

# 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 . . . . .	8
1.1.4	Positions . . . . .	10
1.2	Semantics over the Syntax . . . . .	11
1.3	Rewrite Systems and Properties . . . . .	12
1.3.1	Lifting of Rewrite Rules . . . . .	12
1.3.2	Consistency Preservation . . . . .	13
1.3.3	Full Lifting . . . . .	14
1.4	Transformation testing . . . . .	14
1.4.1	Definition and first Properties . . . . .	14
1.4.2	Invariant conservation . . . . .	15
1.5	Rewrite Rules . . . . .	17
1.5.1	Elimination of the Equivalences . . . . .	17
1.5.2	Eliminate Implication . . . . .	19
1.5.3	Eliminate all the True and False in the formula . . . . .	20
1.5.4	PushNeg . . . . .	24
1.5.5	Push Inside . . . . .	26
1.6	The Full Transformations . . . . .	31
1.6.1	Abstract Definition . . . . .	31
1.6.2	Conjunctive Normal Form . . . . .	32
1.6.3	Disjunctive Normal Form . . . . .	33
1.7	More aggressive simplifications: Removing true and false at the beginning . . . . .	33
1.7.1	Transformation . . . . .	33
1.7.2	More invariants . . . . .	34
1.7.3	The new CNF and DNF transformation . . . . .	35
1.8	Link with Multiset Version . . . . .	36
1.8.1	Transformation to Multiset . . . . .	36
1.8.2	Equisatisfiability of the two Versions . . . . .	36
<b>2</b>	<b>Resolution-based techniques</b>	<b>41</b>
2.1	Resolution . . . . .	41
2.1.1	Simplification Rules . . . . .	41
2.1.2	Unconstrained Resolution . . . . .	42
2.1.3	Inference Rule . . . . .	43
2.1.4	Lemma about the Simplified State . . . . .	48
2.1.5	Resolution and Invariants . . . . .	49

2.2	Superposition . . . . .	57
2.2.1	We can now define the rules of the calculus . . . . .	62
<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**  $'v \text{ propo} =$   
 $FT \mid FF \mid FVar\ 'v \mid FNot\ 'v \text{ propo} \mid FAnd\ 'v \text{ propo}\ 'v \text{ propo} \mid FOr\ 'v \text{ propo}\ 'v \text{ propo}$   
 $\mid FImp\ 'v \text{ propo}\ 'v \text{ propo} \mid FEq\ 'v \text{ propo}\ 'v \text{ propo}$

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**  $'v \text{ connective} = CT \mid CF \mid CVar\ 'v \mid CNot \mid CAnd \mid COr \mid CImp \mid CEq$

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

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

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

**lemma**  $propo\text{-}induct\text{-}arity[case\text{-}names\ nullary\ unary\ binary]:$

**fixes**  $\varphi\ \psi :: 'v \text{ propo}$   
**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**  $P\ \psi$   
 $\langle proof \rangle$

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
<proof>
```

```
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
<proof>
```

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
<proof>
```

### 1.1.2 Properties of the Abstraction

First we can define simplification rules.

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

$wf\text{-}conn\ CT\ l \implies conn\ CT\ l = FT$   
 $wf\text{-}conn\ CF\ l \implies conn\ CF\ l = FF$   
 $wf\text{-}conn\ (CVar\ x)\ l \implies conn\ (CVar\ x)\ l = FVar\ x$   
 $\langle proof \rangle$

**lemma** *wf-conn-list-decomp[simp]*:

$wf\text{-}conn\ CT\ l \longleftrightarrow l = []$   
 $wf\text{-}conn\ CF\ l \longleftrightarrow l = []$   
 $wf\text{-}conn\ (CVar\ x)\ l \longleftrightarrow l = []$   
 $wf\text{-}conn\ CNot\ (\xi\ @\ \varphi\ \# \ \xi') \longleftrightarrow \xi = [] \wedge \xi' = []$   
 $\langle proof \rangle$

**lemma** *wf-conn-list*:

$wf\text{-}conn\ c\ l \implies conn\ c\ l = FT \longleftrightarrow (c = CT \wedge l = [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FF \longleftrightarrow (c = CF \wedge l = [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FVar\ x \longleftrightarrow (c = CVar\ x \wedge l = [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FAnd\ a\ b \longleftrightarrow (c = CAnd \wedge l = a\ \# \ b\ \# \ [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FOr\ a\ b \longleftrightarrow (c = COr \wedge l = a\ \# \ b\ \# \ [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FEq\ a\ b \longleftrightarrow (c = CEq \wedge l = a\ \# \ b\ \# \ [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FImp\ a\ b \longleftrightarrow (c = CImp \wedge l = a\ \# \ b\ \# \ [])$   
 $wf\text{-}conn\ c\ l \implies conn\ c\ l = FNot\ a \longleftrightarrow (c = CNot \wedge l = a\ \# \ [])$   
 $\langle proof \rangle$

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

**lemma** *list-length2-decomp*:  $length\ l = 2 \implies (\exists\ a\ b.\ l = a\ \# \ b\ \# \ [])$

$\langle proof \rangle$

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

**lemma** *wf-conn-bin-list-length*:

**fixes**  $l :: 'v\ propo\ list$   
**assumes**  $conn: c \in binary\text{-}connectives$   
**shows**  $length\ l = 2 \longleftrightarrow wf\text{-}conn\ c\ l$

$\langle proof \rangle$

**lemma** *wf-conn-not-list-length[iff]*:

**fixes**  $l :: 'v\ propo\ list$   
**shows**  $wf\text{-}conn\ CNot\ l \longleftrightarrow length\ l = 1$

$\langle proof \rangle$

Decomposing the Not into an element is moreover very useful.

**lemma** *wf-conn-Not-decomp*:

**fixes**  $l :: 'v\ propo\ list$  **and**  $a :: 'v$   
**assumes**  $corr: wf\text{-}conn\ CNot\ l$   
**shows**  $\exists\ a.\ l = [a]$

$\langle proof \rangle$

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*:

$length\ l = length\ l' \implies wf\text{-}conn\ c\ l \longleftrightarrow wf\text{-}conn\ c\ l'$

$\langle proof \rangle$

**lemma** *wf-conn-no-arity-change-helper*:  
 $\text{length } (\xi @ \varphi \# \xi') = \text{length } (\xi @ \varphi' \# \xi')$   
 $\langle \text{proof} \rangle$

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

**lemma** *conn-inj-not*:  
**assumes** *correct*: *wf-conn* *c* *l*  
**and** *conn*: *conn* *c* *l* = *FNot*  $\psi$   
**shows** *c* = *CNot* **and** *l* = [ $\psi$ ]  
 $\langle \text{proof} \rangle$

**lemma** *conn-inj*:  
**fixes** *c* *ca* :: '*v* *connective* **and** *l*  $\psi$  *s* :: '*v* *propo* *list*  
**assumes** *corr*: *wf-conn* *ca* *l*  
**and** *corr'*: *wf-conn* *c*  $\psi$  *s*  
**and** *eq*: *conn* *ca* *l* = *conn* *c*  $\psi$  *s*  
**shows** *ca* = *c*  $\wedge$   $\psi$  *s* = *l*  
 $\langle \text{proof} \rangle$

### 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*:  $\psi \in \text{set } l \implies \text{wf-conn } c \ l \implies \varphi \preceq \psi \implies \varphi \preceq \text{conn } c \ l$

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*  $\varphi \preceq \psi \implies \varphi \preceq \psi$   
 $\langle \text{proof} \rangle$

**lemma** *subformula-in-binary-conn*:  
**assumes** *conn*: *c*  $\in$  *binary-connectives*  
**shows**  $f \preceq \text{conn } c \ [f, g]$   
**and**  $g \preceq \text{conn } c \ [f, g]$   
 $\langle \text{proof} \rangle$

**lemma** *subformula-trans*:  
 $\psi \preceq \psi' \implies \varphi \preceq \psi \implies \varphi \preceq \psi'$   
 $\langle \text{proof} \rangle$

**lemma** *subformula-leaf*:  
**fixes**  $\varphi \ \psi$  :: '*v* *propo*  
**assumes** *incl*:  $\varphi \preceq \psi$   
**and** *simple*:  $\psi = \text{FT} \vee \psi = \text{FF} \vee \psi = \text{FVar } x$   
**shows**  $\varphi = \psi$   
 $\langle \text{proof} \rangle$

**lemma** *subformula-not-incl-eq*:



**assumes**  $\varphi \preceq \text{conn } c \ l$   
**and**  $\text{wf-conn } c \ l$   
**and**  $\forall \psi. \psi \in \text{set } l \longrightarrow \neg \varphi \preceq \psi$   
**shows**  $\varphi = \text{conn } c \ l$   
 $\langle \text{proof} \rangle$

**lemma** *wf-subformula-conn-cases*:

$\text{wf-conn } c \ l \implies \varphi \preceq \text{conn } c \ l \longleftrightarrow (\varphi = \text{conn } c \ l \vee (\exists \psi. \psi \in \text{set } l \wedge \varphi \preceq \psi))$   
 $\langle \text{proof} \rangle$

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

$\varphi \preceq \text{FAnd } \psi \ \psi' \longleftrightarrow (\varphi = \text{FAnd } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi') \text{ (is ?P FAnd)}$   
 $\varphi \preceq \text{FOr } \psi \ \psi' \longleftrightarrow (\varphi = \text{FOr } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$   
 $\varphi \preceq \text{FEq } \psi \ \psi' \longleftrightarrow (\varphi = \text{FEq } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$   
 $\varphi \preceq \text{FImp } \psi \ \psi' \longleftrightarrow (\varphi = \text{FImp } \psi \ \psi' \vee \varphi \preceq \psi \vee \varphi \preceq \psi')$   
 $\langle \text{proof} \rangle$

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

$\text{wf-conn } \text{CNot } [\varphi]$   
 $\text{wf-conn } \text{CT } []$   
 $\text{wf-conn } \text{CF } []$   
 $\text{wf-conn } (\text{CVar } x) []$   
 $\text{wf-conn } \text{CAnd } [\varphi, \psi]$   
 $\text{wf-conn } \text{COr } [\varphi, \psi]$   
 $\text{wf-conn } \text{CImp } [\varphi, \psi]$   
 $\text{wf-conn } \text{CEq } [\varphi, \psi]$   
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

**assumes**  $\text{wf: wf-conn } c \ l$   
**shows**  $\varphi \preceq \text{conn } c \ l \longleftrightarrow (\varphi = \text{conn } c \ l \vee (\exists \psi \in \text{set } l. \varphi \preceq \psi)) \text{ (is ?A } \longleftrightarrow ?B)$   
 $\langle \text{proof} \rangle$

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

$\varphi \preceq \text{FT} \longleftrightarrow \varphi = \text{FT}$   
 $\varphi \preceq \text{FF} \longleftrightarrow \varphi = \text{FF}$   
 $\varphi \preceq \text{FVar } x \longleftrightarrow \varphi = \text{FVar } x$   
 $\langle \text{proof} \rangle$

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**

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

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

**fixes**  $\xi \ \xi' :: 'v \text{ propo list}$  **and**  $\psi :: 'v \text{ propo}$  **and**  $c :: 'v \text{ connective}$   
**assumes**  $\text{corr: wf-conn } c \ l$  **and**  $\text{incl: } \psi \in \text{set } l$

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

The set of variables is compatible with the subformula order.

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

$\varphi \preceq \psi \implies \text{vars-of-prop } \varphi \subseteq \text{vars-of-prop } \psi$   
 $\langle \text{proof} \rangle$

#### 1.1.4 Positions

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

**datatype**  $\text{sign} = L \mid R$

We use  $\text{nil}$  instead of  $\varepsilon$ .

