False}$

**lemma** *simple-decomp*:

*simple*  $\varphi \longleftrightarrow (\varphi = FT \vee \varphi = FF \vee (\exists x. \varphi = FVar\ x))$   
**by** (*cases*  $\varphi$ ) *auto*

**lemma** *subformula-conn-decomp-simple*:

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

**assumes** *s*: *simple*  $\psi$

**shows**  $\varphi \preceq FNot\ \psi \longleftrightarrow (\varphi = FNot\ \psi \vee \varphi = \psi)$

**proof** –

**have**  $\varphi \preceq conn\ CNot\ [\psi] \longleftrightarrow (\varphi = conn\ CNot\ [\psi] \vee (\exists \psi \in set\ [\psi]. \varphi \preceq \psi))$

**using** *subformula-conn-decomp wf-conn-helper-facts(1)* **by** *metis*

**then show**  $\varphi \preceq FNot\ \psi \longleftrightarrow (\varphi = FNot\ \psi \vee \varphi = \psi)$  **using** *s* **by** (*auto simp: simple-decomp*)

**qed**

**lemma** *subformula-conn-decomp-explicit[simp]*:

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

**shows**

$\varphi \preceq FNot\ FT \longleftrightarrow (\varphi = FNot\ FT \vee \varphi = FT)$

$\varphi \preceq FNot\ FF \longleftrightarrow (\varphi = FNot\ FF \vee \varphi = FF)$

$\varphi \preceq FNot\ (FVar\ x) \longleftrightarrow (\varphi = FNot\ (FVar\ x) \vee \varphi = FVar\ x)$

**by** (*auto simp: subformula-conn-decomp-simple*)

**fun** *simple-not-symb* **where**

*simple-not-symb* (*FNot*  $\varphi$ ) = (*simple*  $\varphi$ ) |

*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]*:

$\neg simple-not\ (FNot\ (FAnd\ \varphi\ \psi))$

$\neg simple-not\ (FNot\ (FOr\ \varphi\ \psi))$

**by** *auto*

**lemma** *simple-not-step-exists*:

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

**assumes** *no-equiv*  $\varphi$  **and** *no-imp*  $\varphi$

**shows**  $\psi \preceq \varphi \implies \neg simple-not-symb\ \psi \implies \exists \psi'. pushNeg\ \psi\ \psi'$

**apply** (*induct*  $\psi$ , *auto*)

**apply** (*rename-tac*  $\psi$ , *case-tac*  $\psi$ , *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**  $\varphi :: 'v\ propo$

**assumes** *noTB*:  $\neg simple-not\ \varphi$  **and** *no-equiv*: *no-equiv*  $\varphi$  **and** *no-imp*: *no-imp*  $\varphi$

**shows**  $\exists \psi\ \psi'. \psi \preceq \varphi \wedge pushNeg\ \psi\ \psi'$

**proof** –

**have**  $\forall x. simple-not-symb\ (FF :: 'v\ propo) \wedge simple-not-symb\ FT \wedge simple-not-symb\ (FVar\ (x :: 'v))$   
**by** *auto*

**moreover** {

**fix** *c*: *'v* *connective* **and** *l*: *'v* *propo* *list* **and**  $\psi :: 'v\ propo$

**have** *H*:  $pushNeg\ (conn\ c\ l)\ \psi \implies \neg simple-not-symb\ (conn\ c\ l)$

**by** (*cases* *conn c l* *rule: pushNeg.cases*) *auto*

```

}
moreover {
  fix  $x :: 'v$ 
  have  $H'$ : simple-not FT simple-not FF simple-not (FVar x)
  by simp-all
}
moreover {
  fix  $\psi :: 'v$  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*:  
 $\text{no-T-F-except-top-level } (F\text{Not } (F\text{And } \varphi \psi)) \implies \text{no-T-F-except-top-level } (F\text{Or } (F\text{Not } \varphi) (F\text{Not } \psi))$   
**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*:  
 $\text{no-T-F-except-top-level } (F\text{Not } (F\text{Or } \varphi \psi)) \implies \text{no-T-F-except-top-level } (F\text{And } (F\text{Not } \varphi) (F\text{Not } \psi))$   
**by** *auto*

**lemma** *no-T-F-symb-pushNeg*:  
 $\text{no-T-F-symb } (F\text{Or } (F\text{Not } \varphi') (F\text{Not } \psi'))$   
 $\text{no-T-F-symb } (F\text{And } (F\text{Not } \varphi') (F\text{Not } \psi'))$   
 $\text{no-T-F-symb } (F\text{Not } (F\text{Not } \varphi'))$   
**by** *auto*

**lemma** *propo-rew-step-pushNeg-no-T-F-symb*:  
 $\text{propo-rew-step pushNeg } \varphi \psi \implies \text{no-T-F-except-top-level } \varphi \implies \text{no-T-F-symb } \varphi \implies \text{no-T-F-symb } \psi$   
**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:  $\text{no-T-F } \varphi \implies \text{no-T-F-symb } \varphi \implies \text{no-T-F-symb } \varphi'$ 
and wf: wf-conn c ( $\xi @ \varphi \# \xi'$ )
and  $n$ :  $\text{conn } c (\xi @ \varphi \# \xi') = FF \vee \text{conn } c (\xi @ \varphi \# \xi') = FT \vee \text{no-T-F } (\text{conn } c (\xi @ \varphi \# \xi'))$ 
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  $c \neq CF \wedge c \neq CT \wedge \text{wf-conn } c (\xi @ \varphi' \# \xi')$ 
  using wf-conn-no-arity-change-helper wf-conn-no-arity-change by metis
moreover have  $n'$ :  $\text{no-T-F } (\text{conn } c (\xi @ \varphi' \# \xi'))$  using  $n$  by (simp add: wf wf-conn-list(1,2))
moreover
{
  have  $\text{no-T-F } \varphi$ 
  by (metis Un-iff all-subformula-st-decomp list.set-intros(1) n' wf no-T-F-def set-append)
  moreover then have  $\text{no-T-F-symb } \varphi$ 
  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  $\text{no-}T\text{-}F\text{-symb } (\text{conn } c (\xi @ \varphi' \# \xi'))$  by (simp add:  $x$ )  
 qed

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

*propo-rew-step pushNeg  $\varphi \psi \implies \text{no-}T\text{-}F \varphi \implies \text{no-}T\text{-}F \psi$*

**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**  $\text{rel} = \text{this}(1)$  and  $\text{IH} = \text{this}(2)$  and  $\text{wf} = \text{this}(3)$  and  $\text{no-}T\text{-}F = \text{this}(4)$

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

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

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

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

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

**and** *no-equiv  $\varphi$  and no-imp  $\varphi$  and no-T-F-except-top-level  $\varphi$*

**shows** *no-equiv  $\psi$  and no-imp  $\psi$  and no-T-F-except-top-level  $\psi$*

**proof** –

{

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

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

```

**lemma**  $\text{pushNeg-full-propo-rew-step:}$

```

fixes  $\varphi \psi :: 'v \text{ propo}$ 
assumes
   $\text{no-equiv } \varphi$  and
   $\text{no-imp } \varphi$  and

```

*full (propo-rew-step pushNeg)  $\varphi$   $\psi$  and*  
*no-T-F-except-top-level  $\varphi$*   
**shows** *simple-not  $\psi$*   
**using** *assms full-propo-rew-step-subformula pushNeg-inv(1,2) simple-not-rew by blast*

### 1.5.5 Push Inside

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

**for** *c c'* :: '*v* *connective* **where**  
*push-conn-inside-l[simp]:*  $c = CAnd \vee c = COr \Longrightarrow c' = CAnd \vee c' = COr$   
 $\Longrightarrow$  *push-conn-inside c c' (conn c [conn c' [ $\varphi 1$ ,  $\varphi 2$ ],  $\psi$ ])*  
 $(conn c' [conn c [\varphi 1,  $\psi$ ], conn c [\varphi 2,  $\psi$ ]]) \mid$   
*push-conn-inside-r[simp]:*  $c = CAnd \vee c = COr \Longrightarrow c' = CAnd \vee c' = COr$   
 $\Longrightarrow$  *push-conn-inside c c' (conn c [ $\psi$ , conn c' [ $\varphi 1$ ,  $\varphi 2$ ]])*  
 $(conn c' [conn c [ $\psi$ ,  $\varphi 1$ ], conn c [ $\psi$ ,  $\varphi 2$ ]])$

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

**lemma** *push-conn-inside-consistent:* *preserve-models (push-conn-inside c c')*  
**unfolding** *preserve-models-def* **by** (*simp add: push-conn-inside-explicit*)

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

**abbreviation** *c-in-c'-symb c c'  $\varphi \equiv \neg$ not-c-in-c'-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]*:

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

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

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

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

*c-in-c'-only*  $c \ c' \ FF$   
*c-in-c'-only*  $c \ c' \ FT$   
*c-in-c'-only*  $c \ c' \ (FVar \ x)$   
*c-in-c'-only*  $c \ c' \ (FNot \ FF)$   
*c-in-c'-only*  $c \ c' \ (FNot \ FT)$   
*c-in-c'-only*  $c \ c' \ (FNot \ (FVar \ x))$   
**unfolding** *c-in-c'-only-def* **by** *auto*

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

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

**proof** (*induct rule: not-c-in-c'-symb.induct*)

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

**next**

**case** (*not-c-in-c'-symb-l*  $\varphi' \ \varphi'' \ \psi'$ ) **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**  $\text{not-c-in-c'-symb } c \ c' \ (\text{conn } c \ [\psi, \varphi])$   
**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'*:

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

**proof** –

**have**  $?A \longleftrightarrow (c\text{-in-c'-symb } c \ c' \ (\text{conn } c \ [\varphi, \psi])$   
 $\quad \wedge (\forall \xi \in \text{set } [\varphi, \psi]. \text{all-subformula-st } (c\text{-in-c'-symb } c \ c' \ \xi))$   
**using** *all-subformula-st-decomp wf* **unfolding** *c-in-c'-only-def* **by** *fastforce*  
**also have**  $\dots \longleftrightarrow (c\text{-in-c'-symb } c \ c' \ (\text{conn } c \ [\psi, \varphi])$

$\wedge (\forall \xi \in \text{set } [\psi, \varphi]. \text{all-subformula-st } (c\text{-in-}c'\text{-symb } c \ c') \ \xi))$   
**using** *not-c-in-c'-symb-commute'* *wf* **by** *auto*  
**also**  
**have** *wf-conn* *c*  $[\psi, \varphi]$  **using** *wf-conn-no-arity-change* *wf* **by** (*metis length-Cons*)  
**then have** (*c-in-c'-symb* *c* *c'* (*conn* *c*  $[\psi, \varphi]$ )  
 $\wedge (\forall \xi \in \text{set } [\psi, \varphi]. \text{all-subformula-st } (c\text{-in-}c'\text{-symb } c \ c') \ \xi))$   
 $\longleftrightarrow ?B$   
**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**  
*c-in-c'-symb* *c* *c'* *FT*  
*c-in-c'-symb* *c* *c'* *FF*  
*c-in-c'-symb* *c* *c'* (*FVar* *x*)  
*wf-conn* *c* [*conn* *c'*  $[\varphi 1, \varphi 2]$ ,  $\psi$ ]  $\implies$  *wf-conn* *c'*  $[\varphi 1, \varphi 2]$   
 $\implies \neg$  *c-in-c'-only* *c* *c'* (*conn* *c* [*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** *c-in-c'-symb* *c* *c'* (*FNot*  $\psi$ )  
**proof** –  
{  
**fix**  $\xi :: 'v \text{ propo}$   
**have** *not-c-in-c'-symb* *c* *c'* (*FNot*  $\psi$ )  $\implies$  *False*  
**apply** (*induct FNot*  $\psi$  *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*:  $c = C\text{And} \vee c = C\text{Or}$  **and** *c'*:  $c' = C\text{And} \vee c' = C\text{Or}$   
**shows**  $\psi \preceq \varphi \implies \neg$  *c-in-c'-symb* *c* *c'*  $\psi \implies \exists \psi'. \text{push-conn-inside } c \ c' \ \psi \ \psi'$   
**apply** (*induct*  $\psi$  *rule: propo-induct-arity*)  
**apply** *auto[2]*  
**proof** –  
**fix**  $\psi 1 \ \psi 2 \ \varphi' :: 'v \text{ propo}$   
**assume** *IH* $\psi 1$ :  $\psi 1 \preceq \varphi \implies \neg$  *c-in-c'-symb* *c* *c'*  $\psi 1 \implies \text{Ex } (\text{push-conn-inside } c \ c' \ \psi 1)$   
**and** *IH* $\psi 2$ :  $\psi 2 \preceq \varphi \implies \neg$  *c-in-c'-symb* *c* *c'*  $\psi 2 \implies \text{Ex } (\text{push-conn-inside } c \ c' \ \psi 2)$   
**and**  $\varphi'$ :  $\varphi' = F\text{And } \psi 1 \ \psi 2 \vee \varphi' = F\text{Or } \psi 1 \ \psi 2 \vee \varphi' = F\text{Imp } \psi 1 \ \psi 2 \vee \varphi' = F\text{Eq } \psi 1 \ \psi 2$   
**and** *in* $\varphi$ :  $\varphi' \preceq \varphi$  **and** *n0*:  $\neg$  *c-in-c'-symb* *c* *c'*  $\varphi'$   
**then have** *n*: *not-c-in-c'-symb* *c* *c'*  $\varphi'$  **by** *auto*  
{  
**assume**  $\varphi'$ :  $\varphi' = \text{conn } c \ [\psi 1, \psi 2]$   
**obtain** *a* *b* **where**  $\psi 1 = \text{conn } c' \ [a, b] \vee \psi 2 = \text{conn } c' \ [a, b]$   
**using** *n*  $\varphi'$  **apply** (*induct rule: not-c-in-c'-symb.induct*)  
**using** *c* **by** *force+*  
**then have** *Ex* (*push-conn-inside* *c* *c'*  $\varphi'$ )  
**unfolding**  $\varphi'$  **apply** *auto*  
**using** *push-conn-inside.intros(1)* *c* *c'* **apply** *blast*

```

    using push-conn-inside.intros(2) c c' by blast
  }
  moreover {
    assume  $\varphi'$ :  $\varphi' \neq \text{conn } c [\psi 1, \psi 2]$ 
    have  $\forall \varphi c \text{ ca}. \exists \varphi 1 \psi 1 \psi 2 \psi 1' \psi 2' \varphi 2'. \text{conn } (c::'v \text{ connective}) [\varphi 1, \text{conn } \text{ca} [\psi 1, \psi 2]] = \varphi$ 
       $\vee \text{conn } c [\text{conn } \text{ca} [\psi 1', \psi 2'], \varphi 2'] = \varphi \vee c\text{-in-}c'\text{-symb } c \text{ ca } \varphi$ 
    by (metis not-c-in-c'-symb.cases)
    then have  $\text{Ex } (\text{push-conn-inside } c \text{ c'} \varphi')$ 
    by (metis (no-types) c c' n push-conn-inside-l push-conn-inside-r)
  }
  ultimately show  $\text{Ex } (\text{push-conn-inside } c \text{ c'} \varphi')$  by blast
qed

```

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

```

  fixes  $\varphi :: 'v \text{ propo}$ 
  assumes noTB:  $\neg c\text{-in-}c'\text{-only } c \text{ c'} \varphi$ 
  and c:  $c = C\text{And} \vee c = C\text{Or}$  and c':  $c' = C\text{And} \vee c' = C\text{Or}$ 
  shows  $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{push-conn-inside } c \text{ c'} \psi \psi'$ 
proof -
  have test-symb-false-nullary:
     $\forall x. c\text{-in-}c'\text{-symb } c \text{ c'} (FF:: 'v \text{ propo}) \wedge c\text{-in-}c'\text{-symb } c \text{ c'} FT$ 
     $\wedge c\text{-in-}c'\text{-symb } c \text{ c'} (FVar (x:: 'v))$ 
  by auto
  moreover {
    fix  $x :: 'v$ 
    have  $H': c\text{-in-}c'\text{-symb } c \text{ c'} FT \wedge c\text{-in-}c'\text{-symb } c \text{ c'} FF \wedge c\text{-in-}c'\text{-symb } c \text{ c'} (FVar x)$ 
    by simp+
  }
  moreover {
    fix  $\psi :: 'v \text{ propo}$ 
    have  $\psi \preceq \varphi \implies \neg c\text{-in-}c'\text{-symb } c \text{ c'} \psi \implies \exists \psi'. \text{push-conn-inside } c \text{ c'} \psi \psi'$ 
    by (auto simp: assms(2) c' c-in-c'-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  $\text{no-T-F } \psi$ 
    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  $\text{no-T-F } \varphi$ 
    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  $\varphi': \text{no-T-F } \varphi'$  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  $c: c \in \text{binary-connectives}$ 
  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  $c \ c' :: 'v$  connective and  $\varphi \ \psi :: 'v$  propo
shows propo-rew-step (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  $\varphi$ , 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  $IH = \text{this}(2)$  and  $wf = \text{this}(3)$ 
  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  $wf \ ca$  list-length2-decomp 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 simple-not-symb (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 rew simple-propo-rew-step-push-conn-inside-inv[OF rew] IH local.wf 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**

*propo-rew-step* (*push-conn-inside* c  $c'$ )  $\varphi \varphi'$  **and**  
*wf-conn* c ( $\xi @ \varphi \# \xi'$ ) **and**  
*simple-not-symb* (conn c ( $\xi @ \varphi \# \xi'$ )) **and**  
*simple-not-symb*  $\varphi'$

**shows** *simple-not-symb* (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**  $tel = this(1)$  **and**  $wf = this(2)$  **and**  
 $IH = this(3)$  **and**  $wf' = this(4)$  **and**  $simple' = this(5)$  **and**  $simple = this(6)$

**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 ( $\xi @ \text{conn } c' (\chi s @ \varphi' \# \chi s') \# \xi'$ )

**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*:  $\chi s = []$   $\chi s' = []$  **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  $c' \varphi \psi \implies \psi \neq FT \wedge \psi \neq FF$

by (induct rule: push-conn-inside.induct, auto)

lemma push-conn-inside-inv:

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

assumes full (propo-rew-step (push-conn-inside c c'))  $\varphi \psi$

and no-equiv  $\varphi$  and no-imp  $\varphi$  and no-T-F-except-top-level  $\varphi$  and simple-not  $\varphi$

shows no-equiv  $\psi$  and no-imp  $\psi$  and no-T-F-except-top-level  $\psi$  and simple-not  $\psi$

proof –

```
{
  {
    fix  $\varphi \psi :: 'v \text{ propo}$ 
    have H: push-conn-inside c c'  $\varphi \psi \implies$  all-subformula-st simple-not-symb  $\varphi$ 
       $\implies$  all-subformula-st simple-not-symb  $\psi$ 
      by (induct  $\varphi \psi$  rule: push-conn-inside.induct, auto)
  } note H = this
```

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

have H: propo-rew-step (push-conn-inside c c')  $\varphi \psi \implies$  all-subformula-st simple-not-symb  $\varphi$   
 $\implies$  all-subformula-st simple-not-symb  $\psi$

apply (induct  $\varphi \psi$  rule: propo-rew-step.induct)

using H apply simp

proof (rename-tac  $\varphi \varphi'$  ca  $\psi s \psi s'$ , case-tac ca rule: connective-cases-arity)

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

and x:: 'v

assume wf-conn c ( $\xi @ \varphi \# \xi'$ )

and  $c = CT \vee c = CF \vee c = CVar x$

then have  $\xi @ \varphi \# \xi' = []$  by auto

then have False by auto

then show all-subformula-st simple-not-symb (conn c ( $\xi @ \varphi' \# \xi'$ )) by blast

next

fix  $\varphi \varphi' :: 'v \text{ propo}$  and ca:: 'v connective and  $\xi \xi' :: 'v \text{ propo list}$

and x:: 'v

assume rel: propo-rew-step (push-conn-inside c c')  $\varphi \varphi'$

and  $\varphi\text{-}\varphi'$ : 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 = CNot

have empty:  $\xi = [] \wedge \xi' = []$  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  $\varphi\text{-}\varphi'$  append-Nil conn.simps(4) all-subformula-st-decomp-explicit(3)  
 simple-not-symb.simps(1))

next

fix  $\varphi \varphi' :: 'v \text{ propo}$  and ca:: 'v connective and  $\xi \xi' :: 'v \text{ propo list}$

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  $\chi \vee \chi \notin \text{set } (\xi @ \varphi' \# \xi')$ 
    using  $wf'$  by (metis (no-types) \varphi' all-subformula-st-decomp calculation insert-iff
      list.set(2))
}
then have  $\forall \varphi. \varphi \in \text{set } (\xi @ \varphi' \# \xi') \longrightarrow \text{all-subformula-st simple-not-symb } \varphi$ 
  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  $c$   $c'$ )  $\varphi \varphi' \Longrightarrow wf\text{-conn } ca (\xi @ \varphi \# \xi')$ 
     $\Longrightarrow \text{simple-not-symb } (\text{conn } ca (\xi @ \varphi \# \xi')) \Longrightarrow \text{simple-not-symb } \varphi'$ 
     $\Longrightarrow \text{simple-not-symb } (\text{conn } ca (\xi @ \varphi' \# \xi'))$ 
  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  $\psi$ 
  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  $\varphi \psi :: 'v$  propo
  have  $H$ : propo-rew-step (push-conn-inside  $c$   $c'$ )  $\varphi \psi \Longrightarrow \text{no-T-F-except-top-level } \varphi$ 
     $\Longrightarrow \text{no-T-F-except-top-level } \psi$ 
  proof –
    assume  $rel$ : propo-rew-step (push-conn-inside  $c$   $c'$ )  $\varphi \psi$ 
    and no-T-F-except-top-level  $\varphi$ 
    then have no-T-F  $\varphi \vee \varphi = FF \vee \varphi = FT$ 
      by (metis no-T-F-symb-except-tolevel-all-subformula-st-no-T-F-symb)
    moreover {
      assume  $\varphi = FF \vee \varphi = FT$ 
      then have False using rel propo-rew-step-push-conn-inside by blast
    }
}

```

```

    then have no-T-F-except-top-level  $\psi$  by blast
  }
  moreover {
    assume no-T-F  $\varphi \wedge \varphi \neq FF \wedge \varphi \neq FT$ 
    then have no-T-F  $\psi$  using rel push-conn-insidec-in-c'-symb-no-T-F by blast
    then have no-T-F-except-top-level  $\psi$  using no-T-F-no-T-F-except-top-level by blast
  }
  ultimately show no-T-F-except-top-level  $\psi$  by blast
qed
}
moreover {
  fix ca :: 'v connective and  $\xi \xi' :: 'v$  propo list and  $\varphi \varphi' :: 'v$  propo
  assume rel: propo-rew-step (push-conn-inside c c')  $\varphi \varphi'$ 
  assume corr: wf-conn ca ( $\xi @ \varphi \# \xi'$ )
  then have c: ca  $\neq CT \wedge ca \neq CF$  by auto
  assume no-T-F: no-T-F-symb-except-toplevel (conn ca ( $\xi @ \varphi \# \xi'$ ))
  have no-T-F-symb-except-toplevel (conn ca ( $\xi @ \varphi' \# \xi'$ ))
  proof
    have c: ca  $\neq CT \wedge ca \neq CF$  using corr by auto
    have  $\zeta: \forall \zeta \in \text{set } (\xi @ \varphi \# \xi'). \zeta \neq FT \wedge \zeta \neq FF$ 
      using corr no-T-F no-T-F-symb-except-toplevel-if-is-a-true-false by blast
    then have  $\varphi \neq FT \wedge \varphi \neq FF$  by auto
    from rel this have  $\varphi' \neq FT \wedge \varphi' \neq FF$ 
    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  $\forall \zeta \in \text{set } (\xi @ \varphi' \# \xi'). \zeta \neq FT \wedge \zeta \neq FF$  using  $\zeta$  by auto
    moreover have wf-conn ca ( $\xi @ \varphi' \# \xi'$ )
      using corr wf-conn-no-arity-change by (metis wf-conn-no-arity-change-helper)
    ultimately show no-T-F-symb (conn ca ( $\xi @ \varphi' \# \xi'$ )) using no-T-F-symb.intros c by metis
  qed
}
ultimately show no-T-F-except-top-level  $\psi$ 
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  $\varphi \psi :: 'v$  propo
  have H: push-conn-inside c c'  $\varphi \psi \implies$  no-equiv  $\varphi \implies$  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  $\varphi \psi :: 'v$  propo
  have H: push-conn-inside c c'  $\varphi \psi \implies$  no-imp  $\varphi \implies$  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*  $\varphi$  **and**  
   *no-imp*  $\varphi$  **and**  
   *full* (*propo-rew-step* (*push-conn-inside*  $c \ c'$ ))  $\varphi \ \psi$  **and**  
   *no-T-F-except-top-level*  $\varphi$  **and**  
   *simple-not*  $\varphi$  **and**  
    $c = CAnd \vee c = COr$  **and**  
    $c' = CAnd \vee c' = COr$   
**shows** *c-in-c'-only*  $c \ c' \ \psi$   
**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*  $\varphi \Longrightarrow$  *only-c-inside-symb*  $c \ \varphi$  |  
*simple-cnot-only-c-inside*[*simp*]: *simple*  $\varphi \Longrightarrow$  *only-c-inside-symb*  $c \ (FNot \ \varphi)$  |  
*only-c-inside-into-only-c-inside*: *wf-conn*  $c \ l \Longrightarrow$  *only-c-inside-symb*  $c \ (conn \ c \ l)$

**lemma** *only-c-inside-symb-simp*[*simp*]:  
*only-c-inside-symb*  $c \ FF$  *only-c-inside-symb*  $c \ FT$  *only-c-inside-symb*  $c \ (FVar \ x)$  **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*  $c \ \psi \longleftrightarrow$  (*simple*  $\psi$   
    $\vee (\exists \ \varphi'. \ \psi = FNot \ \varphi' \wedge \text{simple } \varphi')$   
    $\vee (\exists \ l. \ \psi = conn \ c \ l \wedge \text{wf-conn } c \ l)$ )  
**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**  $c :: 'v \text{ connective}$   
**assumes**  $c: c \neq CNot$   
**shows** *only-c-inside-symb*  $c \ (FNot \ \psi) \longleftrightarrow$  *simple*  $\psi$   
**apply** (*auto simp: only-c-inside-symb.intros(3)*)  
**by** (*induct FNot*  $\psi$  *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*  $c \ (FNot \ \psi) \longleftrightarrow$  *simple*  $\psi$   
**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*  $c \ \varphi \longleftrightarrow$   
    $(\forall \psi. \ \psi \preceq \varphi \longrightarrow (\text{simple } \psi \vee (\exists \ \varphi'. \ \psi = FNot \ \varphi' \wedge \text{simple } \varphi')$   
      $\vee (\exists \ l. \ \psi = conn \ c \ l \wedge \text{wf-conn } c \ l)))$   
**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*  $c\ \varphi$  **and** *incl*:  $conn\ c'\ l \preceq \varphi$  **and** *wf*:  $wf\text{-}conn\ c'\ l$   
**shows** *False*

**proof** –

**let**  $? \psi = conn\ c'\ l$   
**have**  $simple\ ? \psi \vee (\exists\ \varphi'.\ ? \psi = FNot\ \varphi' \wedge simple\ \varphi') \vee (\exists\ l.\ ? \psi = conn\ c\ l \wedge wf\text{-}conn\ c\ l)$   
**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-connn-list*(1–3))  
**moreover**  
 $\{$   
 $\quad$  **fix**  $\varphi'$   
 $\quad$  **have**  $? \psi \neq FNot\ \varphi'$  **using**  $c'$  *conn-inj-not*(1) *wf* **by** *blast*  
 $\}$   
**ultimately obtain**  $l :: 'v\ propo\ list$  **where**  $? \psi = conn\ c\ l \wedge wf\text{-}conn\ c\ l$  **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-connn.simps*) +

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

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

**proof** –

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

**qed**

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

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

**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  $\varphi_1 \varphi_2$ )
note  $IH\varphi_1 = this(1)$  and  $IH\varphi_2 = this(2)$  and  $\varphi = this(3)$  and  $only = this(5)$  and  $wf = this(4)$ 
and  $no-equiv = this(6)$  and  $no-imp = this(7)$  and  $simple-not = this(8)$ 
then have  $l: l = [\varphi_1, \varphi_2]$  by (meson wf-conn-list(4-7))
let  $? \varphi = conn\ c\ l$ 

obtain  $c_1\ l_1\ c_2\ l_2$  where  $\varphi_1: \varphi_1 = conn\ c_1\ l_1$  and  $wf\varphi_1: wf-conn\ c_1\ l_1$ 
and  $\varphi_2: \varphi_2 = conn\ c_2\ l_2$  and  $wf\varphi_2: wf-conn\ c_2\ l_2$  using exists-c-conn by metis
then have  $c\text{-in-}only\varphi_1: c\text{-in-}c'\text{-only}\ c\ c' (conn\ c_1\ l_1)$  and  $c\text{-in-}c'\text{-only}\ c\ c' (conn\ c_2\ l_2)$ 
using only l unfolding c-in-c'-only-def using assms(1) by auto
have  $inc\varphi_1: \varphi_1 \preceq ?\varphi$  and  $inc\varphi_2: \varphi_2 \preceq ?\varphi$ 
using  $\varphi_1\ \varphi_2\ \varphi\ local.wf$  by (metis conn.simps(5-8) helper-fact subformula-in-binary-conn(1,2))+

have  $c_1\text{-eq}: c_1 \neq CEq$  and  $c_2\text{-eq}: c_2 \neq CEq$ 
unfolding no-equiv-def using  $inc\varphi_1\ inc\varphi_2$  by (metis  $\varphi_1\ \varphi_2\ wf\varphi_1\ wf\varphi_2\ assms(1)\ no-equiv$ 
 $no-equiv\text{-eq}(1)\ no-equiv\text{-symb.elims}(3)\ no-equiv\text{-symb-conn-characterization}\ wf-conn-list(4,5)$ 
 $no-equiv\text{-def}\ subformula\text{-all}\ subformula\text{-st}$ )+
have  $c_1\text{-imp}: c_1 \neq CImp$  and  $c_2\text{-imp}: c_2 \neq CImp$ 
using no-imp by (metis  $\varphi_1\ \varphi_2\ all\text{-subformula}\text{-st-decomp-explicit-imp}(2,3)\ assms(1)$ 
 $conn.simps(5,6)\ l\ no\text{-imp}\text{-Imp}(1)\ no\text{-imp}\text{-symb.elims}(3)\ no\text{-imp}\text{-symb-conn-characterization}$ 
 $wf\varphi_1\ wf\varphi_2\ all\text{-subformula}\text{-st-decomp}\ no\text{-imp}\text{-symb-conn-characterization}$ )+
have  $c_1c: c_1 \neq c'$ 
proof
assume  $c_1c: c_1 = c'$ 
then obtain  $\xi_1\ \xi_2$  where  $l_1: l_1 = [\xi_1, \xi_2]$ 
by (metis assms(2) connective.distinct(37,39) helper-fact  $wf\varphi_1\ wf-conn.simps$ 
 $wf-conn-list-decomp(1-3)$ )
have  $c\text{-in-}c'\text{-only}\ c\ c' (conn\ c\ [conn\ c'\ l_1, \varphi_2])$  using  $c_1c\ l\ only\ \varphi_1$  by auto
moreover have  $not\text{-}c\text{-in-}c'\text{-symb}\ c\ c' (conn\ c\ [conn\ c'\ l_1, \varphi_2])$ 
using  $l_1\ \varphi_1\ c_1c\ l\ local.wf\ not\text{-}c\text{-in-}c'\text{-symb}\ l\ wf\varphi_1$  by blast
ultimately show False using  $\varphi_1\ c_1c\ l\ l_1\ local.wf\ not\text{-}c\text{-in-}c'\text{-simp}(4)\ wf\varphi_1$  by blast
qed
then have  $(\varphi_1 = conn\ c\ l_1 \wedge wf-conn\ c\ l_1) \vee (\exists \psi_1. \varphi_1 = FNot\ \psi_1) \vee simple\ \varphi_1$ 
by (metis  $\varphi_1\ assms(1-3)\ c_1\text{-eq}\ c_1\text{-imp}\ simple.elims(3)\ wf\varphi_1\ wf-conn-list(4)\ wf-conn-list(5-7)$ )
moreover {
assume  $\varphi_1 = conn\ c\ l_1 \wedge wf-conn\ c\ l_1$ 
then have only-c-inside  $c\ \varphi_1$ 
by (metis  $IH\varphi_1\ \varphi_1\ all\text{-subformula}\text{-st-decomp-imp}\ inc\varphi_1\ no-equiv\ no-equiv\text{-def}\ no-imp\ no-imp\text{-def}$ 
 $c\text{-in-}only\varphi_1\ only\text{-c-inside}\text{-def}\ only\text{-c-inside-into-}only\text{-c-inside}\ simple\text{-not}\ simple\text{-not}\text{-def}$ 
 $subformula\text{-all}\ subformula\text{-st}$ )
}
moreover {
assume  $\exists \psi_1. \varphi_1 = FNot\ \psi_1$ 
then obtain  $\psi_1$  where  $\varphi_1 = FNot\ \psi_1$  by metis
then have only-c-inside  $c\ \varphi_1$ 
by (metis  $all\text{-subformula}\text{-st}\text{-def}\ assms(1)\ connective.distinct(37,39)\ inc\varphi_1$ 
 $only\text{-c-inside}\text{-decomp-not}\ simple\text{-not}\ simple\text{-not}\text{-def}\ simple\text{-not}\text{-symb.simps}(1))$ 
}
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  $c \ [\varphi 1, \text{conn } c' \ l2]$ )
  using  $c2c \ l \ \text{only } \varphi 2$  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  $c \ [\varphi 1, \text{conn } c' \ l2]$ )
  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 ( $\varphi 2 = \text{conn } c \ l2 \wedge wf\text{-conn } c \ l2$ )  $\vee (\exists \psi 2. \varphi 2 = FNot \ \psi 2) \vee \text{simple } \varphi 2$ 
  using  $c2\text{-eq}$  by (metis  $\varphi 2 \ assms(1-3) \ c2\text{-eq } c2\text{-imp } \text{simple.elims}(3) \ wf \ \varphi 2 \ wf\text{-conn-list}(4-7)$ )
moreover {
  assume  $\varphi 2 = \text{conn } c \ l2 \wedge wf\text{-conn } c \ l2$ 
  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  $\exists \psi 2. \varphi 2 = FNot \ \psi 2$ 
  then obtain  $\psi 2$  where  $\varphi 2 = FNot \ \psi 2$  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  $l \ \text{only-c-inside} \varphi 1 \ \text{only-c-inside} \varphi 2$  by auto
qed

