

A Formalization of the SCL(FOL) Calculus: Simple Clause Learning for First-Order Logic

Martin Desharnais

February 6, 2026

Abstract

This Isabelle/HOL formalization covers the unexecutable specification of Simple Clause Learning for first-order logic without equality [2, 1]: SCL(FOL). The main results are formal proofs of soundness, non-redundancy of learned clauses, termination, and refutational completeness. Compared to the unformalized version, the formalized calculus is simpler, a number of results were generalized, and the non-redundancy statement was strengthened. We found and corrected one bug in a previously published version of the SCL Backtrack rule. Compared to related formalizations, we introduce a new technique for showing termination based on non-redundant clause learning.

Contents

1	Abstract Renaming	3
1.1	Interpretation to Prove That Assumptions Are Consistent . . .	4
2	Abstract Substitution Extra	4
3	Extra Lemmas	5
3.1	Set Extra	5
3.2	Finite Set Extra	5
3.3	Product Type Extra	5
3.4	List Extra	5
3.5	Sublist Extra	6
3.6	Multiset Extra	6
3.6.1	Calculus Extra	6
3.7	Clausal Calculus Extra	6
3.7.1	Clausal Calculus Only	6
3.7.2	Clausal Calculus and Abstract Substitution	7
3.8	First Order Terms Extra	7
3.8.1	First Order Terms Only	7
3.8.2	First Order Terms And Abstract Substitution	9

3.8.3	Minimal, Idempotent Most General Unifier	14
3.8.4	Renaming Extra	15
4	SCL State	16
5	SCL(FOL) Calculus	22
5.1	Lemmas About (\prec_B)	22
5.2	Rules	22
5.3	Well-Defined	24
5.4	Some rules are right unique	26
5.5	Miscellaneous Lemmas	26
6	Invariants	28
6.1	Initial Literals Generalize Learned, Trail, and Conflict Literals	28
6.2	Trail Literals Are Ground	30
6.3	Trail Literals Are Defined Only Once	31
6.4	Trail Closures Are False In Subtrails	32
6.5	Trail Literals Were Propagated or Decided	33
6.6	Trail Atoms Are Less Than Bound	35
6.7	Trail Resolved Literals Have Unique Polarity	36
6.8	Trail And Conflict Closures Are Ground	37
6.9	Trail And Conflict Closures Are Ground And False	39
6.10	Learned Clauses Are Non-empty	40
6.11	Backtrack Follows Conflict Resolution	41
6.12	Miscellaneous Lemmas	42
7	Soundness	42
7.1	Sound Trail	42
7.2	Sound State	43
7.3	Initial State Is Sound	43
7.4	SCL Rules Preserve Soundness	43
8	Strategies	44
9	Monotonicity w.r.t. the Bounding Atom	45
9.1	Examples	51
9.2	Miscellaneous Lemmas	52
9.3	Well-Defined	52
9.4	Strict Partial Order	53
9.5	Strict Total (w.r.t. Elements in Trail) Order	54
9.6	Well-Founded	54
9.7	Extension on All Literals	55
9.7.1	Well-Founded	57
9.8	Alternative only for terms	57

10 Reasonable Steps	59
10.1 Invariants	60
10.1.1 No Conflict After Decide	60
10.2 Miscellaneous Lemmas	61
11 Regular Steps	61
11.1 Invariants	62
11.1.1 Almost No Conflict With Trail	62
11.1.2 Backtrack Follows Regular Conflict Resolution	64
11.2 Miscellaneous Lemmas	65
12 Resolve in Regular Runs	66
13 Clause Redundancy	67
14 Trail-Induced Ordering	69
14.1 Miscellaneous Lemmas	69
14.2 Strict Partial Order	69
14.3 Properties	69
15 Dynamic Non-Redundancy	69
16 Static Non-Redundancy	72
16.1 Basic Results	73
16.1.1 Minimal-element characterization of well-foundedness	73
17 Extra Lemmas	75
17.1 Set Extra	75
17.2 Prod Extra	75
17.3 FSet Extra	76
18 Termination	76
18.1 SCL without backtracking terminates	76
18.2 Backtracking can only be done finitely often	78
18.3 Regular SCL terminates	79
theory <i>Abstract-Renaming-Apart</i>	
imports <i>Main</i>	
begin	

1 Abstract Renaming

```

locale renaming-apart =
  fixes
    renaming :: 'a set  $\Rightarrow$  'a  $\Rightarrow$  'a
  assumes
    renaming-correct: finite V  $\implies$  renaming V x  $\notin$  V and

```

inj-renaming: finite V \implies inj (renaming V)