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

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

**lemma** *finite-pos*:  $\text{finite } (\text{pos } \varphi)$

$\langle \text{proof} \rangle$

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

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

**lemma** *cons-inject*:

$\text{inj } ((\#) \ s)$   
 $\langle \text{proof} \rangle$

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

$\text{finite } s \implies \text{card } (\text{insert } \square \ \{L \ \# \ p \mid p. p \in s\}) = 1 + \text{card } \{L \ \# \ p \mid p. p \in s\}$   
 $\langle \text{proof} \rangle$

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

$\text{card } (\text{pos } (FNot \ \varphi)) = 1 + \text{card } (\text{pos } \varphi)$   
 $\langle \text{proof} \rangle$

**lemma** *card-seperate*:

**assumes**  $\text{finite } s1$  **and**  $\text{finite } s2$   
**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**  $\text{card } (?L \cup ?R) = \text{card } ?L + \text{card } ?R$ )  
 $\langle \text{proof} \rangle$

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

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

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

**shows**  $\text{card } (\text{vars-of-prop } \varphi) \leq \text{prop-size } \varphi$

$\langle \text{proof} \rangle$

**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*:  $c \in \text{binary-connectives} \vee c = \text{CNot} \implies \text{wf-conn } c (\varphi \# l) \implies \text{path-to } p \varphi \varphi' \implies$

*path-to* (*L* # *p*) (*conn* *c* ( $\varphi \# l$ ))  $\varphi'$  |

*path-to-r*:  $c \in \text{binary-connectives} \implies \text{wf-conn } c (\psi \# \varphi \# []) \implies \text{path-to } p \varphi \varphi' \implies$

*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*  $\varphi \varphi' \implies \varphi' \preceq \varphi$

$\langle \text{proof} \rangle$

**lemma** *subformula-path-exists*:

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

**shows**  $\varphi' \preceq \varphi \implies \exists p. \text{path-to } p \varphi \varphi'$

$\langle \text{proof} \rangle$

**fun** *replace-at* :: *sign list*  $\Rightarrow$  *'v propo*  $\Rightarrow$  *'v propo*  $\Rightarrow$  *'v propo* **where**

*replace-at* [] -  $\psi = \psi$  |

*replace-at* (*L* # *l*) (*FAnd*  $\varphi \varphi'$ )  $\psi = \text{FAnd } (\text{replace-at } l \varphi \psi) \varphi'$  |

*replace-at* (*R* # *l*) (*FAnd*  $\varphi \varphi'$ )  $\psi = \text{FAnd } \varphi (\text{replace-at } l \varphi' \psi)$  |

*replace-at* (*L* # *l*) (*FOr*  $\varphi \varphi'$ )  $\psi = \text{FOr } (\text{replace-at } l \varphi \psi) \varphi'$  |

*replace-at* (*R* # *l*) (*FOr*  $\varphi \varphi'$ )  $\psi = \text{FOr } \varphi (\text{replace-at } l \varphi' \psi)$  |

*replace-at* (*L* # *l*) (*FEq*  $\varphi \varphi'$ )  $\psi = \text{FEq } (\text{replace-at } l \varphi \psi) \varphi'$  |

*replace-at* (*R* # *l*) (*FEq*  $\varphi \varphi'$ )  $\psi = \text{FEq } \varphi (\text{replace-at } l \varphi' \psi)$  |

*replace-at* (*L* # *l*) (*FImp*  $\varphi \varphi'$ )  $\psi = \text{FImp } (\text{replace-at } l \varphi \psi) \varphi'$  |

*replace-at* (*R* # *l*) (*FImp*  $\varphi \varphi'$ )  $\psi = \text{FImp } \varphi (\text{replace-at } l \varphi' \psi)$  |

*replace-at* (*L* # *l*) (*FNot*  $\varphi$ )  $\psi = \text{FNot } (\text{replace-at } l \varphi \psi)$

## 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*  $\Rightarrow$  *bool*)  $\Rightarrow$  *'v propo*  $\Rightarrow$  *bool* (**infix**  $\models$  50) **where**

$\mathcal{A} \models \text{FT} = \text{True}$  |

$\mathcal{A} \models \text{FF} = \text{False}$  |

$\mathcal{A} \models \text{FVar } v = (\mathcal{A} \ v)$  |

$\mathcal{A} \models \text{FNot } \varphi = (\neg(\mathcal{A} \models \varphi))$  |

$\mathcal{A} \models \text{FAnd } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \wedge \mathcal{A} \models \varphi_2)$  |

$\mathcal{A} \models \text{FOr } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \vee \mathcal{A} \models \varphi_2)$  |

$\mathcal{A} \models \text{FImp } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \longrightarrow \mathcal{A} \models \varphi_2)$  |

$\mathcal{A} \models \text{FEq } \varphi_1 \varphi_2 = (\mathcal{A} \models \varphi_1 \longleftrightarrow \mathcal{A} \models \varphi_2)$

**definition** *evalf* (**infix**  $\models_f$  50) **where**

*evalf*  $\varphi \psi = (\forall A. A \models \varphi \longrightarrow A \models \psi)$

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

**theorem** *deduction-theorem*:

$\varphi \models^f \psi \longleftrightarrow (\forall A. A \models FImp \varphi \psi)$   
 $\langle proof \rangle$

A shorter proof:

**lemma**  $\varphi \models^f \psi \longleftrightarrow (\forall A. A \models FImp \varphi \psi)$   
 $\langle proof \rangle$

**definition** *same-over-set*::  $('v \Rightarrow bool) \Rightarrow ('v \Rightarrow bool) \Rightarrow 'v \text{ set} \Rightarrow bool$  **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**  $A \models \varphi \longleftrightarrow B \models \varphi$   
 $\langle proof \rangle$

**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* ::  $('v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow bool) \Rightarrow 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow bool$   
**for**  $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow bool$  **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**  $\text{propo-rew-step } r \ \varphi \ \varphi' \implies \exists \ \psi \ \psi'. \ \psi \preceq \varphi \wedge \psi' \preceq \varphi' \wedge r \ \psi \ \psi'$   
 $\langle proof \rangle$

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**  $\psi \preceq \varphi \implies r \ \psi \ \psi' \implies (\exists \varphi'. \ \psi' \preceq \varphi' \wedge \text{propo-rew-step } r \ \varphi \ \varphi')$   
 $\langle \text{proof} \rangle$

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

**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**  $A \models conn \ c \ l \longleftrightarrow A \models conn \ c \ l'$   
 $\langle \text{proof} \rangle$

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**  $r :: 'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow bool$   
**assumes** *propo-rew-step*  $r \ \varphi \ \varphi'$   
**shows**  $\exists \psi \ \psi' \ p. \ r \ \psi \ \psi' \wedge \text{path-to } p \ \varphi \ \psi \wedge \text{replace-at } p \ \varphi \ \psi' = \varphi'$   
 $\langle \text{proof} \rangle$

### 1.3.2 Consistency Preservation

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

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

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

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

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

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

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

### 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]:  
 $\text{preserve-models } r \implies \text{preserve-models } (\text{full } (\text{propo-rew-step } r))$   
 ⟨proof⟩

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

## 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]:  
 $\text{test-symb } FT \implies \text{all-subformula-st test-symb } FT$   
 $\text{test-symb } FF \implies \text{all-subformula-st test-symb } FF$   
 $\text{test-symb } (FVar \ x) \implies \text{all-subformula-st test-symb } (FVar \ x)$   
 ⟨proof⟩

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

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

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

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

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

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

*<proof>*

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

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

**shows** *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>*

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

**lemma** *subformula-all-subformula-st*:

$\psi \preceq \varphi \implies \text{all-subformula-st test-symb } \varphi \implies \text{all-subformula-st test-symb } \psi$

*<proof>*

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

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

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

**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**  $\exists \psi \psi'. \psi \preceq \varphi \wedge r \psi \psi'$

*<proof>*

### 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**  $\text{all-subformula-st test-symb } \psi$   
 $\langle \text{proof} \rangle$

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

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

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

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

## Invariant after all Rewriting

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

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

**lemma** *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**  
**full:**  $\text{full } (\text{propo-rew-step } r) \ \varphi \ \psi$  **and**  
**init:**  $\text{all-subformula-st test-symb } \varphi$   
**shows**  $\text{all-subformula-st test-symb } \psi$   
 $\langle \text{proof} \rangle$

**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**  
**full:**  $\text{full } (\text{propo-rew-step } r) \ \varphi \ \psi$  **and**  
**init:**  $\text{all-subformula-st test-symb } \varphi$   
**shows**  $\text{all-subformula-st test-symb } \psi$   
 $\langle \text{proof} \rangle$

**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**  
**full:**  $\text{full } (\text{propo-rew-step } r) \ \varphi \ \psi$  **and**  
**init:**  $\text{all-subformula-st test-symb } \varphi$   
**shows**  $\text{all-subformula-st test-symb } \psi$   
 $\langle \text{proof} \rangle$

**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* ::  $'v \text{ propo} \Rightarrow 'v \text{ propo} \Rightarrow \text{bool}$  **where**  
 $\text{elim-equiv}[\text{simp}]: \text{elim-equiv } (\text{FEq } \varphi \ \psi) \ (\text{FAnd } (\text{FImp } \varphi \ \psi) \ (\text{FImp } \psi \ \varphi))$

**lemma** *elim-equiv-transformation-consistent:*

$A \models \text{FEq } \varphi \ \psi \longleftrightarrow A \models \text{FAnd } (\text{FImp } \varphi \ \psi) \ (\text{FImp } \psi \ \varphi)$

$\langle \text{proof} \rangle$

**lemma** *elim-equiv-explicit*:  $\text{elim-equiv } \varphi \ \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$   
 $\langle \text{proof} \rangle$

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

**lemma** *elimEquiv-lifted-consistant*:  
*preserve-models (full (propo-rew-step elim-equiv))*  
 $\langle \text{proof} \rangle$

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**  $c :: 'v \text{ connective}$  **and**  $l :: 'v \text{ propo list}$   
**assumes**  $\text{wf}: \text{wf-conn } c \ l$   
**shows**  $\text{no-equiv-symb } (\text{conn } c \ l) \longleftrightarrow c \neq \text{CEq}$   
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *all-subformula-st-decomp-explicit-no-equiv[iff]*:  
**fixes**  $\varphi \ \psi :: 'v \text{ propo}$   
**shows**  
 $\text{no-equiv } (\text{FNot } \varphi) \longleftrightarrow \text{no-equiv } \varphi$   
 $\text{no-equiv } (\text{FAnd } \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$   
 $\text{no-equiv } (\text{FOr } \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$   
 $\text{no-equiv } (\text{FImp } \varphi \ \psi) \longleftrightarrow (\text{no-equiv } \varphi \wedge \text{no-equiv } \psi)$   
 $\langle \text{proof} \rangle$

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

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

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

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

### 1.5.2 Eliminate Implication

After that, we can eliminate the implication symbols.

**inductive** *elim-imp* :: 'v propo  $\Rightarrow$  'v propo  $\Rightarrow$  bool **where**  
 $[\text{simp}]: \text{elim-imp } (F\text{Imp } \varphi \psi) (F\text{Or } (F\text{Not } \varphi) \psi)$

**lemma** *elim-imp-transformation-consistent*:  
 $A \models F\text{Imp } \varphi \psi \longleftrightarrow A \models F\text{Or } (F\text{Not } \varphi) \psi$   
 $\langle \text{proof} \rangle$

**lemma** *elim-imp-explicit*:  $\text{elim-imp } \varphi \psi \implies \forall A. A \models \varphi \longleftrightarrow A \models \psi$   
 $\langle \text{proof} \rangle$

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

**lemma** *elim-imp-lifted-consistant*:  
 $\text{preserve-models } (\text{full } (\text{propo-rew-step elim-imp}))$   
 $\langle \text{proof} \rangle$

**fun** *no-imp-symb* **where**  
 $\text{no-imp-symb } (F\text{Imp } -) = \text{False} \mid$   
 $\text{no-imp-symb } - = \text{True}$

**lemma** *no-imp-symb-conn-characterization*:  
 $\text{wf-conn } c \ l \implies \text{no-imp-symb } (\text{conn } c \ l) \longleftrightarrow c \neq C\text{Imp}$   
 $\langle \text{proof} \rangle$

**definition** *no-imp* **where**  $\text{no-imp} \equiv \text{all-subformula-st no-imp-symb}$   
**declare**  $\text{no-imp-def}[\text{simp}]$

**lemma** *no-imp-Imp[simp]*:  
 $\neg \text{no-imp } (F\text{Imp } \varphi \psi)$   
 $\text{no-imp } FT$   
 $\text{no-imp } FF$   
 $\langle \text{proof} \rangle$

**lemma** *all-subformula-st-decomp-explicit-imp[simp]*:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**shows**  
 $\text{no-imp } (F\text{Not } \varphi) \longleftrightarrow \text{no-imp } \varphi$   
 $\text{no-imp } (F\text{And } \varphi \psi) \longleftrightarrow (\text{no-imp } \varphi \wedge \text{no-imp } \psi)$   
 $\text{no-imp } (F\text{Or } \varphi \psi) \longleftrightarrow (\text{no-imp } \varphi \wedge \text{no-imp } \psi)$   
 $\langle \text{proof} \rangle$

Invariant of the *elim-imp* transformation

**lemma** *elim-imp-no-equiv*:  
 $\text{elim-imp } \varphi \psi \implies \text{no-equiv } \varphi \implies \text{no-equiv } \psi$