```

## Push Conjunction

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

**lemma** *pushConj-consistent: preserve-models 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  $\varphi$  and no-imp  $\varphi$  and no-T-F-except-top-level  $\varphi$  and simple-not  $\varphi$*   
**shows** *no-equiv  $\psi$  and no-imp  $\psi$  and no-T-F-except-top-level  $\psi$  and simple-not  $\psi$*   
**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  $\varphi$  and*  
*no-imp  $\varphi$  and*  
*full (propo-rew-step pushConj)  $\varphi \psi$  and*  
*no-T-F-except-top-level  $\varphi$  and*  
*simple-not  $\varphi$*   
**shows** *and-in-or-only  $\psi$*   
**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: preserve-models 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  $\varphi$  and no-imp  $\varphi$  and no-T-F-except-top-level  $\varphi$  and simple-not  $\varphi$*   
**shows** *no-equiv  $\psi$  and no-imp  $\psi$  and no-T-F-except-top-level  $\psi$  and simple-not  $\psi$*   
**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  $\varphi$  and*  
*no-imp  $\varphi$  and*  
*full (propo-rew-step pushDisj)  $\varphi \psi$  and*  
*no-T-F-except-top-level  $\varphi$  and*  
*simple-not  $\varphi$*   
**shows** *or-in-and-only  $\psi$*

**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)*)

## 1.6 The Full Transformations

### 1.6.1 Abstract Definition

The normal form is a super group of groups

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

*only-c-inside-symb* *c* (*conn* *c'* [ $\varphi 1$ ,  $\varphi 2$ ])  $\Rightarrow$  *c'* = *CAnd*  $\vee$  *c'* = *COr*  $\Rightarrow$  *wf-conn* *c'* [ $\varphi 1$ ,  $\varphi 2$ ]  
 $\Rightarrow$  *c'* = *c*  
**by** (*induct* *conn* *c'* [ $\varphi 1$ ,  $\varphi 2$ ] *rule*: *only-c-inside-symb.induct*, *auto simp: conn-inj*)

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

*only-c-inside* *c* (*conn* *c'* [ $\varphi 1$ ,  $\varphi 2$ ])  $\Rightarrow$  *c'* = *CAnd*  $\vee$  *c'* = *COr*  $\Rightarrow$  *wf-conn* *c'* [ $\varphi 1$ ,  $\varphi 2$ ]  $\Rightarrow$  *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*  $\varphi \Rightarrow$  *grouped-by* *c*  $\varphi$  (**is** ?*O*  $\varphi \Rightarrow$  ?*G*  $\varphi$ )

**proof** (*induct*  $\varphi$  *rule*: *propo-induct-arity*)

**case** (*nullary*  $\varphi$  *x*)  
**then show** ?*G*  $\varphi$  **by** *auto*

**next**

**case** (*unary*  $\psi$ )  
**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**  $\varphi$ -*conn*:  $\varphi =$  *conn* *c* [ $\varphi 1$ ,  $\varphi 2$ ] **and** *wf*: *wf-conn* *c* [ $\varphi 1$ ,  $\varphi 2$ ]

**proof** –

**obtain** *c'' l''* **where**  $\varphi$ -*c''*:  $\varphi =$  *conn* *c''* *l''* **and** *wf*: *wf-conn* *c''* *l''*

**using** *exists-c-conn* **by** *metis*

**then have** *l''*: *l''* = [ $\varphi 1$ ,  $\varphi 2$ ] **using**  $\varphi$  **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*  $\varphi$ -*c'' l''* **by** *metis*

**then have** *c* = *c''*

```

    by (metis  $\varphi$   $\varphi$ -c'' conn-inj conn-inj-not(2) l'' list.distinct(1) list.inject wf
        only-c-inside-symb.cases simple.simps(5-8))
  then show  $\varphi = \text{conn } c \ [\varphi 1, \varphi 2]$  and wf-conn  $c \ [\varphi 1, \varphi 2]$  using  $\varphi$ -c'' wf l'' by auto
qed
have grouped-by  $c \ \varphi 1$  using wf IH $\varphi 1$  IH $\varphi 2$   $\varphi$ -conn only  $\varphi$  unfolding only-c-inside-def by auto
moreover have grouped-by  $c \ \varphi 2$ 
  using wf  $\varphi$  IH $\varphi 1$  IH $\varphi 2$   $\varphi$ -conn only unfolding only-c-inside-def by auto
ultimately show ?G  $\varphi$  using  $\varphi$ -conn connected-is-group local.wf by blast
qed

```

**lemma** grouped-by-false:

```

grouped-by  $c \ (\text{conn } c' \ [\varphi, \psi]) \implies c \neq c' \implies \text{wf-conn } c' \ [\varphi, \psi] \implies \text{False}$ 
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  $c \ \varphi \implies \text{super-grouped-by } c \ c' \ \varphi \mid$   
 connected-is-super-group: super-grouped-by  $c \ c' \ \varphi \implies \text{super-grouped-by } c \ c' \ \psi \implies \text{wf-conn } c \ [\varphi, \psi]$   
 $\implies \text{super-grouped-by } c \ c' \ (\text{conn } c' \ [\varphi, \psi])$

**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  $\varphi \implies \text{no-imp } \varphi \implies \text{simple-not } \varphi \implies \text{c-in-c'-only } c \ c' \ \varphi$ 
 $\implies \text{super-grouped-by } c \ c' \ \varphi$ 
(is ?NE  $\varphi \implies ?NI \ \varphi \implies ?SN \ \varphi \implies ?C \ \varphi \implies ?S \ \varphi$ )

```

**proof** (induct  $\varphi$  rule: propo-induct-arity)

```

case (nullary  $\varphi \ x$ )
then show ?S  $\varphi$  by auto

```

**next**

```

case (unary  $\varphi$ )
then have simple-not-symb (FNot  $\varphi$ )
  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  $\varphi$ , auto)
then show ?S (FNot  $\varphi$ ) by auto

```

**next**

```

case (binary  $\varphi \ \varphi 1 \ \varphi 2$ )
note IH $\varphi 1 = \text{this}(1)$  and IH $\varphi 2 = \text{this}(2)$  and no-equiv = this(4) and no-imp = this(5)
and simpleN = this(6) and c-in-c'-only = this(7) and  $\varphi' = \text{this}(3)$ 
{
  assume  $\varphi = FImp \ \varphi 1 \ \varphi 2 \vee \varphi = FEq \ \varphi 1 \ \varphi 2$ 
  then have False using no-equiv no-imp by auto
  then have ?S  $\varphi$  by auto
}

```

```

moreover {
  assume  $\varphi$ :  $\varphi = \text{conn } c' [\varphi 1, \varphi 2] \wedge \text{wf-conn } c' [\varphi 1, \varphi 2]$ 
  have c-in-c'-only:  $c\text{-in-}c'\text{-only } c \ c' \ \varphi 1 \wedge c\text{-in-}c'\text{-only } c \ c' \ \varphi 2 \wedge c\text{-in-}c'\text{-symb } c \ c' \ \varphi$ 
    using c-in-c'-only  $\varphi'$  unfolding c-in-c'-only-def by auto
  have super-grouped-by  $c \ c' \ \varphi 1$  using  $\varphi \ c'$  no-equiv no-imp simpleN IH  $\varphi 1$  c-in-c'-only by auto
  moreover have super-grouped-by  $c \ c' \ \varphi 2$ 
    using  $\varphi \ c'$  no-equiv no-imp simpleN IH  $\varphi 2$  c-in-c'-only by auto
  ultimately have  $?S \ \varphi$ 
    using super-grouped-by.intros(2)  $\varphi$  by (metis c wf-conn-helper-facts(5,6))
}
moreover {
  assume  $\varphi$ :  $\varphi = \text{conn } c [\varphi 1, \varphi 2] \wedge \text{wf-conn } c [\varphi 1, \varphi 2]$ 
  then have only-c-inside  $c \ \varphi 1 \wedge \text{only-c-inside } c \ \varphi 2$ 
    using c-in-c'-symb-c-implies-only-c-inside  $c \ c' \ c\text{-in-}c'\text{-only}$  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  $c \ (\text{conn } c [\varphi 1, \varphi 2])$ 
    unfolding only-c-inside-def using  $\varphi$ 
    by (simp add: only-c-inside-into-only-c-inside all-subformula-st-decomp)
  then have grouped-by  $c \ \varphi$  using  $\varphi$  only-c-inside-imp-grouped-by  $c$  by blast
  then have  $?S \ \varphi$  using super-grouped-by.intros(1) by metis
}
ultimately show  $?S \ \varphi$  by (metis  $\varphi' \ c \ c' \ cc' \text{conn.simps}$ (5,6) wf-conn-helper-facts(5,6))
qed

```

## 1.6.2 Conjunctive Normal Form

### Definition

**definition** *is-conj-with-TF* **where** *is-conj-with-TF* == *super-grouped-by COr CAnd*

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

**shows**  $\text{no-equiv } \varphi \implies \text{no-imp } \varphi \implies \text{simple-not } \varphi \implies \text{or-in-and-only } \varphi \implies \text{is-conj-with-TF } \varphi$   
**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-equivalent: preserve-models cnf-rew*

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

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



```

apply (unfold cnf-rew-def OO-def)
apply auto
proof –
  fix  $\varphi Eq \varphi Imp \varphi TB \varphi Neg \varphi Disj :: 'v \text{ propo}$ 
  assume  $Eq$ : full (propo-rew-step elim-equiv)  $\varphi \varphi Eq$ 
  then have no-equiv: no-equiv  $\varphi Eq$  using no-equiv-full-propo-rew-step-elim-equiv by blast

  assume  $Imp$ : full (propo-rew-step elim-imp)  $\varphi Eq \varphi Imp$ 
  then have no-imp: no-imp  $\varphi Imp$  using no-imp-full-propo-rew-step-elim-imp by blast
  have no-imp-inv: no-equiv  $\varphi Imp$  using no-equiv  $Imp$  elim-imp-inv by blast

  assume  $TB$ : full (propo-rew-step elimTB)  $\varphi Imp \varphi TB$ 
  then have noTB: no-T-F-except-top-level  $\varphi TB$ 
    using no-imp-inv no-imp elimTB-full-propo-rew-step by blast
  have noTB-inv: no-equiv  $\varphi TB$  no-imp  $\varphi TB$  using elimTB-inv  $TB$  no-imp no-imp-inv by blast+

  assume  $Neg$ : full (propo-rew-step pushNeg)  $\varphi TB \varphi Neg$ 
  then have noNeg: simple-not  $\varphi Neg$ 
    using noTB-inv noTB pushNeg-full-propo-rew-step by blast
  have noNeg-inv: no-equiv  $\varphi Neg$  no-imp  $\varphi Neg$  no-T-F-except-top-level  $\varphi Neg$ 
    using pushNeg-inv  $Neg$  noTB noTB-inv by blast+

  assume  $Disj$ : full (propo-rew-step pushDisj)  $\varphi Neg \varphi Disj$ 
  then have noDisj: or-in-and-only  $\varphi Disj$ 
    using noNeg-inv noNeg pushDisj-full-propo-rew-step by blast
  have noDisj-inv: no-equiv  $\varphi Disj$  no-imp  $\varphi Disj$  no-T-F-except-top-level  $\varphi Disj$ 
    simple-not  $\varphi Disj$ 
  using pushDisj-inv  $Disj$  noNeg noNeg-inv by blast+

  moreover have is-conj-with-TF  $\varphi Disj$ 
    using or-in-and-only-conjunction-in-disj noDisj-inv noDisj by blast
  ultimately show is-cnf  $\varphi Disj$  unfolding is-cnf-def by blast
qed

```

### 1.6.3 Disjunctive Normal Form

#### Definition

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

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

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

#### Full DNF transform

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

**definition** *dnf-rew* **where** *dnf-rew*  $\equiv$   
 (full (propo-rew-step elim-equiv)) OO  
 (full (propo-rew-step elim-imp)) OO  
 (full (propo-rew-step elimTB)) OO

(full (propo-rew-step pushNeg)) OO  
 (full (propo-rew-step pushConj))

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

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

**theorem** *dnf-transformation-correction:*

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

**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))

## 1.7 More aggressive simplifications: Removing true and false at the beginning

### 1.7.1 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  $\varphi$  FT)  $\varphi$  |*  
*ElimTBFull1'*[simp]: *elimTBFull (FAnd FT  $\varphi$ )  $\varphi$  |*

*ElimTBFull2*[simp]: *elimTBFull (FAnd  $\varphi$  FF) FF |*  
*ElimTBFull2'*[simp]: *elimTBFull (FAnd FF  $\varphi$ ) FF |*

*ElimTBFull3*[simp]: *elimTBFull (FOr  $\varphi$  FT) FT |*  
*ElimTBFull3'*[simp]: *elimTBFull (FOr FT  $\varphi$ ) FT |*

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

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

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

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

The transformation is still consistent.

**lemma** *elimTBFull-consistent: preserve-models elimTBFull*

**proof** –

{  
**fix**  $\varphi \psi:: 'b$  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 preserve-models-def by auto
qed

```

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

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

```

  fixes  $\varphi :: 'v \text{ propo}$ 
  shows  $\psi \preceq \varphi \implies \neg \text{no-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTBFull } \psi \ \psi'$ 
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  $\text{Ex } (\text{elimTBFull } \varphi')$  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  $\text{Ex } (\text{elimTBFull } (F\text{Not } \psi))$  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  $\text{Ex } (\text{elimTBFull } \varphi')$  using elimTBFull.intros binary.hyps(3) by blast
qed

```

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

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

```

  fixes  $\varphi :: 'v \text{ propo}$ 
  assumes noTB:  $\neg \text{no-T-F-except-top-level } \varphi$ 
  shows  $\exists \psi \ \psi'. \psi \preceq \varphi \wedge \text{elimTBFull } \psi \ \psi'$ 
proof -
  have test-symb-false-nullary:
     $\forall x. \text{no-T-F-symb-except-toplevel } (FF :: 'v \text{ propo}) \wedge \text{no-T-F-symb-except-toplevel } FT$ 
     $\wedge \text{no-T-F-symb-except-toplevel } (F\text{Var } (x :: 'v))$ 
  by auto
  moreover {
    fix  $c :: 'v \text{ connective}$  and  $l :: 'v \text{ propo list}$  and  $\psi :: 'v \text{ propo}$ 
    have  $H: \text{elimTBFull } (\text{conn } c \ l) \ \psi \implies \neg \text{no-T-F-symb-except-toplevel } (\text{conn } c \ l)$ 
      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

```

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

```

**next**

```

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

```

**moreover** {

```

assume  $c: c \in \text{binary-connectives}$ 
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
 $\text{tl-append2}$ )
have  $\zeta: \forall \zeta \in \text{set } (\xi\ @\ \varphi\ \#\ \xi'). \text{no-T-F } \zeta$ 
using no-T-F unfolding no-T-F-def using corr all-subformula-st-decomp by blast

```

**then have**  $\varphi': \text{no-T-F } \varphi'$  **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)*
 $\text{butlast-append list.distinct(1) list.sel(3)}$ )

**then have**  $\forall \zeta \in \text{set } (\xi\ @\ \varphi'\ \#\ \xi'). \text{no-T-F } \zeta$  **using**  $\zeta\ \varphi'\ ab$  **by** *fastforce*

**moreover**

**have**  $\forall \zeta \in \text{set } (\xi\ @\ \varphi\ \#\ \xi'). \zeta \neq \text{FT} \wedge \zeta \neq \text{FF}$ 
**using**  $\zeta\ \text{corr no-T-F no-T-F-except-top-level-false no-T-F-no-T-F-except-top-level}$  **by** *blast*

**then have**  $\text{no-T-F-symb } (\text{conn } c\ (\xi\ @\ \varphi'\ \#\ \xi'))$

**by** (*metis*  $\varphi'\ l' ab \text{all-subformula-st-test-symb-true-phi } c\ \text{list.distinct(1)}$

$\text{list.set-intros(1,2) no-T-F-symb-except-toplevel-bin-decom}$

$\text{no-T-F-symb-except-toplevel-no-T-F-symb no-T-F-symb-false(1,2) no-T-F-def wf-conn-binary}$

$\text{wf-conn-list(1,2)}$ )

**ultimately have**  $\text{no-T-F } (\text{conn } c\ (\xi\ @\ \varphi'\ \#\ \xi'))$

```

    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  $\text{corr}$  by auto
    ultimately show  $\text{no-}T\text{-}F\text{-symb } (\text{conn } c (\xi @ \zeta' \# \xi'))$ 
    by (metis  $\text{corr}$  wf-conn-no-arity-change wf-conn-no-arity-change-helper no- $T\text{-}F\text{-symb-comp$ )
  qed
}
ultimately show  $\text{no-}T\text{-}F\text{-except-top-level } \psi$ 
using  $\text{full-propo-rew-step-inv-stay-with-inc}$  [of  $\text{elim-equiv no-}T\text{-}F\text{-symb-except-toplevel } \varphi$ ]
   $\text{assms subformula-refl unfolding no-}T\text{-}F\text{-except-top-level-def}$  by metis
qed

```

**lemma** *propo-rew-step-ElimImp-no-T-F*:  $\text{propo-rew-step elim-imp } \varphi \psi \implies \text{no-}T\text{-}F \varphi \implies \text{no-}T\text{-}F \psi$   
**proof** (induct rule: *propo-rew-step.induct*)

```

  case (global-rel  $\varphi' \psi'$ )
  then show  $\text{no-}T\text{-}F \psi'$ 
    using  $\text{elim-imp.cases no-}T\text{-}F\text{-comp-not no-}T\text{-}F\text{-decomp}(1,2)$ 
    by (metis  $\text{no-}T\text{-}F\text{-comp-expanded-explicit}(2))$ 
next
  case (propo-rew-one-step-lift  $\varphi \varphi' c \xi \xi'$ )
  note  $\text{rel} = \text{this}(1)$  and  $\text{IH} = \text{this}(2)$  and  $\text{corr} = \text{this}(3)$  and  $\text{no-}T\text{-}F = \text{this}(4)$ 
  {
    assume  $c: c = CNot$ 
    then have  $\text{empty}: \xi = [] \ \xi' = []$  using  $\text{corr}$  by auto
    then have  $\text{no-}T\text{-}F \varphi$  using  $\text{no-}T\text{-}F c \text{ no-}T\text{-}F\text{-decomp-not}$  by auto
    then have  $\text{no-}T\text{-}F (\text{conn } c (\xi @ \varphi' \# \xi'))$  using  $c \text{ empty no-}T\text{-}F\text{-comp-not IH}$  by auto
  }
  moreover {
    assume  $c: c \in \text{binary-connectives}$ 
    then obtain  $a \ b$  where  $\text{ab}: \xi @ \varphi \# \xi' = [a, b]$ 
      using  $\text{corr list-length2-decomp wf-conn-bin-list-length}$  by metis
    then have  $\varphi: \varphi = a \vee \varphi = b$ 
      by (metis  $\text{append-self-conv2 wf-conn-list-decomp}(4) \text{ wf-conn-unary list.discI list.sel}(3)$ 
         $\text{nth-Cons-0 tl-append2}$ )
    have  $\zeta: \forall \zeta \in \text{set } (\xi @ \varphi \# \xi'). \text{no-}T\text{-}F \zeta$  using  $\text{ab } c \text{ propo-rew-one-step-lift.prem}$  by auto

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

```

```

    then have no-T-F (conn c (ξ @ φ' # ξ')) by auto
  }
  ultimately show no-T-F (conn c (ξ @ φ' # ξ')) using corr wf-conn.cases by blast
qed

```

lemma *elim-imp-inv'*:

```

  fixes φ ψ :: 'v propo
  assumes full (propo-rew-step elim-imp) φ ψ and no-T-F-except-top-level φ
  shows no-T-F-except-top-level ψ
proof -
  {
    {
      fix φ ψ :: 'v propo
      have H: elim-imp φ ψ ⇒ no-T-F-except-top-level φ ⇒ no-T-F-except-top-level ψ
        by (induct φ ψ rule: elim-imp.induct, auto)
    } note H = this
    fix φ ψ :: 'v propo
    have propo-rew-step elim-imp φ ψ ⇒ no-T-F-except-top-level φ ⇒ no-T-F-except-top-level ψ
    proof -
      assume rel: propo-rew-step elim-imp φ ψ
      and no: no-T-F-except-top-level φ
      {
        assume φ = FT ∨ φ = 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 ψ by blast
      }
      moreover {
        assume φ ≠ FT ∧ φ ≠ FF
        then have no-T-F φ
          by (metis no no-T-F-symb-except-toplevel-all-subformula-st-no-T-F-symb)
        then have no-T-F ψ
          using rel propo-rew-step-ElimImp-no-T-F by blast
        then have no-T-F-except-top-level ψ by (simp add: no-T-F-no-T-F-except-top-level)
      }
      ultimately show no-T-F-except-top-level ψ by metis
    qed
  }
  moreover {
    fix c :: 'v connective and ξ ξ' :: 'v propo list and ζ ζ' :: 'v propo
    assume rel: propo-rew-step elim-imp ζ ζ'
    and incl: ζ ≤ φ
    and corr: wf-conn c (ξ @ ζ # ξ')
    and no-T-F: no-T-F-symb-except-toplevel (conn c (ξ @ ζ # ξ'))
    and n: no-T-F-symb-except-toplevel ζ'
    have no-T-F-symb-except-toplevel (conn c (ξ @ ζ' # ξ'))
    proof
      have p: no-T-F-symb (conn c (ξ @ ζ # ξ'))
        by (simp add: corr no-T-F no-T-F-symb-except-toplevel-no-T-F-symb wf-conn-list(1,2))

      have l: ∀ φ ∈ set (ξ @ ζ # ξ'). φ ≠ FT ∧ φ ≠ FF
        using corr wf-conn-no-T-F-symb-iff p by blast
      from rel incl have ζ' ≠ FT ∧ ζ' ≠ FF
      apply (induction ζ ζ' 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

```

### 1.7.3 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**

```

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**  $\text{dnf-rew}'\text{-consistent}$ : preserve-models  $\text{dnf-rew}'$

**by** (simp add:  $\text{dnf-rew}'\text{-def}$  elimEqv-lifted-consistant elim-imp-lifted-consistant  
elimTBFull-consistent preserve-models-OO pushConj-consistent pushNeg-lifted-consistant)

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

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

**unfolding**  $\text{dnf-rew}'\text{-def}$  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**  $\text{cnf-rew}' :: 'a \text{ propo} \Rightarrow 'a \text{ propo} \Rightarrow \text{bool}$  **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**  $\text{cnf-rew}'\text{-consistent}$ : preserve-models  $\text{cnf-rew}'$

**by** (simp add:  $\text{cnf-rew}'\text{-def}$  elimEqv-lifted-consistant elim-imp-lifted-consistant  
elimTBFull-consistent preserve-models-OO pushDisj-consistent pushNeg-lifted-consistant)

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

$\text{cnf-rew}' \varphi \varphi' \implies \text{is-cnf } \varphi'$

**unfolding**  $\text{cnf-rew}'\text{-def}$  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** *Nested-Multisets-Ordinals.Multiset-More Prop-Normalisation*  
*Entailment-Definition.Partial-Herbrand-Interpretation*  
**begin**

## 1.8 Link with Multiset Version

### 1.8.1 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* = {}

### 1.8.2 Equisatisfiability of the two Versions

**lemma** *is-conj-with-TF-FNot*:  
*is-conj-with-TF* (*FNot*  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = \textit{FVar } v \vee \varphi = \textit{FF} \vee \varphi = \textit{FT}$ )  
**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*  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = \textit{FVar } v \vee \varphi = \textit{FF} \vee \varphi = \textit{FT}$ )  
**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*]:  $\neg no-T-F \textit{FF}$  **and**  
 $no-T-F \textit{FT}$ [*simp*]:  $\neg no-T-F \textit{FT}$   
**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]:  $\neg$  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  $\{\text{mset-of-conj } \varphi\}$ )  
**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: ( $\models$ ) uses a function assigning *True* or *False*, while ( $\models_s$ ) uses a set where being in the list means entailment of a literal.

```

theorem cnf-eval-true-clss:
  fixes  $\varphi :: 'v \text{ propo}$ 
  assumes is-cnf  $\varphi$ 
  shows  $\text{eval } A \ \varphi \longleftrightarrow \text{Partial-Herbrand-Interpretation.true-clss } (\{\text{Pos } v \mid v. A \ v\} \cup \{\text{Neg } v \mid v. \neg A \ v\})$ 
    (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  $\varphi = \{\text{mset-of-conj } \varphi\}$  mset-of-formula  $\psi = \{\text{mset-of-conj } \psi\}$ 
    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

```

```

function formula-of-mset :: 'a clause  $\Rightarrow$  'a propo where
  (formula-of-mset  $\varphi =$ 
    (if  $\varphi = \{\#\}$  then FF
      else
        let  $v = (\text{SOME } v. v \in \# \ \varphi);$ 
         $v' = (\text{if } \text{is-pos } v \text{ then } \text{FVar } (\text{atm-of } v) \text{ else } \text{FNot } (\text{FVar } (\text{atm-of } v)))$  in
        if remove1-mset  $v \ \varphi = \{\#\}$  then  $v'$ 
        else FOr  $v' (\text{formula-of-mset } (\text{remove1-mset } v \ \varphi)))$ )

```

```

by auto
termination
  apply (relation ⟨measure size⟩)
  apply (auto simp: size-mset-remove1-mset-le-iff)
  by (meson multiset-nonemptyE someI-ex)

lemma formula-of-mset-empty[simp]: ⟨formula-of-mset {#} = FF⟩
  by (auto simp: Let-def)

lemma formula-of-mset-empty-iff[iff]: ⟨formula-of-mset  $\varphi$  = FF  $\longleftrightarrow$   $\varphi$  = {#}⟩
  by (induction  $\varphi$ ) (auto simp: Let-def)

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

function formula-of-msets :: 'a literal multiset set  $\Rightarrow$  'a propo where
  ⟨formula-of-msets  $\varphi$ s =
    (if  $\varphi$ s = {}  $\vee$  infinite  $\varphi$ s then FT
     else
       let  $v$  = (SOME  $v$ .  $v \in \varphi$ s);
        $v'$  = formula-of-mset  $v$  in
       if  $\varphi$ s - { $v$ } = {} then  $v'$ 
       else FAnd  $v'$  (formula-of-msets ( $\varphi$ s - { $v$ })))⟩
  by auto
termination
  apply (relation ⟨measure card⟩)
  apply (auto simp: some-in-eq)
  by (metis all-not-in-conv card-gt-0-iff diff-less lessI)

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

lemma remove1-mset-empty-iff:
  ⟨remove1-mset  $v$   $\varphi$  = {#}  $\longleftrightarrow$  ( $\varphi$  = {#}  $\vee$   $\varphi$  = {# $v$ #})⟩
  using remove1-mset-eqE by force