1.1 Interpretation to Prove That Assumptions Are Consistent

experiment begin

definition *renaming-apart-nats where*

renaming-apart-nats V = (let m = Max V in ($\lambda x. \text{Suc } (x + m)$))

interpretation *renaming-apart-nats: renaming-apart renaming-apart-nats*
<proof>

end

end

theory *Ordered-Resolution-Prover-Extra*

imports

Ordered-Resolution-Prover.Abstract-Substitution

begin

2 Abstract Substitution Extra

lemma (*in substitution-ops*) *subst-atm-of-eqI:*

L · l $\sigma_L = K · l \sigma_K \implies \text{atm-of } L · a \sigma_L = \text{atm-of } K · a \sigma_K$
<proof>

lemma (*in substitution-ops*) *set-mset-subst-cls-conv: set-mset (C · σ) = ($\lambda L. L · l$*

σ) ‘ set-mset C

<proof>

end

theory *SCL-FOL*

imports

Main

HOL-Library.FSet

Saturation-Framework.Calculus

Saturation-Framework-Extensions.Clausal-Calculus

Ordered-Resolution-Prover.Clausal-Logic

Ordered-Resolution-Prover.Abstract-Substitution

Ordered-Resolution-Prover.Herbrand-Interpretation

First-Order-Terms.Subsumption

First-Order-Terms.Term

First-Order-Terms.Unification

Abstract-Renaming-Apart

Ordered-Resolution-Prover-Extra

begin

3 Extra Lemmas

3.1 Set Extra

lemma *not-in-iff*: $L \notin xs \iff (\forall y \in xs. L \neq y)$
<proof>

lemma *disjoint-iff'*: $A \cap B = \{\}$ $\iff (\forall a \in A. a \notin B) \wedge (\forall b \in B. b \notin A)$
<proof>

lemma *set-filter-insert-conv*:
 $\{x \in insert\ y\ S. P\ x\} = (if\ P\ y\ then\ insert\ y\ else\ id)\ \{x \in S. P\ x\}$
<proof>

lemma *not-empty-if-mem*: $x \in X \implies X \neq \{\}$
<proof>

3.2 Finite Set Extra

lemma *finite-induct'* [*case-names empty singleton insert-insert, induct set: finite*]:
— Discharging $x \notin F$ entails extra work.
assumes *finite* F
assumes $P\ \{\}$
and *singleton*: $\bigwedge x. P\ \{x\}$
and *insert-insert*: $\bigwedge x\ y\ F. finite\ F \implies x \neq y \implies x \notin F \implies y \notin F \implies P$
 $(insert\ y\ F) \implies P\ (insert\ x\ (insert\ y\ F))$
shows $P\ F$
<proof>

3.3 Product Type Extra

lemma *insert-Times*: $insert\ a\ A \times B = Pair\ a\ ' B \cup A \times B$
<proof>

lemma *Times-insert*: $A \times insert\ b\ B = (\lambda x. (x, b))\ ' A \cup A \times B$
<proof>

lemma *insert-Times-insert'*:
 $insert\ a\ A \times insert\ b\ B = insert\ (a, b)\ ((Pair\ a\ ' B) \cup ((\lambda x. (x, b))\ ' A) \cup (A \times B))$
(is ?lhs = ?rhs)
<proof>

3.4 List Extra

lemma *lt-lengthD*:
assumes *i-lt-xs*: $i < length\ xs$
shows $\exists xs1\ xi\ xs2. xs = xs1\ @\ xi\ \#\ xs2 \wedge length\ xs1 = i$
<proof>

lemma *lt-lt-lengthD*:

assumes *i-lt-xs*: $i < \text{length } xs$ **and** *j-lt-xs*: $j < \text{length } xs$ **and**

i-lt-j: $i < j$

shows $\exists xs1\ xi\ xs2\ xj\ xs3. xs = xs1 @ xi \# xs2 @ xj \# xs3 \wedge \text{length } xs1 = i \wedge \text{length } (xs1 @ xi \# xs2) = j$

<proof>

3.5 Sublist Extra

lemma *not-mem-strict-suffix*:

shows *strict-suffix* $xs\ (y \# ys) \implies y \notin \text{set } ys \implies y \notin \text{set } xs$

<proof>

lemma *not-mem-strict-suffix'*:

shows *strict-suffix* $xs\ (y \# ys) \implies f\ y \notin f\ ' \text{set } ys \implies f\ y \notin f\ ' \text{set } xs$

<proof>

3.6 Multiset Extra

lemma *multp_{DM}-implies-one-step*:

multp_{DM} $R\ M\ N \implies \exists I\ J\ K. N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge (\forall k \in \#K. \exists x \in \#J. R\ k\ x)$

<proof>

lemma *multp_{HO}-implies-one-step*:

multp_{HO} $R\ M\ N \implies \exists I\ J\ K. N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge (\forall k \in \#K. \exists x \in \#J. R\ k\ x)$

<proof>

lemma *Multiset-Bex-plus-iff*: $(\exists x \in \#(M1 + M2). P\ x) \longleftrightarrow (\exists x \in \#M1. P\ x) \vee (\exists x \in \#M2. P\ x)$