$\langle \text{proof} \rangle$

**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*  $\psi$

$\langle \text{proof} \rangle$

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

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

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

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

### 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*  $\varphi \ FT$ )  $\varphi \mid$

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

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

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

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

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

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

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

*ElimTB5*: *elimTB* (*FNot*  $FT$ )  $FF \mid$

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

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

$\langle \text{proof} \rangle$

**inductive** *no-T-F-symb* ::  $'v \text{ propo} \Rightarrow \text{bool}$  **where**

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

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

*wf-conn*  $c \ \psi s \implies$

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

$\langle \text{proof} \rangle$

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

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

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

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

*no-T-F-symb* (*FImp*  $\varphi$   $\psi$ )  $\longleftrightarrow (\forall \chi \in \text{set } [\varphi, \psi]. \chi \neq FF \wedge \chi \neq FT)$   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-false*[*simp*]:  
**fixes**  $c :: 'v$  *connective*  
**shows**  
 $\neg \text{no-T-F-symb } (FT :: 'v \text{ propo})$   
 $\neg \text{no-T-F-symb } (FF :: 'v \text{ propo})$   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-bool*[*simp*]:  
**fixes**  $x :: 'v$   
**shows** *no-T-F-symb* (*FVar*  $x$ )  
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-fnot-imp*:  
 $\neg \text{no-T-F-symb } (FNot \varphi) \implies \varphi = FT \vee \varphi = FF$   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-fnot*[*simp*]:  
 $\text{no-T-F-symb } (FNot \varphi) \longleftrightarrow \neg(\varphi = FT \vee \varphi = FF)$   
 $\langle \text{proof} \rangle$

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*  $\varphi \implies \text{no-T-F-symb-except-toplevel } \varphi$

**lemma** *no-T-F-symb-except-toplevel-bool*:  
**fixes**  $x :: 'v$   
**shows** *no-T-F-symb-except-toplevel* (*FVar*  $x$ )  
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *no-T-F-symb-except-top-level-false-example[simp]*:

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

**assumes**  $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$

**shows**

$\neg \text{no-T-F-symb-except-toplevel } (FAnd \ \varphi \ \psi)$

$\neg \text{no-T-F-symb-except-toplevel } (FOr \ \varphi \ \psi)$

$\neg \text{no-T-F-symb-except-toplevel } (FImp \ \varphi \ \psi)$

$\neg \text{no-T-F-symb-except-toplevel } (FEq \ \varphi \ \psi)$

$\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-except-top-level-false-not[simp]*:

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

**assumes**  $\varphi = FT \vee \varphi = FF$

**shows**

$\neg \text{no-T-F-symb-except-toplevel } (FNot \ \varphi)$

$\langle \text{proof} \rangle$

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

**definition** *no-T-F-except-top-level* **where**

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

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

**and**  $FT \in \text{set } l \vee FF \in \text{set } l$

**shows**  $\neg \text{no-T-F-except-top-level } (\text{conn } c \ l)$

$\langle \text{proof} \rangle$

**lemma** *no-T-F-except-top-level-false-example[simp]*:

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

**assumes**  $\varphi = FT \vee \psi = FT \vee \varphi = FF \vee \psi = FF$

**shows**

$\neg \text{no-T-F-except-top-level } (FAnd \ \varphi \ \psi)$

$\neg \text{no-T-F-except-top-level } (FOr \ \varphi \ \psi)$

$\neg \text{no-T-F-except-top-level } (FEq \ \varphi \ \psi)$

$\neg \text{no-T-F-except-top-level } (FImp \ \varphi \ \psi)$

$\langle \text{proof} \rangle$

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

*no-T-F-symb-except-toplevel*  $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies \text{no-T-F-symb } \varphi$

$\langle \text{proof} \rangle$

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

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

*no-T-F-except-top-level*  $\varphi \implies \varphi \neq FF \implies \varphi \neq FT \implies \text{no-T-F } \varphi$

$\langle \text{proof} \rangle$

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

*no-T-F*  $\varphi \implies \text{no-T-F-except-top-level } \varphi$

$\langle \text{proof} \rangle$

**lemma** *no-T-F-except-top-level-simp*[simp]: *no-T-F-except-top-level FF no-T-F-except-top-level FT*  
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-no-T-F-except-top-level'*[simp]:  
*no-T-F-except-top-level  $\varphi \longleftrightarrow (\varphi = FF \vee \varphi = FT \vee \text{no-T-F } \varphi)$*   
 $\langle \text{proof} \rangle$

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

**lemma** *no-T-F-bin-decomp-expanded*[simp]:  
**assumes** *c*: *c* = *CAnd*  $\vee$  *c* = *COr*  $\vee$  *c* = *CEq*  $\vee$  *c* = *CImp*  
**shows** *no-T-F (conn c [ $\varphi$ ,  $\psi$ ])  $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$*   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-comp-expanded-explicit*[simp]:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**shows**  
*no-T-F (FAnd  $\varphi \psi$ )  $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$*   
*no-T-F (FOr  $\varphi \psi$ )  $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$*   
*no-T-F (FEq  $\varphi \psi$ )  $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$*   
*no-T-F (FImp  $\varphi \psi$ )  $\longleftrightarrow (\text{no-T-F } \varphi \wedge \text{no-T-F } \psi)$*   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-comp-not*[simp]:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**shows** *no-T-F (FNot  $\varphi$ )  $\longleftrightarrow \text{no-T-F } \varphi$*   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-decomp*:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**assumes**  $\varphi$ : *no-T-F (FAnd  $\varphi \psi$ )  $\vee$  no-T-F (FOr  $\varphi \psi$ )  $\vee$  no-T-F (FEq  $\varphi \psi$ )  $\vee$  no-T-F (FImp  $\varphi \psi$ )  
**shows** *no-T-F  $\psi$  and no-T-F  $\varphi$*   
 $\langle \text{proof} \rangle$*

**lemma** *no-T-F-decomp-not*:  
**fixes**  $\varphi :: 'v \text{ propo}$   
**assumes**  $\varphi$ : *no-T-F (FNot  $\varphi$ )*  
**shows** *no-T-F  $\varphi$*   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-symb-except-toplevel-step-exists*:  
**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**assumes** *no-equiv  $\varphi$  and no-imp  $\varphi$*   
**shows**  $\psi \preceq \varphi \implies \neg \text{no-T-F-symb-except-toplevel } \psi \implies \exists \psi'. \text{elimTB } \psi \psi'$   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-except-top-level-rew*:  
**fixes**  $\varphi :: 'v \text{ propo}$   
**assumes** *noTB*:  $\neg \text{no-T-F-except-top-level } \varphi$  **and** *no-equiv*: *no-equiv  $\varphi$  and no-imp*: *no-imp  $\varphi$*   
**shows**  $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{elimTB } \psi \psi'$   
 $\langle \text{proof} \rangle$

**lemma** *elimTB-inv*:

**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**assumes** *full (propo-rew-step elimTB)  $\varphi \psi$*   
**and** *no-equiv  $\varphi$  and no-imp  $\varphi$*   
**shows** *no-equiv  $\psi$  and no-imp  $\psi$*

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

### 1.5.4 PushNeg

Push the negation inside the formula, until the literal.

**inductive** *pushNeg* **where**

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

**lemma** *pushNeg-transformation-consistent*:

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *pushNeg-lifted-consistant*:

*preserve-models (full (propo-rew-step pushNeg))*

$\langle \text{proof} \rangle$

**fun** *simple* **where**

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

**lemma** *simple-decomp*:

*simple  $\varphi \longleftrightarrow (\varphi = \text{FT} \vee \varphi = \text{FF} \vee (\exists x. \varphi = \text{FVar } x))$*

$\langle \text{proof} \rangle$

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

**fixes**  $\varphi \psi :: 'v \text{ propo}$   
**assumes** *s: simple  $\psi$*   
**shows**  $\varphi \preceq \text{FNot } \psi \longleftrightarrow (\varphi = \text{FNot } \psi \vee \varphi = \psi)$



$\langle \text{proof} \rangle$

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

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

**shows**

$\varphi \preceq \text{FNot } FT \longleftrightarrow (\varphi = \text{FNot } FT \vee \varphi = FT)$

$\varphi \preceq \text{FNot } FF \longleftrightarrow (\varphi = \text{FNot } FF \vee \varphi = FF)$

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

$\langle \text{proof} \rangle$

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

*simple-not-symb* ( $\text{FNot } \varphi$ ) = (*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]*:

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

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

$\langle \text{proof} \rangle$

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

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

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

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

$\langle \text{proof} \rangle$

**lemma** *simple-not-rew*:

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

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

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

$\langle \text{proof} \rangle$

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

*no-T-F-except-top-level* ( $\text{FNot } (\text{FAnd } \varphi \psi)$ )  $\implies$  *no-T-F-except-top-level* ( $\text{FOr } (\text{FNot } \varphi) (\text{FNot } \psi)$ )

$\langle \text{proof} \rangle$

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

*no-T-F-except-top-level* ( $\text{FNot } (\text{FOr } \varphi \psi)$ )  $\implies$  *no-T-F-except-top-level* ( $\text{FAnd } (\text{FNot } \varphi) (\text{FNot } \psi)$ )

$\langle \text{proof} \rangle$

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

*no-T-F-symb* ( $\text{FOr } (\text{FNot } \varphi') (\text{FNot } \psi')$ )

*no-T-F-symb* ( $\text{FAnd } (\text{FNot } \varphi') (\text{FNot } \psi')$ )

*no-T-F-symb* ( $\text{FNot } (\text{FNot } \varphi')$ )

$\langle \text{proof} \rangle$

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

*propo-rew-step pushNeg*  $\varphi \psi \implies$  *no-T-F-except-top-level*  $\varphi \implies$  *no-T-F-symb*  $\varphi \implies$  *no-T-F-symb*  $\psi$

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

**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*  $\psi$

$\langle \text{proof} \rangle$

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

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

**assumes**

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

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

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

*no-T-F-except-top-level*  $\varphi$

**shows** *simple-not*  $\psi$

$\langle \text{proof} \rangle$

### 1.5.5 Push Inside

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

**for**  $c c' :: 'v \text{ connective}$  **where**

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

$\Longrightarrow \text{push-conn-inside } c c' (\text{conn } c [\text{conn } c' [\varphi 1, \varphi 2], \psi])$

$(\text{conn } c' [\text{conn } c [\varphi 1, \psi], \text{conn } c [\varphi 2, \psi]]) \mid$

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

$\Longrightarrow \text{push-conn-inside } c c' (\text{conn } c [\psi, \text{conn } c' [\varphi 1, \varphi 2]])$

$(\text{conn } c' [\text{conn } c [\psi, \varphi 1], \text{conn } c [\psi, \varphi 2]])$

**lemma** *push-conn-inside-explicit*:  $\text{push-conn-inside } c c' \varphi \psi \Longrightarrow \forall A. A \models \varphi \longleftrightarrow A \models \psi$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

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

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

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

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

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

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

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

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

$\vee \xi = FNot (FVar x) \Longrightarrow \text{False}$

$\langle \text{proof} \rangle$

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

$\neg \text{not-c-in-c'-symb } c \ c' \ FF$   
 $\neg \text{not-c-in-c'-symb } c \ c' \ FT$   
 $\neg \text{not-c-in-c'-symb } c \ c' \ (FVar \ x)$   
 $\neg \text{not-c-in-c'-symb } c \ c' \ (FNot \ FF)$   
 $\neg \text{not-c-in-c'-symb } c \ c' \ (FNot \ FT)$   
 $\neg \text{not-c-in-c'-symb } c \ c' \ (FNot \ (FVar \ x))$   
 $\langle \text{proof} \rangle$

**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*  $c \ c' \ (FNot \ (FVar \ x))$   
 $\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

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

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

**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**  $?A \longleftrightarrow ?B$ )  
 $\langle \text{proof} \rangle$

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

**fixes**  $\varphi1 \ \varphi2 \ \psi :: 'v \ \text{propo}$  **and**  $x :: 'v$   
**shows**  
 $c\text{-in-c'-symb } c \ c' \ FT$   
 $c\text{-in-c'-symb } c \ c' \ FF$   
 $c\text{-in-c'-symb } c \ c' \ (FVar \ x)$   
 $\text{wf-conn } c \ [\text{conn } c' \ [\varphi1, \varphi2], \psi] \implies \text{wf-conn } c' \ [\varphi1, \varphi2]$   
 $\implies \neg c\text{-in-c'-only } c \ c' \ (\text{conn } c \ [\text{conn } c' \ [\varphi1, \varphi2], \psi])$   
 $\langle \text{proof} \rangle$

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

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

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

**fixes**  $\varphi :: 'v \ \text{propo}$   
**assumes**  $c: c = CAnd \vee c = COr$  **and**  $c': c' = CAnd \vee c' = COr$

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

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

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

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

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

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

<proof>

**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>

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

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

<proof>

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

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

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

<proof>

**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*  $c \ (\xi @ \varphi' \# \xi')$ )

<proof>

**lemma** *push-conn-inside-not-true-false*:

*push-conn-inside*  $c \ c' \ \varphi \ \psi \implies \psi \neq FT \wedge \psi \neq FF$

<proof>

**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>

**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*  $c$   $c'$   $\psi$   
 $\langle proof \rangle$

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

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

**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*  $\psi$   
 $\vee (\exists \varphi'. \psi = FNot \varphi' \wedge \text{simple } \varphi')$   
 $\vee (\exists l. \psi = \text{conn } c \ l \wedge \text{wf-conn } c \ l))$   
 $\langle proof \rangle$