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

lemma grouped-by-COr-formula-of-mset: ⟨grouped-by COr (formula-of-mset  $\varphi$ )⟩
proof (induction ⟨size  $\varphi$ ⟩ arbitrary:  $\varphi$ )
  case 0
  then show ?case by (subst formula-of-mset.simps) (auto simp: Let-def)
next
  case (Suc  $n$ ) note IH = this(1) and s = this(2)
  then have ⟨ $n$  = size (remove1-mset (SOME  $v$ .  $v \in \# \varphi$ )  $\varphi$ )⟩ if ⟨ $\varphi \neq \{ \# \}$ ⟩
    using that by (auto simp: size-Diff-singleton-if some-in-eq)
  then show ?case
    using IH[of ⟨remove1-mset (SOME  $v$ .  $v \in \# \varphi$ )  $\varphi$ ]
    by (subst formula-of-mset.simps) (auto simp: Let-def grouped-by-COr-FOr)
qed
lemma no-T-F-formula-of-mset: ⟨no-T-F (formula-of-mset  $\varphi$ )⟩ if ⟨formula-of-mset  $\varphi \neq FF$ ⟩ for  $\varphi$ 
  using that
proof (induction ⟨size  $\varphi$ ⟩ arbitrary:  $\varphi$ )
  case 0
  then show ?case by (subst formula-of-mset.simps) (auto simp: Let-def no-T-F-def
    all-subformula-st-def)
next

```

**case** (Suc n) **note** IH = this(1) **and** s = this(2) **and** FF = this(3)  
**then have**  $\langle n = \text{size } (\text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi) \rangle$  **if**  $\langle \varphi \neq \{\#\} \rangle$   
**using that by** (auto simp: size-Diff-singleton-if some-in-eq)  
**moreover have**  $\langle \text{no-T-F } (\text{FVar } (\text{atm-of } (\text{SOME } v. v \in \# \varphi))) \rangle$   
**by** (auto simp: no-T-F-def)  
**ultimately show** ?case  
**using** IH[of  $\langle \text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi \rangle$ ] FF  
**by**(subst formula-of-mset.simps) (auto simp: Let-def grouped-by-COr-FOr)  
**qed**

**lemma** mset-of-conj-formula-of-mset[simp]:  $\langle \text{mset-of-conj}(\text{formula-of-mset } \varphi) = \varphi \rangle$  **for**  $\varphi$   
**proof** (induction  $\langle \text{size } \varphi \rangle$  arbitrary:  $\varphi$ )

**case** 0  
**then show** ?case **by** (subst formula-of-mset.simps) (auto simp: Let-def no-T-F-def  
all-subformula-st-def)

**next**

**case** (Suc n) **note** IH = this(1) **and** s = this(2)  
**then have**  $\langle n = \text{size } (\text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi) \rangle$  **if**  $\langle \varphi \neq \{\#\} \rangle$   
**using that by** (auto simp: size-Diff-singleton-if some-in-eq)  
**moreover have**  $\langle \text{no-T-F } (\text{FVar } (\text{atm-of } (\text{SOME } v. v \in \# \varphi))) \rangle$   
**by** (auto simp: no-T-F-def)  
**ultimately show** ?case  
**using** IH[of  $\langle \text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi \rangle$ ]  
**by**(subst formula-of-mset.simps) (auto simp: some-in-eq Let-def grouped-by-COr-FOr remove1-mset-empty-iff)  
**qed**

**lemma** mset-of-formula-formula-of-mset [simp]:  $\langle \text{mset-of-formula } (\text{formula-of-mset } \varphi) = \{\varphi\} \rangle$  **for**  $\varphi$   
**proof** (induction  $\langle \text{size } \varphi \rangle$  arbitrary:  $\varphi$ )

**case** 0  
**then show** ?case **by** (subst formula-of-mset.simps) (auto simp: Let-def no-T-F-def  
all-subformula-st-def)

**next**

**case** (Suc n) **note** IH = this(1) **and** s = this(2)  
**then have**  $\langle n = \text{size } (\text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi) \rangle$  **if**  $\langle \varphi \neq \{\#\} \rangle$   
**using that by** (auto simp: size-Diff-singleton-if some-in-eq)  
**moreover have**  $\langle \text{no-T-F } (\text{FVar } (\text{atm-of } (\text{SOME } v. v \in \# \varphi))) \rangle$   
**by** (auto simp: no-T-F-def)  
**ultimately show** ?case  
**using** IH[of  $\langle \text{remove1-mset } (\text{SOME } v. v \in \# \varphi) \varphi \rangle$ ]  
**by**(subst formula-of-mset.simps) (auto simp: some-in-eq Let-def grouped-by-COr-FOr remove1-mset-empty-iff)  
**qed**

**lemma** formula-of-mset-is-cnf:  $\langle \text{is-cnf } (\text{formula-of-mset } \varphi) \rangle$

**by** (auto simp: is-cnf-def is-conj-with-TF-def grouped-by-COr-formula-of-mset no-T-F-formula-of-mset  
intro!: grouped-is-super-grouped)

**lemma** eval-clss-iff:

**assumes**  $\langle \text{consistent-interp } A \rangle$  **and**  $\langle \text{total-over-set } A \text{ UNIV} \rangle$   
**shows**  $\langle \text{eval } (\text{fun-of-set } A) (\text{formula-of-mset } \varphi) \longleftrightarrow \text{Partial-Herbrand-Interpretation.true-clss } A \{\varphi\} \rangle$   
**apply** (subst cnf-eval-true-clss[OF formula-of-mset-is-cnf])  
**using** assms  
**apply** (auto simp add: true-clss-def fun-of-set-def consistent-interp-def total-over-set-def)  
**apply** (case-tac L)  
**by** (fastforce simp add: true-clss-def fun-of-set-def consistent-interp-def total-over-set-def)+

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

*is-conj-with-TF* (*FAnd*  $\varphi 1$   $\varphi 2$ )  $\longleftrightarrow$  *is-conj-with-TF*  $\varphi 1 \wedge$  *is-conj-with-TF*  $\varphi 2$   
**unfolding** *is-conj-with-TF-def* **by** (*subst super-grouped-by.simps*) *auto*

**lemma** *is-CNF-Fand*:

$\langle \text{is-cnf } (FAnd \varphi \psi) \longleftrightarrow (\text{is-cnf } \varphi \wedge \text{no-T-F } \varphi) \wedge \text{is-cnf } \psi \wedge \text{no-T-F } \psi \rangle$   
**by** (*auto simp: is-cnf-def is-conj-with-TF-Fand-iff*)

**lemma** *no-T-F-formula-of-mset-iff*:  $\langle \text{no-T-F } (\text{formula-of-mset } \varphi) \longleftrightarrow \varphi \neq \{\#\} \rangle$

**proof** (*induction*  $\langle \text{size } \varphi \rangle$  *arbitrary:*  $\varphi$ *)*

**case** 0

**then show** ?*case* **by** (*subst formula-of-mset.simps*) (*auto simp: Let-def no-T-F-def all-subformula-st-def*)

**next**

**case** (*Suc* *n*) **note** *IH* = *this*(1) **and** *s* = *this*(2)

**then have**  $\langle n = \text{size } (\text{remove1-mset } (SOME \ v. \ v \in \# \ \varphi) \ \varphi) \rangle$  **if**  $\langle \varphi \neq \{\#\} \rangle$

**using** *that* **by** (*auto simp: size-Diff-singleton-if some-in-eq*)

**moreover have**  $\langle \text{no-T-F } (FVar \ (\text{atm-of } (SOME \ v. \ v \in \# \ \varphi))) \rangle$

**by** (*auto simp: no-T-F-def*)

**ultimately show** ?*case*

**using** *IH*[*of*  $\langle \text{remove1-mset } (SOME \ v. \ v \in \# \ \varphi) \ \varphi \rangle$ ]

**by**(*subst formula-of-mset.simps*) (*auto simp: some-in-eq Let-def grouped-by-COr-FOr remove1-mset-empty-iff*)  
**qed**

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

**assumes**  $\langle \text{finite } \varphi \rangle$  **and**  $\langle \{\#\} \notin \varphi \rangle$  **and**  $\langle \varphi \neq \{\} \rangle$

**shows**  $\langle \text{no-T-F } (\text{formula-of-msets } \varphi) \rangle$

**using** *assms* **apply** (*induction*  $\langle \text{card } \varphi \rangle$  *arbitrary:*  $\varphi$ *)*

**subgoal by** (*subst formula-of-msets.simps*) (*auto simp: no-T-F-def all-subformula-st-def*)[]

**subgoal**

**apply** (*subst formula-of-msets.simps*)

**apply** (*auto split: simp: Let-def formula-of-mset-is-cnf is-CNF-Fand no-T-F-formula-of-mset-iff some-in-eq*)

**apply** (*metis* (*mono-tags*, *lifting*) *some-eq-ex*)

**done**

**done**

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

**assumes**  $\langle \text{finite } \varphi \rangle$  **and**  $\langle \{\#\} \notin \varphi \rangle$

**shows**  $\langle \text{is-cnf } (\text{formula-of-msets } \varphi) \rangle$

**using** *assms* **apply** (*induction*  $\langle \text{card } \varphi \rangle$  *arbitrary:*  $\varphi$ *)*

**subgoal by** (*subst formula-of-msets.simps*) (*auto simp: is-cnf-def is-conj-with-TF-def*)[]

**subgoal**

**apply** (*subst formula-of-msets.simps*)

**apply** (*auto split: simp: Let-def formula-of-mset-is-cnf is-CNF-Fand no-T-F-formula-of-mset-iff some-in-eq intro: no-T-F-formula-of-msets*)

**apply** (*metis* (*mono-tags*, *lifting*) *some-eq-ex*)

**done**

**done**

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

**assumes**  $\langle \text{finite } \varphi \rangle$

**shows**  $\langle \text{mset-of-formula } (\text{formula-of-msets } \varphi) = \varphi \rangle$

**using** *assms* **apply** (*induction*  $\langle \text{card } \varphi \rangle$  *arbitrary:*  $\varphi$ *)*

**subgoal by** (*subst formula-of-msets.simps*) (*auto simp: is-cnf-def is-conj-with-TF-def*)[]

**subgoal**

**apply** (*subst formula-of-msets.simps*)

```

apply (auto split: simp: Let-def formula-of-mset-is-cnf is-CNF-Fand
        no-T-F-formula-of-mset-iff some-in-eq intro: no-T-F-formula-of-msets)
done
done

lemma
  assumes  $\langle \text{consistent-interp } A \rangle$  and  $\langle \text{total-over-set } A \text{ UNIV} \rangle$  and  $\langle \text{finite } \varphi \rangle$  and  $\langle \{\#\} \notin \varphi \rangle$ 
  shows  $\langle \text{eval } (\text{fun-of-set } A) (\text{formula-of-msets } \varphi) \longleftrightarrow \text{Partial-Herbrand-Interpretation.true-clss } A \varphi \rangle$ 
  apply (subst cnf-eval-true-clss[OF is-cnf-formula-of-msets[OF assms(3-4)]])
  using assms(3) unfolding mset-of-formula-formula-of-msets[OF assms(3)]
  by (induction  $\varphi$ )
    (use eval-clss-iff[OF assms(1,2)] in  $\langle \text{simp-all add: cnf-eval-true-clss formula-of-mset-is-cnf} \rangle$ )

end
theory Prop-Resolution
imports Entailment-Definition.Partial-Herbrand-Interpretation
         Weidenbach-Book-Base.WB-List-More
         Weidenbach-Book-Base.Wellfounded-More

begin

```





## Chapter 2

# Resolution-based techniques

This chapter contains the formalisation of resolution and superposition.

### 2.1 Resolution

#### 2.1.1 Simplification Rules

**inductive** *simplify* :: 'v clause-set  $\Rightarrow$  'v clause-set  $\Rightarrow$  bool **for** *N* :: 'v clause set **where**

*tautology-deletion*:

$add\_mset (Pos P) (add\_mset (Neg P) A) \in N \implies simplify\ N\ (N - \{add\_mset (Pos P) (add\_mset (Neg P) A)\})$

*condensation*:

$add\_mset L (add\_mset L A) \in N \implies simplify\ N\ (N - \{add\_mset L (add\_mset L A)\} \cup \{add\_mset L A\})$

*subsumption*:

$A \in N \implies A \subset\# B \implies B \in N \implies simplify\ N\ (N - \{B\})$

**lemma** *simplify-preserve-models'*:

**fixes** *N N'* :: 'v clause-set

**assumes** *simplify N N'*

**and** *total-over-m I N*

**shows**  $I \models_s N' \longrightarrow I \models_s N$

**using** *assms*

**proof** (*induct rule: simplify.induct*)

**case** (*tautology-deletion P A*)

**then have**  $I \models add\_mset (Pos P) (add\_mset (Neg P) A)$

**by** (*fastforce dest: mk-disjoint-insert*)

**then show** ?case **by** (*metis Un-Diff-cancel2 true-clss-singleton true-clss-union*)

**next**

**case** (*condensation A P*)

**then show** ?case

**by** (*fastforce dest: mk-disjoint-insert*)

**next**

**case** (*subsumption A B*)

**have**  $A \neq B$  **using** *subsumption.hyps(2)* **by** *auto*

**then have**  $I \models_s N - \{B\} \implies I \models A$  **using**  $\langle A \in N \rangle$  **by** (*simp add: true-clss-def*)

**moreover have**  $I \models A \implies I \models B$  **using**  $\langle A \subset\# B \rangle$  **by** *auto*

**ultimately show** ?case **by** (*metis insert-Diff-single true-clss-insert*)

**qed**

**lemma** *simplify-preserve-models*:

```

fixes  $N\ N' :: 'v\ \text{clause-set}$ 
assumes simplify  $N\ N'$ 
and total-over-m  $I\ N$ 
shows  $I \models_s N \longrightarrow I \models_s N'$ 
using assms apply (induct rule: simplify.induct)
using true-clss-def by fastforce+
```

**lemma** *simplify-preserve-models''*:

```

fixes  $N\ N' :: 'v\ \text{clause-set}$ 
assumes simplify  $N\ N'$ 
and total-over-m  $I\ N'$ 
shows  $I \models_s N \longrightarrow I \models_s N'$ 
using assms apply (induct rule: simplify.induct)
using true-clss-def by fastforce+
```

**lemma** *simplify-preserve-models-eq*:

```

fixes  $N\ N' :: 'v\ \text{clause-set}$ 
assumes simplify  $N\ N'$ 
and total-over-m  $I\ N$ 
shows  $I \models_s N \longleftrightarrow I \models_s N'$ 
using simplify-preserve-models simplify-preserve-models' assms by blast
```

**lemma** *simplify-preserves-finite*:

```

assumes simplify  $\psi\ \psi'$ 
shows finite  $\psi \longleftrightarrow \text{finite } \psi'$ 
using assms by (induct rule: simplify.induct, auto simp add: remove-def)
```

**lemma** *rtranclp-simplify-preserves-finite*:

```

assumes rtranclp simplify  $\psi\ \psi'$ 
shows finite  $\psi \longleftrightarrow \text{finite } \psi'$ 
using assms by (induct rule: rtranclp-induct) (auto simp add: simplify-preserves-finite)
```

**lemma** *simplify-atms-of-ms*:

```

assumes simplify  $\psi\ \psi'$ 
shows atms-of-ms  $\psi' \subseteq \text{atms-of-ms } \psi$ 
using assms unfolding atms-of-ms-def
proof (induct rule: simplify.induct)
  case (tautology-deletion  $A\ P$ )
  then show ?case by auto
next
  case (condensation  $P\ A$ )
  moreover have  $A + \{\#P\} + \{\#P\} \in \psi \implies \exists x \in \psi. \text{atm-of } P \in \text{atm-of } ' \text{set-mset } x$ 
    by (metis Un-iff atms-of-def atms-of-plus atms-of-singleton insert-iff)
  ultimately show ?case by (auto simp add: atms-of-def)
next
  case (subsumption  $A\ P$ )
  then show ?case by auto
qed
```

**lemma** *rtranclp-simplify-atms-of-ms*:

```

assumes rtranclp simplify  $\psi\ \psi'$ 
shows atms-of-ms  $\psi' \subseteq \text{atms-of-ms } \psi$ 
using assms apply (induct rule: rtranclp-induct)
  apply (fastforce intro: simplify-atms-of-ms)
using simplify-atms-of-ms by blast
```

**lemma** *factoring-imp-simplify*:  
**assumes**  $\{\#L, L\# \} + C \in N$   
**shows**  $\exists N'. \text{ simplify } N N'$   
**proof** –  
**have**  $\text{add-mset } L (\text{add-mset } L C) \in N$  **using** *assms* **by** (*simp add: add commute union-lcomm*)  
**from** *condensation[OF this]* **show** *?thesis* **by** *blast*  
**qed**

## 2.1.2 Unconstrained Resolution

**type-synonym** *'v uncon-state* = *'v clause-set*

**inductive** *uncon-res* :: *'v uncon-state*  $\Rightarrow$  *'v uncon-state*  $\Rightarrow$  *bool* **where**

*resolution*:

$\{\#Pos\ p\# \} + C \in N \implies \{\#Neg\ p\# \} + D \in N \implies (\text{add-mset } (Pos\ p)\ C, \text{add-mset } (Neg\ P)\ D) \notin \text{already-used}$

$\implies \text{uncon-res } N (N \cup \{C + D\}) \mid$

*factoring*:  $\{\#L\# \} + \{\#L\# \} + C \in N \implies \text{uncon-res } N (\text{insert } (\text{add-mset } L\ C)\ N)$

**lemma** *uncon-res-increasing*:

**assumes** *uncon-res* *S S'* **and**  $\psi \in S$

**shows**  $\psi \in S'$

**using** *assms* **by** (*induct rule: uncon-res.induct*) *auto*

**lemma** *rtranclp-uncon-inference-increasing*:

**assumes** *rtranclp uncon-res* *S S'* **and**  $\psi \in S$

**shows**  $\psi \in S'$

**using** *assms* **by** (*induct rule: rtranclp-induct*) (*auto simp add: uncon-res-increasing*)

## Subsumption

**definition** *subsumes* :: *'a literal multiset*  $\Rightarrow$  *'a literal multiset*  $\Rightarrow$  *bool* **where**

*subsumes*  $\chi\ \chi' \iff$

$(\forall I. \text{total-over-m } I\ \{\chi'\} \longrightarrow \text{total-over-m } I\ \{\chi\})$

$\wedge (\forall I. \text{total-over-m } I\ \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models \chi')$

**lemma** *subsumes-refl[simp]*:

*subsumes*  $\chi\ \chi$

**unfolding** *subsumes-def* **by** *auto*

**lemma** *subsumes-subsumption*:

**assumes** *subsumes* *D*  $\chi$

**and**  $C \subset\# D$  **and**  $\neg \text{tautology } \chi$

**shows** *subsumes* *C*  $\chi$  **unfolding** *subsumes-def*

**using** *assms* *subsumption-total-over-m* *subsumption-chained* **unfolding** *subsumes-def*

**by** (*blast intro!: subset-mset.less-imp-le*)

**lemma** *subsumes-tautology*:

**assumes** *subsumes*  $(\text{add-mset } (Pos\ P)\ (\text{add-mset } (Neg\ P)\ C))\ \chi$

**shows** *tautology*  $\chi$

**using** *assms* **unfolding** *subsumes-def* **by** (*auto simp add: tautology-def*)

## 2.1.3 Inference Rule

**type-synonym** *'v state* = *'v clause-set*  $\times$  (*'v clause*  $\times$  *'v clause*) *set*

**inductive** *inference-clause* :: 'v state  $\Rightarrow$  'v clause  $\times$  ('v clause  $\times$  'v clause) set  $\Rightarrow$  bool

(**infix**  $\Rightarrow_{\text{Res}}$  100) **where**

*resolution*:

$\{\#Pos\ p\# \} + C \in N \implies \{\#Neg\ p\# \} + D \in N \implies (\{\#Pos\ p\# \} + C, \{\#Neg\ p\# \} + D) \notin$   
*already-used*

$\implies \text{inference-clause } (N, \text{already-used}) (C + D, \text{already-used} \cup \{(\{\#Pos\ p\# \} + C, \{\#Neg\ p\# \} + D)\}) \mid$

*factoring*:  $\{\#L, L\# \} + C \in N \implies \text{inference-clause } (N, \text{already-used}) (C + \{\#L\# \}, \text{already-used})$

**inductive** *inference* :: 'v state  $\Rightarrow$  'v state  $\Rightarrow$  bool **where**

*inference-step*: *inference-clause* *S* (*clause*, *already-used*)

$\implies \text{inference } S (\text{fst } S \cup \{\text{clause}\}, \text{already-used})$

**abbreviation** *already-used-inv*

:: 'a literal multiset set  $\times$  ('a literal multiset  $\times$  'a literal multiset) set  $\Rightarrow$  bool **where**

*already-used-inv* state  $\equiv$

$(\forall (A, B) \in \text{snd state}. \exists p. \text{Pos } p \in \# A \wedge \text{Neg } p \in \# B \wedge$   
 $((\exists \chi \in \text{fst state}. \text{subsumes } \chi ((A - \{\#Pos\ p\# \}) + (B - \{\#Neg\ p\# \})))$   
 $\vee \text{tautology } ((A - \{\#Pos\ p\# \}) + (B - \{\#Neg\ p\# \}))))$

**lemma** *inference-clause-preserves-already-used-inv*:

**assumes** *inference-clause* *S* *S'*

**and** *already-used-inv* *S*

**shows** *already-used-inv* (*fst* *S*  $\cup$  {*fst* *S'*}, *snd* *S'*)

**using** *assms* **apply** (*induct* rule: *inference-clause.induct*)

**by** *fastforce*+

**lemma** *inference-preserves-already-used-inv*:

**assumes** *inference* *S* *S'*

**and** *already-used-inv* *S*

**shows** *already-used-inv* *S'*

**using** *assms*

**proof** (*induct* rule: *inference.induct*)

**case** (*inference-step* *S* *clause* *already-used*)

**then show** ?*case*

**using** *inference-clause-preserves-already-used-inv*[of *S* (*clause*, *already-used*)] **by** *simp*

**qed**

**lemma** *rtranclp-inference-preserves-already-used-inv*:

**assumes** *rtranclp* *inference* *S* *S'*

**and** *already-used-inv* *S*

**shows** *already-used-inv* *S'*

**using** *assms* **apply** (*induct* rule: *rtranclp-induct*, *simp*)

**using** *inference-preserves-already-used-inv* **unfolding** *tautology-def* **by** *fast*

**lemma** *subsumes-condensation*:

**assumes** *subsumes* (*C* + {*#L*#} + {*#L*#}) *D*

**shows** *subsumes* (*C* + {*#L*#}) *D*

**using** *assms* **unfolding** *subsumes-def* **by** *simp*

**lemma** *simplify-preserves-already-used-inv*:

**assumes** *simplify* *N* *N'*

**and** *already-used-inv* (*N*, *already-used*)

**shows** *already-used-inv* (*N'*, *already-used*)

```

using assms
proof (induct rule: simplify.induct)
  case (condensation C L)
  then show ?case
    using subsumes-condensation by simp fast
next
{
  fix a:: 'a and A :: 'a set and P
  have  $(\exists x \in \text{Set.remove } a \ A. P \ x) \longleftrightarrow (\exists x \in A. x \neq a \wedge P \ x)$  by auto
} note ex-member-remove = this
{
  fix a a0 :: 'v clause and A :: 'v clause-set and y
  assume  $a \in A$  and  $a0 \subset\# a$ 
  then have  $(\exists x \in A. \text{subsumes } x \ y) \longleftrightarrow (\text{subsumes } a \ y \vee (\exists x \in A. x \neq a \wedge \text{subsumes } x \ y))$ 
    by auto
} note tt2 = this
case (subsumption A B) note  $A = \text{this}(1)$  and  $AB = \text{this}(2)$  and  $B = \text{this}(3)$  and  $\text{inv} = \text{this}(4)$ 
show ?case
proof (standard, standard)
  fix x a b
  assume  $x: x \in \text{snd } (N - \{B\}, \text{already-used})$  and [simp]:  $x = (a, b)$ 
  obtain p where  $p: \text{Pos } p \in\# a \wedge \text{Neg } p \in\# b$  and
     $q: (\exists \chi \in N. \text{subsumes } \chi (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})))$ 
     $\vee \text{tautology } (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\}))$ 
  using inv x by fastforce
  consider (taut)  $\text{tautology } (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})) \mid$ 
     $(\chi) \chi$  where  $\chi \in N$   $\text{subsumes } \chi (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\}))$ 
     $\neg \text{tautology } (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\}))$ 
  using q by auto
  then show
     $\exists p. \text{Pos } p \in\# a \wedge \text{Neg } p \in\# b$ 
     $\wedge ((\exists \chi \in \text{fst } (N - \{B\}, \text{already-used}). \text{subsumes } \chi (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})))$ 
     $\vee \text{tautology } (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})))$ 
  proof cases
    case taut
    then show ?thesis using p by auto
  next
    case  $\chi$  note  $H = \text{this}$ 
    show ?thesis using p A AB B subsumes-subsumption[OF - AB H(3)] H(1,2) by fastforce
  qed
qed
next
case (tautology-deletion P C)
then show ?case
proof clarify
  fix a b
  assume  $\text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) C) \in N$ 
  assume  $\text{already-used-inv } (N, \text{already-used})$ 
  and  $(a, b) \in \text{snd } (N - \{\text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) C)\}, \text{already-used})$ 
  then obtain p where
     $\text{Pos } p \in\# a \wedge \text{Neg } p \in\# b \wedge$ 
     $((\exists \chi \in \text{fst } (N \cup \{\text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) C)\}, \text{already-used}).$ 
     $\text{subsumes } \chi (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})))$ 
     $\vee \text{tautology } (a - \{\# \text{Pos } p\} + (b - \{\# \text{Neg } p\})))$ 
  by fastforce
  moreover have  $\text{tautology } (\text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) C))$  by auto

```

**ultimately show**

$\exists p. \text{Pos } p \in \# a \wedge \text{Neg } p \in \# b \wedge$   
 $((\exists \chi \in \text{fst } (N - \{\text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) C)\}, \text{already-used}).$   
 $\text{subsumes } \chi (\text{remove1-mset } (\text{Pos } p) a + \text{remove1-mset } (\text{Neg } p) b)) \vee$   
 $\text{tautology } (\text{remove1-mset } (\text{Pos } p) a + \text{remove1-mset } (\text{Neg } p) b))$   
**by** (*metis* (*no-types*) *Diff-iff Un-insert-right empty-iff fst-conv insertE subsumes-tautology*  
*sup-bot.right-neutral*)

**qed**

**qed**

**lemma**

*factoring-satisfiable*:  $I \models \text{add-mset } L (\text{add-mset } L C) \longleftrightarrow I \models \text{add-mset } L C$  **and**

*resolution-satisfiable*:

*consistent-interp*  $I \implies I \models \text{add-mset } (\text{Pos } p) C \implies I \models \text{add-mset } (\text{Neg } p) D \implies I \models C + D$  **and**

*factoring-same-vars*:  $\text{atms-of } (\text{add-mset } L (\text{add-mset } L C)) = \text{atms-of } (\text{add-mset } L C)$

**unfolding** *true-cls-def consistent-interp-def* **by** (*fastforce split: if-split-asm*)**+**

**lemma** *inference-increasing*:

**assumes** *inference*  $S S'$  **and**  $\psi \in \text{fst } S$

**shows**  $\psi \in \text{fst } S'$

**using** *assms* **by** (*induct rule: inference.induct, auto*)

**lemma** *rtranclp-inference-increasing*:

**assumes** *rtranclp inference*  $S S'$  **and**  $\psi \in \text{fst } S$

**shows**  $\psi \in \text{fst } S'$

**using** *assms* **by** (*induct rule: rtranclp-induct, auto simp add: inference-increasing*)

**lemma** *inference-clause-already-used-increasing*:

**assumes** *inference-clause*  $S S'$

**shows**  $\text{snd } S \subseteq \text{snd } S'$

**using** *assms* **by** (*induct rule: inference-clause.induct, auto*)

**lemma** *inference-already-used-increasing*:

**assumes** *inference*  $S S'$

**shows**  $\text{snd } S \subseteq \text{snd } S'$

**using** *assms* **apply** (*induct rule: inference.induct*)

**using** *inference-clause-already-used-increasing* **by** *fastforce*

**lemma** *inference-clause-preserve-models*:

**fixes**  $N N' :: 'v \text{ clause-set}$

**assumes** *inference-clause*  $T T'$

**and** *total-over-m*  $I (\text{fst } T)$

**and** *consistent: consistent-interp*  $I$

**shows**  $I \models_s \text{fst } T \longleftrightarrow I \models_s \text{fst } T \cup \{\text{fst } T'\}$

**using** *assms* **apply** (*induct rule: inference-clause.induct*)

**unfolding** *consistent-interp-def true-clss-def* **by** *auto force+*

**lemma** *inference-preserve-models*:

**fixes**  $N N' :: 'v \text{ clause-set}$

**assumes** *inference*  $T T'$

**and** *total-over-m*  $I (\text{fst } T)$

**and** *consistent: consistent-interp*  $I$

**shows**  $I \models_s \text{fst } T \longleftrightarrow I \models_s \text{fst } T'$

**using** *assms* **apply** (*induct rule: inference.induct*)  
**using** *inference-clause-preserve-models* **by** *fastforce*

**lemma** *inference-clause-preserves-atms-of-ms*:  
**assumes** *inference-clause*  $S\ S'$   
**shows** *atms-of-ms* (*fst* ( $\text{fst } S \cup \{\text{fst } S'\}$ , *snd*  $S'$ ))  $\subseteq$  *atms-of-ms* (*fst*  $S$ )  
**using** *assms* **by** (*induct rule: inference-clause.induct*) (*auto dest!: atms-of-atms-of-ms-mono*)

**lemma** *inference-preserves-atms-of-ms*:  
**fixes**  $N\ N' :: 'v\ \text{clause-set}$   
**assumes** *inference*  $T\ T'$   
**shows** *atms-of-ms* (*fst*  $T'$ )  $\subseteq$  *atms-of-ms* (*fst*  $T$ )  
**using** *assms* **apply** (*induct rule: inference.induct*)  
**using** *inference-clause-preserves-atms-of-ms* **by** *fastforce*

**lemma** *inference-preserves-total*:  
**fixes**  $N\ N' :: 'v\ \text{clause-set}$   
**assumes** *inference* ( $N$ , *already-used*) ( $N'$ , *already-used'*)  
**shows** *total-over-m*  $I\ N \implies$  *total-over-m*  $I\ N'$   
**using** *assms* *inference-preserves-atms-of-ms* **unfolding** *total-over-m-def* *total-over-set-def*  
**by** *fastforce*

**lemma** *rtranclp-inference-preserves-total*:  
**assumes** *rtranclp inference*  $T\ T'$   
**shows** *total-over-m*  $I\ (\text{fst } T) \implies$  *total-over-m*  $I\ (\text{fst } T')$   
**using** *assms* **by** (*induct rule: rtranclp-induct*, *auto simp add: inference-preserves-total*)

**lemma** *rtranclp-inference-preserve-models*:  
**assumes** *rtranclp inference*  $N\ N'$   
**and** *total-over-m*  $I\ (\text{fst } N)$   
**and** *consistent: consistent-interp*  $I$   
**shows**  $I \models_s \text{fst } N \longleftrightarrow I \models_s \text{fst } N'$   
**using** *assms* **apply** (*induct rule: rtranclp-induct*)  
**apply** (*simp add: inference-preserve-models*)  
**using** *inference-preserve-models rtranclp-inference-preserves-total* **by** *blast*

**lemma** *inference-preserves-finite*:  
**assumes** *inference*  $\psi\ \psi'$  **and** *finite* (*fst*  $\psi$ )  
**shows** *finite* (*fst*  $\psi'$ )  
**using** *assms* **by** (*induct rule: inference.induct*, *auto simp add: simplify-preserves-finite*)

**lemma** *inference-clause-preserves-finite-snd*:  
**assumes** *inference-clause*  $\psi\ \psi'$  **and** *finite* (*snd*  $\psi$ )  
**shows** *finite* (*snd*  $\psi'$ )  
**using** *assms* **by** (*induct rule: inference-clause.induct*, *auto*)

**lemma** *inference-preserves-finite-snd*:  
**assumes** *inference*  $\psi\ \psi'$  **and** *finite* (*snd*  $\psi$ )  
**shows** *finite* (*snd*  $\psi'$ )  
**using** *assms* *inference-clause-preserves-finite-snd* **by** (*induct rule: inference.induct*, *fastforce*)

**lemma** *rtranclp-inference-preserves-finite*:

```

assumes rtranclp inference  $\psi$   $\psi'$  and finite (fst  $\psi$ )
shows finite (fst  $\psi'$ )
using assms by (induct rule: rtranclp-induct)
  (auto simp add: simplify-preserves-finite inference-preserves-finite)

lemma consistent-interp-insert:
  assumes consistent-interp  $I$ 
  and atm-of  $P \notin \text{atm-of } I$ 
  shows consistent-interp (insert  $P$   $I$ )
proof –
  have  $P: \text{insert } P \ I = I \cup \{P\}$  by auto
  show ?thesis unfolding  $P$ 
  apply (rule consistent-interp-disjoint)
  using assms by (auto simp: image-iff)
qed

lemma simplify-clause-preserves-sat:
  assumes simp: simplify  $\psi$   $\psi'$ 
  and satisfiable  $\psi'$ 
  shows satisfiable  $\psi$ 
  using assms
proof induction
  case (tautology-deletion  $P$   $A$ ) note  $AP = \text{this}(1)$  and  $\text{sat} = \text{this}(2)$ 
  let  $?A' = \text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) \ A)$ 
  let  $? \psi' = \psi - \{?A'\}$ 
  obtain  $I$  where
     $I: I \models_s ? \psi'$  and
    cons: consistent-interp  $I$  and
    tot: total-over-m  $I$   $? \psi'$ 
  using sat unfolding satisfiable-def by auto
  { assume  $\text{Pos } P \in I \vee \text{Neg } P \in I$ 
    then have  $I \models ?A'$  by auto
    then have  $I \models_s \psi$  using  $I$  by (metis insert-Diff tautology-deletion.hyps true-clss-insert)
    then have ?case using cons tot by auto
  }
  moreover {
    assume Pos: Pos  $P \notin I$  and Neg: Neg  $P \notin I$ 
    then have consistent-interp ( $I \cup \{\text{Pos } P\}$ ) using cons by simp
    moreover have  $I'A: I \cup \{\text{Pos } P\} \models ?A'$  by auto
    have  $\{\text{Pos } P\} \cup I \models_s \psi - \{?A'\}$ 
      using ( $I \models_s \psi - \{?A'\}$ ) true-clss-union-increase' by blast
    then have  $I \cup \{\text{Pos } P\} \models_s \psi$ 
      by (metis (no-types) Un-empty-right Un-insert-left Un-insert-right I'A insert-Diff
        sup-bot.left-neutral tautology-deletion.hyps true-clss-insert)
    ultimately have ?case using satisfiable-carac' by blast
  }
  ultimately show ?case by blast
next
  case (condensation  $L$   $A$ ) note  $AL = \text{this}(1)$  and  $\text{sat} = \text{this}(2)$ 
  let  $?A' = \text{add-mset } L \ A$ 
  let  $?A = \text{add-mset } L (\text{add-mset } L \ A)$ 
  have f3: simplify  $\psi (\psi - \{?A\} \cup \{?A'\})$ 
    using  $AL$  simplify.condensation by blast
  obtain  $LL$  :: 'a literal set where
     $f4: LL \models_s \psi - \{?A\} \cup \{?A'\}$ 
     $\wedge$  consistent-interp  $LL$ 

```



```

     $\wedge$  total-over-m LL ( $\psi - \{?A\} \cup \{?A'\}$ )
  using sat by (meson satisfiable-def)
have f5: insert ( $A + \{\#L\# \} + \{\#L\# \}$ ) ( $\psi - \{A + \{\#L\# \} + \{\#L\# \}\}$ ) =  $\psi$ 
  using AL by fastforce
have atms-of ( $?A'$ ) = atms-of ( $?A$ )
  by simp
then show ?case
  using f5 f4 f3 by (metis Un-insert-right add-mset-add-single atms-of-ms-insert satisfiable-carac
    simplify-preserve-models' sup-bot.right-neutral total-over-m-def)
next
case (subsumption A B) note A = this(1) and AB = this(2) and B = this(3) and sat = this(4)
let  $? \psi' = \psi - \{B\}$ 
obtain I where I:  $I \models ? \psi'$  and cons: consistent-interp I and tot: total-over-m I  $? \psi'$ 
  using sat unfolding satisfiable-def by auto
have  $I \models A$  using A I by (metis AB Diff-iff subset-mset.less-irrefl singletonD true-clss-def)
then have  $I \models B$  using AB subset-mset.less-imp-le true-clss-mono-leD by blast
then have  $I \models \psi$  using I by (metis insert-Diff-single true-clss-insert)
then show ?case using cons satisfiable-carac' by blast
qed

```

**lemma** simplify-preserves-unsat:

```

  assumes inference  $\psi \ \psi'$ 
  shows satisfiable (fst  $\psi'$ )  $\longrightarrow$  satisfiable (fst  $\psi$ )
  using assms apply (induct rule: inference.induct)
  using satisfiable-decreasing by (metis fst-conv)+

```

**lemma** inference-preserves-unsat:

```

  assumes inference** S S'
  shows satisfiable (fst S')  $\longrightarrow$  satisfiable (fst S)
  using assms apply (induct rule: rtranclp-induct)
  apply simp-all
  using simplify-preserves-unsat by blast

```

**datatype** 'v sem-tree = Node 'v 'v sem-tree 'v sem-tree | Leaf

**fun** sem-tree-size :: 'v sem-tree  $\Rightarrow$  nat **where**

```

sem-tree-size Leaf = 0 |
sem-tree-size (Node - ag ad) = 1 + sem-tree-size ag + sem-tree-size ad

```

**lemma** sem-tree-size[case-names bigger]:

```

( $\bigwedge xs:: 'v \text{ sem-tree. } (\bigwedge ys:: 'v \text{ sem-tree. } \text{sem-tree-size } ys < \text{sem-tree-size } xs \implies P \ ys) \implies P \ xs$ )
 $\implies P \ xs$ 
by (fact Nat.measure-induct-rule)

```

**fun** partial-interps :: 'v sem-tree  $\Rightarrow$  'v partial-interp  $\Rightarrow$  'v clause-set  $\Rightarrow$  bool **where**

```

partial-interps Leaf I  $\psi$  = ( $\exists \chi. \neg I \models \chi \wedge \chi \in \psi \wedge \text{total-over-m } I \ \{\chi\}$ ) |
partial-interps (Node v ag ad) I  $\psi \longleftrightarrow$ 
  (partial-interps ag (I  $\cup \{\text{Pos } v\}$ )  $\psi \wedge \text{partial-interps } ad \ (I \cup \{\text{Neg } v\}) \ \psi$ )

```

**lemma** simplify-preserve-partial-leaf:

```

simplify N N'  $\implies$  partial-interps Leaf I N  $\implies$  partial-interps Leaf I N'
apply (induct rule: simplify.induct)
  using union-lcomm apply auto[1]
  apply (simp)

```

**apply** (*metis atms-of-remdups-mset remdups-mset-singleton-sum true-cls-add-mset union-single-eq-member*)  
**apply** *auto*  
**by** (*metis atms-of-ms-empty-set subsumption-total-over-m total-over-m-def total-over-m-insert*  
*total-over-set-empty true-cls-mono-leD*)

**lemma** *simplify-preserve-partial-tree*:  
**assumes** *simplify N N'*  
**and** *partial-interps t I N*  
**shows** *partial-interps t I N'*  
**using** *assms* **apply** (*induct t arbitrary: I, simp*)  
**using** *simplify-preserve-partial-leaf* **by** *metis*

**lemma** *inference-preserve-partial-tree*:  
**assumes** *inference S S'*  
**and** *partial-interps t I (fst S)*  
**shows** *partial-interps t I (fst S')*  
**using** *assms* **apply** (*induct t arbitrary: I, simp-all*)  
**by** (*meson inference-increasing*)

**lemma** *rtranclp-inference-preserve-partial-tree*:  
**assumes** *rtranclp inference N N'*  
**and** *partial-interps t I (fst N)*  
**shows** *partial-interps t I (fst N')*  
**using** *assms* **apply** (*induct rule: rtranclp-induct, auto*)  
**using** *inference-preserve-partial-tree* **by** *force*

**function** *build-sem-tree* :: '*v* :: linorder set  $\Rightarrow$  '*v* clause-set  $\Rightarrow$  '*v* sem-tree **where**  
*build-sem-tree atms  $\psi$  =*  
*(if atms = {}  $\vee$   $\neg$  finite atms*  
*then Leaf*  
*else Node (Min atms) (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )*  
*(build-sem-tree (Set.remove (Min atms) atms)  $\psi$ ))*  
**by** *auto*  
**termination**  
**apply** (*relation measure ( $\lambda(A, -). \text{card } A$ ), simp-all*)  
**apply** (*metis Min-in card-Diff1-less remove-def*)+  
**done**  
**declare** *build-sem-tree.induct[case-names tree]*

**lemma** *unsatisfiable-empty[simp]*:  
 $\neg$ *unsatisfiable {}*  
**unfolding** *satisfiable-def* **apply** *auto*  
**using** *consistent-interp-def* **unfolding** *total-over-m-def total-over-set-def atms-of-ms-def* **by** *blast*

**lemma** *partial-interps-build-sem-tree-atms-general*:  
**fixes**  $\psi :: 'v :: \text{linorder clause-set}$  **and**  $p :: 'v \text{ literal list}$   
**assumes** *unsat: unsatisfiable  $\psi$*  **and** *finite  $\psi$*  **and** *consistent-interp I*  
**and** *finite atms*  
**and** *atms-of-ms  $\psi = \text{atms} \cup \text{atms-of-s } I$*  **and** *atms  $\cap \text{atms-of-s } I = \{\}$*   
**shows** *partial-interps (build-sem-tree atms  $\psi$ ) I  $\psi$*   
**using** *assms*  
**proof** (*induct arbitrary: I rule: build-sem-tree.induct*)  
**case** (*1 atms  $\psi$  Ia*) **note** *IH1 = this(1)* **and** *IH2 = this(2)* **and** *unsat = this(3)* **and** *finite = this(4)*

```

and cons = this(5) and f = this(6) and un = this(7) and disj = this(8)
{
  assume atms: atms = {}
  then have atmsIa: atms-of-ms  $\psi$  = atms-of-s Ia using un by auto
  then have total-over-m Ia  $\psi$  unfolding total-over-m-def atmsIa by auto
  then have  $\chi$ :  $\exists \chi \in \psi. \neg Ia \models \chi$ 
    using unsat cons unfolding true-clss-def satisfiable-def by auto
  then have build-sem-tree atms  $\psi$  = Leaf using atms by auto
  moreover
    have tot:  $\bigwedge \chi. \chi \in \psi \implies \text{total-over-m Ia } \{\chi\}$ 
    unfolding total-over-m-def total-over-set-def atms-of-ms-def atms-of-s-def
    using atmsIa atms-of-ms-def by fastforce
  have partial-interps Leaf Ia  $\psi$ 
    using  $\chi$  tot by (auto simp add: total-over-m-def total-over-set-def atms-of-ms-def)

  ultimately have ?case by metis
}
moreover {
  assume atms: atms  $\neq \{\}$ 
  have build-sem-tree atms  $\psi$  = Node (Min atms) (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
    (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
    using build-sem-tree.simps[of atms  $\psi$ ] f atms by metis

  have consistent-interp (Ia  $\cup \{\text{Pos (Min atms)}\}$ ) unfolding consistent-interp-def
    by (metis Int-iff Min-in Un-iff atm-of-uminus atms cons consistent-interp-def disj empty-iff
      f in-atms-of-s-decomp insert-iff literal.distinct(1) literal.exhaust-sel literal.sel(2)
      uminus-Neg uminus-Pos)
  moreover have atms-of-ms  $\psi$  = Set.remove (Min atms) atms  $\cup$  atms-of-s (Ia  $\cup \{\text{Pos (Min atms)}\}$ )
    using Min-in atms f un by fastforce
  moreover have disj': Set.remove (Min atms) atms  $\cap$  atms-of-s (Ia  $\cup \{\text{Pos (Min atms)}\}$ ) =  $\{\}$ 
    by simp (metis disj disjoint-iff-not-equal member-remove)
  moreover have finite (Set.remove (Min atms) atms) using f by (simp add: remove-def)
  ultimately have subtree1: partial-interps (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
    (Ia  $\cup \{\text{Pos (Min atms)}\}$ )  $\psi$ 
    using IH1[of Ia  $\cup \{\text{Pos (Min (atms))}\}$ ] atms f unsat finite by metis

  have consistent-interp (Ia  $\cup \{\text{Neg (Min atms)}\}$ ) unfolding consistent-interp-def
    by (metis Int-iff Min-in Un-iff atm-of-uminus atms cons consistent-interp-def disj empty-iff
      f in-atms-of-s-decomp insert-iff literal.distinct(1) literal.exhaust-sel literal.sel(2)
      uminus-Neg)
  moreover have atms-of-ms  $\psi$  = Set.remove (Min atms) atms  $\cup$  atms-of-s (Ia  $\cup \{\text{Neg (Min atms)}\}$ )
    using  $\langle \text{atms-of-ms } \psi = \text{Set.remove (Min atms) atms } \cup \text{atms-of-s (Ia } \cup \{\text{Pos (Min atms)}\}) \rangle$  by
blast

  moreover have disj': Set.remove (Min atms) atms  $\cap$  atms-of-s (Ia  $\cup \{\text{Neg (Min atms)}\}$ ) =  $\{\}$ 
    using disj by auto
  moreover have finite (Set.remove (Min atms) atms) using f by (simp add: remove-def)
  ultimately have subtree2: partial-interps (build-sem-tree (Set.remove (Min atms) atms)  $\psi$ )
    (Ia  $\cup \{\text{Neg (Min atms)}\}$ )  $\psi$ 
    using IH2[of Ia  $\cup \{\text{Neg (Min (atms))}\}$ ] atms f unsat finite by metis

  then have ?case
    using IH1 subtree1 subtree2 f local.finite unsat atms by simp
}
ultimately show ?case by metis
qed

```

**lemma** *partial-interps-build-sem-tree-atms*:  
**fixes**  $\psi :: 'v :: \text{linorder clause-set}$  **and**  $p :: 'v \text{ literal list}$   
**assumes** *unsat*: *unsatisfiable*  $\psi$  **and** *finite*: *finite*  $\psi$   
**shows** *partial-interps* (*build-sem-tree* (*atms-of-ms*  $\psi$ )  $\psi$ )  $\{\}$   $\psi$   
**proof** –  
**have** *consistent-interp*  $\{\}$  **unfolding** *consistent-interp-def* **by** *auto*  
**moreover** **have** *atms-of-ms*  $\psi = \text{atms-of-ms } \psi \cup \text{atms-of-s } \{\}$  **unfolding** *atms-of-s-def* **by** *auto*  
**moreover** **have** *atms-of-ms*  $\psi \cap \text{atms-of-s } \{\} = \{\}$  **unfolding** *atms-of-s-def* **by** *auto*  
**moreover** **have** *finite* (*atms-of-ms*  $\psi$ ) **unfolding** *atms-of-ms-def* **using** *finite* **by** *simp*  
**ultimately** **show** *partial-interps* (*build-sem-tree* (*atms-of-ms*  $\psi$ )  $\psi$ )  $\{\}$   $\psi$   
**using** *partial-interps-build-sem-tree-atms-general*[*of*  $\psi \{\}$  *atms-of-ms*  $\psi$ ] *assms* **by** *metis*  
**qed**

**lemma** *can-decrease-count*:  
**fixes**  $\psi'' :: 'v \text{ clause-set} \times ('v \text{ clause} \times 'v \text{ clause} \times 'v) \text{ set}$   
**assumes** *count*  $\chi \ L = n$   
**and**  $L \in \# \chi$  **and**  $\chi \in \text{fst } \psi$   
**shows**  $\exists \psi' \chi'. \text{inference}^{**} \psi \psi' \wedge \chi' \in \text{fst } \psi' \wedge (\forall L. L \in \# \chi \longleftrightarrow L \in \# \chi')$   
 $\wedge \text{count } \chi' \ L = 1$   
 $\wedge (\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi')$   
 $\wedge (I \models \chi \longleftrightarrow I \models \chi')$   
 $\wedge (\forall I'. \text{total-over-m } I' \{\chi\} \longrightarrow \text{total-over-m } I' \{\chi'\})$

**using** *assms*

**proof** (*induct n arbitrary:  $\chi \psi$* )

**case** 0

**then** **show** ?*case* **by** (*simp add: not-in-iff[symmetric]*)

**next**

**case** (*Suc n  $\chi$* )

**note** *IH* = *this*(1) **and** *count* = *this*(2) **and** *L* = *this*(3) **and**  $\chi$  = *this*(4)

{

**assume**  $n = 0$

**then** **have** *inference*<sup>\*\*</sup>  $\psi \psi$

**and**  $\chi \in \text{fst } \psi$

**and**  $\forall L. (L \in \# \chi) \longleftrightarrow (L \in \# \chi)$

**and** *count*  $\chi \ L = (1::\text{nat})$

**and**  $\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi$

**by** (*auto simp add: count L  $\chi$* )

**then** **have** ?*case* **by** *metis*

}

**moreover** {

**assume**  $n > 0$

**then** **have**  $\exists C. \chi = C + \{\#L, L\# \}$

**by** (*metis Suc-inject union-mset-add-mset-right add-mset-add-single count-add-mset count-inI less-not-refl3 local.count mset-add zero-less-Suc*)

**then** **obtain** *C* **where**  $\chi = C + \{\#L, L\# \}$  **by** *metis*

**let**  $? \chi' = C + \{\#L\# \}$

**let**  $? \psi' = (\text{fst } \psi \cup \{? \chi'\}, \text{snd } \psi)$

**have**  $\varphi: \forall \varphi \in \text{fst } \psi. (\varphi \in \text{fst } \psi \vee \varphi \neq ? \chi') \longleftrightarrow \varphi \in \text{fst } ? \psi'$  **unfolding** *C* **by** *auto*

**have** *inf*: *inference*  $\psi ? \psi'$

**using** *C factoring  $\chi$  prod.collapse union-commute inference-step*

**by** (*metis add-mset-add-single*)

**moreover** **have** *count'*: *count*  $? \chi' \ L = n$  **using** *C count* **by** *auto*

**moreover** **have**  $L \chi'$ :  $L \in \# ? \chi'$  **by** *auto*

**moreover** **have**  $\chi' \psi'$ :  $? \chi' \in \text{fst } ? \psi'$  **by** *auto*

ultimately obtain  $\psi''$  and  $\chi''$   
where  
*inference\*\**  $? \psi' \psi''$  and  
 $\alpha: \chi'' \in \text{fst } \psi''$  and  
 $\forall La. (La \in \# ? \chi') \longleftrightarrow (La \in \# \chi'')$  and  
 $\beta: \text{count } \chi'' L = (1::\text{nat})$  and  
 $\varphi': \forall \varphi. \varphi \in \text{fst } ? \psi' \longrightarrow \varphi \in \text{fst } \psi''$  and  
 $I\chi: I \models ? \chi' \longleftrightarrow I \models \chi''$  and  
 $\text{tot}: \forall I'. \text{total-over-m } I' \{? \chi'\} \longrightarrow \text{total-over-m } I' \{\chi''\}$   
using *IH*[*of*  $? \chi' ? \psi'$ ] *count'*  $L\chi' \chi' \psi'$  by *blast*

then have *inference\*\**  $\psi \psi''$   
and  $\forall La. (La \in \# \chi) \longleftrightarrow (La \in \# \chi'')$   
using *inf unfolding C* by *auto*  
moreover have  $\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi''$  using  $\varphi \varphi'$  by *metis*  
moreover have  $I \models \chi \longleftrightarrow I \models \chi''$  using  $I\chi$  *unfolding true-cls-def C* by *auto*  
moreover have  $\forall I'. \text{total-over-m } I' \{\chi\} \longrightarrow \text{total-over-m } I' \{\chi''\}$   
using *tot unfolding C total-over-m-def* by *auto*  
ultimately have *?case* using  $\varphi \varphi' \alpha \beta$  by *metis*

}

ultimately show *?case* by *auto*

qed

**lemma** *can-decrease-tree-size*:  
*fixes*  $\psi :: 'v \text{ state}$  and *tree*  $:: 'v \text{ sem-tree}$   
*assumes* *finite* (*fst*  $\psi$ ) and *already-used-inv*  $\psi$   
and *partial-interps tree I* (*fst*  $\psi$ )  
*shows*  $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{inference** } \psi \psi' \wedge \text{partial-interps tree}' I (\text{fst } \psi')$   
 $\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0)$   
using *assms*

**proof** (*induct arbitrary: I rule: sem-tree-size*)  
*case* (*bigger xs I*) **note** *IH* = *this*(1) and *finite* = *this*(2) and *a-u-i* = *this*(3) and *part* = *this*(4)

{  
*assume* *sem-tree-size xs* = 0  
then have *?case* using *part* by *blast*  
}

moreover {  
*assume* *sn0*: *sem-tree-size xs* > 0  
obtain *ag ad v* **where** *xs*: *xs* = *Node v ag ad* using *sn0* by (*cases xs, auto*)  
{  
*assume* *sem-tree-size ag* = 0 and *sem-tree-size ad* = 0  
then have *ag*: *ag* = *Leaf* and *ad*: *ad* = *Leaf* by (*cases ag, auto*) (*cases ad, auto*)

then obtain  $\chi \chi'$  where  
 $\chi: \neg I \cup \{\text{Pos } v\} \models \chi$  and  
 $\text{tot}\chi: \text{total-over-m } (I \cup \{\text{Pos } v\}) \{\chi\}$  and  
 $\chi\psi: \chi \in \text{fst } \psi$  and  
 $\chi': \neg I \cup \{\text{Neg } v\} \models \chi'$  and  
 $\text{tot}\chi': \text{total-over-m } (I \cup \{\text{Neg } v\}) \{\chi'\}$  and  
 $\chi'\psi: \chi' \in \text{fst } \psi$   
using *part unfolding xs* by *auto*  
have *Posv*: *Pos v*  $\notin \# \chi$  using  $\chi$  *unfolding true-cls-def true-lit-def* by *auto*  
have *Negv*: *Neg v*  $\notin \# \chi'$  using  $\chi'$  *unfolding true-cls-def true-lit-def* by *auto*  
{

```

assume  $Neg\chi$ :  $Neg\ v \notin \# \chi$ 
have  $\neg I \models \chi$  using  $\chi$   $Posv$  unfolding  $true\text{-cls}\text{-def}$   $true\text{-lit}\text{-def}$  by auto
moreover have  $total\text{-over}\text{-}m\ I\ \{\chi\}$ 
  using  $Posv\ Neg\chi\ atm\text{-imp}\text{-}pos\text{-or}\text{-}neg\text{-lit}\ tot\chi$  unfolding  $total\text{-over}\text{-}m\text{-def}$   $total\text{-over}\text{-}set\text{-def}$ 
  by fastforce
ultimately have  $partial\text{-interp}\ Leaf\ I\ (fst\ \psi)$ 
and  $sem\text{-tree}\text{-}size\ Leaf < sem\text{-tree}\text{-}size\ xs$ 
and  $inference^{**}\ \psi\ \psi$ 
  unfolding  $xs$  by  $(auto\ simp\ add: \chi\psi)$ 
}
moreover {
  assume  $Pos\chi$ :  $Pos\ v \notin \# \chi'$ 
  then have  $I\chi$ :  $\neg I \models \chi'$  using  $\chi'$   $Posv$  unfolding  $true\text{-cls}\text{-def}$   $true\text{-lit}\text{-def}$  by auto
  moreover have  $total\text{-over}\text{-}m\ I\ \{\chi'\}$ 
    using  $Negv\ Pos\chi\ atm\text{-imp}\text{-}pos\text{-or}\text{-}neg\text{-lit}\ tot\chi'$ 
    unfolding  $total\text{-over}\text{-}m\text{-def}$   $total\text{-over}\text{-}set\text{-def}$  by fastforce
  ultimately have  $partial\text{-interp}\ Leaf\ I\ (fst\ \psi)$  and
     $sem\text{-tree}\text{-}size\ Leaf < sem\text{-tree}\text{-}size\ xs$  and
     $inference^{**}\ \psi\ \psi$ 
    using  $\chi'\psi\ I\chi$  unfolding  $xs$  by auto
}
moreover {
  assume  $neg$ :  $Neg\ v \in \# \chi$  and  $pos$ :  $Pos\ v \in \# \chi'$ 
  then obtain  $\psi'\ \chi^2$  where  $inf$ :  $rtranclp\ inference\ \psi\ \psi'$  and  $\chi^2\text{incl}$ :  $\chi^2 \in fst\ \psi'$ 
    and  $\chi\chi^2\text{-incl}$ :  $\forall L. L \in \# \chi \longleftrightarrow L \in \# \chi^2$ 
    and  $count\chi^2$ :  $count\ \chi^2\ (Neg\ v) = 1$ 
    and  $\varphi$ :  $\forall \varphi::'v\ literal\ multiset. \varphi \in fst\ \psi \longrightarrow \varphi \in fst\ \psi'$ 
    and  $I\chi$ :  $I \models \chi \longleftrightarrow I \models \chi^2$ 
    and  $tot\text{-imp}\chi$ :  $\forall I'. total\text{-over}\text{-}m\ I'\ \{\chi\} \longrightarrow total\text{-over}\text{-}m\ I'\ \{\chi^2\}$ 
    using  $can\text{-decrease}\text{-}count[of\ \chi\ Neg\ v\ count\ \chi\ (Neg\ v)\ \psi\ I]\ \chi\psi\ \chi'\psi$  by auto

  have  $\chi' \in fst\ \psi'$  by  $(simp\ add: \chi'\psi\ \varphi)$ 
  with  $pos$ 
  obtain  $\psi''\ \chi^{2'}$  where
     $inf'$ :  $inference^{**}\ \psi'\ \psi''$ 
    and  $\chi^{2'}\text{-incl}$ :  $\chi^{2'} \in fst\ \psi''$ 
    and  $\chi'\chi^{2'}\text{-incl}$ :  $\forall L::'v\ literal. (L \in \# \chi') = (L \in \# \chi^{2'})$ 
    and  $count\chi^{2'}$ :  $count\ \chi^{2'}\ (Pos\ v) = (1::nat)$ 
    and  $\varphi'$ :  $\forall \varphi::'v\ literal\ multiset. \varphi \in fst\ \psi' \longrightarrow \varphi \in fst\ \psi''$ 
    and  $I\chi'$ :  $I \models \chi' \longleftrightarrow I \models \chi^{2'}$ 
    and  $tot\text{-imp}\chi'$ :  $\forall I'. total\text{-over}\text{-}m\ I'\ \{\chi'\} \longrightarrow total\text{-over}\text{-}m\ I'\ \{\chi^{2'}\}$ 
    using  $can\text{-decrease}\text{-}count[of\ \chi'\ Pos\ v\ count\ \chi'\ (Pos\ v)\ \psi'\ I]\$  by auto

  define  $C$  where  $C$ :  $C = \chi^2 - \{\#Neg\ v\ \#\}$ 

  then have  $\chi^2$ :  $\chi^2 = C + \{\#Neg\ v\ \#\}$  and  $negC$ :  $Neg\ v \notin \# C$  and  $posC$ :  $Pos\ v \notin \# C$ 
    using  $\chi\chi^2\text{-incl}\ neg$  apply auto]
    using  $C\ \chi\chi^2\text{-incl}\ neg\ count\chi^2\ count\text{-eq}\text{-zero}\text{-iff}$  apply fastforce
    using  $C\ Posv\ \chi\chi^2\text{-incl}\ in\text{-diff}D$  by fastforce

  obtain  $C'$  where
     $\chi^{2'}$ :  $\chi^{2'} = C' + \{\#Pos\ v\ \#\}$  and
     $posC'$ :  $Pos\ v \notin \# C'$  and
     $negC'$ :  $Neg\ v \notin \# C'$ 
  proof –
    assume  $a1$ :  $\bigwedge C'. \llbracket \chi^{2'} = C' + \{\#Pos\ v\ \#\}; Pos\ v \notin \# C'; Neg\ v \notin \# C' \rrbracket \implies thesis$ 

```

```

have f2:  $\bigwedge n. (n::nat) - n = 0$ 
  by simp
have Neg v  $\notin \# \chi^{2'} - \{\#Pos v\}$ 
  using Negv  $\chi^{2'} \text{-incl}$  by (auto simp: not-in-iff)
have count  $\{\#Pos v\} (Pos v) = 1$ 
  by simp
then show ?thesis
  by (metis  $\chi^{2'} \text{-incl}$   $\langle Neg v \notin \# \chi^{2'} - \{\#Pos v\} \rangle$  a1 count $\chi^{2'}$  count-diff f2
      insert-DiffM2 less-numeral-extra(3) mem-Collect-eq pos set-mset-def)
qed

have already-used-inv  $\psi'$ 
  using rtranclp-inference-preserves-already-used-inv[of  $\psi \psi'$ ] a-u-i inf by blast
then have a-u-i- $\psi''$ : already-used-inv  $\psi''$ 
  using rtranclp-inference-preserves-already-used-inv a-u-i inf' unfolding tautology-def
  by simp

have totC: total-over-m I {C}
  using tot-imp $\chi$  tot $\chi$  total-over-m-remove[of I Pos v C] negC posC unfolding  $\chi^2$ 
  by (metis total-over-m-sum uminus-Neg uminus-of-uminus-id)
have totC': total-over-m I {C'}
  using tot-imp $\chi'$  tot $\chi'$  total-over-m-sum total-over-m-remove[of I Neg v C'] negC' posC'
  unfolding  $\chi^{2'}$  by (metis total-over-m-sum uminus-Neg)
have  $\neg I \models C + C'$ 
  using  $\chi$  I $\chi$   $\chi'$  I $\chi'$  unfolding  $\chi^2$   $\chi^{2'}$  true-cls-def by auto
then have part-I- $\psi'''$ : partial-interps Leaf I (fst  $\psi'' \cup \{C + C'\}$ )
  using totC totC' by simp
  (metis  $\neg I \models C + C'$  atms-of-ms-singleton total-over-m-def total-over-m-sum)
{
  assume  $(\{\#Pos v\} + C', \{\#Neg v\} + C) \notin \text{snd } \psi''$ 
  then have inf'': inference  $\psi''$  (fst  $\psi'' \cup \{C + C'\}$ , snd  $\psi'' \cup \{(\chi^{2'}, \chi^2)\}$ )
    using add.commute  $\varphi' \chi^{2'} \text{-incl}$   $\langle \chi^{2'} \in \text{fst } \psi'' \rangle$  unfolding  $\chi^2$   $\chi^{2'}$ 
    by (metis prod.collapse inference-step resolution)
  have inference**  $\psi$  (fst  $\psi'' \cup \{C + C'\}$ , snd  $\psi'' \cup \{(\chi^{2'}, \chi^2)\}$ )
    using inf inf' inf'' rtranclp-trans by auto
  moreover have sem-tree-size Leaf < sem-tree-size xs unfolding xs by auto
  ultimately have ?case using part-I- $\psi'''$  by (metis fst-conv)
}
moreover {
  assume a:  $(\{\#Pos v\} + C', \{\#Neg v\} + C) \in \text{snd } \psi''$ 
  then have  $(\exists \chi \in \text{fst } \psi''. (\forall I. \text{total-over-m } I \{C + C'\} \longrightarrow \text{total-over-m } I \{\chi\})$ 
     $\wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C))$ 
     $\vee \text{tautology } (C' + C)$ 
  proof -
    obtain p where p: Pos p  $\in \# (\{\#Pos v\} + C')$  and
      n: Neg p  $\in \# (\{\#Neg v\} + C)$  and
      decomp:  $((\exists \chi \in \text{fst } \psi''. (\forall I. \text{total-over-m } I \{(\{\#Pos v\} + C') - \{\#Pos p\}$ 
         $+ ((\{\#Neg v\} + C) - \{\#Neg p\})\}$ 
         $\longrightarrow \text{total-over-m } I \{\chi\})$ 
         $\wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi$ 
         $\longrightarrow I \models (\{\#Pos v\} + C') - \{\#Pos p\} + ((\{\#Neg v\} + C) - \{\#Neg p\})))$ 
         $\vee \text{tautology } ((\{\#Pos v\} + C') - \{\#Pos p\} + ((\{\#Neg v\} + C) - \{\#Neg p\})))$ 
      using a by (blast intro: allE[OF a-u-i- $\psi''$ ][unfolded subsumes-def Ball-def],
        of  $(\{\#Pos v\} + C', \{\#Neg v\} + C))$ 

```

```

{ assume  $p \neq v$ 
  then have  $Pos\ p \in \# \ C' \wedge Neg\ p \in \# \ C$  using  $p\ n$  by force
  then have ?thesis unfolding Bex-def by auto
}
moreover {
  assume  $p = v$ 
  then have ?thesis using decomp by (metis add.commute add-diff-cancel-left')
}
ultimately show ?thesis by auto
qed
moreover {
  assume  $\exists \chi \in fst\ \psi''. (\forall I. total-over-m\ I\ \{C+C'\} \longrightarrow total-over-m\ I\ \{\chi\})$ 
   $\wedge (\forall I. total-over-m\ I\ \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)$ 
  then obtain  $\vartheta$  where  $\vartheta: \vartheta \in fst\ \psi''$  and
   $tot\text{-}\vartheta\text{-}CC': \forall I. total-over-m\ I\ \{C+C'\} \longrightarrow total-over-m\ I\ \{\vartheta\}$  and
   $\vartheta\text{-}inv: \forall I. total-over-m\ I\ \{\vartheta\} \longrightarrow I \models \vartheta \longrightarrow I \models C' + C$  by blast
  have partial-interps Leaf  $I\ (fst\ \psi'')$ 
  using  $tot\text{-}\vartheta\text{-}CC'\ \vartheta\ \vartheta\text{-}inv\ totC\ totC' \hookrightarrow I \models C + C'$  total-over-m-sum by fastforce
  moreover have  $sem\text{-}tree\text{-}size\ Leaf < sem\text{-}tree\text{-}size\ xs$  unfolding xs by auto
  ultimately have ?case by (metis inf inf' rtranclp-trans)
}
moreover {
  assume  $tautCC': tautology\ (C' + C)$ 
  have  $total-over-m\ I\ \{C'+C\}$  using  $totC\ totC'\ total-over-m\text{-}sum$  by auto
  then have  $\neg tautology\ (C' + C)$ 
  using  $\hookrightarrow I \models C + C'$  unfolding add.commute[of  $C\ C'$ ] total-over-m-def
  unfolding tautology-def by auto
  then have False using  $tautCC'$  unfolding tautology-def by auto
}
ultimately have ?case by auto
}
ultimately have ?case by auto
}
ultimately have ?case using part by (metis (no-types) sem-tree-size.simps(1))
}
moreover {
  assume  $size-ag: sem\text{-}tree\text{-}size\ ag > 0$ 
  have  $sem\text{-}tree\text{-}size\ ag < sem\text{-}tree\text{-}size\ xs$  unfolding xs by auto
  moreover have partial-interps  $ag\ (I \cup \{Pos\ v\})\ (fst\ \psi)$ 
  and partad: partial-interps  $ad\ (I \cup \{Neg\ v\})\ (fst\ \psi)$ 
  using part partial-interps.simps(2) unfolding xs by metis+
  moreover have  $sem\text{-}tree\text{-}size\ ag < sem\text{-}tree\text{-}size\ xs \longrightarrow finite\ (fst\ \psi) \longrightarrow already-used\text{-}inv\ \psi$ 
 $\longrightarrow (partial-interps\ ag\ (I \cup \{Pos\ v\})\ (fst\ \psi) \longrightarrow$ 
 $(\exists tree'\ \psi'. inference^{**}\ \psi\ \psi' \wedge partial-interps\ tree'\ (I \cup \{Pos\ v\})\ (fst\ \psi')$ 
 $\wedge (sem\text{-}tree\text{-}size\ tree' < sem\text{-}tree\text{-}size\ ag \vee sem\text{-}tree\text{-}size\ ag = 0)))$ 
  using IH by auto
  ultimately obtain  $\psi' :: 'v\ state$  and  $tree' :: 'v\ sem\text{-}tree$  where
   $inf: inference^{**}\ \psi\ \psi'$ 
  and part: partial-interps  $tree'\ (I \cup \{Pos\ v\})\ (fst\ \psi')$ 
  and size:  $sem\text{-}tree\text{-}size\ tree' < sem\text{-}tree\text{-}size\ ag \vee sem\text{-}tree\text{-}size\ ag = 0$ 
  using finite part rtranclp.rtrancl-refl a-u-i by blast

  have partial-interps  $ad\ (I \cup \{Neg\ v\})\ (fst\ \psi')$ 
  using rtranclp-inference-preserve-partial-tree inf partad by metis
  then have partial-interps (Node  $v\ tree'\ ad)\ I\ (fst\ \psi')$  using part by auto
  then have ?case using inf size size-ag part unfolding xs by fastforce
}

```