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

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

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

**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-conn*  $c' \ l$   
**shows** *False*  
 $\langle proof \rangle$

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

**lemma** *c-in-c'-symb-decomp-level1*:  
**fixes**  $l :: 'v \text{ propo list}$  **and**  $c \ c' \ ca :: 'v \text{ connective}$   
**shows**  $\text{wf-conn } ca \ l \implies ca \neq c \implies c\text{-in-}c'\text{-symb } c \ c' (\text{conn } ca \ l)$   
 $\langle \text{proof} \rangle$

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

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

## Push Conjunction

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

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

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

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

**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*  $\psi$   
 $\langle \text{proof} \rangle$

**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*  $\psi$   
 $\langle \text{proof} \rangle$

## Push Disjunction

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

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

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

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

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

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

**lemma** *pushDisj-inv*:  
**fixes**  $\varphi \ \psi :: 'v \text{ propo}$   
**assumes** *full* (*propo-rew-step* *pushDisj*)  $\varphi \ \psi$   
**and** *no-equiv*  $\varphi$  **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*  $\psi$   
 $\langle \text{proof} \rangle$

**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*  $\psi$   
 $\langle \text{proof} \rangle$

## 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*))  
 $\langle \text{proof} \rangle$

**lemma** *only-c-inside-symb-c-eq-c'*:  
*only-c-inside-symb* *c* (*conn* *c'*  $[\varphi1, \varphi2]$ )  $\Rightarrow$   $c' = CAnd \vee c' = COr \Rightarrow$  *wf-conn* *c'*  $[\varphi1, \varphi2]$   
 $\Rightarrow c' = c$   
 $\langle \text{proof} \rangle$

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

*only-c-inside*  $c$  (*conn*  $c'$  [ $\varphi 1$ ,  $\varphi 2$ ])  $\implies c' = CAnd \vee c' = COr \implies wf\text{-}conn\ c' [\varphi 1, \varphi 2] \implies c = c'$   
 $\langle proof \rangle$

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

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

**shows** *only-c-inside*  $c\ \varphi \implies grouped\text{-}by\ c\ \varphi$  (**is**  $?O\ \varphi \implies ?G\ \varphi$ )

$\langle proof \rangle$

**lemma** *grouped-by-false*:

*grouped-by*  $c$  (*conn*  $c'$  [ $\varphi$ ,  $\psi$ ])  $\implies c \neq c' \implies wf\text{-}conn\ c' [\varphi, \psi] \implies False$

$\langle proof \rangle$

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

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

*grouped-is-super-grouped*[*simp*]: *grouped-by*  $c\ \varphi \implies super\text{-}grouped\text{-}by\ c\ c'\ \varphi$  |

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

**lemma** *simple-cnf*[*simp*]:

*super-grouped-by*  $c\ c'\ FT$

*super-grouped-by*  $c\ c'\ FF$

*super-grouped-by*  $c\ c'\ (FVar\ x)$

*super-grouped-by*  $c\ c'\ (FNot\ FT)$

*super-grouped-by*  $c\ c'\ (FNot\ FF)$

*super-grouped-by*  $c\ c'\ (FNot\ (FVar\ x))$

$\langle proof \rangle$

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

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

**shows** *no-equiv*  $\varphi \implies no\text{-}imp\ \varphi \implies simple\text{-}not\ \varphi \implies c\text{-in-}c'\text{-only}\ c\ c'\ \varphi$

$\implies super\text{-}grouped\text{-}by\ c\ c'\ \varphi$

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

$\langle proof \rangle$

## 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** *no-equiv*  $\varphi \implies no\text{-}imp\ \varphi \implies simple\text{-}not\ \varphi \implies or\text{-in-and-only}\ \varphi \implies is\text{-conj-with-TF}\ \varphi$

$\langle proof \rangle$

**definition** *is-cnf* **where**

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

### 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*



$(full \ (propo\text{-}rew\text{-}step \ elimTB)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ pushNeg)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ pushDisj))$

**lemma** *cnf-rew-equivalent: preserve-models cnf-rew*  
 $\langle proof \rangle$

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

### 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  $\varphi \implies no\text{-}imp \ \varphi \implies simple\text{-}not \ \varphi \implies and\text{-}in\text{-}or\text{-}only \ \varphi \implies is\text{-}disj\text{-}with\text{-}TF \ \varphi$*   
 $\langle proof \rangle$

**definition** *is-dnf :: 'a propo  $\Rightarrow$  bool where*  
*is-dnf  $\varphi \longleftrightarrow is\text{-}disj\text{-}with\text{-}TF \ \varphi \wedge no\text{-}T\text{-}F\text{-}except\text{-}top\text{-}level \ \varphi$*

#### Full DNF transform

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

**definition** *dnf-rew where dnf-rew  $\equiv$*   
 $(full \ (propo\text{-}rew\text{-}step \ elim\text{-}equiv)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ elim\text{-}imp)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ elimTB)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ pushNeg)) \ OO$   
 $(full \ (propo\text{-}rew\text{-}step \ pushConj))$

**lemma** *dnf-rew-consistent: preserve-models dnf-rew*  
 $\langle proof \rangle$

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

*dnf-rew  $\varphi \ \varphi' \implies is\text{-}dnf \ \varphi'$*   
 $\langle proof \rangle$

## 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 |*

$ElimTBFull3[simp]: elimTBFull (FOr \varphi FT) FT \mid$   
 $ElimTBFull3'[simp]: elimTBFull (FOr FT \varphi) FT \mid$   
  
 $ElimTBFull4[simp]: elimTBFull (FOr \varphi FF) \varphi \mid$   
 $ElimTBFull4'[simp]: elimTBFull (FOr FF \varphi) \varphi \mid$   
  
 $ElimTBFull5[simp]: elimTBFull (FNot FT) FF \mid$   
 $ElimTBFull5'[simp]: elimTBFull (FNot FF) FT \mid$   
  
 $ElimTBFull6-l[simp]: elimTBFull (FImp FT \varphi) \varphi \mid$   
 $ElimTBFull6-l'[simp]: elimTBFull (FImp FF \varphi) FT \mid$   
 $ElimTBFull6-r[simp]: elimTBFull (FImp \varphi FT) FT \mid$   
 $ElimTBFull6-r'[simp]: elimTBFull (FImp \varphi FF) (FNot \varphi) \mid$   
  
 $ElimTBFull7-l[simp]: elimTBFull (FEq FT \varphi) \varphi \mid$   
 $ElimTBFull7-l'[simp]: elimTBFull (FEq FF \varphi) (FNot \varphi) \mid$   
 $ElimTBFull7-r[simp]: elimTBFull (FEq \varphi FT) \varphi \mid$   
 $ElimTBFull7-r'[simp]: elimTBFull (FEq \varphi FF) (FNot \varphi) \mid$

The transformation is still consistent.

**lemma** *elimTBFull-consistent: preserve-models elimTBFull*  
 $\langle proof \rangle$

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

$\langle proof \rangle$

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**  $\exists \psi \psi'. \psi \preceq \varphi \wedge \text{elimTBFull } \psi \psi'$

$\langle proof \rangle$

**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*  $\psi$

$\langle proof \rangle$

### 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*: *propo-rew-step elim-equiv*  $\varphi \psi \implies \text{no-T-F } \varphi \implies \text{no-T-F } \psi$   
 $\langle proof \rangle$

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

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

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

### 1.7.3 The new CNF and DNF transformation

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

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

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

**lemma** *dnf-rew'-consistent: preserve-models dnf-rew'*  
 $\langle \text{proof} \rangle$

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

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

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

*(full (propo-rew-step elimTBFULL)) OO*  
*(full (propo-rew-step elim-equiv)) OO*  
*(full (propo-rew-step elim-imp)) OO*  
*(full (propo-rew-step pushNeg)) OO*  
*(full (propo-rew-step pushDisj))*

**lemma** *cnf-rew'-consistent: preserve-models cnf-rew'*  
 $\langle \text{proof} \rangle$

**theorem** *cnf'-transformation-correction*:  
*cnf-rew'  $\varphi \varphi' \implies \text{is-cnf } \varphi'$*   
 $\langle \text{proof} \rangle$

**end**

**theory** *Prop-Logic-Multiset*

**imports** *Nested-Multisets-Ordinals.Multiset-More Prop-Normalisation*

## 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*  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = \textit{FVar } v \vee \varphi = \textit{FF} \vee \varphi = \textit{FT}$ )  
 $\langle \textit{proof} \rangle$

**lemma** *grouped-by-COr-FNot*:

*grouped-by COr* (*FNot*  $\varphi$ )  $\longleftrightarrow$  ( $\exists v. \varphi = \textit{FVar } v \vee \varphi = \textit{FF} \vee \varphi = \textit{FT}$ )  
 $\langle \textit{proof} \rangle$

**lemma**

**shows** *no-T-F-FF[simp]*:  $\neg \textit{no-T-F FF}$  and  
*no-T-F-FT[simp]*:  $\neg \textit{no-T-F FT}$   
 $\langle \textit{proof} \rangle$

**lemma** *grouped-by-CAnd-FAnd*:

*grouped-by CAnd* (*FAnd*  $\varphi 1$   $\varphi 2$ )  $\longleftrightarrow$  *grouped-by CAnd*  $\varphi 1 \wedge$  *grouped-by CAnd*  $\varphi 2$   
 $\langle \textit{proof} \rangle$

**lemma** *grouped-by-COr-FOr*:

*grouped-by COr* (*FOr*  $\varphi 1$   $\varphi 2$ )  $\longleftrightarrow$  *grouped-by COr*  $\varphi 1 \wedge$  *grouped-by COr*  $\varphi 2$   
 $\langle \textit{proof} \rangle$

**lemma** *grouped-by-COr-FAnd[simp]*:  $\neg$  *grouped-by COr* (*FAnd*  $\varphi 1$   $\varphi 2$ )

$\langle \textit{proof} \rangle$

**lemma** *grouped-by-COr-FEq[simp]*:  $\neg$  *grouped-by COr* (*FEq*  $\varphi 1$   $\varphi 2$ )

$\langle \textit{proof} \rangle$

**lemma** [*simp*]:  $\neg$  *grouped-by COr* (*FImp*  $\varphi$   $\psi$ )

$\langle \textit{proof} \rangle$

**lemma** [*simp*]:  $\neg$  *is-conj-with-TF* (*FImp*  $\varphi$   $\psi$ )

$\langle \textit{proof} \rangle$

**lemma** *[simp]:  $\neg$  is-conj-with-TF (FEq  $\varphi$   $\psi$ )*  
 $\langle \text{proof} \rangle$

**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  $\varphi 2$*   
 $\langle \text{proof} \rangle$

**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  $\varphi 2$*   
 $\langle \text{proof} \rangle$

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

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

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

**theorem** *cnf-eval-true-cls:*

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

**assumes** *is-cnf  $\varphi$*

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

$\langle \text{proof} \rangle$

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

*$\langle \text{formula-of-mset } \varphi =$*

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

*else*

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

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

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

*else FOr  $v' (\text{formula-of-mset (remove1-mset } v \varphi))$ )*

$\rangle$

$\langle \text{proof} \rangle$

**termination**

$\langle \text{proof} \rangle$

**lemma** *formula-of-mset-empty[simp]:  $\langle \text{formula-of-mset } \{\#\} = FF \rangle$*

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

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

*$\langle \text{formula-of-msets } \varphi s =$*

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

*else*

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

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

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

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

$\rangle$

$\langle \text{proof} \rangle$   
**termination**  
 $\langle \text{proof} \rangle$

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

**lemma** *remove1-mset-empty-iff*:  
 $\langle \text{remove1-mset } v \ \varphi = \{\#\} \longleftrightarrow (\varphi = \{\#\} \vee \varphi = \{\#v\#\}) \rangle$   
 $\langle \text{proof} \rangle$

**definition** *fun-of-set* **where**  
 $\langle \text{fun-of-set } A \ x = (\text{if } \text{Pos } x \in A \text{ then } \text{True} \text{ else if } \text{Neg } x \in A \text{ then } \text{False} \text{ else } \text{undefined}) \rangle$

**lemma** *grouped-by-COr-formula-of-mset*:  $\langle \text{grouped-by } \text{COr } (\text{formula-of-mset } \varphi) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *no-T-F-formula-of-mset*:  $\langle \text{no-T-F } (\text{formula-of-mset } \varphi) \rangle$  **if**  $\langle \text{formula-of-mset } \varphi \neq \text{FF} \rangle$  **for**  $\varphi$   
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *eval-cls-iff*:  
**assumes**  $\langle \text{consistent-interp } A \rangle$  **and**  $\langle \text{total-over-set } A \ \text{UNIV} \rangle$   
**shows**  $\langle \text{eval } (\text{fun-of-set } A) (\text{formula-of-mset } \varphi) \longleftrightarrow \text{Partial-Herbrand-Interpretation.true-cls } A \ \{\varphi\} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *is-conj-with-TF-Fand-iff*:  
 $\langle \text{is-conj-with-TF } (\text{FAnd } \varphi_1 \ \varphi_2) \longleftrightarrow \text{is-conj-with-TF } \varphi_1 \wedge \text{is-conj-with-TF } \varphi_2 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *is-CNF-Fand*:  
 $\langle \text{is-cnf } (\text{FAnd } \varphi \ \psi) \longleftrightarrow (\text{is-cnf } \varphi \wedge \text{no-T-F } \varphi) \wedge \text{is-cnf } \psi \wedge \text{no-T-F } \psi \rangle$   
 $\langle \text{proof} \rangle$

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

**lemma** *no-T-F-formula-of-msets*:  
**assumes**  $\langle \text{finite } \varphi \rangle$  **and**  $\langle \{\#\} \notin \varphi \rangle$  **and**  $\langle \varphi \neq \{\} \rangle$   
**shows**  $\langle \text{no-T-F } (\text{formula-of-msets } (\varphi)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *is-cnf-formula-of-msets*:  
**assumes**  $\langle \text{finite } \varphi \rangle$  **and**  $\langle \{\#\} \notin \varphi \rangle$   
**shows**  $\langle \text{is-cnf } (\text{formula-of-msets } \varphi) \rangle$   
 $\langle \text{proof} \rangle$

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

⟨*proof*⟩

**lemma**

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

**shows** ⟨*eval* (*fun-of-set* *A*) (*formula-of-msets*  $\varphi$ )  $\longleftrightarrow$  *Partial-Herbrand-Interpretation.true-cls* *A*  $\varphi$ ⟩

⟨*proof*⟩

**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*:

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

*condensation*:

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

*subsumption*:

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

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

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

**assumes** *simplify N N'*

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

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

*<proof>*

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

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

**assumes** *simplify N N'*

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

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

*<proof>*

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

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

**assumes** *simplify N N'*

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

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

*<proof>*

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

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

**assumes** *simplify N N'*

**and** *total-over-m*  $I\ N$   
**shows**  $I \models_s N \longleftrightarrow I \models_s N'$   
 $\langle \text{proof} \rangle$

**lemma** *simplify-preserves-finite*:  
**assumes** *simplify*  $\psi\ \psi'$   
**shows** *finite*  $\psi \longleftrightarrow \text{finite } \psi'$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-simplify-preserves-finite*:  
**assumes** *rtranclp simplify*  $\psi\ \psi'$   
**shows** *finite*  $\psi \longleftrightarrow \text{finite } \psi'$   
 $\langle \text{proof} \rangle$

**lemma** *simplify-atms-of-ms*:  
**assumes** *simplify*  $\psi\ \psi'$   
**shows** *atms-of-ms*  $\psi' \subseteq \text{atms-of-ms } \psi$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-simplify-atms-of-ms*:  
**assumes** *rtranclp simplify*  $\psi\ \psi'$   
**shows** *atms-of-ms*  $\psi' \subseteq \text{atms-of-ms } \psi$   
 $\langle \text{proof} \rangle$

**lemma** *factoring-imp-simplify*:  
**assumes**  $\{\#L, L\# \} + C \in N$   
**shows**  $\exists N'. \text{simplify } N\ N'$   
 $\langle \text{proof} \rangle$

## 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*:

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

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

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

**lemma** *uncon-res-increasing*:  
**assumes** *uncon-res*  $S\ S'$  **and**  $\psi \in S$   
**shows**  $\psi \in S'$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-uncon-inference-increasing*:  
**assumes** *rtranclp uncon-res*  $S\ S'$  **and**  $\psi \in S$   
**shows**  $\psi \in S'$   
 $\langle \text{proof} \rangle$

## Subsumption

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

*subsumes*  $\chi\ \chi' \longleftrightarrow$

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

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

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

*subsumes*  $\chi$   $\chi$   
 $\langle \text{proof} \rangle$

**lemma** *subsumes-subsumption*:

**assumes** *subsumes*  $D$   $\chi$   
**and**  $C \subset\# D$  **and**  $\neg \text{tautology } \chi$   
**shows** *subsumes*  $C$   $\chi$   $\langle \text{proof} \rangle$

**lemma** *subsumes-tautology*:

**assumes** *subsumes* (*add-mset* ( $\text{Pos } P$ ) (*add-mset* ( $\text{Neg } P$ )  $C$ ))  $\chi$   
**shows** *tautology*  $\chi$   
 $\langle \text{proof} \rangle$

### 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*:

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

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

*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*)

$\implies \text{inference } S$  (*fst*  $S \cup \{\text{clause}\}$ , *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*  $\equiv$

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

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

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

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

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

$\langle \text{proof} \rangle$

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

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

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

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

$\langle \text{proof} \rangle$

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

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

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

**shows** *already-used-inv*  $S'$   
 $\langle \text{proof} \rangle$

**lemma** *subsumes-condensation*:  
**assumes** *subsumes*  $(C + \{\#L\# \} + \{\#L\# \}) D$   
**shows** *subsumes*  $(C + \{\#L\# \}) D$   
 $\langle \text{proof} \rangle$

**lemma** *simplify-preserved-already-used-inv*:  
**assumes** *simplify*  $N N'$   
**and** *already-used-inv*  $(N, \text{already-used})$   
**shows** *already-used-inv*  $(N', \text{already-used})$   
 $\langle \text{proof} \rangle$

**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*:  $\text{atms-of } (\text{add-mset } L (\text{add-mset } L C)) = \text{atms-of } (\text{add-mset } L C)$   
 $\langle \text{proof} \rangle$

**lemma** *inference-increasing*:  
**assumes** *inference*  $S S'$  **and**  $\psi \in \text{fst } S$   
**shows**  $\psi \in \text{fst } S'$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-inference-increasing*:  
**assumes** *rtranclp inference*  $S S'$  **and**  $\psi \in \text{fst } S$   
**shows**  $\psi \in \text{fst } S'$   
 $\langle \text{proof} \rangle$

**lemma** *inference-clause-already-used-increasing*:  
**assumes** *inference-clause*  $S S'$   
**shows**  $\text{snd } S \subseteq \text{snd } S'$   
 $\langle \text{proof} \rangle$

**lemma** *inference-already-used-increasing*:  
**assumes** *inference*  $S S'$   
**shows**  $\text{snd } S \subseteq \text{snd } S'$   
 $\langle \text{proof} \rangle$

**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**  $I \models_s \text{fst } T \longleftrightarrow I \models_s \text{fst } T \cup \{\text{fst } T'\}$   
 $\langle \text{proof} \rangle$

**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**  $I \models_s \text{fst } T \longleftrightarrow I \models_s \text{fst } T'$   
 $\langle \text{proof} \rangle$

**lemma** *inference-clause-preserves-atms-of-ms*:  
**assumes** *inference-clause*  $S S'$   
**shows**  $\text{atms-of-ms } (\text{fst } (S \cup \{\text{fst } S'\}, \text{snd } S')) \subseteq \text{atms-of-ms } (\text{fst } S)$   
 $\langle \text{proof} \rangle$

**lemma** *inference-preserves-atms-of-ms*:  
**fixes**  $N N' :: 'v \text{ clause-set}$   
**assumes** *inference*  $T T'$   
**shows**  $\text{atms-of-ms } (\text{fst } T') \subseteq \text{atms-of-ms } (\text{fst } T)$   
 $\langle \text{proof} \rangle$

**lemma** *inference-preserves-total*:  
**fixes**  $N N' :: 'v \text{ clause-set}$   
**assumes** *inference*  $(N, \text{already-used}) (N', \text{already-used}')$   
**shows**  $\text{total-over-m } I N \implies \text{total-over-m } I N'$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-inference-preserves-total*:  
**assumes** *rtranclp inference*  $T T'$   
**shows**  $\text{total-over-m } I (\text{fst } T) \implies \text{total-over-m } I (\text{fst } T')$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-inference-preserve-models*:  
**assumes** *rtranclp inference*  $N N'$   
**and**  $\text{total-over-m } I (\text{fst } N)$   
**and** *consistent*: *consistent-interp*  $I$   
**shows**  $I \models_s \text{fst } N \longleftrightarrow I \models_s \text{fst } N'$   
 $\langle \text{proof} \rangle$

**lemma** *inference-preserves-finite*:  
**assumes** *inference*  $\psi \psi'$  **and** *finite*  $(\text{fst } \psi)$   
**shows** *finite*  $(\text{fst } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *inference-clause-preserves-finite-snd*:  
**assumes** *inference-clause*  $\psi \psi'$  **and** *finite*  $(\text{snd } \psi)$   
**shows** *finite*  $(\text{snd } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *inference-preserves-finite-snd*:  
**assumes** *inference*  $\psi \psi'$  **and** *finite*  $(\text{snd } \psi)$   
**shows** *finite*  $(\text{snd } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *rtranclp-inference-preserves-finite*:  
**assumes** *rtranclp inference*  $\psi \psi'$  **and** *finite*  $(\text{fst } \psi)$   
**shows** *finite*  $(\text{fst } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *consistent-interp-insert*:  
**assumes** *consistent-interp I*  
**and** *atm-of P*  $\notin$  *atm-of 'I*  
**shows** *consistent-interp (insert P I)*  
 $\langle$ *proof* $\rangle$

**lemma** *simplify-clause-preserves-sat*:  
**assumes** *simp: simplify  $\psi$   $\psi'$*   
**and** *satisfiable  $\psi'$*   
**shows** *satisfiable  $\psi$*   
 $\langle$ *proof* $\rangle$

**lemma** *simplify-preserves-unsat*:  
**assumes** *inference  $\psi$   $\psi'$*   
**shows** *satisfiable (fst  $\psi'$ )  $\longrightarrow$  satisfiable (fst  $\psi$ )*  
 $\langle$ *proof* $\rangle$

**lemma** *inference-preserves-unsat*:  
**assumes** *inference\*\* S S'*  
**shows** *satisfiable (fst S')  $\longrightarrow$  satisfiable (fst S)*  
 $\langle$ *proof* $\rangle$

**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]*:  
 $(\bigwedge xs:: 'v \text{ sem-tree. } (\bigwedge ys:: 'v \text{ sem-tree. } \text{sem-tree-size } ys < \text{sem-tree-size } xs \implies P \text{ } ys) \implies P \text{ } xs)$   
 $\implies P \text{ } xs$   
 $\langle$ *proof* $\rangle$

**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  $\psi$*   $\longleftrightarrow$   
 $(\text{partial-interps } ag (I \cup \{ \text{Pos } v \}) \psi \wedge \text{partial-interps } ad (I \cup \{ \text{Neg } v \}) \psi)$

**lemma** *simplify-preserve-partial-leaf*:  
*simplify N N'  $\implies$  partial-interps Leaf I N  $\implies$  partial-interps Leaf I N'*  
 $\langle$ *proof* $\rangle$

**lemma** *simplify-preserve-partial-tree*:  
**assumes** *simplify N N'*  
**and** *partial-interps t I N*  
**shows** *partial-interps t I N'*  
 $\langle$ *proof* $\rangle$

**lemma** *inference-preserve-partial-tree*:  
**assumes** *inference S S'*  
**and** *partial-interps t I (fst S)*  
**shows** *partial-interps t I (fst S')*

$\langle \text{proof} \rangle$

**lemma** *rtranclp-inference-preserve-partial-tree*:

**assumes** *rtranclp inference*  $N N'$

**and** *partial-interps*  $t I$  (*fst*  $N$ )

**shows** *partial-interps*  $t I$  (*fst*  $N'$ )

$\langle \text{proof} \rangle$

**function** *build-sem-tree* ::  $'v :: \text{linorder set} \Rightarrow 'v \text{ clause-set} \Rightarrow 'v \text{ sem-tree}$  **where**

*build-sem-tree*  $\text{atms } \psi =$

(if  $\text{atms} = \{\}$   $\vee \neg \text{finite atms}$

then *Leaf*

else *Node* (*Min*  $\text{atms}$ ) (*build-sem-tree* (*Set.remove* (*Min*  $\text{atms}$ )  $\text{atms}$ )  $\psi$ )

(*build-sem-tree* (*Set.remove* (*Min*  $\text{atms}$ )  $\text{atms}$ )  $\psi$ ))

$\langle \text{proof} \rangle$

**termination**

$\langle \text{proof} \rangle$

**declare** *build-sem-tree.induct*[*case-names tree*]

**lemma** *unsatisfiable-empty[simp]*:

$\neg \text{unsatisfiable } \{\}$

$\langle \text{proof} \rangle$

**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*  $\text{atms}$

**and**  $\text{atms-of-ms } \psi = \text{atms} \cup \text{atms-of-s } I$  **and**  $\text{atms} \cap \text{atms-of-s } I = \{\}$

**shows** *partial-interps* (*build-sem-tree*  $\text{atms } \psi$ )  $I \psi$

$\langle \text{proof} \rangle$

**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*  $\psi$ )  $\psi$ )  $\{\} \psi$

$\langle \text{proof} \rangle$

**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**  $\exists \psi' \chi'. \text{inference}^{**} \psi \psi' \wedge \chi' \in \text{fst } \psi' \wedge (\forall L. L \in \# \chi \longleftrightarrow L \in \# \chi')$