```

}
moreover {
  assume size-ad: sem-tree-size ad > 0
  have sem-tree-size ad < sem-tree-size xs unfolding xs by auto
  moreover have partag: partial-interps ag ( $I \cup \{Pos\ v\}$ ) (fst  $\psi$ ) and
    partial-interps ad ( $I \cup \{Neg\ v\}$ ) (fst  $\psi$ )
    using part partial-interps.simps(2) unfolding xs by metis+
  moreover have sem-tree-size ad < sem-tree-size xs  $\longrightarrow$  finite (fst  $\psi$ )  $\longrightarrow$  already-used-inv  $\psi$ 
     $\longrightarrow$  (partial-interps ad ( $I \cup \{Neg\ v\}$ ) (fst  $\psi$ )
       $\longrightarrow$  ( $\exists$  tree'  $\psi'$ . inference**  $\psi\ \psi' \wedge$  partial-interps tree' ( $I \cup \{Neg\ v\}$ ) (fst  $\psi'$ )
         $\wedge$  (sem-tree-size tree' < sem-tree-size ad  $\vee$  sem-tree-size ad = 0)))
    using IH by auto
  ultimately obtain  $\psi' :: 'v$  state and tree' ::  $'v$  sem-tree where
    inf: inference**  $\psi\ \psi'$ 
    and part: partial-interps tree' ( $I \cup \{Neg\ v\}$ ) (fst  $\psi'$ )
    and size: sem-tree-size tree' < sem-tree-size ad  $\vee$  sem-tree-size ad = 0
    using finite part rtranclp.rtrancl-refl a-u-i by blast

  have partial-interps ag ( $I \cup \{Pos\ v\}$ ) (fst  $\psi'$ )
    using rtranclp-inference-preserve-partial-tree inf partag by metis
  then have partial-interps (Node v ag tree') I (fst  $\psi'$ ) using part by auto
  then have ?case using inf size size-ad unfolding xs by fastforce
}
ultimately have ?case by auto
}
ultimately show ?case by auto
qed

```

```

lemma inference-completeness-inv:
  fixes  $\psi :: 'v :: linorder$  state
  assumes
    unsat:  $\neg$ satisfiable (fst  $\psi$ ) and
    finite: finite (fst  $\psi$ ) and
    a-u-v: already-used-inv  $\psi$ 
  shows  $\exists \psi'. (inference^{**} \psi\ \psi' \wedge \{\#\} \in fst\ \psi')$ 
proof –
  obtain tree where partial-interps tree {} (fst  $\psi$ )
    using partial-interps-build-sem-tree-atms assms by metis
  then show ?thesis
    using unsat finite a-u-v
  proof (induct tree arbitrary:  $\psi$  rule: sem-tree-size)
    case (bigger tree  $\psi$ ) note H = this
    {
      fix  $\chi$ 
      assume tree: tree = Leaf
      obtain  $\chi$  where  $\chi: \neg \{\} \models \chi$  and tot $\chi$ : total-over-m {} { $\chi$ } and  $\chi\psi$ :  $\chi \in fst\ \psi$ 
        using H unfolding tree by auto
      moreover have { $\#\}$  =  $\chi$ 
        using tot $\chi$  unfolding total-over-m-def total-over-set-def by fastforce
      moreover have inference**  $\psi\ \psi$  by auto
      ultimately have ?case by metis
    }
  moreover {
    fix v tree1 tree2
    assume tree: tree = Node v tree1 tree2
    obtain

```

```

    tree'  $\psi'$  where inf: inference**  $\psi \ \psi'$  and
    part': partial-interps tree' {} (fst  $\psi'$ ) and
    decrease: sem-tree-size tree' < sem-tree-size tree  $\vee$  sem-tree-size tree = 0
    using can-decrease-tree-size[of  $\psi$ ] H(2,4,5) unfolding tautology-def by meson
    have sem-tree-size tree' < sem-tree-size tree using decrease unfolding tree by auto
    moreover have finite (fst  $\psi'$ ) using rtranclp-inference-preserves-finite inf H(4) by metis
    moreover have unsatisfiable (fst  $\psi'$ )
    using inference-preserves-unsat inf bigger.premis(2) by blast
    moreover have already-used-inv  $\psi'$ 
    using H(5) inf rtranclp-inference-preserves-already-used-inv[of  $\psi \ \psi'$ ] by auto
    ultimately have ?case using inf rtranclp-trans part' H(1) by fastforce
  }
  ultimately show ?case by (cases tree, auto)

```

qed

qed

**lemma** *inference-completeness*:

```

  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes unsat:  $\neg \text{satisfiable (fst } \psi)$ 
  and finite: finite (fst  $\psi$ )
  and snd  $\psi = \{\}$ 
  shows  $\exists \psi'. (\text{rtranclp inference } \psi \ \psi' \wedge \{\#\} \in \text{fst } \psi')$ 

```

**proof** –

```

  have already-used-inv  $\psi$  unfolding assms by auto
  then show ?thesis using assms inference-completeness-inv by blast

```

qed

**lemma** *inference-soundness*:

```

  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes rtranclp inference  $\psi \ \psi'$  and  $\{\#\} \in \text{fst } \psi'$ 
  shows unsatisfiable (fst  $\psi$ )
  using assms by (meson rtranclp-inference-preserve-models satisfiable-def true-clc-empty
    true-clss-def)

```

**lemma** *inference-soundness-and-completeness*:

```

  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes finite: finite (fst  $\psi$ )
  and snd  $\psi = \{\}$ 
  shows  $(\exists \psi'. (\text{inference** } \psi \ \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \text{unsatisfiable (fst } \psi)$ 
  using assms inference-completeness inference-soundness by metis

```

## 2.1.4 Lemma about the Simplified State

**abbreviation** *simplified*  $\psi \equiv (\text{no-step simplify } \psi)$

**lemma** *simplified-count*:

```

  assumes simp: simplified  $\psi$  and  $\chi: \chi \in \psi$ 
  shows count  $\chi \ L \leq 1$ 

```

**proof** –

```

  {
    let ? $\chi' = \chi - \{\#L, L\# \}$ 
    assume count  $\chi \ L \geq 2$ 
    then have f1: count  $(\chi - \{\#L, L\# \} + \{\#L, L\# \}) \ L = \text{count } \chi \ L$ 
    by simp
    then have  $L \in \# \ \chi - \{\#L\# \}$ 
    by (metis (no-types) add.left-neutral add-diff-cancel-left' count-union diff-diff-add)
  }

```

```

    diff-single-trivial insert-DiffM mem-Collect-eq multi-member-this not-gr0 set-mset-def)
  then have  $\chi': \{\#L, L\# \} + ?\chi' = \chi$ 
    using f1 in-diffD insert-DiffM by fastforce

  have  $\exists \psi'. \text{ simplify } \psi \ \psi'$ 
    by (metis (no-types, hide-lams)  $\chi \ \chi'$  factoring-imp-simplify)
  then have False using simp by auto
}
then show ?thesis by arith
qed

lemma simplified-no-both:
  assumes simp: simplified  $\psi$  and  $\chi: \chi \in \psi$ 
  shows  $\neg (L \in \# \chi \wedge \neg L \in \# \chi)$ 
proof (rule ccontr)
  assume  $\neg \neg (L \in \# \chi \wedge \neg L \in \# \chi)$ 
  then have  $L \in \# \chi \wedge \neg L \in \# \chi$  by metis
  then obtain  $\chi'$  where  $\chi = \text{add-mset } (\text{Pos } (\text{atm-of } L)) (\text{add-mset } (\text{Neg } (\text{atm-of } L)) \ \chi')$ 
    by (cases L) (auto dest!: multi-member-split simp: add-eq-conv-ex)
  then show False using  $\chi$  simp tautology-deletion by fast
qed

lemma add-mset-Neg-Pos-commute[simp]:
   $\text{add-mset } (\text{Neg } P) (\text{add-mset } (\text{Pos } P) \ C) = \text{add-mset } (\text{Pos } P) (\text{add-mset } (\text{Neg } P) \ C)$ 
  by (rule add-mset-commute)

lemma simplified-not-tautology:
  assumes simplified  $\{\psi\}$ 
  shows  $\sim \text{tautology } \psi$ 
proof (rule ccontr)
  assume  $\sim ?thesis$ 
  then obtain  $p$  where  $\text{Pos } p \in \# \psi \wedge \text{Neg } p \in \# \psi$  using tautology-decomp by metis
  then obtain  $\chi$  where  $\psi = \chi + \{\# \text{Pos } p\# \} + \{\# \text{Neg } p\# \}$ 
    by (auto dest!: multi-member-split simp: add-eq-conv-ex)
  then have  $\sim \text{ simplified } \{\psi\}$  by (auto intro: tautology-deletion)
  then show False using assms by auto
qed

lemma simplified-remove:
  assumes simplified  $\{\psi\}$ 
  shows simplified  $\{\psi - \{\#l\#\}\}$ 
proof (rule ccontr)
  assume ns:  $\neg \text{ simplified } \{\psi - \{\#l\#\}\}$ 
  {
    assume  $l \notin \# \psi$ 
    then have  $\psi - \{\#l\#\} = \psi$  by simp
    then have False using ns assms by auto
  }
  moreover {
    assume  $l\psi: l \in \# \psi$ 
    have  $A: \bigwedge A. A \in \{\psi - \{\#l\#\}\} \longleftrightarrow \text{add-mset } l \ A \in \{\psi\}$  by (auto simp add:  $l\psi$ )
    obtain  $l'$  where  $l': \text{ simplify } \{\psi - \{\#l\#\}\} \ l'$  using ns by metis
    then have  $\exists l'. \text{ simplify } \{\psi\} \ l'$ 
    proof (induction rule: simplify.induct)
      case (tautology-deletion P A)
      then have  $\{\# \text{Neg } P\# \} + (\{\# \text{Pos } P\# \} + (A + \{\#l\#\})) \in \{\psi\}$ 

```

```

      using A by auto
    then show ?thesis
      using simplified-no-both by fastforce
  next
    case (condensation L A)
    have add-mset l (add-mset L (add-mset L A)) ∈ {ψ}
      using condensation.hyps unfolding A by blast
    then have {#L, L#} + (A + {#l#}) ∈ {ψ}
      by auto
    then show ?case
      using factoring-imp-simplify by blast
  next
    case (subsumption A B)
    then show ?case by blast
  qed
  then have False using assms(1) by blast
}
ultimately show False by auto
qed

```

```

lemma in-simplified-simplified:
  assumes simp: simplified ψ and incl: ψ' ⊆ ψ
  shows simplified ψ'
proof (rule ccontr)
  assume ¬ ?thesis
  then obtain ψ'' where simplify ψ' ψ'' by metis
  then have ∃ l'. simplify ψ l'
  proof (induction rule: simplify.induct)
    case (tautology-deletion A P)
    then show ?thesis using simplify.tautology-deletion[of A P ψ] incl by blast
  next
    case (condensation A L)
    then show ?case using simplify.condensation[of A L ψ] incl by blast
  next
    case (subsumption A B)
    then show ?case using simplify.subsumption[of A ψ B] incl by auto
  qed
  then show False using assms(1) by blast
qed

```

```

lemma simplified-in:
  assumes simplified ψ
  and N ∈ ψ
  shows simplified {N}
  using assms by (metis Set.set-insert empty-subsetI in-simplified-simplified insert-mono)

```

```

lemma subsumes-imp-formula:
  assumes ψ ≤# φ
  shows {ψ} ⊨p φ
  unfolding true-clss-cls-def apply auto
  using assms true-clss-mono-leD by blast

```

```

lemma simplified-imp-distinct-mset-tauto:
  assumes simp: simplified ψ'
  shows distinct-mset-set ψ' and ∀ χ ∈ ψ'. ¬tautology χ

```

```

proof –
  show  $\forall \chi \in \psi'. \neg \text{tautology } \chi$ 
    using simp by (auto simp add: simplified-in simplified-not-tautology)

  show distinct-mset-set  $\psi'$ 
    proof (rule ccontr)
      assume  $\neg ?thesis$ 
      then obtain  $\chi$  where  $\chi \in \psi'$  and  $\neg \text{distinct-mset } \chi$  unfolding distinct-mset-set-def by auto
      then obtain  $L$  where  $\text{count } \chi \ L \geq 2$ 
        unfolding distinct-mset-def
        by (meson count-greater-eq-one-iff le-antisym simp simplified-count)
      then show False by (metis Suc-1 'χ ∈ ψ' not-less-eq-eq simp simplified-count)
    qed
qed

lemma simplified-no-more-full1-simplified:
  assumes simplified  $\psi$ 
  shows  $\neg \text{full1 simplify } \psi \ \psi'$ 
  using assms unfolding full1-def by (meson tranclpD)

```

## 2.1.5 Resolution and Invariants

```

inductive resolution :: 'v state  $\Rightarrow$  'v state  $\Rightarrow$  bool where
  full1-simp: full1 simplify  $N \ N' \Longrightarrow \text{resolution } (N, \text{already-used}) \ (N', \text{already-used}) \mid$ 
  inferring: inference  $(N, \text{already-used}) \ (N', \text{already-used}') \Longrightarrow \text{simplified } N$ 
     $\Longrightarrow \text{full simplify } N' \ N'' \Longrightarrow \text{resolution } (N, \text{already-used}) \ (N'', \text{already-used}')$ 

```

### Invariants

```

lemma resolution-finite:
  assumes resolution  $\psi \ \psi'$  and finite (fst  $\psi$ )
  shows finite (fst  $\psi'$ )
  using assms by (induct rule: resolution.induct)
    (auto simp add: full1-def full-def rtranclp-simplify-preserves-finite
      dest: tranclp-into-rtranclp inference-preserves-finite)

lemma rtranclp-resolution-finite:
  assumes resolution**  $\psi \ \psi'$  and finite (fst  $\psi$ )
  shows finite (fst  $\psi'$ )
  using assms by (induct rule: rtranclp-induct, auto simp add: resolution-finite)

lemma resolution-finite-snd:
  assumes resolution  $\psi \ \psi'$  and finite (snd  $\psi$ )
  shows finite (snd  $\psi'$ )
  using assms apply (induct rule: resolution.induct, auto simp add: inference-preserves-finite-snd)
  using inference-preserves-finite-snd snd-conv by metis

lemma rtranclp-resolution-finite-snd:
  assumes resolution**  $\psi \ \psi'$  and finite (snd  $\psi$ )
  shows finite (snd  $\psi'$ )
  using assms by (induct rule: rtranclp-induct, auto simp add: resolution-finite-snd)

lemma resolution-always-simplified:
  assumes resolution  $\psi \ \psi'$ 
  shows simplified (fst  $\psi'$ )
  using assms by (induct rule: resolution.induct)

```

(*auto simp add: full1-def full-def*)

**lemma** *trancpl-resolution-always-simplified:*

**assumes** *trancpl resolution  $\psi \psi'$*

**shows** *simplified (fst  $\psi'$ )*

**using** *assms by (induct rule: trancpl.induct, auto simp add: resolution-always-simplified)*

**lemma** *resolution-atms-of:*

**assumes** *resolution  $\psi \psi'$  and finite (fst  $\psi$ )*

**shows** *atms-of-ms (fst  $\psi'$ )  $\subseteq$  atms-of-ms (fst  $\psi$ )*

**using** *assms apply (induct rule: resolution.induct)*

**apply**(*simp add: rtrancpl-simplify-atms-of-ms trancpl-into-rtrancpl full1-def*)

**by** (*metis (no-types, lifting) contra-subsetD fst-conv full-def*

*inference-preserves-atms-of-ms rtrancpl-simplify-atms-of-ms subsetI*)

**lemma** *rtrancpl-resolution-atms-of:*

**assumes** *resolution\*\*  $\psi \psi'$  and finite (fst  $\psi$ )*

**shows** *atms-of-ms (fst  $\psi'$ )  $\subseteq$  atms-of-ms (fst  $\psi$ )*

**using** *assms apply (induct rule: rtrancpl-induct)*

**using** *resolution-atms-of rtrancpl-resolution-finite by blast+*

**lemma** *resolution-include:*

**assumes** *res: resolution  $\psi \psi'$  and finite: finite (fst  $\psi$ )*

**shows** *fst  $\psi' \subseteq$  simple-clss (atms-of-ms (fst  $\psi$ ))*

**proof** –

**have** *finite': finite (fst  $\psi'$ ) using local.finite res resolution-finite by blast*

**have** *simplified (fst  $\psi'$ ) using res finite' resolution-always-simplified by blast*

**then have** *fst  $\psi' \subseteq$  simple-clss (atms-of-ms (fst  $\psi'$ ))*

**using** *simplified-in-simple-clss finite' simplified-imp-distinct-mset-tauto[of fst  $\psi'$ ] by auto*

**moreover have** *atms-of-ms (fst  $\psi'$ )  $\subseteq$  atms-of-ms (fst  $\psi$ )*

**using** *res finite resolution-atms-of[of  $\psi \psi'$ ] by auto*

**ultimately show** *?thesis by (meson atms-of-ms-finite local.finite order.trans rev-finite-subset simple-clss-mono)*

**qed**

**lemma** *rtrancpl-resolution-include:*

**assumes** *res: trancpl resolution  $\psi \psi'$  and finite: finite (fst  $\psi$ )*

**shows** *fst  $\psi' \subseteq$  simple-clss (atms-of-ms (fst  $\psi$ ))*

**using** *assms apply (induct rule: trancpl.induct)*

**apply** (*simp add: resolution-include*)

**by** (*meson simple-clss-mono order-trans resolution-include*

*rtrancpl-resolution-atms-of rtrancpl-resolution-finite trancpl-into-rtrancpl*)

**abbreviation** *already-used-all-simple*

**::** (*'a literal multiset  $\times$  'a literal multiset*) *set  $\Rightarrow$  'a set  $\Rightarrow$  bool* **where**

*already-used-all-simple already-used vars  $\equiv$*

( $\forall (A, B) \in \text{already-used. simplified } \{A\} \wedge \text{simplified } \{B\} \wedge \text{atms-of } A \subseteq \text{vars} \wedge \text{atms-of } B \subseteq \text{vars}$ )

**lemma** *already-used-all-simple-vars-incl:*

**assumes** *vars  $\subseteq$  vars'*

**shows** *already-used-all-simple a vars  $\implies$  already-used-all-simple a vars'*

**using** *assms by fast*

**lemma** *inference-clause-preserves-already-used-all-simple:*

**assumes** *inference-clause S S'*

**and** *already-used-all-simple (snd S) vars*

```

and simplified (fst S)
and atms-of-ms (fst S)  $\subseteq$  vars
shows already-used-all-simple (snd (fst S  $\cup$  {fst S'}, snd S')) vars
using assms
proof (induct rule: inference-clause.induct)
case (factoring L C N already-used)
then show ?case by (simp add: simplified-in factoring-imp-simplify)
next
case (resolution P C N D already-used) note H = this
show ?case apply clarify
proof -
fix A B v
assume (A, B)  $\in$  snd (fst (N, already-used))
 $\cup$  {fst (C + D, already-used  $\cup$  {({#Pos P#} + C, {#Neg P#} + D))},
snd (C + D, already-used  $\cup$  {({#Pos P#} + C, {#Neg P#} + D))})
then have (A, B)  $\in$  already-used  $\vee$  (A, B) = ({#Pos P#} + C, {#Neg P#} + D) by auto
moreover {
assume (A, B)  $\in$  already-used
then have simplified {A}  $\wedge$  simplified {B}  $\wedge$  atms-of A  $\subseteq$  vars  $\wedge$  atms-of B  $\subseteq$  vars
using H(4) by auto
}
moreover {
assume eq: (A, B) = ({#Pos P#} + C, {#Neg P#} + D)
then have simplified {A} using simplified-in H(1,5) by auto
moreover have simplified {B} using eq simplified-in H(2,5) by auto
moreover have atms-of A  $\subseteq$  atms-of-ms N
using eq H(1)
using atms-of-atms-of-ms-mono[of A N] by auto
moreover have atms-of B  $\subseteq$  atms-of-ms N
using eq H(2) atms-of-atms-of-ms-mono[of B N] by auto
ultimately have simplified {A}  $\wedge$  simplified {B}  $\wedge$  atms-of A  $\subseteq$  vars  $\wedge$  atms-of B  $\subseteq$  vars
using H(6) by auto
}
ultimately show simplified {A}  $\wedge$  simplified {B}  $\wedge$  atms-of A  $\subseteq$  vars  $\wedge$  atms-of B  $\subseteq$  vars
by fast
qed
qed

```

```

lemma inference-preserves-already-used-all-simple:
assumes inference S S'
and already-used-all-simple (snd S) vars
and simplified (fst S)
and atms-of-ms (fst S)  $\subseteq$  vars
shows already-used-all-simple (snd S') vars
using assms
proof (induct rule: inference.induct)
case (inference-step S clause already-used)
then show ?case
using inference-clause-preserves-already-used-all-simple[of S (clause, already-used) vars]
by auto
qed

```

```

lemma already-used-all-simple-inv:
assumes resolution S S'
and already-used-all-simple (snd S) vars
and atms-of-ms (fst S)  $\subseteq$  vars

```

```

  shows already-used-all-simple (snd S') vars
  using assms
proof (induct rule: resolution.induct)
  case (full1-simp N N')
  then show ?case by simp
next
  case (inferring N already-used N' already-used' N'')
  then show already-used-all-simple (snd (N'', already-used')) vars
    using inference-preserves-already-used-all-simple[of (N, already-used)] by simp
qed

lemma rtranclp-already-used-all-simple-inv:
  assumes resolution** S S'
  and already-used-all-simple (snd S) vars
  and atms-of-ms (fst S)  $\subseteq$  vars
  and finite (fst S)
  shows already-used-all-simple (snd S') vars
  using assms
proof (induct rule: rtranclp-induct)
  case base
  then show ?case by simp
next
  case (step S' S'') note infstar = this(1) and IH = this(3) and res = this(2) and
    already = this(4) and atms = this(5) and finite = this(6)
  have already-used-all-simple (snd S') vars using IH already atms finite by simp
  moreover have atms-of-ms (fst S')  $\subseteq$  atms-of-ms (fst S)
    by (simp add: infstar local.finite rtranclp-resolution-atms-of)
  then have atms-of-ms (fst S')  $\subseteq$  vars using atms by auto
  ultimately show ?case
    using already-used-all-simple-inv[OF res] by simp
qed

lemma inference-clause-simplified-already-used-subset:
  assumes inference-clause S S'
  and simplified (fst S)
  shows snd S  $\subset$  snd S'
  using assms apply (induct rule: inference-clause.induct)
  using factoring-imp-simplify apply (simp; blast)
  using factoring-imp-simplify by force

lemma inference-simplified-already-used-subset:
  assumes inference S S'
  and simplified (fst S)
  shows snd S  $\subset$  snd S'
  using assms apply (induct rule: inference.induct)
  by (metis inference-clause-simplified-already-used-subset snd-conv)

lemma resolution-simplified-already-used-subset:
  assumes resolution S S'
  and simplified (fst S)
  shows snd S  $\subset$  snd S'
  using assms apply (induct rule: resolution.induct, simp-all add: full1-def)
  apply (meson tranclpD)
  by (metis inference-simplified-already-used-subset fst-conv snd-conv)

lemma tranclp-resolution-simplified-already-used-subset:

```



**assumes** *tracp resolution S S'*  
**and** *simplified (fst S)*  
**shows** *snd S  $\subset$  snd S'*  
**using** *assms apply (induct rule: tracp.induct)*  
**using** *resolution-simplified-already-used-subset apply metis*  
**by** (*meson tracp-resolution-always-simplified resolution-simplified-already-used-subset less-trans*)

**abbreviation** *already-used-top vars  $\equiv$  simple-clss vars  $\times$  simple-clss vars*

**lemma** *already-used-all-simple-in-already-used-top:*

**assumes** *already-used-all-simple s vars and finite vars*  
**shows** *s  $\subseteq$  already-used-top vars*

**proof**

**fix** *x*  
**assume** *x-s: x  $\in$  s*  
**obtain** *A B where x: x = (A, B) by (cases x, auto)*  
**then have** *simplified {A} and atms-of A  $\subseteq$  vars using assms(1) x-s by fastforce+*  
**then have** *A: A  $\in$  simple-clss vars*  
   **using** *simple-clss-mono[of atms-of A vars] x assms(2)*  
   *simplified-imp-distinct-mset-tauto[of {A}]*  
   *distinct-mset-not-tautology-implies-in-simple-clss by fast*  
**moreover have** *simplified {B} and atms-of B  $\subseteq$  vars using assms(1) x-s x by fast+*  
**then have** *B: B  $\in$  simple-clss vars*  
   **using** *simplified-imp-distinct-mset-tauto[of {B}]*  
   *distinct-mset-not-tautology-implies-in-simple-clss*  
   *simple-clss-mono[of atms-of B vars] x assms(2) by fast*  
**ultimately show** *x  $\in$  simple-clss vars  $\times$  simple-clss vars*  
   **unfolding** *x by auto*

**qed**

**lemma** *already-used-top-finite:*

**assumes** *finite vars*  
**shows** *finite (already-used-top vars)*  
**using** *simple-clss-finite assms by auto*

**lemma** *already-used-top-increasing:*

**assumes** *var  $\subseteq$  var' and finite var'*  
**shows** *already-used-top var  $\subseteq$  already-used-top var'*  
**using** *assms simple-clss-mono by auto*

**lemma** *already-used-all-simple-finite:*

**fixes** *s :: ('a literal multiset  $\times$  'a literal multiset) set and vars :: 'a set*  
**assumes** *already-used-all-simple s vars and finite vars*  
**shows** *finite s*  
**using** *assms already-used-all-simple-in-already-used-top[OF assms(1)]*  
*rev-finite-subset[OF already-used-top-finite[of vars]] by auto*

**abbreviation** *card-simple vars  $\psi \equiv$  card (already-used-top vars  $- \psi$ )*

**lemma** *resolution-card-simple-decreasing:*

**assumes** *res: resolution  $\psi \psi'$*   
**and** *a-u-s: already-used-all-simple (snd  $\psi$ ) vars*  
**and** *finite-v: finite vars*  
**and** *finite-fst: finite (fst  $\psi$ )*  
**and** *finite-snd: finite (snd  $\psi$ )*

**and** *simp*: *simplified* (*fst*  $\psi$ )  
**and** *atms-of-ms* (*fst*  $\psi$ )  $\subseteq$  *vars*  
**shows** *card-simple vars* (*snd*  $\psi'$ ) < *card-simple vars* (*snd*  $\psi$ )  
**proof** –  
**let** *?vars* = *vars*  
**let** *?top* = *simple-clss* *?vars*  $\times$  *simple-clss* *?vars*  
**have** 1: *card-simple vars* (*snd*  $\psi$ ) = *card* *?top* – *card* (*snd*  $\psi$ )  
**using** *card-Diff-subset finite-snd already-used-all-simple-in-already-used-top*[*OF* *a-u-s*]  
*finite-v* **by** *metis*  
**have** *a-u-s'*: *already-used-all-simple* (*snd*  $\psi'$ ) *vars*  
**using** *already-used-all-simple-inv res a-u-s assms*(7) **by** *blast*  
**have** *f*: *finite* (*snd*  $\psi'$ ) **using** *already-used-all-simple-finite a-u-s' finite-v* **by** *auto*  
**have** 2: *card-simple vars* (*snd*  $\psi'$ ) = *card* *?top* – *card* (*snd*  $\psi'$ )  
**using** *card-Diff-subset*[*OF* *f*] *already-used-all-simple-in-already-used-top*[*OF* *a-u-s' finite-v*]  
**by** *auto*  
**have** *card* (*already-used-top vars*)  $\geq$  *card* (*snd*  $\psi'$ )  
**using** *already-used-all-simple-in-already-used-top*[*OF* *a-u-s' finite-v*]  
*card-mono*[*of already-used-top vars snd*  $\psi'$ ] *already-used-top-finite*[*OF* *finite-v*] **by** *metis*  
**then show** *?thesis*  
**using** *psubset-card-mono*[*OF* *f resolution-simplified-already-used-subset*[*OF* *res simp*]]  
**unfolding** 1 2 **by** *linarith*  
**qed**

**lemma** *trancpl-resolution-card-simple-decreasing*:

**assumes** *trancpl resolution*  $\psi$   $\psi'$  **and** *finite-fst*: *finite* (*fst*  $\psi$ )  
**and** *already-used-all-simple* (*snd*  $\psi$ ) *vars*  
**and** *atms-of-ms* (*fst*  $\psi$ )  $\subseteq$  *vars*  
**and** *finite-v*: *finite vars*  
**and** *finite-snd*: *finite* (*snd*  $\psi$ )  
**and** *simplified* (*fst*  $\psi$ )  
**shows** *card-simple vars* (*snd*  $\psi'$ ) < *card-simple vars* (*snd*  $\psi$ )  
**using** *assms*  
**proof** (*induct rule: trancpl-induct*)  
**case** (*base*  $\psi'$ )  
**then show** *?case* **by** (*simp add: resolution-card-simple-decreasing*)  
**next**  
**case** (*step*  $\psi'$   $\psi''$ ) **note** *res* = *this*(1) **and** *res'* = *this*(2) **and** *a-u-s* = *this*(5) **and**  
*atms* = *this*(6) **and** *f-v* = *this*(7) **and** *f-fst* = *this*(4) **and** *H* = *this*  
**then have** *card-simple vars* (*snd*  $\psi'$ ) < *card-simple vars* (*snd*  $\psi$ ) **by** *auto*  
**moreover have** *a-u-s'*: *already-used-all-simple* (*snd*  $\psi'$ ) *vars*  
**using** *rtrancpl-already-used-all-simple-inv*[*OF* *trancpl-into-rtrancpl*[*OF* *res*] *a-u-s atms f-fst*]  
**have** *finite* (*fst*  $\psi'$ )  
**by** (*meson finite-fst res rtrancpl-resolution-finite trancpl-into-rtrancpl*)  
**moreover have** *finite* (*snd*  $\psi'$ ) **using** *already-used-all-simple-finite*[*OF* *a-u-s' f-v*]  
**moreover have** *simplified* (*fst*  $\psi'$ ) **using** *res trancpl-resolution-always-simplified* **by** *blast*  
**moreover have** *atms-of-ms* (*fst*  $\psi'$ )  $\subseteq$  *vars*  
**by** (*meson atms f-fst order.trans res rtrancpl-resolution-atms-of trancpl-into-rtrancpl*)  
**ultimately show** *?case*  
**using** *resolution-card-simple-decreasing*[*OF* *res' a-u-s' f-v*] *f-v*  
*less-trans*[*of card-simple vars* (*snd*  $\psi''$ ) *card-simple vars* (*snd*  $\psi'$ )  
*card-simple vars* (*snd*  $\psi$ )]  
**by** *blast*  
**qed**

**lemma** *tracp-resolution-card-simple-decreasing-2*:  
**assumes** *tracp resolution*  $\psi \psi'$   
**and** *finite-fst*: *finite* (*fst*  $\psi$ )  
**and** *empty-snd*: *snd*  $\psi = \{\}$   
**and** *simplified* (*fst*  $\psi$ )  
**shows** *card-simple* (*atms-of-ms* (*fst*  $\psi$ )) (*snd*  $\psi'$ )  $<$  *card-simple* (*atms-of-ms* (*fst*  $\psi$ )) (*snd*  $\psi$ )  
**proof** –  
**let** *?vars* = *atms-of-ms* (*fst*  $\psi$ )  
**have** *already-used-all-simple* (*snd*  $\psi$ ) *?vars* **unfolding** *empty-snd* **by** *auto*  
**moreover** **have** *atms-of-ms* (*fst*  $\psi$ )  $\subseteq$  *?vars* **by** *auto*  
**moreover** **have** *finite-v*: *finite* *?vars* **using** *finite-fst* **by** *auto*  
**moreover** **have** *finite-snd*: *finite* (*snd*  $\psi$ ) **unfolding** *empty-snd* **by** *auto*  
**ultimately** **show** *?thesis*  
**using** *assms*(1,2,4) *tracp-resolution-card-simple-decreasing*[*of*  $\psi \psi'$ ] **by** *presburger*  
**qed**

## Well-Foundness of the Relation

**lemma** *wf-simplified-resolution*:  
**assumes** *f-vars*: *finite vars*  
**shows** *wf*  $\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$   
**proof** –  
{  
**fix** *a b* :: *'v::linorder state*  
**assume**  $(b, a) \in \{(y, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$   
**then have**  
*atms-of-ms* (*fst* *a*)  $\subseteq$  *vars* **and**  
*simp*: *simplified* (*fst* *a*) **and**  
*finite* (*snd* *a*) **and**  
*finite* (*fst* *a*) **and**  
*a-u-v*: *already-used-all-simple* (*snd* *a*) *vars* **and**  
*res*: *resolution* *a b* **by** *auto*  
**have** *finite* (*already-used-top vars*) **using** *f-vars* *already-used-top-finite* **by** *blast*  
**moreover** **have** *already-used-top vars*  $\subseteq$  *already-used-top vars* **by** *auto*  
**moreover** **have** *snd b*  $\subseteq$  *already-used-top vars*  
**using** *already-used-all-simple-in-already-used-top*[*of* *snd b vars*]  
*a-u-v* *already-used-all-simple-inv*[*OF res*] (*finite* (*fst a*)) (*atms-of-ms* (*fst a*)  $\subseteq$  *vars*) *f-vars*  
**by** *presburger*  
**moreover** **have** *snd a*  $\subset$  *snd b* **using** *resolution-simplified-already-used-subset*[*OF res simp*] .  
**ultimately** **have** *finite* (*already-used-top vars*)  $\wedge$  *already-used-top vars*  $\subseteq$  *already-used-top vars*  
 $\wedge$  *snd b*  $\subseteq$  *already-used-top vars*  $\wedge$  *snd a*  $\subset$  *snd b* **by** *metis*  
}  
**then show** *?thesis* **using** *wf-bounded-set*[*of*  $\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$   $\lambda\cdot$ . *already-used-top vars* *snd*] **by** *auto*  
**qed**

**lemma** *wf-simplified-resolution'*:  
**assumes** *f-vars*: *finite vars*  
**shows** *wf*  $\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \neg \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$   
**unfolding** *wf-def*  
**apply** (*simp add*: *resolution-always-simplified*)

by (metis (mono-tags, hide-lams) fst-conv resolution-always-simplified)

**lemma** wf-resolution:

**assumes** f-vars: finite vars

**shows** wf ( $\{(y: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x \ y\} \cup \{(y, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \neg \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x \ y\})$  **(is** wf (?R  $\cup$  ?S))

**proof** –

**have** Domain ?R Int Range ?S = {} **using** resolution-always-simplified **by** auto blast

**then show** wf (?R  $\cup$  ?S)

**using** wf-simplified-resolution[OF f-vars] wf-simplified-resolution'[OF f-vars] wf-Un[of ?R ?S]

**by** fast

**qed**

**lemma** rtranclp-simplify-already-used-inv:

**assumes** simplify\*\* S S'

**and** already-used-inv (S, N)

**shows** already-used-inv (S', N)

**using** assms **apply** induction

**using** simplify-preserves-already-used-inv **by** fast+

**lemma** full1-simplify-already-used-inv:

**assumes** full1 simplify S S'

**and** already-used-inv (S, N)

**shows** already-used-inv (S', N)

**using** assms tranclp-into-rtranclp[of simplify S S'] rtranclp-simplify-already-used-inv

**unfolding** full1-def **by** fast

**lemma** full-simplify-already-used-inv:

**assumes** full simplify S S'

**and** already-used-inv (S, N)

**shows** already-used-inv (S', N)

**using** assms rtranclp-simplify-already-used-inv **unfolding** full-def **by** fast

**lemma** resolution-already-used-inv:

**assumes** resolution S S'

**and** already-used-inv S

**shows** already-used-inv S'

**using** assms

**proof** induction

**case** (full1-simp N N' already-used)

**then show** ?case **using** full1-simplify-already-used-inv **by** fast

**next**

**case** (inferring N already-used N' already-used' N'') **note** inf = this(1) **and** full = this(3) **and** a-u-v = this(4)

**then show** ?case

**using** inference-preserves-already-used-inv[OF inf a-u-v] full-simplify-already-used-inv full

**by** fast

**qed**

**lemma** rtranclp-resolution-already-used-inv:

**assumes** resolution\*\* S S'

**and** already-used-inv S

**shows** already-used-inv S'

**using** assms **apply** induction

**using** resolution-already-used-inv **by** fast+