$\wedge \text{count } \chi' L = 1$

$\wedge (\forall \varphi. \varphi \in \text{fst } \psi \longrightarrow \varphi \in \text{fst } \psi')$

$\wedge (I \models \chi \longleftrightarrow I \models \chi')$

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

$\langle \text{proof} \rangle$

**lemma** *can-decrease-tree-size*:

**fixes**  $\psi :: 'v \text{ state}$  **and**  $\text{tree} :: 'v \text{ sem-tree}$

**assumes** *finite* (*fst*  $\psi$ ) **and** *already-used-inv*  $\psi$

**and** *partial-interps*  $\text{tree } I$  (*fst*  $\psi$ )

**shows**  $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{inference}^{**} \psi \psi' \wedge \text{partial-interps } \text{tree}' I (\text{fst } \psi')$

$\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *inference-completeness-inv*:

**fixes**  $\psi :: 'v :: \text{linorder state}$

**assumes**

*unsat*:  $\neg \text{satisfiable (fst } \psi)$  **and**

*finite*:  $\text{finite (fst } \psi)$  **and**

*a-u-v*: *already-used-inv*  $\psi$

**shows**  $\exists \psi'. (\text{inference}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$

$\langle \text{proof} \rangle$

**lemma** *inference-completeness*:

**fixes**  $\psi :: 'v :: \text{linorder state}$

**assumes** *unsat*:  $\neg \text{satisfiable (fst } \psi)$

**and** *finite*:  $\text{finite (fst } \psi)$

**and** *snd*  $\psi = \{\}$

**shows**  $\exists \psi'. (\text{rtranclp inference } \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$

$\langle \text{proof} \rangle$

**lemma** *inference-soundness*:

**fixes**  $\psi :: 'v :: \text{linorder state}$

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

**shows**  $\neg \text{satisfiable (fst } \psi)$

$\langle \text{proof} \rangle$

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

**fixes**  $\psi :: 'v :: \text{linorder state}$

**assumes** *finite*:  $\text{finite (fst } \psi)$

**and** *snd*  $\psi = \{\}$

**shows**  $(\exists \psi'. (\text{inference}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \neg \text{satisfiable (fst } \psi)$

$\langle \text{proof} \rangle$

## 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**  $\text{count } \chi L \leq 1$

$\langle \text{proof} \rangle$

**lemma** *simplified-no-both*:

**assumes** *simp*: *simplified*  $\psi$  **and**  $\chi: \chi \in \psi$

**shows**  $\neg (L \in \# \chi \wedge \neg L \in \# \chi)$

$\langle \text{proof} \rangle$

**lemma** *add-mset-Neg-Pos-commute[simp]*:

$\text{add-mset (Neg } P) (\text{add-mset (Pos } P) C) = \text{add-mset (Pos } P) (\text{add-mset (Neg } P) C)$

$\langle \text{proof} \rangle$

**lemma** *simplified-not-tautology*:

**assumes** *simplified*  $\{\psi\}$

**shows**  $\sim \text{tautology } \psi$

$\langle \text{proof} \rangle$



**lemma** *simplified-remove*:  
**assumes** *simplified*  $\{\psi\}$   
**shows** *simplified*  $\{\psi - \{\#l\#\}\}$   
 $\langle proof \rangle$

**lemma** *in-simplified-simplified*:  
**assumes** *simp*: *simplified*  $\psi$  **and** *incl*:  $\psi' \subseteq \psi$   
**shows** *simplified*  $\psi'$   
 $\langle proof \rangle$

**lemma** *simplified-in*:  
**assumes** *simplified*  $\psi$   
**and**  $N \in \psi$   
**shows** *simplified*  $\{N\}$   
 $\langle proof \rangle$

**lemma** *subsumes-imp-formula*:  
**assumes**  $\psi \leq \# \varphi$   
**shows**  $\{\psi\} \models_p \varphi$   
 $\langle proof \rangle$

**lemma** *simplified-imp-distinct-mset-tauto*:  
**assumes** *simp*: *simplified*  $\psi'$   
**shows** *distinct-mset-set*  $\psi'$  **and**  $\forall \chi \in \psi'. \neg \text{tautology } \chi$   
 $\langle proof \rangle$

**lemma** *simplified-no-more-full1-simplified*:  
**assumes** *simplified*  $\psi$   
**shows**  $\neg \text{full1 simplify } \psi \psi'$   
 $\langle proof \rangle$

## 2.1.5 Resolution and Invariants

**inductive** *resolution* ::  $'v \text{ state} \Rightarrow 'v \text{ state} \Rightarrow \text{bool}$  **where**  
*full1-simp*: *full1 simplify*  $N N' \Longrightarrow \text{resolution } (N, \text{already-used}) (N', \text{already-used})$  |  
*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*  $(fst \psi')$   
 $\langle proof \rangle$

**lemma** *rtrancp-resolution-finite*:  
**assumes** *resolution*\*\*  $\psi \psi'$  **and** *finite*  $(fst \psi)$   
**shows** *finite*  $(fst \psi')$   
 $\langle proof \rangle$

**lemma** *resolution-finite-snd*:  
**assumes** *resolution*  $\psi \psi'$  **and** *finite*  $(snd \psi)$   
**shows** *finite*  $(snd \psi')$   
 $\langle proof \rangle$

**lemma** *rtrancp-resolution-finite-snd*:  
**assumes** *resolution\*\**  $\psi \ \psi'$  **and** *finite* (*snd*  $\psi$ )  
**shows** *finite* (*snd*  $\psi'$ )  
 $\langle$ *proof* $\rangle$

**lemma** *resolution-always-simplified*:  
**assumes** *resolution*  $\psi \ \psi'$   
**shows** *simplified* (*fst*  $\psi'$ )  
 $\langle$ *proof* $\rangle$

**lemma** *trancp-resolution-always-simplified*:  
**assumes** *trancp resolution*  $\psi \ \psi'$   
**shows** *simplified* (*fst*  $\psi'$ )  
 $\langle$ *proof* $\rangle$

**lemma** *resolution-atms-of*:  
**assumes** *resolution*  $\psi \ \psi'$  **and** *finite* (*fst*  $\psi$ )  
**shows** *atms-of-ms* (*fst*  $\psi'$ )  $\subseteq$  *atms-of-ms* (*fst*  $\psi$ )  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancp-resolution-atms-of*:  
**assumes** *resolution\*\**  $\psi \ \psi'$  **and** *finite* (*fst*  $\psi$ )  
**shows** *atms-of-ms* (*fst*  $\psi'$ )  $\subseteq$  *atms-of-ms* (*fst*  $\psi$ )  
 $\langle$ *proof* $\rangle$

**lemma** *resolution-include*:  
**assumes** *res: resolution*  $\psi \ \psi'$  **and** *finite: finite* (*fst*  $\psi$ )  
**shows** *fst*  $\psi' \subseteq$  *simple-clss* (*atms-of-ms* (*fst*  $\psi$ ))  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancp-resolution-include*:  
**assumes** *res: trancp resolution*  $\psi \ \psi'$  **and** *finite: finite* (*fst*  $\psi$ )  
**shows** *fst*  $\psi' \subseteq$  *simple-clss* (*atms-of-ms* (*fst*  $\psi$ ))  
 $\langle$ *proof* $\rangle$

**abbreviation** *already-used-all-simple*  
 $:: ('a \text{ literal multiset} \times 'a \text{ literal multiset}) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$  **where**  
*already-used-all-simple* *already-used* *vars*  $\equiv$   
 $(\forall (A, B) \in \text{already-used}. \text{simplified } \{A\} \wedge \text{simplified } \{B\} \wedge \text{atms-of } A \subseteq \text{vars} \wedge \text{atms-of } B \subseteq \text{vars})$

**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'*  
 $\langle$ *proof* $\rangle$

**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*  $\cup$   $\{\text{fst } S'\}$ , *snd* *S'*)) *vars*  
 $\langle$ *proof* $\rangle$

**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*  
 $\langle$ *proof* $\rangle$

**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*  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancpl-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*  
 $\langle$ *proof* $\rangle$

**lemma** *inference-clause-simplified-already-used-subset*:  
**assumes** *inference-clause* *S S'*  
**and** *simplified* (*fst S*)  
**shows** *snd S*  $\subset$  *snd S'*  
 $\langle$ *proof* $\rangle$

**lemma** *inference-simplified-already-used-subset*:  
**assumes** *inference* *S S'*  
**and** *simplified* (*fst S*)  
**shows** *snd S*  $\subset$  *snd S'*  
 $\langle$ *proof* $\rangle$

**lemma** *resolution-simplified-already-used-subset*:  
**assumes** *resolution* *S S'*  
**and** *simplified* (*fst S*)  
**shows** *snd S*  $\subset$  *snd S'*  
 $\langle$ *proof* $\rangle$

**lemma** *trancpl-resolution-simplified-already-used-subset*:  
**assumes** *trancpl resolution* *S S'*  
**and** *simplified* (*fst S*)  
**shows** *snd S*  $\subset$  *snd S'*  
 $\langle$ *proof* $\rangle$

**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*  
 $\langle$ *proof* $\rangle$

**lemma** *already-used-top-finite*:  
**assumes** *finite vars*  
**shows** *finite* (*already-used-top vars*)  
 $\langle$ *proof* $\rangle$

**lemma** *already-used-top-increasing*:  
**assumes**  $var \subseteq var'$  **and** *finite*  $var'$   
**shows** *already-used-top*  $var \subseteq \text{already-used-top } var'$   
 $\langle \text{proof} \rangle$

**lemma** *already-used-all-simple-finite*:  
**fixes**  $s :: ('a \text{ literal multiset} \times 'a \text{ literal multiset}) \text{ set}$  **and**  $vars :: 'a \text{ set}$   
**assumes** *already-used-all-simple*  $s \text{ vars}$  **and** *finite*  $vars$   
**shows** *finite*  $s$   
 $\langle \text{proof} \rangle$

**abbreviation** *card-simple vars*  $\psi \equiv \text{card } (\text{already-used-top } vars - \psi)$

**lemma** *resolution-card-simple-decreasing*:  
**assumes** *res*: *resolution*  $\psi \psi'$   
**and** *a-u-s*: *already-used-all-simple*  $(\text{snd } \psi) \text{ vars}$   
**and** *finite-v*: *finite*  $vars$   
**and** *finite-fst*: *finite*  $(\text{fst } \psi)$   
**and** *finite-snd*: *finite*  $(\text{snd } \psi)$   
**and** *simp*: *simplified*  $(\text{fst } \psi)$   
**and** *atms-of-ms*  $(\text{fst } \psi) \subseteq vars$   
**shows** *card-simple vars*  $(\text{snd } \psi') < \text{card-simple vars } (\text{snd } \psi)$   
 $\langle \text{proof} \rangle$

**lemma** *trancp-resolution-card-simple-decreasing*:  
**assumes** *trancp resolution*  $\psi \psi'$  **and** *finite-fst*: *finite*  $(\text{fst } \psi)$   
**and** *already-used-all-simple*  $(\text{snd } \psi) \text{ vars}$   
**and** *atms-of-ms*  $(\text{fst } \psi) \subseteq vars$   
**and** *finite-v*: *finite*  $vars$   
**and** *finite-snd*: *finite*  $(\text{snd } \psi)$   
**and** *simplified*  $(\text{fst } \psi)$   
**shows** *card-simple vars*  $(\text{snd } \psi') < \text{card-simple vars } (\text{snd } \psi)$   
 $\langle \text{proof} \rangle$

**lemma** *trancp-resolution-card-simple-decreasing-2*:  
**assumes** *trancp resolution*  $\psi \psi'$   
**and** *finite-fst*: *finite*  $(\text{fst } \psi)$   
**and** *empty-snd*:  $\text{snd } \psi = \{\}$   
**and** *simplified*  $(\text{fst } \psi)$   
**shows** *card-simple*  $(\text{atms-of-ms } (\text{fst } \psi)) (\text{snd } \psi') < \text{card-simple } (\text{atms-of-ms } (\text{fst } \psi)) (\text{snd } \psi)$   
 $\langle \text{proof} \rangle$

## Well-Foundness of the Relation

**lemma** *wf-simplified-resolution*:  
**assumes** *f-vars*: *finite*  $vars$   
**shows** *wf*  $\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq vars \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$   
 $\langle \text{proof} \rangle$

**lemma** *wf-simplified-resolution'*:  
**assumes** *f-vars*: *finite*  $vars$   
**shows** *wf*  $\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq vars \wedge \neg \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\}$

$\langle \text{proof} \rangle$

**lemma** *wf-resolution*:

**assumes** *f-vars*: *finite vars*

**shows**  $\text{wf } (\{(y:: 'v:: \text{linorder state}, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\} \cup \{(y, x). (\text{atms-of-ms } (\text{fst } x) \subseteq \text{vars} \wedge \neg \text{simplified } (\text{fst } x) \wedge \text{finite } (\text{snd } x) \wedge \text{finite } (\text{fst } x) \wedge \text{already-used-all-simple } (\text{snd } x) \text{ vars}) \wedge \text{resolution } x y\})$  **(is**  $\text{wf } (?R \cup ?S))$