```

lemma rtanclp-simplify-preserves-unsat:
  assumes simplify**  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms apply induction
  using simplify-clause-preserves-sat by blast

lemma full1-simplify-preserves-unsat:
  assumes full1 simplify  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms rtanclp-simplify-preserves-unsat[of  $\psi$   $\psi'$ ] tranclp-into-rtranclp
  unfolding full1-def by metis

lemma full-simplify-preserves-unsat:
  assumes full simplify  $\psi$   $\psi'$ 
  shows satisfiable  $\psi' \longrightarrow$  satisfiable  $\psi$ 
  using assms rtanclp-simplify-preserves-unsat[of  $\psi$   $\psi'$ ] unfolding full-def by metis

lemma resolution-preserves-unsat:
  assumes resolution  $\psi$   $\psi'$ 
  shows satisfiable (fst  $\psi'$ )  $\longrightarrow$  satisfiable (fst  $\psi$ )
  using assms apply (induct rule: resolution.induct)
  using full1-simplify-preserves-unsat apply (metis fst-conv)
  using full-simplify-preserves-unsat simplify-preserves-unsat by fastforce

lemma rtranclp-resolution-preserves-unsat:
  assumes resolution**  $\psi$   $\psi'$ 
  shows satisfiable (fst  $\psi'$ )  $\longrightarrow$  satisfiable (fst  $\psi$ )
  using assms apply induction
  using resolution-preserves-unsat by fast

lemma rtranclp-simplify-preserve-partial-tree:
  assumes simplify**  $N$   $N'$ 
  and partial-interps  $t$   $I$   $N$ 
  shows partial-interps  $t$   $I$   $N'$ 
  using assms apply (induction, simp)
  using simplify-preserve-partial-tree by metis

lemma full1-simplify-preserve-partial-tree:
  assumes full1 simplify  $N$   $N'$ 
  and partial-interps  $t$   $I$   $N$ 
  shows partial-interps  $t$   $I$   $N'$ 
  using assms rtranclp-simplify-preserve-partial-tree[of  $N$   $N'$   $t$   $I$ ] tranclp-into-rtranclp
  unfolding full1-def by fast

lemma full-simplify-preserve-partial-tree:
  assumes full simplify  $N$   $N'$ 
  and partial-interps  $t$   $I$   $N$ 
  shows partial-interps  $t$   $I$   $N'$ 
  using assms rtranclp-simplify-preserve-partial-tree[of  $N$   $N'$   $t$   $I$ ] tranclp-into-rtranclp
  unfolding full-def by fast

lemma resolution-preserve-partial-tree:
  assumes resolution  $S$   $S'$ 
  and partial-interps  $t$   $I$  (fst  $S$ )
  shows partial-interps  $t$   $I$  (fst  $S'$ )

```

```

using assms apply induction
using full1-simplify-preserve-partial-tree fst-conv apply metis
using full-simplify-preserve-partial-tree inference-preserve-partial-tree by fastforce

lemma rtrancp-resolution-preserve-partial-tree:
  assumes resolution** S S'
  and partial-interps t I (fst S)
  shows partial-interps t I (fst S')
  using assms apply induction
  using resolution-preserve-partial-tree by fast+
  thm nat-less-induct nat.induct

lemma nat-ge-induct[case-names 0 Suc]:
  assumes P 0
  and  $\bigwedge n. (\bigwedge m. m < \text{Suc } n \implies P m) \implies P (\text{Suc } n)$ 
  shows P n
  using assms apply (induct rule: nat-less-induct)
  by (rename-tac n, case-tac n) auto

lemma wf-always-more-step-False:
  assumes wf R
  shows  $(\forall x. \exists z. (z, x) \in R) \implies \text{False}$ 
  using assms unfolding wf-def by (meson Domain.DomainI assms wfE-min)

lemma finite-finite-mset-element-of-mset[simp]:
  assumes finite N
  shows finite {f  $\varphi$  L |  $\varphi$  L.  $\varphi \in N \wedge L \in \# \varphi \wedge P \varphi L$ }
  using assms
proof (induction N rule: finite-induct)
  case empty
  show ?case by auto
next
  case (insert x N) note finite = this(1) and IH = this(3)
  have  $\{f \varphi L \mid \varphi L. (\varphi = x \vee \varphi \in N) \wedge L \in \# \varphi \wedge P \varphi L\} \subseteq \{f x L \mid L. L \in \# x \wedge P x L\}$ 
   $\cup \{f \varphi L \mid \varphi L. \varphi \in N \wedge L \in \# \varphi \wedge P \varphi L\}$  by auto
  moreover have finite {f x L | L. L  $\in$  # x} by auto
  ultimately show ?case using IH finite-subset by fastforce
qed

definition sum-count-ge-2 :: 'a multiset set  $\Rightarrow$  nat ( $\Xi$ ) where
sum-count-ge-2  $\equiv$  folding.F ( $\lambda \varphi. (+)(\text{sum-mset } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L\}\}) 0$ 

interpretation sum-count-ge-2:
  folding  $\lambda \varphi. (+)(\text{sum-mset } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L\}) 0$ 
rewrites
  folding.F ( $\lambda \varphi. (+)(\text{sum-mset } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L\}) 0 = \text{sum-count-ge-2}$ 
proof –
  show folding ( $\lambda \varphi. (+) (\text{sum-mset } (\text{image-mset } (\text{count } \varphi) \{\# L \in \# \varphi. 2 \leq \text{count } \varphi L\}\}))$ 
  by standard auto
  then interpret sum-count-ge-2:
  folding  $\lambda \varphi. (+)(\text{sum-mset } \{\# \text{count } \varphi L \mid L \in \# \varphi. 2 \leq \text{count } \varphi L\}) 0$  .
  show folding.F ( $\lambda \varphi. (+) (\text{sum-mset } (\text{image-mset } (\text{count } \varphi) \{\# L \in \# \varphi. 2 \leq \text{count } \varphi L\}\})) 0$ 
   $= \text{sum-count-ge-2}$  by (auto simp add: sum-count-ge-2-def)
qed

```

**lemma** *finite-incl-le-setsum*:  
*finite* ( $B :: 'a$  multiset set)  $\implies A \subseteq B \implies \Xi A \leq \Xi B$   
**proof** (*induction arbitrary:A rule: finite-induct*)  
  **case** *empty*  
  **then show** *?case* **by** *simp*  
**next**  
  **case** (*insert a F*) **note** *finite = this(1)* **and** *aF = this(2)* **and** *IH = this(3)* **and** *AF = this(4)*  
  **show** *?case*  
  **proof** (*cases a ∈ A*)  
  **assume**  $a \notin A$   
  **then have**  $A \subseteq F$  **using** *AF* **by** *auto*  
  **then show** *?case* **using** *IH[of A]* **by** (*simp add: aF local.finite*)  
**next**  
  **assume**  $aA: a \in A$   
  **then have**  $A - \{a\} \subseteq F$  **using** *AF* **by** *auto*  
  **then have**  $\Xi (A - \{a\}) \leq \Xi F$  **using** *IH* **by** *blast*  
  **then show** *?case*  
  **proof** –  
  **obtain**  $nn :: nat \Rightarrow nat \Rightarrow nat$  **where**  
   $\forall x0\ x1. (\exists v2. x0 = x1 + v2) = (x0 = x1 + nn\ x0\ x1)$   
  **by** *moura*  
  **then have**  $\Xi F = \Xi (A - \{a\}) + nn\ (\Xi F)\ (\Xi (A - \{a\}))$   
  **by** (*meson*  $\langle \Xi (A - \{a\}) \leq \Xi F \rangle$  *le-iff-add*)  
  **then show** *?thesis*  
  **by** (*metis* (*no-types*) *le-iff-add aA aF add.assoc finite.insertI finite-subset*  
  *insert.premis local.finite sum-count-ge-2.insert sum-count-ge-2.remove*)  
  **qed**  
**qed**  
**qed**

**lemma** *simplify-finite-measure-decrease*:  
*simplify*  $N\ N' \implies \text{finite } N \implies \text{card } N' + \Xi N' < \text{card } N + \Xi N$   
**proof** (*induction rule: simplify.induct*)  
  **case** (*tautology-deletion P A*) **note**  $an = \text{this}(1)$  **and**  $fin = \text{this}(2)$   
  **let**  $?N' = N - \{\text{add-mset } (Pos\ P)\ (\text{add-mset } (Neg\ P)\ A)\}$   
  **have**  $\text{card } ?N' < \text{card } N$   
  **by** (*meson card-Diff1-less tautology-deletion.hyps tautology-deletion.premis*)  
  **moreover have**  $?N' \subseteq N$  **by** *auto*  
  **then have**  $\text{sum-count-ge-2 } ?N' \leq \text{sum-count-ge-2 } N$  **using** *finite-incl-le-setsum[OF fin]* **by** *blast*  
  **ultimately show** *?case* **by** *linarith*  
**next**  
  **case** (*condensation L A*) **note**  $AN = \text{this}(1)$  **and**  $fin = \text{this}(2)$   
  **let**  $?C' = \text{add-mset } L\ A$   
  **let**  $?C = \text{add-mset } L\ ?C'$   
  **let**  $?N' = N - \{?C\} \cup \{?C'\}$   
  **have**  $\text{card } ?N' \leq \text{card } N$   
  **using**  $AN$  **by** (*metis* (*no-types, lifting*) *Diff-subset Un-empty-right Un-insert-right card.remove*  
  *card-insert-if card-mono fin finite-Diff order-refl*)  
  **moreover have**  $\Xi \{?C'\} < \Xi \{?C\}$   
  **proof** –  
  **have** *mset-decomp*:  
   $\{\# La \in \# A. (L = La \longrightarrow La \in \# A) \wedge (L \neq La \longrightarrow 2 \leq \text{count } A\ La)\#\}$   
   $= \{\# La \in \# A. L \neq La \wedge 2 \leq \text{count } A\ La\#\} +$   
   $\{\# La \in \# A. L = La \wedge \text{Suc } 0 \leq \text{count } A\ L\#\}$   
  **by** (*auto simp: multiset-eq-iff ac-simps*)

```

have mset-decomp2: {# La ∈# A. L ≠ La → 2 ≤ count A La#} =
  {# La ∈# A. L ≠ La ∧ 2 ≤ count A La#} + replicate-mset (count A L) L
by (auto simp: multiset-eq-iff)
have *: (∑ x∈#B. if L = x then Suc (count A x) else count A x) ≤
  (∑ x∈#B. if L = x then Suc (count (add-mset L A) x) else count (add-mset L A) x)
for B
by (auto intro!: sum-mset-mono)
show ?thesis
  using *[of {# La ∈# A. L ≠ La ∧ 2 ≤ count A La#}]
  by (auto simp: mset-decomp mset-decomp2 filter-mset-eq)
qed
have ∃ ?N' < ∃ N
proof cases
  assume a1: ?C' ∈ N
  then show ?thesis
  proof -
    have f2: ∧m M. insert (m::'a literal multiset) (M - {m}) = M ∪ {} ∨ m ∉ M
    using Un-empty-right insert-Diff by blast
    have f3: ∧m M Ma. insert (m::'a literal multiset) M - insert m Ma = M - insert m Ma
    by simp
    then have f4: ∧M m. M - {m::'a literal multiset} = M ∪ {} ∨ m ∈ M
    using Diff-insert-absorb Un-empty-right by fastforce
    have f5: insert ?C N = N
    using f3 f2 Un-empty-right condensation.hyps insert-iff by fastforce
    have ∧m M. insert (m::'a literal multiset) M = M ∪ {} ∨ m ∉ M
    using f3 f2 Un-empty-right add.right-neutral insert-iff by fastforce
    then have ∃ (N - {?C}) < ∃ N
    using f5 f4 by (metis Un-empty-right ⟨∃ {?C'} < ∃ {?C}⟩
      add.right-neutral add-diff-cancel-left' add-gr-0 diff-less fin finite.emptyI not-le
      sum-count-ge-2.empty sum-count-ge-2.insert-remove trans-le-add2)
    then show ?thesis
    using f3 f2 a1 by (metis (no-types) Un-empty-right Un-insert-right condensation.hyps
      insert-iff multi-self-add-other-not-self)
  qed
next
  assume ?C' ∉ N
  have mset-decomp:
    {# La ∈# A. (L = La → Suc 0 ≤ count A La) ∧ (L ≠ La → 2 ≤ count A La)#}
    = {# La ∈# A. L ≠ La ∧ 2 ≤ count A La#} +
      {# La ∈# A. L = La ∧ Suc 0 ≤ count A L#}
    by (auto simp: multiset-eq-iff ac-simps)
  have mset-decomp2: {# La ∈# A. L ≠ La → 2 ≤ count A La#} =
    {# La ∈# A. L ≠ La ∧ 2 ≤ count A La#} + replicate-mset (count A L) L
  by (auto simp: multiset-eq-iff)

  show ?thesis
  using ⟨∃ {?C'} < ∃ {?C}⟩ condensation.hyps fin
    sum-count-ge-2.remove[of - ?C] ⟨?C' ∉ N⟩
  by (auto simp: mset-decomp mset-decomp2 filter-mset-eq)
qed
ultimately show ?case by linarith
next
case (subsumption A B) note AN = this(1) and AB = this(2) and BN = this(3) and fin = this(4)
have card (N - {B}) < card N using BN by (meson card-Diff1-less subsumption.prem)
moreover have ∃ (N - {B}) ≤ ∃ N
  by (simp add: Diff-subset finite-incl-le-setsum subsumption.prem)

```



ultimately show ?case by linarith  
qed

lemma *simplify-terminates*:

wf {(N', N). finite N ∧ simplify N N'}  
apply (rule wfP-if-measure[of finite simplify λN. card N + ∑ N])  
using simplify-finite-measure-decrease by blast

lemma *wf-terminates*:

assumes wf r  
shows  $\exists N'. (N', N) \in r^* \wedge (\forall N''. (N'', N') \notin r)$

proof –

let ?P = λN. ( $\exists N'. (N', N) \in r^* \wedge (\forall N''. (N'', N') \notin r)$ )

have  $\forall x. (\forall y. (y, x) \in r \longrightarrow ?P y) \longrightarrow ?P x$

proof clarify

fix x

assume H:  $\forall y. (y, x) \in r \longrightarrow ?P y$

{ assume  $\exists y. (y, x) \in r$

then obtain y where  $y: (y, x) \in r$  by blast

then have ?P y using H by blast

then have ?P x using y by (meson rtrancl.rtrancl-into-rtrancl)

}

moreover {

assume  $\neg(\exists y. (y, x) \in r)$

then have ?P x by auto

}

ultimately show ?P x by blast

qed

moreover have  $(\forall x. (\forall y. (y, x) \in r \longrightarrow ?P y) \longrightarrow ?P x) \longrightarrow \text{All } ?P$

using assms unfolding wf-def by (rule allE)

ultimately have All ?P by blast

then show ?P N by blast

qed

lemma *rtranclp-simplify-terminates*:

assumes fin: finite N

shows  $\exists N'. \text{simplify}^{**} N N' \wedge \text{simplified } N'$

proof –

have H:  $\{(N', N). \text{finite } N \wedge \text{simplify } N N'\} = \{(N', N). \text{simplify } N N' \wedge \text{finite } N\}$  by auto

then have wf: wf {(N', N). simplify N N' ∧ finite N}

using simplify-terminates by (simp add: H)

obtain N' where N':  $(N', N) \in \{(b, a). \text{simplify } a b \wedge \text{finite } a\}^*$  and

more:  $\forall N''. (N'', N') \notin \{(b, a). \text{simplify } a b \wedge \text{finite } a\}$

using Prop-Resolution.wf-terminates[OF wf, of N] by blast

have 1:  $\text{simplify}^{**} N N'$

using N' by (induction rule: rtrancl.induct) auto

then have finite N' using fin rtranclp-simplify-preserves-finite by blast

then have 2:  $\forall N''. \neg \text{simplify } N' N''$  using more by auto

show ?thesis using 1 2 by blast

qed

lemma *finite-simplified-full1-simp*:

assumes finite N

shows  $\text{simplified } N \vee (\exists N'. \text{full1 simplify } N N')$

**using** *rtrancp-simplify-terminates*[*OF assms*] **unfolding** *full1-def*  
**by** (*metis Nitpick.rtrancp-unfold*)

**lemma** *finite-simplified-full-simp*:

**assumes** *finite N*

**shows**  $\exists N'. \text{full simplify } N N'$

**using** *rtrancp-simplify-terminates*[*OF assms*] **unfolding** *full-def* **by** *metis*

**lemma** *can-decrease-tree-size-resolution*:

**fixes**  $\psi :: 'v \text{ state}$  **and**  $\text{tree} :: 'v \text{ sem-tree}$

**assumes** *finite (fst  $\psi$ )* **and** *already-used-inv  $\psi$*

**and** *partial-interps tree I (fst  $\psi$ )*

**and** *simplified (fst  $\psi$ )*

**shows**  $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{resolution}^{**} \psi \psi' \wedge \text{partial-interps tree}' I (\text{fst } \psi')$

$\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0)$

**using** *assms*

**proof** (*induct arbitrary: I rule: sem-tree-size*)

**case** (*bigger xs I*) **note** *IH = this(1)* **and** *finite = this(2)* **and** *a-u-i = this(3)* **and** *part = this(4)*  
**and** *simp = this(5)*

{ **assume** *sem-tree-size xs = 0*  
**then have** *?case using part by blast*  
}

**moreover** {

**assume** *sn0: sem-tree-size xs > 0*

**obtain** *ag ad v* **where** *xs: xs = Node v ag ad* **using** *sn0* **by** (*cases xs, auto*)

{

**assume** *sem-tree-size ag = 0  $\wedge$  sem-tree-size ad = 0*

**then have** *ag: ag = Leaf* **and** *ad: ad = Leaf* **by** (*cases ag, auto, cases ad, auto*)

**then obtain**  $\chi \chi'$  **where**

$\chi: \neg I \cup \{\text{Pos } v\} \models \chi$  **and**

*tot $\chi$ : total-over-m ( $I \cup \{\text{Pos } v\}$ )  $\{\chi\}$*  **and**

$\chi\psi: \chi \in \text{fst } \psi$  **and**

$\chi': \neg I \cup \{\text{Neg } v\} \models \chi'$  **and**

*tot $\chi'$ : total-over-m ( $I \cup \{\text{Neg } v\}$ )  $\{\chi'\}$*  **and**  $\chi'\psi: \chi' \in \text{fst } \psi$

**using** *part unfolding xs by auto*

**have** *Posv: Pos v  $\notin \# \chi$*  **using**  $\chi$  **unfolding** *true-cls-def true-lit-def* **by** *auto*

**have** *Negv: Neg v  $\notin \# \chi'$*  **using**  $\chi'$  **unfolding** *true-cls-def true-lit-def* **by** *auto*

{

**assume** *Neg $\chi$ : Neg v  $\notin \# \chi$*

**then have**  $\neg I \models \chi$  **using**  $\chi$  *Posv* **unfolding** *true-cls-def true-lit-def* **by** *auto*

**moreover have** *total-over-m I  $\{\chi\}$*

**using** *Posv Neg $\chi$  atm-imp-pos-or-neg-lit tot $\chi$*  **unfolding** *total-over-m-def total-over-set-def*  
**by** *fastforce*

**ultimately have** *partial-interps Leaf I (fst  $\psi$ )*

**and** *sem-tree-size Leaf < sem-tree-size xs*

**and** *resolution $^{**}$   $\psi \psi$*

**unfolding** *xs* **by** (*auto simp add:  $\chi\psi$* )

}

**moreover** {

**assume** *Pos $\chi$ : Pos v  $\notin \# \chi'$*

**then have**  $I\chi: \neg I \models \chi'$  **using**  $\chi'$  *Posv* **unfolding** *true-cls-def true-lit-def* **by** *auto*

**moreover have** *total-over-m I  $\{\chi'\}$*

**using** *Negv Pos $\chi$  atm-imp-pos-or-neg-lit tot $\chi'$*

```

    unfolding total-over-m-def total-over-set-def by fastforce
ultimately have partial-interps Leaf I (fst  $\psi$ )
  and sem-tree-size Leaf < sem-tree-size xs
  and resolution**  $\psi$   $\psi$ 
  using  $\chi'\psi$   $I_\chi$  unfolding xs by auto
}
moreover {
  assume neg: Neg v  $\in\#$   $\chi$  and pos: Pos v  $\in\#$   $\chi'$ 
  have count  $\chi$  (Neg v) = 1
    using simplified-count[OF simp  $\chi\psi$ ] neg
    by (simp add: dual-order.antisym)
  have count  $\chi'$  (Pos v) = 1
    using simplified-count[OF simp  $\chi'\psi$ ] pos
    by (simp add: dual-order.antisym)

  obtain C where  $\chi C$ :  $\chi = \text{add-mset (Neg v) } C$  and negC: Neg v  $\notin\#$  C and posC: Pos v  $\notin\#$ 
C
    by (metis (no-types, lifting) One-nat-def Posv  $\langle \text{count } \chi \text{ (Neg v) } = 1 \rangle$ 
       $\langle \text{count } \chi' \text{ (Pos v) } = 1 \rangle$  count-add-mset count-greater-eq-Suc-zero-iff insert-DiffM
      le-numeral-extra(2) nat.inject pos)

  obtain C' where
     $\chi C'$ :  $\chi' = \text{add-mset (Pos v) } C'$  and
    posC': Pos v  $\notin\#$  C' and
    negC': Neg v  $\notin\#$  C'
    by (metis (no-types, lifting) Negv One-nat-def  $\langle \text{count } \chi' \text{ (Pos v) } = 1 \rangle$  count-add-mset
      count-eq-zero-iff mset-add nat.inject pos)

  have totC: total-over-m I {C}
    using tot $\chi$  tot-over-m-remove[of I Pos v C] negC posC unfolding  $\chi C$  by auto
  have totC': total-over-m I {C'}
    using tot $\chi'$  total-over-m-sum tot-over-m-remove[of I Neg v C'] negC' posC'
    unfolding  $\chi C'$  by auto
  have  $\neg I \models C + C'$ 
    using  $\chi$   $\chi'$   $\chi C$   $\chi C'$  by auto
  then have part-I- $\psi'''$ : partial-interps Leaf I (fst  $\psi \cup \{C + C'\}$ )
    using totC totC'  $\neg I \models C + C'$  by (metis Un-insert-right insertI1
      partial-interps.simps(1) total-over-m-sum)
  {
    assume (add-mset (Pos v) C', add-mset (Neg v) C)  $\notin \text{snd } \psi$ 
    then have inf'': inference  $\psi$  (fst  $\psi \cup \{C + C'\}$ , snd  $\psi \cup \{(\chi', \chi)\}$ )
      by (metis  $\chi'\psi$   $\chi C$   $\chi C'$   $\chi\psi$  add-mset-add-single inference-clause.resolution
        inference-step prod.collapse union-commute)
    obtain N' where full: full simplify (fst  $\psi \cup \{C + C'\}$ ) N'
      by (metis finite-simplified-full-simp fst-conv inf'' inference-preserves-finite
        local.finite)
    have resolution  $\psi$  (N', snd  $\psi \cup \{(\chi', \chi)\}$ )
      using resolution.intros(2)[OF - simp full, of snd  $\psi$  snd  $\psi \cup \{(\chi', \chi)\}$ ] inf''
      by (metis surjective-pairing)
    moreover have partial-interps Leaf I N'
      using full-simplify-preserve-partial-tree[OF full part-I- $\psi'''$ ] .
    moreover have sem-tree-size Leaf < sem-tree-size xs unfolding xs by auto
    ultimately have ?case
      by (metis (no-types) prod.sel(1) rtranclp.rtrancl-into-rtrancl rtranclp.rtrancl-refl)
  }
}
moreover {

```

```

assume  $a$ : ( $\{\#Pos\ v\# \} + C', \{\#Neg\ v\# \} + C) \in \text{snd } \psi$ 
then have ( $\exists \chi \in \text{fst } \psi. (\forall I. \text{total-over-m } I \{C+C'\} \longrightarrow \text{total-over-m } I \{\chi\})$ 
   $\wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C) \vee \text{tautology } (C' + C)$ )
proof –
  obtain  $p$  where  $p$ :  $Pos\ p \in \# (\{\#Pos\ v\# \} + C') \wedge Neg\ p \in \# (\{\#Neg\ v\# \} + C)$ 
     $\wedge ((\exists \chi \in \text{fst } \psi. (\forall I. \text{total-over-m } I \{(\{\#Pos\ v\# \} + C') - \{\#Pos\ p\# \} + ((\{\#Neg\ v\# \} + C) - \{\#Neg\ p\# \})) \longrightarrow \text{total-over-m } I \{\chi\}) \wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models (\{\#Pos\ v\# \} + C') - \{\#Pos\ p\# \} + ((\{\#Neg\ v\# \} + C) - \{\#Neg\ p\# \}))) \vee \text{tautology } ((\{\#Pos\ v\# \} + C') - \{\#Pos\ p\# \} + ((\{\#Neg\ v\# \} + C) - \{\#Neg\ p\# \})))$ 
    using  $a$  by ( $\text{blast intro: allE[OF a-u-i[unfolding subsumes-def Ball-def],$ 
       $\text{of } (\{\#Pos\ v\# \} + C', \{\#Neg\ v\# \} + C)]$ )
    { assume  $p \neq v$ 
      then have  $Pos\ p \in \# C' \wedge Neg\ p \in \# C$  using  $p$  by force
      then have ?thesis by auto
    }
    moreover {
      assume  $p = v$ 
      then have ?thesis using  $p$  by (metis add.commute add-diff-cancel-left)
    }
    ultimately show ?thesis by auto
  qed
moreover {
  assume  $\exists \chi \in \text{fst } \psi. (\forall I. \text{total-over-m } I \{C+C'\} \longrightarrow \text{total-over-m } I \{\chi\})$ 
     $\wedge (\forall I. \text{total-over-m } I \{\chi\} \longrightarrow I \models \chi \longrightarrow I \models C' + C)$ 
  then obtain  $\vartheta$  where
     $\vartheta$ :  $\vartheta \in \text{fst } \psi$  and
     $\text{tot-}\vartheta$ - $CC'$ :  $\forall I. \text{total-over-m } I \{C+C'\} \longrightarrow \text{total-over-m } I \{\vartheta\}$  and
     $\vartheta$ -inv:  $\forall I. \text{total-over-m } I \{\vartheta\} \longrightarrow I \models \vartheta \longrightarrow I \models C' + C$  by blast
  have partial-interps Leaf  $I$  ( $\text{fst } \psi$ )
    using  $\text{tot-}\vartheta$ - $CC'$   $\vartheta$   $\vartheta$ -inv  $\text{tot}C$   $\text{tot}C'$   $\langle \neg I \models C + C' \rangle$  total-over-m-sum by fastforce
  moreover have sem-tree-size Leaf  $<$  sem-tree-size  $xs$  unfolding  $xs$  by auto
  ultimately have ?case by blast
}
moreover {
  assume  $\text{taut}CC'$ : tautology  $(C' + C)$ 
  have total-over-m  $I \{C'+C\}$  using  $\text{tot}C$   $\text{tot}C'$  total-over-m-sum by auto
  then have  $\neg \text{tautology } (C' + C)$ 
    using  $\langle \neg I \models C + C' \rangle$  unfolding add.commute[of  $C\ C'$ ] total-over-m-def
    unfolding tautology-def by auto
  then have False using  $\text{taut}CC'$  unfolding tautology-def by auto
}
ultimately have ?case by auto
}
ultimately have ?case by auto
}
ultimately have ?case using part by (metis (no-types) sem-tree-size.simps(1))
}
moreover {
  assume size-ag: sem-tree-size  $ag > 0$ 
  have sem-tree-size  $ag <$  sem-tree-size  $xs$  unfolding  $xs$  by auto
  moreover have partial-interps  $ag$   $(I \cup \{Pos\ v\})$  ( $\text{fst } \psi$ )
  and partad: partial-interps  $ad$   $(I \cup \{Neg\ v\})$  ( $\text{fst } \psi$ )
    using part partial-interps.simps(2) unfolding  $xs$  by metis+
  moreover
    have sem-tree-size  $ag <$  sem-tree-size  $xs \implies \text{finite } (\text{fst } \psi) \implies \text{already-used-inv } \psi$ 
       $\implies \text{partial-interps } ag$   $(I \cup \{Pos\ v\})$  ( $\text{fst } \psi$ )  $\implies \text{simplified } (\text{fst } \psi)$ 

```