$\langle \text{proof} \rangle$

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

**assumes** *simplify\*\** *S S'*

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

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

$\langle \text{proof} \rangle$

**lemma** *full1-simplify-already-used-inv*:

**assumes** *full1 simplify* *S S'*

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

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

$\langle \text{proof} \rangle$

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

**assumes** *full simplify* *S S'*

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

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

$\langle \text{proof} \rangle$

**lemma** *resolution-already-used-inv*:

**assumes** *resolution* *S S'*

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

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

$\langle \text{proof} \rangle$

**lemma** *rtranclp-resolution-already-used-inv*:

**assumes** *resolution\*\** *S S'*

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

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

$\langle \text{proof} \rangle$

**lemma** *rtanclp-simplify-preserves-unsat*:

**assumes** *simplify\*\**  $\psi \psi'$

**shows** *satisfiable*  $\psi' \longrightarrow \text{satisfiable } \psi$

$\langle \text{proof} \rangle$

**lemma** *full1-simplify-preserves-unsat*:

**assumes** *full1 simplify*  $\psi \psi'$

**shows** *satisfiable*  $\psi' \longrightarrow \text{satisfiable } \psi$

$\langle \text{proof} \rangle$

**lemma** *full-simplify-preserves-unsat*:

**assumes** *full simplify*  $\psi \psi'$

**shows** *satisfiable*  $\psi' \longrightarrow \text{satisfiable } \psi$

$\langle \text{proof} \rangle$

**lemma** *resolution-preserves-unsat*:

**assumes** *resolution*  $\psi \psi'$

**shows** *satisfiable* (*fst*  $\psi'$ )  $\longrightarrow$  *satisfiable* (*fst*  $\psi$ )  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancp-resolution-preserves-unsat*:  
**assumes** *resolution*\*\*  $\psi$   $\psi'$   
**shows** *satisfiable* (*fst*  $\psi'$ )  $\longrightarrow$  *satisfiable* (*fst*  $\psi$ )  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancp-simplify-preserve-partial-tree*:  
**assumes** *simplify*\*\*  $N$   $N'$   
**and** *partial-interps*  $t$   $I$   $N$   
**shows** *partial-interps*  $t$   $I$   $N'$   
 $\langle$ *proof* $\rangle$

**lemma** *full1-simplify-preserve-partial-tree*:  
**assumes** *full1 simplify*  $N$   $N'$   
**and** *partial-interps*  $t$   $I$   $N$   
**shows** *partial-interps*  $t$   $I$   $N'$   
 $\langle$ *proof* $\rangle$

**lemma** *full-simplify-preserve-partial-tree*:  
**assumes** *full simplify*  $N$   $N'$   
**and** *partial-interps*  $t$   $I$   $N$   
**shows** *partial-interps*  $t$   $I$   $N'$   
 $\langle$ *proof* $\rangle$

**lemma** *resolution-preserve-partial-tree*:  
**assumes** *resolution*  $S$   $S'$   
**and** *partial-interps*  $t$   $I$  (*fst*  $S$ )  
**shows** *partial-interps*  $t$   $I$  (*fst*  $S'$ )  
 $\langle$ *proof* $\rangle$

**lemma** *rtrancp-resolution-preserve-partial-tree*:  
**assumes** *resolution*\*\*  $S$   $S'$   
**and** *partial-interps*  $t$   $I$  (*fst*  $S$ )  
**shows** *partial-interps*  $t$   $I$  (*fst*  $S'$ )  
 $\langle$ *proof* $\rangle$   
**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**  $P n$   
 $\langle$ *proof* $\rangle$

**lemma** *wf-always-more-step-False*:  
**assumes** *wf*  $R$   
**shows**  $(\forall x. \exists z. (z, x) \in R) \implies \text{False}$   
 $\langle$ *proof* $\rangle$

**lemma** *finite-finite-mset-element-of-mset[simp]*:  
**assumes** *finite*  $N$   
**shows** *finite*  $\{f \varphi L \mid \varphi L. \varphi \in N \wedge L \in \# \varphi \wedge P \varphi L\}$   
 $\langle$ *proof* $\rangle$

**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 = *sum-count-ge-2*

$\langle \text{proof} \rangle$

**lemma** *finite-incl-le-setsum*:

*finite* ( $B :: 'a \text{ multiset set}$ )  $\Longrightarrow A \subseteq B \Longrightarrow \Xi A \leq \Xi B$

$\langle \text{proof} \rangle$

**lemma** *simplify-finite-measure-decrease*:

*simplify*  $N N' \Longrightarrow \text{finite } N \Longrightarrow \text{card } N' + \Xi N' < \text{card } N + \Xi N$

$\langle \text{proof} \rangle$

**lemma** *simplify-terminates*:

*wf*  $\{(N', N). \text{finite } N \wedge \text{simplify } N N'\}$

$\langle \text{proof} \rangle$

**lemma** *wf-terminates*:

**assumes** *wf*  $r$

**shows**  $\exists N'. (N', N) \in r^* \wedge (\forall N''. (N'', N') \notin r)$

$\langle \text{proof} \rangle$

**lemma** *rtranclp-simplify-terminates*:

**assumes** *fin*: *finite*  $N$

**shows**  $\exists N'. \text{simplify}^{**} N N' \wedge \text{simplified } N'$

$\langle \text{proof} \rangle$

**lemma** *finite-simplified-full1-simp*:

**assumes** *finite*  $N$

**shows**  $\text{simplified } N \vee (\exists N'. \text{full1 simplify } N N')$

$\langle \text{proof} \rangle$

**lemma** *finite-simplified-full-simp*:

**assumes** *finite*  $N$

**shows**  $\exists N'. \text{full simplify } N N'$

$\langle \text{proof} \rangle$

**lemma** *can-decrease-tree-size-resolution*:

**fixes**  $\psi :: 'v \text{ state}$  **and** *tree* :: 'v *sem-tree*

**assumes** *finite* (*fst*  $\psi$ ) **and** *already-used-inv*  $\psi$

**and** *partial-interps tree*  $I$  (*fst*  $\psi$ )

**and** *simplified* (*fst*  $\psi$ )

**shows**  $\exists (\text{tree}' :: 'v \text{ sem-tree}) \psi'. \text{resolution}^{**} \psi \psi' \wedge \text{partial-interps tree}' I (\text{fst } \psi')$

$\wedge (\text{sem-tree-size tree}' < \text{sem-tree-size tree} \vee \text{sem-tree-size tree} = 0)$

$\langle \text{proof} \rangle$

**lemma** *resolution-completeness-inv*:

**fixes**  $\psi :: 'v :: \text{linorder state}$

**assumes**

*unsat*:  $\neg \text{satisfiable}$  (*fst*  $\psi$ ) **and**

*finite*: *finite* (*fst*  $\psi$ ) **and**  
*a-u-v*: *already-used-inv*  $\psi$   
**shows**  $\exists \psi'. (\text{resolution}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *resolution-preserves-already-used-inv*:  
**assumes** *resolution*  $S S'$   
**and** *already-used-inv*  $S$   
**shows** *already-used-inv*  $S'$   
 $\langle \text{proof} \rangle$

**lemma** *rtrancp-resolution-preserves-already-used-inv*:  
**assumes** *resolution*<sup>\*\*</sup>  $S S'$   
**and** *already-used-inv*  $S$   
**shows** *already-used-inv*  $S'$   
 $\langle \text{proof} \rangle$

**lemma** *resolution-completeness*:  
**fixes**  $\psi :: 'v :: \text{linorder state}$   
**assumes** *unsat*:  $\neg \text{satisfiable } (\text{fst } \psi)$   
**and** *finite*: *finite* (*fst*  $\psi$ )  
**and** *snd*  $\psi = \{\}$   
**shows**  $\exists \psi'. (\text{resolution}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')$   
 $\langle \text{proof} \rangle$

**lemma** *rtrancp-preserves-sat*:  
**assumes** *simplify*<sup>\*\*</sup>  $S S'$   
**and** *satisfiable*  $S$   
**shows** *satisfiable*  $S'$   
 $\langle \text{proof} \rangle$

**lemma** *resolution-preserves-sat*:  
**assumes** *resolution*  $S S'$   
**and** *satisfiable* (*fst*  $S$ )  
**shows** *satisfiable* (*fst*  $S'$ )  
 $\langle \text{proof} \rangle$

**lemma** *rtrancp-resolution-preserves-sat*:  
**assumes** *resolution*<sup>\*\*</sup>  $S S'$   
**and** *satisfiable* (*fst*  $S$ )  
**shows** *satisfiable* (*fst*  $S'$ )  
 $\langle \text{proof} \rangle$

**lemma** *resolution-soundness*:  
**fixes**  $\psi :: 'v :: \text{linorder state}$   
**assumes** *resolution*<sup>\*\*</sup>  $\psi \psi'$  **and**  $\{\#\} \in \text{fst } \psi'$   
**shows** *unsatisfiable* (*fst*  $\psi$ )  
 $\langle \text{proof} \rangle$

**lemma** *resolution-soundness-and-completeness*:  
**fixes**  $\psi :: 'v :: \text{linorder state}$   
**assumes** *finite*: *finite* (*fst*  $\psi$ )  
**and** *snd*: *snd*  $\psi = \{\}$   
**shows**  $(\exists \psi'. (\text{resolution}^{**} \psi \psi' \wedge \{\#\} \in \text{fst } \psi')) \longleftrightarrow \text{unsatisfiable } (\text{fst } \psi)$   
 $\langle \text{proof} \rangle$



**lemma** *simplified-falsity*:

**assumes** *simp*: *simplified*  $\psi$   
**and**  $\{\#\} \in \psi$   
**shows**  $\psi = \{\{\#\}\}$

$\langle proof \rangle$

**lemma** *simplify-falsity-in-preserved*:

**assumes** *simplify*  $\chi s \chi s'$   
**and**  $\{\#\} \in \chi s$   
**shows**  $\{\#\} \in \chi s'$

$\langle proof \rangle$

**lemma** *rtranclp-simplify-falsity-in-preserved*:

**assumes** *simplify*<sup>\*\*</sup>  $\chi s \chi s'$   
**and**  $\{\#\} \in \chi s$   
**shows**  $\{\#\} \in \chi s'$

$\langle proof \rangle$

**lemma** *resolution-falsity-get-falsity-alone*:

**assumes** *finite* (*fst*  $\psi$ )  
**shows**  $(\exists \psi'. (resolution^{**} \psi \psi' \wedge \{\#\} \in fst \psi')) \longleftrightarrow (\exists a-u-v. resolution^{**} \psi (\{\{\#\}\}, a-u-v))$   
**(is**  $?A \longleftrightarrow ?B$ )

$\langle proof \rangle$

**theorem** *resolution-soundness-and-completeness'*:

**fixes**  $\psi :: 'v :: linorder \text{ state}$   
**assumes**  
*finite*: *finite* (*fst*  $\psi$ ) **and**  
*snd*: *snd*  $\psi = \{\}$   
**shows**  $(\exists a-u-v. (resolution^{**} \psi (\{\{\#\}\}, a-u-v))) \longleftrightarrow unsatisfiable (fst \psi)$

$\langle proof \rangle$

**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*:

$S \models_h C \longleftrightarrow \{Pos \ P | P. P \in S\} \cup \{Neg \ P | P. P \notin S\} \models C$

$\langle proof \rangle$

**lemma** *herbrand-consistent-interp*:

*consistent-interp*  $(\{Pos \ P | P. P \in S\} \cup \{Neg \ P | P. P \notin S\})$

$\langle proof \rangle$

**lemma** *herbrand-total-over-set*:

*total-over-set* ( $\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$ )  $T$   
 $\langle \text{proof} \rangle$

**lemma** *herbrand-total-over-m*:

*total-over-m* ( $\{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\}$ )  $T$   
 $\langle \text{proof} \rangle$

**lemma** *herbrand-interp-iff-partial-interp-clss*:

$S \models_{hs} C \longleftrightarrow \{\text{Pos } P \mid P. P \in S\} \cup \{\text{Neg } P \mid P. P \notin S\} \models_s C$   
 $\langle \text{proof} \rangle$

**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* ( $-\hat{<}^{bsup}>-\hat{<}^{esup}>$ )

**locale** *selection* =

**fixes**  $S :: 'a \text{ clause} \Rightarrow 'a \text{ clause}$

**assumes**

$S\text{-selects-subseteq}: \bigwedge C. S \ C \leq \# \ C$  **and**

$S\text{-selects-neg-lits}: \bigwedge C \ L. L \in \# \ S \ C \implies \text{is-neg } L$

**locale** *ground-resolution-with-selection* =

*selection*  $S$  **for**  $S :: ('a :: \text{wellorder}) \text{ clause} \Rightarrow 'a \text{ clause}$