```

     $\implies \exists \text{tree}' \psi'. \text{resolution}^{**} \psi \psi' \wedge \text{partial-interps tree}' (I \cup \{\text{Pos } v\}) (\text{fst } \psi')$ 
     $\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size ag} \vee \text{sem-tree-size ag} = 0)$ 
    using  $\text{IH}[\text{of ag } I \cup \{\text{Pos } v\}]$  by auto
  ultimately obtain  $\psi' :: 'v \text{ state and tree}' :: 'v \text{ sem-tree where}$ 
     $\text{inf: resolution}^{**} \psi \psi'$ 
    and  $\text{part: partial-interps tree}' (I \cup \{\text{Pos } v\}) (\text{fst } \psi')$ 
    and  $\text{size: sem-tree-size tree}' < \text{sem-tree-size ag} \vee \text{sem-tree-size ag} = 0$ 
    using  $\text{finite part rtranclp.rtrancl-refl a-u-i simp}$  by blast

  have  $\text{partial-interps ad } (I \cup \{\text{Neg } v\}) (\text{fst } \psi')$ 
    using  $\text{rtranclp-resolution-preserve-partial-tree inf partad}$  by fast
  then have  $\text{partial-interps (Node v tree' ad) } I (\text{fst } \psi')$  using  $\text{part}$  by auto
  then have  $?case$  using  $\text{inf size size-ag part unfolding xs}$  by fastforce
}
moreover {
  assume  $\text{size-ad: sem-tree-size ad} > 0$ 
  have  $\text{sem-tree-size ad} < \text{sem-tree-size xs}$  unfolding  $\text{xs}$  by auto
  moreover
    have
       $\text{partag: partial-interps ag } (I \cup \{\text{Pos } v\}) (\text{fst } \psi)$  and
       $\text{partial-interps ad } (I \cup \{\text{Neg } v\}) (\text{fst } \psi)$ 
      using  $\text{part partial-interps.simps(2) unfolding xs}$  by metis+
    moreover have  $\text{sem-tree-size ad} < \text{sem-tree-size xs} \longrightarrow \text{finite (fst } \psi) \longrightarrow \text{already-used-inv } \psi$ 
       $\longrightarrow (\text{partial-interps ad } (I \cup \{\text{Neg } v\}) (\text{fst } \psi) \longrightarrow \text{simplified (fst } \psi))$ 
       $\longrightarrow (\exists \text{tree}' \psi'. \text{resolution}^{**} \psi \psi' \wedge \text{partial-interps tree}' (I \cup \{\text{Neg } v\}) (\text{fst } \psi')$ 
         $\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size ad} \vee \text{sem-tree-size ad} = 0))$ 
      using  $\text{IH}$  by blast
    ultimately obtain  $\psi' :: 'v \text{ state and tree}' :: 'v \text{ sem-tree where}$ 
       $\text{inf: resolution}^{**} \psi \psi'$ 
      and  $\text{part: partial-interps tree}' (I \cup \{\text{Neg } v\}) (\text{fst } \psi')$ 
      and  $\text{size: sem-tree-size tree}' < \text{sem-tree-size ad} \vee \text{sem-tree-size ad} = 0$ 
      using  $\text{finite part rtranclp.rtrancl-refl a-u-i simp}$  by blast

    have  $\text{partial-interps ag } (I \cup \{\text{Pos } v\}) (\text{fst } \psi')$ 
      using  $\text{rtranclp-resolution-preserve-partial-tree inf partag}$  by fast
    then have  $\text{partial-interps (Node v ag tree') } I (\text{fst } \psi')$  using  $\text{part}$  by auto
    then have  $?case$  using  $\text{inf size size-ad unfolding xs}$  by fastforce
  }
  ultimately have  $?case$  by auto
}
ultimately show  $?case$  by auto
qed

```

**lemma** *resolution-completeness-inv:*

**fixes**  $\psi :: 'v :: \text{linorder state}$

**assumes**

$\text{unsat: } \neg \text{satisfiable (fst } \psi)$  **and**

$\text{finite: finite (fst } \psi)$  **and**

$\text{a-u-v: already-used-inv } \psi$

**shows**  $\exists \psi'. (\text{resolution}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$

**proof** –

**obtain**  $\text{tree where partial-interps tree } \{\}$   $(\text{fst } \psi)$

using  $\text{partial-interps-build-sem-tree-atms assms}$  by metis

**then show**  $?thesis$

using  $\text{unsat finite a-u-v}$

**proof** (*induct tree arbitrary:  $\psi$  rule: sem-tree-size*)

```

case (bigger tree  $\psi$ ) note  $H = \text{this}$ 
{
  fix  $\chi$ 
  assume  $\text{tree}: \text{tree} = \text{Leaf}$ 
  obtain  $\chi$  where  $\chi: \neg \{\} \models \chi$  and  $\text{tot}\chi: \text{total-over-m } \{\} \{\chi\}$  and  $\chi\psi: \chi \in \text{fst } \psi$ 
    using  $H$  unfolding  $\text{tree}$  by  $\text{auto}$ 
  moreover have  $\{\#\} = \chi$ 
    using  $H$   $\text{atms-empty-iff-empty tot}\chi$ 
    unfolding  $\text{true-cls-def total-over-m-def total-over-set-def}$  by  $\text{fastforce}$ 
  moreover have  $\text{resolution}^{**} \psi \psi$  by  $\text{auto}$ 
  ultimately have  $?case$  by  $\text{metis}$ 
}
moreover {
  fix  $v \text{ tree1 tree2}$ 
  assume  $\text{tree}: \text{tree} = \text{Node } v \text{ tree1 tree2}$ 
  obtain  $\psi_0$  where  $\psi_0: \text{resolution}^{**} \psi \psi_0$  and  $\text{simp}: \text{simplified } (\text{fst } \psi_0)$ 
  proof -
    { assume  $\text{simplified } (\text{fst } \psi)$ 
      moreover have  $\text{resolution}^{**} \psi \psi$  by  $\text{auto}$ 
      ultimately have  $\text{thesis}$  using  $\text{that}$  by  $\text{blast}$ 
    }
    moreover {
      assume  $\neg \text{simplified } (\text{fst } \psi)$ 
      then have  $\exists \psi'. \text{full1 simplify } (\text{fst } \psi) \psi'$ 
        by ( $\text{metis Nitpick.rtranclp-unfold bigger.prem}(3) \text{full1-def}$ 
           $\text{rtranclp-simplify-terminates}$ )
      then obtain  $N$  where  $\text{full1 simplify } (\text{fst } \psi) N$  by  $\text{metis}$ 
      then have  $\text{resolution } \psi (N, \text{snd } \psi)$ 
        using  $\text{resolution.intros}(1)[\text{of } \text{fst } \psi N \text{snd } \psi]$  by  $\text{auto}$ 
      moreover have  $\text{simplified } N$ 
        using  $\langle \text{full1 simplify } (\text{fst } \psi) N \rangle$  unfolding  $\text{full1-def}$  by  $\text{blast}$ 
      ultimately have  $?thesis$  using  $\text{that}$  by  $\text{force}$ 
    }
    ultimately show  $?thesis$  by  $\text{auto}$ 
  }
qed

have  $p: \text{partial-interps tree } \{\} (\text{fst } \psi_0)$ 
and  $\text{uns}: \text{unsatisfiable } (\text{fst } \psi_0)$ 
and  $f: \text{finite } (\text{fst } \psi_0)$ 
and  $a\text{-}u\text{-}v: \text{already-used-inv } \psi_0$ 
  using  $\psi_0 \text{ bigger.prem}(1) \text{rtranclp-resolution-preserve-partial-tree}$  apply  $\text{blast}$ 
  using  $\psi_0 \text{ bigger.prem}(2) \text{rtranclp-resolution-preserves-unsat}$  apply  $\text{blast}$ 
  using  $\psi_0 \text{ bigger.prem}(3) \text{rtranclp-resolution-finite}$  apply  $\text{blast}$ 
  using  $\text{rtranclp-resolution-already-used-inv}[OF \psi_0 \text{ bigger.prem}(4)]$  by  $\text{blast}$ 
obtain  $\text{tree}' \psi'$  where
   $\text{inf}: \text{resolution}^{**} \psi_0 \psi'$  and
   $\text{part}': \text{partial-interps tree}' \{\} (\text{fst } \psi')$  and
   $\text{decrease}: \text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0$ 
  using  $\text{can-decrease-tree-size-resolution}[OF f a\text{-}u\text{-}v p \text{simp}]$  unfolding  $\text{tautology-def}$ 
  by  $\text{meson}$ 
have  $s: \text{sem-tree-size tree}' < \text{sem-tree-size tree}$  using  $\text{decrease}$  unfolding  $\text{tree}$  by  $\text{auto}$ 
have  $\text{fin}: \text{finite } (\text{fst } \psi')$ 
  using  $f \text{inf rtranclp-resolution-finite}$  by  $\text{blast}$ 
have  $\text{unsat}: \text{unsatisfiable } (\text{fst } \psi')$ 
  using  $\text{rtranclp-resolution-preserves-unsat inf uns}$  by  $\text{metis}$ 

```

```

    have a-u-i': already-used-inv  $\psi'$ 
      using a-u-v inf rtrancpl-resolution-already-used-inv[of  $\psi_0 \psi'$ ] by auto
    have ?case
      using inf rtrancpl-trans[of resolution]  $H(1)[OF s \text{ part}' \text{ unsat fin a-u-i}'] \psi_0$  by blast
  }
  ultimately show ?case by (cases tree, auto)
qed
qed

```

```

lemma resolution-preserves-already-used-inv:
  assumes resolution S S'
  and already-used-inv S
  shows already-used-inv S'
  using assms
  apply (induct rule: resolution.induct)
  apply (rule full1-simplify-already-used-inv; simp)
  apply (rule full-simplify-already-used-inv, simp)
  apply (rule inference-preserves-already-used-inv, simp)
  apply blast
done

```

```

lemma rtrancpl-resolution-preserves-already-used-inv:
  assumes resolution** S S'
  and already-used-inv S
  shows already-used-inv S'
  using assms
  apply (induct rule: rtrancpl-induct)
  apply simp
  using resolution-preserves-already-used-inv by fast

```

```

lemma resolution-completeness:
  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes unsat:  $\neg \text{satisfiable (fst } \psi)$ 
  and finite: finite (fst  $\psi$ )
  and snd  $\psi = \{\}$ 
  shows  $\exists \psi'. (\text{resolution** } \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$ 
proof -
  have already-used-inv  $\psi$  unfolding assms by auto
  then show ?thesis using assms resolution-completeness-inv by blast
qed

```

```

lemma rtrancpl-preserves-sat:
  assumes simplify** S S'
  and satisfiable S
  shows satisfiable S'
  using assms apply induction
  apply simp
  by (meson satisfiable-carac satisfiable-def simplify-preserve-models-eq)

```

```

lemma resolution-preserves-sat:
  assumes resolution S S'
  and satisfiable (fst S)
  shows satisfiable (fst S')
  using assms apply (induction rule: resolution.induct)
  using rtrancpl-preserves-sat trancpl-into-rtrancpl unfolding full1-def apply fastforce
  by (metis fst-conv full-def inference-preserve-models rtrancpl-preserves-sat)

```

*satisfiable-carac' satisfiable-def)*

**lemma** *rtrancpl-resolution-preserves-sat:*

**assumes** *resolution\*\* S S'*  
**and** *satisfiable (fst S)*  
**shows** *satisfiable (fst S')*  
**using** *assms apply (induction rule: rtrancpl-induct)*  
**apply** *simp*  
**using** *resolution-preserves-sat by blast*

**lemma** *resolution-soundness:*

**fixes**  $\psi :: 'v :: \text{linorder state}$   
**assumes** *resolution\*\*  $\psi \psi'$  and  $\{\#\} \in \text{fst } \psi'$*   
**shows** *unsatisfiable (fst  $\psi$ )*  
**using** *assms by (meson rtrancpl-resolution-preserves-sat satisfiable-def true-cls-empty true-clss-def)*

**lemma** *resolution-soundness-and-completeness:*

**fixes**  $\psi :: 'v :: \text{linorder state}$   
**assumes** *finite: finite (fst  $\psi$ )*  
**and** *snd: snd  $\psi = \{\}$*   
**shows**  $(\exists \psi'. (\text{resolution** } \psi \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \text{unsatisfiable (fst } \psi)$   
**using** *assms resolution-completeness resolution-soundness by metis*

**lemma** *simplified-falsity:*

**assumes** *simp: simplified  $\psi$*   
**and**  $\{\#\} \in \psi$   
**shows**  $\psi = \{\{\#\}\}$   
**proof** (rule *ccontr*)  
**assume**  $H: \neg ?thesis$   
**then obtain**  $\chi$  **where**  $\chi \in \psi$  **and**  $\chi \neq \{\#\}$  **using** *assms(2) by blast*  
**then have**  $\{\#\} \subsetneq \chi$  **by** (*simp add: subset-mset.zero-less-iff-neq-zero*)  
**then have** *simplify  $\psi (\psi - \{\chi\})$*   
**using** *simplify.subsumption[OF assms(2)  $\langle\{\#\} \subsetneq \chi\rangle \langle\chi \in \psi\rangle$  by blast*  
**then show** *False using simp by blast*  
**qed**

**lemma** *simplify-falsity-in-preserved:*

**assumes** *simplify  $\chi s \chi s'$*   
**and**  $\{\#\} \in \chi s$   
**shows**  $\{\#\} \in \chi s'$   
**using** *assms*  
**by** *induction auto*

**lemma** *rtrancpl-simplify-falsity-in-preserved:*

**assumes** *simplify\*\*  $\chi s \chi s'$*   
**and**  $\{\#\} \in \chi s$   
**shows**  $\{\#\} \in \chi s'$   
**using** *assms*  
**by** *induction (auto intro: simplify-falsity-in-preserved)*

**lemma** *resolution-falsity-get-falsity-alone:*

**assumes** *finite (fst  $\psi$ )*  
**shows**  $(\exists \psi'. (\text{resolution** } \psi \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow (\exists a-u-v. \text{resolution** } \psi (\{\{\#\}\}, a-u-v))$   
**(is**  $?A \longleftrightarrow ?B$ **)**



```

proof
  assume ?B
  then show ?A by auto
next
  assume ?A
  then obtain  $\chi s$   $a-u-v$  where  $\chi s$ : resolution**  $\psi$  ( $\chi s$ ,  $a-u-v$ ) and  $F$ :  $\{\#\} \in \chi s$  by auto
  { assume simplified  $\chi s$ 
    then have ?B using simplified-falsity[OF - F]  $\chi s$  by blast
  }
  moreover {
    assume  $\neg$  simplified  $\chi s$ 
    then obtain  $\chi s'$  where full1 simplify  $\chi s$   $\chi s'$ 
      by (metis  $\chi s$  assms finite-simplified-full1-simp fst-conv rtranclp-resolution-finite)
    then have  $\{\#\} \in \chi s'$ 
      unfolding full1-def by (meson F rtranclp-simplify-falsity-in-preserved
        trancplp-into-rtranclp)
    then have ?B
      by (metis  $\chi s$  (full1 simplify  $\chi s$   $\chi s'$ ) fst-conv full1-simp resolution-always-simplified
        rtranclp.rtrancl-into-rtrancl simplified-falsity)
  }
  ultimately show ?B by blast
qed

theorem resolution-soundness-and-completeness':
  fixes  $\psi :: 'v :: \text{linorder state}$ 
  assumes
    finite: finite (fst  $\psi$ ) and
    snd: snd  $\psi = \{\}$ 
  shows  $(\exists a-u-v. (\text{resolution}^{**} \psi (\{\{\#\}\}, a-u-v))) \longleftrightarrow \text{unsatisfiable (fst } \psi)$ 
  using assms resolution-completeness resolution-soundness resolution-falsity-get-falsity-alone
  by metis

```

```

end
theory Prop-Superposition
imports Entailment-Definition.Partial-Herbrand-Interpretation Ordered-Resolution-Prover.Herbrand-Interpretation
begin

```

## 2.2 Superposition

```

no-notation Herbrand-Interpretation.true-cls (infix  $\models$  50)
notation Herbrand-Interpretation.true-cls (infix  $\models_h$  50)

no-notation Herbrand-Interpretation.true-clss (infix  $\models_s$  50)
notation Herbrand-Interpretation.true-clss (infix  $\models_{hs}$  50)

lemma herbrand-interp-iff-partial-interp-cls:
   $S \models_h C \longleftrightarrow \{Pos\ P|P. P \in S\} \cup \{Neg\ P|P. P \notin S\} \models C$ 
  unfolding Herbrand-Interpretation.true-cls-def Partial-Herbrand-Interpretation.true-cls-def
  by auto

lemma herbrand-consistent-interp:
  consistent-interp ( $\{Pos\ P|P. P \in S\} \cup \{Neg\ P|P. P \notin S\}$ )
  unfolding consistent-interp-def by auto

lemma herbrand-total-over-set:

```

*total-over-set* ( $\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$ ) *T*  
**unfolding** *total-over-set-def* **by** *auto*

**lemma** *herbrand-total-over-m*:

*total-over-m* ( $\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$ ) *T*  
**unfolding** *total-over-m-def* **by** (*auto simp add: herbrand-total-over-set*)

**lemma** *herbrand-interp-iff-partial-interp-clss*:

$S \models_{hs} C \longleftrightarrow \{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\} \models_s C$   
**unfolding** *true-clss-def Ball-def herbrand-interp-iff-partial-interp-clss*  
*Partial-Herbrand-Interpretation.true-clss-def* **by** *auto*

**definition** *clss-lt* :: '*a*::wellorder clause-set  $\Rightarrow$  '*a* clause  $\Rightarrow$  '*a* clause-set **where**  
*clss-lt* *N C* =  $\{D \in N. D < C\}$

**notation** (*latex output*)

*clss-lt* ( $-\langle \hat{b}sup \rangle - \langle \hat{e}sup \rangle$ )

**locale** *selection* =

**fixes** *S* :: '*a* clause  $\Rightarrow$  '*a* clause  
**assumes**  
*S-selects-subseteq*:  $\bigwedge C. S C \leq \# C$  **and**  
*S-selects-neg-lits*:  $\bigwedge C L. L \in \# S C \implies \text{is-neg } L$

**locale** *ground-resolution-with-selection* =

*selection S* **for** *S* :: ('*a* :: wellorder) clause  $\Rightarrow$  '*a* clause

**begin**

**context**

**fixes** *N* :: '*a* clause set

**begin**

We do not create an equivalent of  $\delta$ , but we directly defined  $N_C$  by inlining the definition.

**function**

*production* :: '*a* clause  $\Rightarrow$  '*a* interp

**where**

*production C* =  
 $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max-mset } C = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1$   
 $\wedge \neg (\bigcup D \in \{D. D < C\}. \text{production } D) \models_h C \wedge S C = \{\#\}\}$

**by** *auto*

**termination by** (*relation*  $\{(D, C). D < C\}$ ) (*auto simp: wf-less-multiset*)

**declare** *production.simps*[*simp del*]

**definition** *interp* :: '*a* clause  $\Rightarrow$  '*a* interp **where**

*interp C* =  $(\bigcup D \in \{D. D < C\}. \text{production } D)$

**lemma** *production-unfold*:

*production C* =  $\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max-mset } C = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1 \wedge \neg \text{interp } C \models_h C \wedge S C = \{\#\}\}$

**unfolding** *interp-def* **by** (*rule production.simps*)

**abbreviation** *productive A*  $\equiv$  (*production A*  $\neq \{\}$ )

**abbreviation** *produces* :: '*a* clause  $\Rightarrow$  '*a*  $\Rightarrow$  bool **where**

*produces C A*  $\equiv$  *production C* =  $\{A\}$

**lemma** *producesD*:

*produces C A*  $\implies C \in N \wedge C \neq \{\#\} \wedge \text{Pos } A = \text{Max-mset } C \wedge \text{count } C (\text{Pos } A) \leq 1 \wedge$   
 $\neg \text{interp } C \models_h C \wedge S \ C = \{\#\}$

**unfolding** *production-unfold* **by** *auto*

**lemma** *produces C A*  $\implies \text{Pos } A \in \# \ C$

**by** (*simp add: Max-in-lits producesD*)

**lemma** *interp'-def-in-set*:

*interp C* =  $(\bigcup D \in \{D \in N. D < C\}. \text{production } D)$

**unfolding** *interp-def* **apply** *auto*

**unfolding** *production-unfold* **apply** *auto*

**done**

**lemma** *production-iff-produces*:

*produces D A*  $\longleftrightarrow A \in \text{production } D$

**unfolding** *production-unfold* **by** *auto*

**definition** *Interp* :: 'a clause  $\Rightarrow$  'a *interp* **where**

*Interp C* = *interp C*  $\cup$  *production C*

**lemma**

**assumes** *produces C P*

**shows** *Interp C*  $\models_h C$

**unfolding** *Interp-def* *assms* **using** *producesD[OF assms]*

**by** (*metis Max-in-lits Un-insert-right insertI1 pos-literal-in-imp-true-cls*)

**definition** *INTERP* :: 'a *interp* **where**

*INTERP* =  $(\bigcup D \in N. \text{production } D)$

**lemma** *interp-subseteq-Interp[simp]*: *interp C*  $\subseteq$  *Interp C*

**unfolding** *Interp-def* **by** *simp*

**lemma** *Interp-as-UNION*: *Interp C* =  $(\bigcup D \in \{D. D \leq C\}. \text{production } D)$

**unfolding** *Interp-def* *interp-def* *less-eq-multiset-def* **by** *fast*

**lemma** *productive-not-empty*: *productive C*  $\implies C \neq \{\#\}$

**unfolding** *production-unfold* **by** *auto*

**lemma** *productive-imp-produces-Max-literal*: *productive C*  $\implies \text{produces } C (\text{atm-of } (\text{Max-mset } C))$

**unfolding** *production-unfold* **by** (*auto simp del: atm-of-Max-lit*)

**lemma** *productive-imp-produces-Max-atom*: *productive C*  $\implies \text{produces } C (\text{Max } (\text{atms-of } C))$

**unfolding** *atms-of-def* *Max-atm-of-set-mset-commute[OF productive-not-empty]*

**by** (*rule productive-imp-produces-Max-literal*)

**lemma** *produces-imp-Max-literal*: *produces C A*  $\implies A = \text{atm-of } (\text{Max-mset } C)$

**by** (*metis Max-singleton insert-not-empty productive-imp-produces-Max-literal*)

**lemma** *produces-imp-Max-atom*: *produces C A*  $\implies A = \text{Max } (\text{atms-of } C)$

**by** (*metis Max-singleton insert-not-empty productive-imp-produces-Max-atom*)

**lemma** *produces-imp-Pos-in-lits*: *produces C A*  $\implies \text{Pos } A \in \# \ C$

**by** (*auto intro: Max-in-lits dest!: producesD*)

**lemma** *productive-in-N*:  $\text{productive } C \implies C \in N$   
**unfolding** *production-unfold* **by** *auto*

**lemma** *produces-imp-atms-leq*:  $\text{produces } C \ A \implies B \in \text{atms-of } C \implies B \leq A$   
**by** (*metis Max-ge finite-atms-of insert-not-empty productive-imp-produces-Max-atom singleton-inject*)

**lemma** *produces-imp-neg-notin-lits*:  $\text{produces } C \ A \implies \text{Neg } A \notin \# \ C$   
**by** (*rule pos-Max-imp-neg-notin*) (*auto dest: producesD*)

**lemma** *less-eq-imp-interp-subseteq-interp*:  $C \leq D \implies \text{interp } C \subseteq \text{interp } D$   
**unfolding** *interp-def* **by** *auto* (*metis order.strict-trans2*)

**lemma** *less-eq-imp-interp-subseteq-Interp*:  $C \leq D \implies \text{interp } C \subseteq \text{Interp } D$   
**unfolding** *Interp-def* **using** *less-eq-imp-interp-subseteq-interp* **by** *blast*

**lemma** *less-imp-production-subseteq-interp*:  $C < D \implies \text{production } C \subseteq \text{interp } D$   
**unfolding** *interp-def* **by** *fast*

**lemma** *less-eq-imp-production-subseteq-Interp*:  $C \leq D \implies \text{production } C \subseteq \text{Interp } D$   
**unfolding** *Interp-def* **using** *less-imp-production-subseteq-interp*  
**by** (*metis le-imp-less-or-eq le-supI1 sup-ge2*)

**lemma** *less-imp-Interp-subseteq-interp*:  $C < D \implies \text{Interp } C \subseteq \text{interp } D$   
**unfolding** *Interp-def*  
**by** (*auto simp: less-eq-imp-interp-subseteq-interp less-imp-production-subseteq-interp*)

**lemma** *less-eq-imp-Interp-subseteq-Interp*:  $C \leq D \implies \text{Interp } C \subseteq \text{Interp } D$   
**using** *less-imp-Interp-subseteq-interp*  
**unfolding** *Interp-def* **by** (*metis le-imp-less-or-eq le-supI2 subset-refl sup-commute*)

**lemma** *false-Interp-to-true-interp-imp-less-multiset*:  $A \notin \text{Interp } C \implies A \in \text{interp } D \implies C < D$   
**using** *less-eq-imp-interp-subseteq-Interp not-less* **by** *blast*

**lemma** *false-interp-to-true-interp-imp-less-multiset*:  $A \notin \text{interp } C \implies A \in \text{interp } D \implies C < D$   
**using** *less-eq-imp-interp-subseteq-interp not-less* **by** *blast*

**lemma** *false-Interp-to-true-Interp-imp-less-multiset*:  $A \notin \text{Interp } C \implies A \in \text{Interp } D \implies C < D$   
**using** *less-eq-imp-Interp-subseteq-Interp not-less* **by** *blast*

**lemma** *false-interp-to-true-Interp-imp-le-multiset*:  $A \notin \text{interp } C \implies A \in \text{Interp } D \implies C \leq D$   
**using** *less-imp-Interp-subseteq-interp not-less* **by** *blast*

**lemma** *interp-subseteq-INTERP*:  $\text{interp } C \subseteq \text{INTERP}$   
**unfolding** *interp-def INTERP-def* **by** (*auto simp: production-unfold*)

**lemma** *production-subseteq-INTERP*:  $\text{production } C \subseteq \text{INTERP}$   
**unfolding** *INTERP-def* **using** *production-unfold* **by** *blast*

**lemma** *Interp-subseteq-INTERP*:  $\text{Interp } C \subseteq \text{INTERP}$   
**unfolding** *Interp-def* **by** (*auto intro!: interp-subseteq-INTERP production-subseteq-INTERP*)

This lemma corresponds to theorem 2.7.7 page 77 of Weidenbach's book.

**lemma** *produces-imp-in-interp*:  
**assumes** *a-in-c*:  $\text{Neg } A \in \# \ C$  **and** *d*:  $\text{produces } D \ A$

shows  $A \in \text{interp } C$   
**proof** –  
 from  $d$  have  $\text{Max-mset } D = \text{Pos } A$   
 using *production-unfold* **by** *blast*  
 then have  $D < \{\# \text{Neg } A \# \}$   
 by (*meson Max-pos-neg-less-multiset multi-member-last*)  
 moreover have  $\{\# \text{Neg } A \# \} \leq C$   
 by (*rule subset-eq-imp-le-multiset*) (*rule mset-subset-eq-single[OF a-in-c]*)  
 ultimately show *?thesis*  
 using  $d$  **by** (*blast dest: less-eq-imp-interp-subseteq-interp less-imp-production-subseteq-interp*)  
**qed**

**lemma** *neg-notin-Interp-not-produce*:  $\text{Neg } A \in \# C \implies A \notin \text{Interp } D \implies C \leq D \implies \neg \text{produces } D'' A$   
**by** (*auto dest: produces-imp-in-interp less-eq-imp-interp-subseteq-Interp*)

**lemma** *in-production-imp-produces*:  $A \in \text{production } C \implies \text{produces } C A$   
**by** (*metis insert-absorb productive-imp-produces-Max-atom singleton-insert-inj-eq'*)

**lemma** *not-produces-imp-notin-production*:  $\neg \text{produces } C A \implies A \notin \text{production } C$   
**by** (*metis in-production-imp-produces*)

**lemma** *not-produces-imp-notin-interp*:  $(\bigwedge D. \neg \text{produces } D A) \implies A \notin \text{interp } C$   
**unfolding** *interp-def* **by** (*fast intro!: in-production-imp-produces*)

The results below corresponds to Lemma 3.4.

**Nitpicking 0.1.** *If  $D = D'$  and  $D$  is productive,  $I^D \subseteq I_{D'}$  does not hold.*

**lemma** *true-Interp-imp-general*:  
**assumes**  
 $c\text{-le-}d$ :  $C \leq D$  **and**  
 $d\text{-lt-}d'$ :  $D < D'$  **and**  
 $c\text{-at-}d$ :  $\text{Interp } D \models_h C$  **and**  
 $\text{subs: } \text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$   
**shows**  $(\bigcup C \in CC. \text{production } C) \models_h C$   
**proof** (*cases  $\exists A. \text{Pos } A \in \# C \wedge A \in \text{Interp } D$* )  
**case** *True*  
 then obtain  $A$  where  $a\text{-in-}c$ :  $\text{Pos } A \in \# C$  **and**  $a\text{-at-}d$ :  $A \in \text{Interp } D$   
 by *blast*  
 from  $a\text{-at-}d$  have  $A \in \text{interp } D'$   
 using  $d\text{-lt-}d'$  *less-imp-Interp-subseteq-interp* **by** *blast*  
 then show *?thesis*  
 using  $\text{subs } a\text{-in-}c$  **by** (*blast dest: contra-subsetD*)  
**next**  
**case** *False*  
 then obtain  $A$  where  $a\text{-in-}c$ :  $\text{Neg } A \in \# C$  **and**  $A \notin \text{Interp } D$   
 using  $c\text{-at-}d$  **unfolding** *true-cls-def* **by** *blast*  
 then have  $\bigwedge D''. \neg \text{produces } D'' A$   
 using  $c\text{-le-}d$  *neg-notin-Interp-not-produce* **by** *simp*  
 then show *?thesis*  
 using  $a\text{-in-}c$   $\text{subs}$  *not-produces-imp-notin-production* **by** *auto*  
**qed**

**lemma** *true-Interp-imp-interp*:  $C \leq D \implies D < D' \implies \text{Interp } D \models_h C \implies \text{interp } D' \models_h C$

```

using interp-def true-Interp-imp-general by simp

lemma true-Interp-imp-Interp:  $C \leq D \implies D < D' \implies \text{Interp } D \models_h C \implies \text{Interp } D' \models_h C$ 
using Interp-as-UNION interp-subseteq-Interp true-Interp-imp-general by simp

lemma true-Interp-imp-INTERP:  $C \leq D \implies \text{Interp } D \models_h C \implies \text{INTERP} \models_h C$ 
using INTERP-def interp-subseteq-INTERP
  true-Interp-imp-general[OF - le-multiset-right-total]
by simp

lemma true-interp-imp-general:
  assumes
    c-le-d:  $C \leq D$  and
    d-lt-d':  $D < D'$  and
    c-at-d:  $\text{interp } D \models_h C$  and
    subs:  $\text{interp } D' \subseteq (\bigcup C \in CC. \text{production } C)$ 
  shows  $(\bigcup C \in CC. \text{production } C) \models_h C$ 
proof (cases  $\exists A. \text{Pos } A \in \# C \wedge A \in \text{interp } D$ )
case True
  then obtain A where a-in-c:  $\text{Pos } A \in \# C$  and a-at-d:  $A \in \text{interp } D$ 
  by blast
  from a-at-d have  $A \in \text{interp } D'$ 
  using d-lt-d' less-eq-imp-interp-subseteq-interp[OF less-imp-le] by blast
  then show ?thesis
  using subs a-in-c by (blast dest: contra-subsetD)
next
case False
  then obtain A where a-in-c:  $\text{Neg } A \in \# C$  and  $A \notin \text{interp } D$ 
  using c-at-d unfolding true-cls-def by blast
  then have  $\bigwedge D''. \neg \text{produces } D'' A$ 
  using c-le-d by (auto dest: produces-imp-in-interp less-eq-imp-interp-subseteq-interp)
  then show ?thesis
  using a-in-c subs not-produces-imp-notin-production by auto
qed

```

This lemma corresponds to theorem 2.7.7 page 77 of Weidenbach's book. Here the strict maximality is important

```

lemma true-interp-imp-interp:  $C \leq D \implies D < D' \implies \text{interp } D \models_h C \implies \text{interp } D' \models_h C$ 
using interp-def true-interp-imp-general by simp

```

```

lemma true-interp-imp-Interp:  $C \leq D \implies D < D' \implies \text{interp } D \models_h C \implies \text{Interp } D' \models_h C$ 
using Interp-as-UNION interp-subseteq-Interp[of D'] true-interp-imp-general by simp

```

```

lemma true-interp-imp-INTERP:  $C \leq D \implies \text{interp } D \models_h C \implies \text{INTERP} \models_h C$ 
using INTERP-def interp-subseteq-INTERP
  true-interp-imp-general[OF - le-multiset-right-total]
by simp

```

```

lemma productive-imp-false-interp:  $\text{productive } C \implies \neg \text{interp } C \models_h C$ 
unfolding production-unfold by auto

```

This lemma corresponds to theorem 2.7.7 page 77 of Weidenbach's book. Here the strict maximality is important

```

lemma cls-gt-double-pos-no-production:
  assumes D:  $\{\# \text{Pos } P, \text{Pos } P\# \} < C$ 

```

**shows**  $\neg \text{produces } C \ P$   
**proof** –  
 let  $?D = \{\#Pos \ P, \ Pos \ P\#\}$   
 note  $D' = D[\text{unfolded less-multiset}_{HO}]$   
**consider**  
 (P) *count*  $C \ (Pos \ P) \geq 2$   
 | (Q)  $Q$  **where**  $Q > Pos \ P$  **and**  $Q \in \# \ C$   
   **using**  $HOL.spec[OF \ HOL.conjunct2[OF \ D], \ of \ Pos \ P]$  **by** (*auto split: if-split-asm*)  
**then show** *?thesis*  
**proof** *cases*  
   **case**  $Q$   
   **have**  $Q \in \text{set-mset } C$   
     **using**  $Q(2)$  **by** (*auto split: if-split-asm*)  
   **then have**  $Max\text{-mset } C > Pos \ P$   
     **using**  $Q(1) \ Max\text{-gr-iff}$  **by** *blast*  
   **then show** *?thesis*  
     **unfolding** *production-unfold* **by** *auto*  
 next  
   **case**  $P$   
   **then show** *?thesis*  
     **unfolding** *production-unfold* **by** *auto*  
**qed**  
**qed**

This lemma corresponds to theorem 2.7.7 page 77 of Weidenbach's book.

**lemma**  
 assumes  $D: C + \{\#Neg \ P\# \} < D$   
 shows *production*  $D \neq \{P\}$   
**proof** –  
 note  $D' = D[\text{unfolded less-multiset}_{HO}]$   
**consider**  
 (P)  $Neg \ P \in \# \ D$   
 | (Q)  $Q$  **where**  $Q > Neg \ P$  **and** *count*  $D \ Q > \text{count } (C + \{\#Neg \ P\# \}) \ Q$   
   **using**  $HOL.spec[OF \ HOL.conjunct2[OF \ D], \ of \ Neg \ P]$  *count-greater-zero-iff* **by** *fastforce*  
**then show** *?thesis*  
**proof** *cases*  
   **case**  $Q$   
   **have**  $Q \in \text{set-mset } D$   
     **using**  $Q(2) \ gr\text{-implies-not0}$  **by** *fastforce*  
   **then have**  $Max\text{-mset } D > Neg \ P$   
     **using**  $Q(1) \ Max\text{-gr-iff}$  **by** *blast*  
   **then have**  $Max\text{-mset } D > Pos \ P$   
     **using** *less-trans*[*of*  $Pos \ P \ Neg \ P \ Max\text{-mset } D$ ] **by** *auto*  
   **then show** *?thesis*  
     **unfolding** *production-unfold* **by** *auto*  
 next  
   **case**  $P$   
   **then have**  $Max\text{-mset } D > Pos \ P$   
     **by** (*meson*  $Max\text{-ge finite-set-mset le-less-trans linorder-not-le pos-less-neg$ )  
   **then show** *?thesis*  
     **unfolding** *production-unfold* **by** *auto*  
**qed**  
**qed**

**lemma** *in-interp-is-produced*:  
 assumes  $P \in INTERP$

**shows**  $\exists D. D + \{\#Pos P\} \in N \wedge produces (D + \{\#Pos P\}) P$   
**using** *assms unfolding INTERP-def UN-iff production-iff-produces Ball-def*  
**by** (*metis ground-resolution-with-selection.produces-imp-Pos-in-lits insert-DiffM2*  
*ground-resolution-with-selection-axioms not-produces-imp-notin-production*)

**end**  
**end**

### 2.2.1 We can now define the rules of the calculus

**inductive** *superposition-rules* :: 'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  'a clause  $\Rightarrow$  bool **where**  
*factoring*: *superposition-rules* ( $C + \{\#Pos P\} + \{\#Pos P\}$ )  $B (C + \{\#Pos P\})$  |  
*superposition-l*: *superposition-rules* ( $C_1 + \{\#Pos P\}$ ) ( $C_2 + \{\#Neg P\}$ ) ( $C_1 + C_2$ )

**inductive** *superposition* :: 'a clause-set  $\Rightarrow$  'a clause-set  $\Rightarrow$  bool **where**  
*superposition*:  $A \in N \Longrightarrow B \in N \Longrightarrow superposition-rules A B C$   
 $\Longrightarrow superposition N (N \cup \{C\})$

**definition** *abstract-red* :: 'a::wellorder clause  $\Rightarrow$  'a clause-set  $\Rightarrow$  bool **where**  
*abstract-red*  $C N = (clss-lt N C \models_p C)$

**lemma** *herbrand-true-clss-true-clss-clss-herbrand-true-clss*:

**assumes**  
 $AB: A \models_{hs} B$  **and**  
 $BC: B \models_p C$   
**shows**  $A \models_h C$

**proof** –

**let**  $?I = \{Pos P \mid P. P \in A\} \cup \{Neg P \mid P. P \notin A\}$   
**have**  $B: ?I \models_s B$  **using**  $AB$   
**by** (*auto simp add: herbrand-interp-iff-partial-interp-clss*)

**have**  $IH: \bigwedge I. total-over-set I (atms-of C) \Longrightarrow total-over-m I B \Longrightarrow consistent-interp I$   
 $\Longrightarrow I \models_s B \Longrightarrow I \models C$  **using**  $BC$   
**by** (*auto simp add: true-clss-clss-def*)

**show** *?thesis*

**unfolding** *herbrand-interp-iff-partial-interp-clss*  
**by** (*auto intro: IH[of ?I] simp add: herbrand-total-over-set herbrand-total-over-m*  
*herbrand-consistent-interp B*)

**qed**

**lemma** *abstract-red-subset-mset-abstract-red*:

**assumes**  
 $abstr: abstract-red C N$  **and**  
 $c-lt-d: C \subseteq_{\#} D$   
**shows**  $abstract-red D N$

**proof** –

**have**  $\{D \in N. D < C\} \subseteq \{D' \in N. D' < D\}$   
**using** *subset-eq-imp-le-multiset[OF c-lt-d]*  
**by** (*metis (no-types, lifting) Collect-mono order.strict-trans2*)

**then show** *?thesis*

**using**  $abstr$  **unfolding** *abstract-red-def clss-lt-def*  
**by** (*metis (no-types, lifting) c-lt-d subset-mset.diff-add true-clss-clss-mono-r'*  
*true-clss-clss-subset*)

**qed**



**lemma** *true-clss-clss-extended*:

**assumes**

$A \models_p B$  **and**

*tot*: *total-over-m*  $I$   $A$  **and**

*cons*: *consistent-interp*  $I$  **and**

$I \models_s A$

**shows**  $I \models B$

**proof** –

**let**  $?I = I \cup \{Pos\ P \mid P. P \in \text{atms-of } B \wedge P \notin \text{atms-of-s } I\}$

**have** *consistent-interp*  $?I$

**using** *cons* **unfolding** *consistent-interp-def* *atms-of-s-def* *atms-of-def*

**apply** (*auto* 1 5 *simp* *add: image-iff*)

**by** (*metis* *atm-of-uminus* *literal.sel*(1))

**moreover** **have** *tot-I*: *total-over-m*  $?I$   $(A \cup \{B\})$

**proof** –

**obtain**  $aa :: 'a \text{ set} \Rightarrow 'a \text{ literal set} \Rightarrow 'a$  **where**

$f2: \forall x0\ x1. (\exists v2. v2 \in x0 \wedge Pos\ v2 \notin x1 \wedge Neg\ v2 \notin x1)$

$\longleftrightarrow (aa\ x0\ x1 \in x0 \wedge Pos\ (aa\ x0\ x1) \notin x1 \wedge Neg\ (aa\ x0\ x1) \notin x1)$

**by** *moura*

**have**  $\forall a. a \notin \text{atms-of-ms } A \vee Pos\ a \in I \vee Neg\ a \in I$

**using** *tot* **by** (*simp* *add: total-over-m-def* *total-over-set-def*)

**then** **have**  $aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\})\ (I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})$

$\notin \text{atms-of-ms } A \cup \text{atms-of-ms } \{B\} \vee Pos\ (aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\}))$

$(I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})) \in I$

$\cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\}$

$\vee Neg\ (aa\ (\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\}))$

$(I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})) \in I$

$\cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\}$

**by** *auto*

**then** **have** *total-over-set*  $(I \cup \{Pos\ a \mid a. a \in \text{atms-of } B \wedge a \notin \text{atms-of-s } I\})$

$(\text{atms-of-ms } A \cup \text{atms-of-ms } \{B\})$

**using**  $f2$  **by** (*meson* *total-over-set-def*)

**then** **show** *?thesis*

**by** (*simp* *add: total-over-m-def*)

**qed**

**moreover** **have**  $?I \models_s A$

**using**  $I \models A$  **by** *auto*

**ultimately** **have** 1:  $?I \models B$

**using**  $\langle A \models_p B \rangle$  **unfolding** *true-clss-clss-def* **by** *auto*

**let**  $?I' = I \cup \{Neg\ P \mid P. P \in \text{atms-of } B \wedge P \notin \text{atms-of-s } I\}$

**have** *consistent-interp*  $?I'$

**using** *cons* **unfolding** *consistent-interp-def* *atms-of-s-def* *atms-of-def*

**apply** (*auto* 1 5 *simp* *add: image-iff*)

**by** (*metis* *atm-of-uminus* *literal.sel*(2))

**moreover** **have** *tot*: *total-over-m*  $?I'$   $(A \cup \{B\})$

**by** (*smt* *Un-iff* *in-atms-of-s-decomp* *mem-Collect-eq* *tot* *total-over-m-empty* *total-over-m-insert* *total-over-m-union* *total-over-set-def* *total-union*)

**moreover** **have**  $?I' \models_s A$

**using**  $I \models A$  **by** *auto*

**ultimately** **have** 2:  $?I' \models B$

**using**  $\langle A \models_p B \rangle$  **unfolding** *true-clss-clss-def* **by** *auto*

**define**  $BB$  **where**

$\langle BB = \{P. P \in \text{atms-of } B \wedge P \notin \text{atms-of-s } I\} \rangle$

**have** 1:  $\langle I \cup Pos\ 'BB \models B \rangle$

```

    using 1 unfolding BB-def by (simp add: setcompr-eq-image)
have 2:  $\langle I \cup \text{Neg } 'BB \models B \rangle$ 
    using 2 unfolding BB-def by (simp add: setcompr-eq-image)
have  $\langle \text{finite } BB \rangle$ 
    unfolding BB-def by auto
then show ?thesis
    using 1 2 apply (induction BB)
    subgoal by auto
    subgoal for  $x \ BB$ 
        using remove-literal-in-model-tautology[of  $\langle I \cup \text{Pos } 'BB \rangle$ ]
    apply -
    apply (rule ccontr)
    apply (auto simp: Partial-Herbrand-Interpretation.true-cls-def total-over-set-def total-over-m-def
        atms-of-ms-def)

oops
lemma
    assumes
        CP:  $\neg \text{class-lt } N (\{ \#C \# \} + \{ \#E \# \}) \models_p \{ \#C \# \} + \{ \# \text{Neg } P \# \}$  and
        class-lt  $N (\{ \#C \# \} + \{ \#E \# \}) \models_p \{ \#E \# \} + \{ \# \text{Pos } P \# \} \vee \text{class-lt } N (\{ \#C \# \} + \{ \#E \# \}) \models_p$ 
 $\{ \#C \# \} + \{ \# \text{Neg } P \# \}$ 
    shows  $\text{class-lt } N (\{ \#C \# \} + \{ \#E \# \}) \models_p \{ \#E \# \} + \{ \# \text{Pos } P \# \}$ 

oops

locale ground-ordered-resolution-with-redundancy =
    ground-resolution-with-selection +
    fixes redundant ::  $'a::\text{wellorder clause} \Rightarrow 'a \text{ clause-set} \Rightarrow \text{bool}$ 
    assumes
        redundant-iff-abstract:  $\text{redundant } A \ N \longleftrightarrow \text{abstract-red } A \ N$ 
begin

definition saturated ::  $'a \text{ clause-set} \Rightarrow \text{bool}$  where
    saturated  $N \longleftrightarrow$ 
     $(\forall A \ B \ C. A \in N \longrightarrow B \in N \longrightarrow \neg \text{redundant } A \ N \longrightarrow \neg \text{redundant } B \ N \longrightarrow$ 
     $\text{superposition-rules } A \ B \ C \longrightarrow \text{redundant } C \ N \vee C \in N)$ 
lemma (in  $\neg$ )
    assumes  $\langle A \models_p C + E \rangle$ 
    shows  $\langle A \models_p \text{add-mset } L \ C \vee A \models_p \text{add-mset } (-L) \ E \rangle$ 
proof clarify
    assume  $\langle \neg A \models_p \text{add-mset } (-L) \ E \rangle$ 
    then obtain  $I'$  where
        tot':  $\langle \text{total-over-m } I' (A \cup \{ \text{add-mset } (-L) \ E \}) \rangle$  and
        cons':  $\langle \text{consistent-interp } I' \rangle$  and
        I'-A:  $\langle I' \models_s A \rangle$  and
        I'-uL-E:  $\langle \neg I' \models \text{add-mset } (-L) \ E \rangle$ 
    unfolding true-cls-cls-def by auto
    have  $\langle -L \notin I' \rangle \langle \neg I' \models E \rangle$ 
    using I'-uL-E by auto
    moreover have  $\langle \text{atm-of } L \in \text{atm-of } 'I' \rangle$ 
    using tot' unfolding total-over-m-def total-over-set-def
    by (cases L) force+
    ultimately have  $\langle L \in I' \rangle$ 
    by (auto simp: image-iff atm-of-eq-atm-of)

    show  $\langle A \models_p \text{add-mset } L \ C \rangle$ 

```

```

unfolding true-clss-cls-def
proof (intro allI impI conjI)
  fix I
  assume
    tot:  $\langle \text{total-over-}m\ I\ (A \cup \{\text{add-mset}\ L\ C\}) \rangle$  and
    cons:  $\langle \text{consistent-interp}\ I \rangle$  and
    I-A:  $\langle I \models_s A \rangle$ 
  let  $?I = I \cup \{Pos\ P \mid P. P \in \text{atms-of}\ E \wedge P \notin \text{atms-of-s}\ I\}$ 
  have in-C-pm-I:  $\langle L \in \# C \implies L \in I \vee -L \in I \rangle$  for L
    using tot by (cases L) (force simp: total-over-m-def total-over-set-def atms-of-def)+
  have consistent-interp ?I
    using cons unfolding consistent-interp-def atms-of-s-def atms-of-def
    apply (auto 1 5 simp add: image-iff)
    by (metis atm-of-uminus literal.sel(1))
  moreover {
    have tot-I: total-over-m ?I ( $A \cup \{E\}$ )
      using tot total-over-set-def total-union by force
    then have tot-I: total-over-m ?I ( $A \cup \{C+E\}$ )
      using total-union[OF tot] by auto
  }
  moreover have  $?I \models_s A$ 
    using I-A by auto
  ultimately have  $1: ?I \models C + E$ 
    using assms unfolding true-clss-cls-def by auto

  then show  $\langle I \models \text{add-mset}\ L\ C \rangle$ 
    unfolding Partial-Herbrand-Interpretation.true-cls-def
    apply (auto simp: true-cls-def dest: in-C-pm-I)
    oops

```

**lemma**

```

assumes
  saturated: saturated N and
  finite: finite N and
  empty:  $\{\#\} \notin N$ 
shows  $INTERP\ N \models_{hs}\ N$ 
proof (rule ccontr)
  let  $?N_{\mathcal{I}} = INTERP\ N$ 
  assume  $\neg ?thesis$ 
  then have not-empty:  $\{E \in N. \neg ?N_{\mathcal{I}} \models_h E\} \neq \{\}$ 
    unfolding true-clss-def Ball-def by auto
  define D where  $D = Min\ \{E \in N. \neg ?N_{\mathcal{I}} \models_h E\}$ 
  have [simp]:  $D \in N$ 
    unfolding D-def
    by (metis (mono-tags, lifting) Min-in not-empty finite mem-Collect-eq rev-finite-subset subsetI)
  have not-d-interp:  $\neg ?N_{\mathcal{I}} \models_h D$ 
    unfolding D-def
    by (metis (mono-tags, lifting) Min-in finite mem-Collect-eq not-empty rev-finite-subset subsetI)
  have cls-not-D:  $\bigwedge E. E \in N \implies E \neq D \implies \neg ?N_{\mathcal{I}} \models_h E \implies D \leq E$ 
    using finite D-def by auto
  obtain C L where  $D: D = C + \{\#L\#\}$  and LSD:  $L \in \# S\ D \vee (S\ D = \{\#\} \wedge Max-mset\ D = L)$ 
  proof (cases S D = \{\#\})
    case False
    then obtain L where  $L \in \# S\ D$ 
      using Max-in-lits by blast
    moreover {
      then have  $L \in \# D$ 

```

```

    using S-selects-subseteq[of D] by auto
  then have  $D = (D - \{\#L\}) + \{\#L\}$ 
    by auto }
ultimately show ?thesis using that by blast
next
let ?L = Max-mset D
case True
moreover {
  have ?L  $\in \#$  D
    by (metis (no-types, lifting) Max-in-lits  $\langle D \in N \rangle$  empty)
  then have  $D = (D - \{\#?L\}) + \{\#?L\}$ 
    by auto }
ultimately show ?thesis using that by blast
qed
have red:  $\neg$ redundant D N
proof (rule ccontr)
  assume red[simplified]:  $\sim \sim$ redundant D N
  have  $\forall E < D. E \in N \longrightarrow ?N_{\mathcal{I}} \models_h E$ 
    using cls-not-D unfolding not-le[symmetric] by fastforce
  then have  $?N_{\mathcal{I}} \models_{hs} clss-lt\ N\ D$ 
    unfolding clss-lt-def true-clss-def Ball-def by blast
  then show False
    using red not-d-interp unfolding abstract-red-def redundant-iff-abstract
    using herbrand-true-clss-true-clss-cls-herbrand-true-clss by fast
qed

consider
  (L) P where  $L = Pos\ P$  and  $S\ D = \{\#\}$  and  $Max-mset\ D = Pos\ P$ 
| (Lneg) P where  $L = Neg\ P$ 
  using LSD S-selects-neg-lits[of L D] by (cases L) auto
then show False
proof cases
  case L note  $P = this(1)$  and  $S = this(2)$  and  $max = this(3)$ 
  have count D L > 1
  proof (rule ccontr)
    assume  $\sim ?thesis$ 
    then have count: count D L = 1
      unfolding D by (auto simp: not-in-iff)
    have  $\neg ?N_{\mathcal{I}} \models_h D$ 
      using not-d-interp true-interp-imp-INTERP ground-resolution-with-selection-axioms
      by blast
    then have produces N D P
      using not-empty empty finite  $\langle D \in N \rangle$  count L
      true-interp-imp-INTERP unfolding production-iff-produces unfolding production-unfold
      by (auto simp add: max not-empty)
    then have INTERP N  $\models_h D$ 
      unfolding D
      by (metis pos-literal-in-imp-true-cls produces-imp-Pos-in-lits
        production-subseteq-INTERP singletonI subsetCE)
    then show False
      using not-d-interp by blast
  qed
then have  $Pos\ P \in \#$  C
  by (simp add: P D)
then obtain C' where  $C': D = C' + \{\#Pos\ P\} + \{\#Pos\ P\}$ 
  unfolding D by (metis (full-types) P insert-DiffM2)

```

```

have sup: superposition-rules D D (D - {#L#})
  unfolding C' L by (auto simp add: superposition-rules.simps)
have C' + {#Pos P#} < C' + {#Pos P#} + {#Pos P#}
  by auto
moreover have  $\neg ?N_{\mathcal{I}} \models h (D - \{ \#L\# \})$ 
  using not-d-interp unfolding C' L by auto
ultimately have C' + {#Pos P#}  $\notin$  N
  using C' P cls-not-D by fastforce
have D - {#L#} < D
  unfolding C' L by auto
have c'-p-p: C' + {#Pos P#} + {#Pos P#} - {#Pos P#} = C' + {#Pos P#}
  by auto
have redundant (C' + {#Pos P#}) N
  using saturated red sup  $\langle D \in N \rangle \langle C' + \{ \#Pos P\# \} \notin N \rangle$  unfolding saturated-def C' L c'-p-p
  by blast
moreover have C' + {#Pos P#}  $\subseteq$  C' + {#Pos P#} + {#Pos P#}
  by auto
ultimately show False
  using red unfolding C' redundant-iff-abstract by (blast dest:
    abstract-red-subset-mset-abstract-red)
next
case Lneg note L = this(1)
have P: P  $\in$  ?NI
  using not-d-interp unfolding D true-cls-def L by (auto split: if-split-asm)
then obtain E where
  DPN: E + {#Pos P#}  $\in$  N and
  prod: production N (E + {#Pos P#}) = {P}
  using in-interp-is-produced by blast
have  $\langle \neg ?N_{\mathcal{I}} \models h C \rangle$ 
  using not-d-interp P unfolding D Lneg by auto
then have uL-C:  $\langle Pos P \notin \# C \rangle$ 
  using P unfolding Lneg by blast

have sup-EC: superposition-rules (E + {#Pos P#}) (C + {#Neg P#}) (E + C)
  using superposition-l by fast
then have superposition N (N  $\cup$  {E+C})
  using DPN  $\langle D \in N \rangle$  unfolding D L by (auto simp add: superposition.simps)
have
  PMax: Pos P = Max-mset (E + {#Pos P#}) and
  count (E + {#Pos P#}) (Pos P)  $\leq$  1 and
  S (E + {#Pos P#}) = {#} and
   $\neg$ interp N (E + {#Pos P#})  $\models h$  E + {#Pos P#}
  using prod unfolding production-unfold by auto
have Neg P  $\notin$  # E
  using prod produces-imp-neg-notin-lits by force
then have  $\bigwedge y. y \in \# (E + \{ \#Pos P\# \}) \implies$ 
  count (E + {#Pos P#}) (Neg P) < count (C + {#Neg P#}) (Neg P)
  using count-greater-zero-iff by fastforce
moreover have  $\bigwedge y. y \in \# (E + \{ \#Pos P\# \}) \implies y < Neg P$ 
  using PMax by (metis DPN Max-less-iff empty finite-set-mset pos-less-neg
    set-mset-eq-empty-iff)
moreover have E + {#Pos P#}  $\neq$  C + {#Neg P#}
  using prod produces-imp-neg-notin-lits by force
ultimately have E + {#Pos P#} < C + {#Neg P#}
  unfolding less-multisetHO by (metis count-greater-zero-iff less-iff-Suc-add zero-less-Suc)
have ce-lt-d: C + E < D

```

```

unfolding  $D\ L$  by (simp add:  $\langle \bigwedge y. y \in \# E + \{\#Pos\ P\} \implies y < Neg\ P \rangle$  ex-gt-imp-less-multiset)
have  $?N_{\mathcal{I}} \models_h E + \{\#Pos\ P\}$ 
  using  $\langle P \in ?N_{\mathcal{I}} \rangle$  by blast
have  $?N_{\mathcal{I}} \models_h C+E \vee C+E \notin N$ 
  using ce-lt-d cls-not-D unfolding  $D\text{-def}$  by fastforce
have  $Pos\text{-}P\text{-}C\text{-}E: Pos\ P \notin \# C+E$ 
  using  $D \langle P \in \text{ground-resolution-with-selection.INTERP } S\ N \rangle$ 
     $\langle count\ (E + \{\#Pos\ P\})\ (Pos\ P) \leq 1 \rangle$  multi-member-skip not-d-interp
  by (auto simp: not-in-iff)
then have  $\bigwedge y. y \in \# C+E \implies count\ (C+E)\ (Pos\ P) < count\ (E + \{\#Pos\ P\})\ (Pos\ P)$ 
  using set-mset-def by fastforce
have  $\neg \text{redundant}\ (C + E)\ N$ 
proof (rule ccontr)
  assume  $\text{red}'[\text{simplified}]: \neg ?thesis$ 
  have  $\text{abs: } \text{class-lt}\ N\ (C + E) \models_p C + E$ 
    using redundant-iff-abstract red' unfolding abstract-red-def by auto
  moreover
have  $\langle \text{class-lt}\ N\ (C + E) \subseteq \text{class-lt}\ N\ (E + \{\#Pos\ P\}) \rangle$ 
    using ce-lt-d Pos-P-C-E uL-C apply (auto simp: class-lt-def D L)

    using  $Pos\text{-}P\text{-}C\text{-}E$  unfolding less-multisetHO
    apply (auto split: if-splits)
    sorry
then have  $\text{class-lt}\ N\ (E + \{\#Pos\ P\}) \models_p E + \{\#Pos\ P\} \vee$ 
   $\text{class-lt}\ N\ (C + \{\#Neg\ P\}) \models_p C + \{\#Neg\ P\}$ 
proof clarify
  assume  $CP: \neg \text{class-lt}\ N\ (C + \{\#Neg\ P\}) \models_p C + \{\#Neg\ P\}$ 
  { fix  $I$ 
    assume
      total-over-m  $I\ (\text{class-lt}\ N\ (C + E) \cup \{E + \{\#Pos\ P\}\})$  and
      consistent-interp  $I$  and
       $I \models_s \text{class-lt}\ N\ (C + E)$ 
    then have  $I \models C + E$ 
      using abs sorry
    moreover have  $\neg I \models C + \{\#Neg\ P\}$ 
      using  $CP$  unfolding true-class-cls-def
      sorry
    ultimately have  $I \models E + \{\#Pos\ P\}$  by auto
  }
then show  $\text{class-lt}\ N\ (E + \{\#Pos\ P\}) \models_p E + \{\#Pos\ P\}$ 
  unfolding true-class-cls-def sorry
qed
then have  $\text{class-lt}\ N\ (C + E) \models_p E + \{\#Pos\ P\} \vee \text{class-lt}\ N\ (C + E) \models_p C + \{\#Neg\ P\}$ 
proof clarify
  assume  $CP: \neg \text{class-lt}\ N\ (C + E) \models_p C + \{\#Neg\ P\}$ 
  { fix  $I$ 
    assume
      total-over-m  $I\ (\text{class-lt}\ N\ (C + E) \cup \{E + \{\#Pos\ P\}\})$  and
      consistent-interp  $I$  and
       $I \models_s \text{class-lt}\ N\ (C + E)$ 
    then have  $I \models C + E$ 
      using abs sorry
    moreover have  $\neg I \models C + \{\#Neg\ P\}$ 
      using  $CP$  unfolding true-class-cls-def
      sorry
    ultimately have  $I \models E + \{\#Pos\ P\}$  by auto
  }

```

```

    }
    then show clss-lt  $N$   $(C + E) \models_p E + \{\#Pos\ P\# \}$ 
      unfolding true-clss-clss-def by auto
qed
moreover have clss-lt  $N$   $(C + E) \subseteq \text{clss-lt } N (C + \{\#Neg\ P\# \})$ 
  using ce-lt-d order.strict-trans2 unfolding clss-lt-def  $D\ L$ 
  by (blast dest: less-imp-le)
ultimately have redundant  $(C + \{\#Neg\ P\# \})\ N \vee \text{clss-lt } N (C + E) \models_p E + \{\#Pos\ P\# \}$ 
  unfolding redundant-iff-abstract abstract-red-def using true-clss-clss-subset by blast
show False

  sorry
qed
moreover have  $\neg \text{redundant } (E + \{\#Pos\ P\# \})\ N$ 
  sorry
ultimately have CEN:  $C + E \in N$ 
  using  $\langle D \in N \rangle \langle E + \{\#Pos\ P\# \} \in N \rangle$  saturated sup-EC red unfolding saturated-def  $D\ L$ 
  by (metis union-commute)
have CED:  $C + E \neq D$ 
  using  $D$  ce-lt-d by auto
have interp:  $\neg \text{INTERP } N \models_h C + E$ 
  sorry
show False
  using cls-not-D[OF CEN CED interp] ce-lt-d unfolding INTERP-def less-eq-multiset-def by auto
qed
qed

end

lemma tautology-is-redundant:
  assumes tautology  $C$ 
  shows abstract-red  $C\ N$ 
  using assms unfolding abstract-red-def true-clss-clss-def tautology-def by auto

lemma subsumed-is-redundant:
  assumes AB:  $A \subset\# B$ 
  and AN:  $A \in N$ 
  shows abstract-red  $B\ N$ 
proof -
  have  $A \in \text{clss-lt } N\ B$  using AN AB unfolding clss-lt-def
    by (auto dest: subset-eq-imp-le-multiset simp add: dual-order.order-iff-strict)
  then show ?thesis
    using AB unfolding abstract-red-def true-clss-clss-def Partial-Herbrand-Interpretation.true-clss-def
    by blast
qed

inductive redundant :: 'a clause  $\Rightarrow$  'a clause-set  $\Rightarrow$  bool where
  subsumption:  $A \in N \Longrightarrow A \subset\# B \Longrightarrow \text{redundant } B\ N$ 

lemma redundant-is-redundancy-criterion:
  fixes  $A :: 'a :: \text{wellorder clause}$  and  $N :: 'a :: \text{wellorder clause-set}$ 
  assumes redundant  $A\ N$ 
  shows abstract-red  $A\ N$ 
  using assms
proof (induction rule: redundant.induct)
  case (subsumption  $A\ B\ N$ )

```

```

then show ?case
  using subsumed-is-redundant[of A N B] unfolding abstract-red-def clss-lt-def by auto
qed

```

```

lemma redundant-mono:
  redundant A N  $\implies$  A  $\subseteq\#$  B  $\implies$  redundant B N
  apply (induction rule: redundant.induct)
  by (meson subset-mset.less-le-trans subsumption)

```

```

locale truc =
  selection S for S :: nat clause  $\implies$  nat clause
begin

```

```

end

```

```

end

```