**begin**

**context**

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

**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*  $C =$

$\{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max-mset } C = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1$   
 $\wedge \neg (\bigcup D \in \{D. D < C\}. \text{production } D) \models_h C \wedge S \ C = \{\#\}\}$

$\langle \text{proof} \rangle$

**termination**  $\langle \text{proof} \rangle$

**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*  $C = \{A. C \in N \wedge C \neq \{\#\} \wedge \text{Max-mset } C = \text{Pos } A \wedge \text{count } C (\text{Pos } A) \leq 1 \wedge \neg \text{interp}$   
 $C \models_h C \wedge S \ C = \{\#\}\}$

$\langle \text{proof} \rangle$

**abbreviation** *productive*  $A \equiv (\text{production } A \neq \{\})$

**abbreviation** *produces* :: 'a clause  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

*produces*  $C \ A \equiv \text{production } C = \{A\}$

**lemma** *producesD*:

$\text{produces } C \ A \implies C \in N \wedge C \neq \{\#\} \wedge \text{Pos } A = \text{Max-mset } C \wedge \text{count } C \ (\text{Pos } A) \leq 1 \wedge$   
 $\neg \text{interp } C \models_h C \wedge S \ C = \{\#\}$   
 $\langle \text{proof} \rangle$

**lemma** *produces C A  $\implies$  Pos A  $\in \#$  C*

$\langle \text{proof} \rangle$

**lemma** *interp'-def-in-set*:

$\text{interp } C = (\bigcup D \in \{D \in N. D < C\}. \text{production } D)$   
 $\langle \text{proof} \rangle$

**lemma** *production-iff-produces*:

$\text{produces } D \ A \longleftrightarrow A \in \text{production } D$   
 $\langle \text{proof} \rangle$

**definition** *Interp* :: 'a clause  $\Rightarrow$  'a interp **where**

$\text{Interp } C = \text{interp } C \cup \text{production } C$

**lemma**

**assumes** *produces C P*

**shows** *Interp C  $\models_h$  C*

$\langle \text{proof} \rangle$

**definition** *INTERP* :: 'a interp **where**

$\text{INTERP} = (\bigcup D \in N. \text{production } D)$

**lemma** *interp-subseteq-Interp[simp]*:  $\text{interp } C \subseteq \text{Interp } C$

$\langle \text{proof} \rangle$

**lemma** *Interp-as-UNION*:  $\text{Interp } C = (\bigcup D \in \{D. D \leq C\}. \text{production } D)$

$\langle \text{proof} \rangle$

**lemma** *productive-not-empty*:  $\text{productive } C \implies C \neq \{\#\}$

$\langle \text{proof} \rangle$

**lemma** *productive-imp-produces-Max-literal*:  $\text{productive } C \implies \text{produces } C \ (\text{atm-of } (\text{Max-mset } C))$

$\langle \text{proof} \rangle$

**lemma** *productive-imp-produces-Max-atom*:  $\text{productive } C \implies \text{produces } C \ (\text{Max } (\text{atms-of } C))$

$\langle \text{proof} \rangle$

**lemma** *produces-imp-Max-literal*:  $\text{produces } C \ A \implies A = \text{atm-of } (\text{Max-mset } C)$

$\langle \text{proof} \rangle$

**lemma** *produces-imp-Max-atom*:  $\text{produces } C \ A \implies A = \text{Max } (\text{atms-of } C)$

$\langle \text{proof} \rangle$

**lemma** *produces-imp-Pos-in-lits*:  $\text{produces } C \ A \implies \text{Pos } A \in \# \ C$

$\langle \text{proof} \rangle$

**lemma** *productive-in-N*:  $\text{productive } C \implies C \in N$

$\langle \text{proof} \rangle$

**lemma** *produces-imp-atms-leq*:  $\text{produces } C \ A \implies B \in \text{atms-of } C \implies B \leq A$

$\langle \text{proof} \rangle$

**lemma** *produces-imp-neg-notin-lits*:  $\text{produces } C \ A \implies \text{Neg } A \notin \# \ C$

$\langle \text{proof} \rangle$

**lemma** *less-eq-imp-interp-subseteq-interp*:  $C \leq D \implies \text{interp } C \subseteq \text{interp } D$

$\langle \text{proof} \rangle$

**lemma** *less-eq-imp-interp-subseteq-Interp*:  $C \leq D \implies \text{interp } C \subseteq \text{Interp } D$

$\langle \text{proof} \rangle$

**lemma** *less-imp-production-subseteq-interp*:  $C < D \implies \text{production } C \subseteq \text{interp } D$

$\langle \text{proof} \rangle$

**lemma** *less-eq-imp-production-subseteq-Interp*:  $C \leq D \implies \text{production } C \subseteq \text{Interp } D$

$\langle \text{proof} \rangle$

**lemma** *less-imp-Interp-subseteq-interp*:  $C < D \implies \text{Interp } C \subseteq \text{interp } D$

$\langle \text{proof} \rangle$

**lemma** *less-eq-imp-Interp-subseteq-Interp*:  $C \leq D \implies \text{Interp } C \subseteq \text{Interp } D$

$\langle \text{proof} \rangle$

**lemma** *false-Interp-to-true-interp-imp-less-multiset*:  $A \notin \text{Interp } C \implies A \in \text{interp } D \implies C < D$

$\langle \text{proof} \rangle$

**lemma** *false-interp-to-true-interp-imp-less-multiset*:  $A \notin \text{interp } C \implies A \in \text{interp } D \implies C < D$

$\langle \text{proof} \rangle$

**lemma** *false-Interp-to-true-Interp-imp-less-multiset*:  $A \notin \text{Interp } C \implies A \in \text{Interp } D \implies C < D$

$\langle \text{proof} \rangle$

**lemma** *false-interp-to-true-Interp-imp-le-multiset*:  $A \notin \text{interp } C \implies A \in \text{Interp } D \implies C \leq D$

$\langle \text{proof} \rangle$

**lemma** *interp-subseteq-INTERP*:  $\text{interp } C \subseteq \text{INTERP}$

$\langle \text{proof} \rangle$

**lemma** *production-subseteq-INTERP*:  $\text{production } C \subseteq \text{INTERP}$

$\langle \text{proof} \rangle$

**lemma** *Interp-subseteq-INTERP*:  $\text{Interp } C \subseteq \text{INTERP}$

$\langle \text{proof} \rangle$

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**  $A \in \text{interp } C$

$\langle \text{proof} \rangle$

**lemma** *neg-notin-Interp-not-produce*:  $\text{Neg } A \in \# \ C \implies A \notin \text{Interp } D \implies C \leq D \implies \neg \text{produces } D'' \ A$

$\langle \text{proof} \rangle$

**lemma** *in-production-imp-produces*:  $A \in \text{production } C \implies \text{produces } C \ A$

$\langle \text{proof} \rangle$

**lemma** *not-produces-imp-notin-production*:  $\neg \text{produces } C \ A \implies A \notin \text{production } C$   
 $\langle \text{proof} \rangle$

**lemma** *not-produces-imp-notin-interp*:  $(\bigwedge D. \neg \text{produces } D \ A) \implies A \notin \text{interp } C$   
 $\langle \text{proof} \rangle$

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-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**  $(\bigcup C \in CC. \text{production } C) \models_h C$

$\langle \text{proof} \rangle$

**lemma** *true-Interp-imp-interp*:  $C \leq D \implies D < D' \implies \text{Interp } D \models_h C \implies \text{interp } D' \models_h C$   
 $\langle \text{proof} \rangle$

**lemma** *true-Interp-imp-Interp*:  $C \leq D \implies D < D' \implies \text{Interp } D \models_h C \implies \text{Interp } D' \models_h C$   
 $\langle \text{proof} \rangle$

**lemma** *true-Interp-imp-INTERP*:  $C \leq D \implies \text{Interp } D \models_h C \implies \text{INTERP} \models_h C$   
 $\langle \text{proof} \rangle$

**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**  $(\bigcup C \in CC. \text{production } C) \models_h C$

$\langle \text{proof} \rangle$

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*:  $C \leq D \implies D < D' \implies \text{interp } D \models_h C \implies \text{interp } D' \models_h C$   
 $\langle \text{proof} \rangle$

**lemma** *true-interp-imp-Interp*:  $C \leq D \implies D < D' \implies \text{interp } D \models_h C \implies \text{Interp } D' \models_h C$   
 $\langle \text{proof} \rangle$

**lemma** *true-interp-imp-INTERP*:  $C \leq D \implies \text{interp } D \models_h C \implies \text{INTERP} \models_h C$   
 $\langle \text{proof} \rangle$

**lemma** *productive-imp-false-interp*:  $\text{productive } C \implies \neg \text{interp } C \models_h C$   
 $\langle \text{proof} \rangle$

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**  $D: \{\#Pos\ P, Pos\ P\# \} < C$   
**shows**  $\neg produces\ C\ P$   
 $\langle proof \rangle$

This lemma corresponds to theorem 2.7.7 page 77 of Weidenbach's book.

**lemma**  
**assumes**  $D: C + \{\#Neg\ P\# \} < D$   
**shows**  $production\ D \neq \{P\}$   
 $\langle proof \rangle$

**lemma** *in-interp-is-produced*:  
**assumes**  $P \in INTERP$   
**shows**  $\exists D. D + \{\#Pos\ P\# \} \in N \wedge produces\ (D + \{\#Pos\ P\# \})\ P$   
 $\langle proof \rangle$

**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**  $A \models_h C$   
 $\langle proof \rangle$

**lemma** *abstract-red-subset-mset-abstract-red*:  
**assumes**  
 $abstr: abstract-red\ C\ N$  **and**  
 $c-lt-d: C \subseteq_{\#} D$   
**shows**  $abstract-red\ D\ N$   
 $\langle proof \rangle$

**lemma** *true-clss-clss-extended*:  
**assumes**  
 $A \models_p B$  **and**  
 $tot: total-over-m\ I\ A$  **and**  
 $cons: consistent-interp\ I$  **and**  
 $I-A: I \models_s A$   
**shows**  $I \models B$   
 $\langle proof \rangle$

**lemma**

**assumes**

$CP: \neg \text{class-lt } N (\{ \#C\# \} + \{ \#E\# \}) \models_p \{ \#C\# \} + \{ \#Neg P\# \}$  **and**  
 $\text{class-lt } N (\{ \#C\# \} + \{ \#E\# \}) \models_p \{ \#E\# \} + \{ \#Pos P\# \} \vee \text{class-lt } N (\{ \#C\# \} + \{ \#E\# \}) \models_p$   
 $\{ \#C\# \} + \{ \#Neg P\# \}$   
**shows**  $\text{class-lt } N (\{ \#C\# \} + \{ \#E\# \}) \models_p \{ \#E\# \} + \{ \#Pos P\# \}$

$\langle \text{proof} \rangle$

**locale** *ground-ordered-resolution-with-redundancy* =

*ground-resolution-with-selection* +

**fixes** *redundant* :: 'a::wellorder clause  $\Rightarrow$  'a clause-set  $\Rightarrow$  bool

**assumes**

*redundant-iff-abstract*:  $\text{redundant } A \ N \longleftrightarrow \text{abstract-red } A \ N$

**begin**

**definition** *saturated* :: 'a clause-set  $\Rightarrow$  bool **where**

$\text{saturated } N \longleftrightarrow$

$(\forall A \ B \ C. A \in N \longrightarrow B \in N \longrightarrow \neg \text{redundant } A \ N \longrightarrow \neg \text{redundant } B \ N \longrightarrow$   
 $\text{superposition-rules } A \ B \ C \longrightarrow \text{redundant } C \ N \vee C \in N)$

**lemma** (**in**  $-$ )

**assumes**  $\langle A \models_p C + E \rangle$

**shows**  $\langle A \models_p \text{add-mset } L \ C \vee A \models_p \text{add-mset } (-L) \ E \rangle$

$\langle \text{proof} \rangle$

**lemma**

**assumes**

*saturated*:  $\text{saturated } N$  **and**

*finite*:  $\text{finite } N$  **and**

*empty*:  $\{ \# \} \notin N$

**shows**  $\text{INTERP } N \models_{hs} N$

$\langle \text{proof} \rangle$

**end**

**lemma** *tautology-is-redundant*:

**assumes** *tautology*  $C$

**shows**  $\text{abstract-red } C \ N$

$\langle \text{proof} \rangle$

**lemma** *subsumed-is-redundant*:

**assumes**  $AB: A \subset\# B$

**and**  $AN: A \in N$

**shows**  $\text{abstract-red } B \ N$

$\langle \text{proof} \rangle$

**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**  $\text{redundant } A \ N$

**shows**  $\text{abstract-red } A \ N$

$\langle \text{proof} \rangle$

**lemma** *redundant-mono*:

*redundant A N  $\implies A \subseteq\# B \implies$  redundant B N*  
 *$\langle proof \rangle$*

**locale** *truc* =  
    *selection S for S :: nat clause  $\Rightarrow$  nat clause*  
**begin**  
  
**end**  
  
**end**