<proof>

lemma *multp-singleton-rightD*:

assumes *multp* $R\ M\ \{\#x\#}$ **and** *transp* R

shows $y \in \#M \implies R\ y\ x$

<proof>

3.6.1 Calculus Extra

lemma (*in consequence-relation*) *entails-one-formula*: $N \models U \implies D \in U \implies N \models \{D\}$

<proof>

3.7 Clausal Calculus Extra

3.7.1 Clausal Calculus Only

lemma *true-cls-iff-set-mset-eq*: *set-mset* $C = \text{set-mset } D \implies I \models C \longleftrightarrow I \models D$

<proof>

lemma *true-cls-if-set-mset-eq*: $(\forall D \in \mathcal{D}. \exists C \in \mathcal{C}. \text{set-mset } D = \text{set-mset } C) \implies I \models_s \mathcal{C} \implies I \models_s \mathcal{D}$
 ⟨proof⟩

lemma *entails-cls-insert*: $N \models_e \text{insert } C \ U \longleftrightarrow N \models_e \{C\} \wedge N \models_e U$
 ⟨proof⟩

lemma *Collect-lits-from-atms-conv*: $\{L. P \ (\text{atm-of } L)\} = (\bigcup x \in \{x. P \ x\}. \{\text{Pos } x, \text{Neg } x\})$
 (is ?lhs = ?rhs)
 ⟨proof⟩

3.7.2 Clausal Calculus and Abstract Substitution

lemma (in substitution) *is-ground-lit-Pos[simp]*: *is-ground-atm atm* \implies *is-ground-lit (Pos atm)*
 ⟨proof⟩

lemma (in substitution) *is-ground-lit-Neg[simp]*: *is-ground-atm atm* \implies *is-ground-lit (Neg atm)*
 ⟨proof⟩

3.8 First Order Terms Extra

3.8.1 First Order Terms Only

lemma *atm-of-eq-uminus-if-lit-eq*: $L = - K \implies \text{atm-of } L = \text{atm-of } K$
 ⟨proof⟩

lemma *subst-subst-eq-subst-subst-if-subst-eq-substI*:
assumes $t \cdot \sigma = u \cdot \delta$ **and**
t-inter-δ-empty: $\text{vars-term } t \cap \text{subst-domain } \delta = \{\}$ **and**
u-inter-σ-empty: $\text{vars-term } u \cap \text{subst-domain } \sigma = \{\}$
shows
range-vars σ ∩ subst-domain δ = {} $\implies t \cdot \sigma \cdot \delta = u \cdot \sigma \cdot \delta$
range-vars δ ∩ subst-domain σ = {} $\implies t \cdot \delta \cdot \sigma = u \cdot \delta \cdot \sigma$
 ⟨proof⟩

lemma *subst-compose-in-unifiersI*:
assumes $t \cdot \sigma = u \cdot \delta$ **and**
vars-term t ∩ subst-domain δ = {} **and**
vars-term u ∩ subst-domain σ = {}
shows
range-vars σ ∩ subst-domain δ = {} $\implies \sigma \circ_s \delta \in \text{unifiers } \{(t, u)\}$
range-vars δ ∩ subst-domain σ = {} $\implies \delta \circ_s \sigma \in \text{unifiers } \{(t, u)\}$
 ⟨proof⟩

lemma *subst-ident-if-not-in-domain*: $x \notin \text{subst-domain } \mu \implies \mu x = \text{Var } x$
 ⟨proof⟩

lemma *is-renaming* ($\text{Var}(x := \text{Var } x')$)
 ⟨*proof*⟩

lemma *ex-mgu-if-subst-eq-subst-and-disj-vars*:
fixes $t\ u :: ('f, 'v)\ \text{Term.term}$ **and** $\sigma_t\ \sigma_u :: ('f, 'v)\ \text{subst}$
assumes $t \cdot \sigma_t = u \cdot \sigma_u$ **and** $\text{vars-term } t \cap \text{vars-term } u = \{\}$
shows $\exists \mu :: ('f, 'v)\ \text{subst}.$ *Unification.mgu* $t\ u = \text{Some } \mu$
 ⟨*proof*⟩

lemma *restrict-subst-domain-subst-composition*:
fixes $\sigma_A\ \sigma_B\ A\ B$
assumes
 distinct-domains: $A \cap B = \{\}$ **and**
 distinct-range: $\forall x \in A. \text{vars-term } (\sigma_A\ x) \cap \text{subst-domain } \sigma_B = \{\}$
defines $\sigma \equiv \text{restrict-subst-domain } A\ \sigma_A\ \circ_s\ \sigma_B$
shows $x \in A \implies \sigma\ x = \sigma_A\ x$ $x \in B \implies \sigma\ x = \sigma_B\ x$
 ⟨*proof*⟩

lemma *merge-substs-on-disjoint-domains*:
fixes $\sigma_A\ \sigma_B\ A\ B$
assumes *distinct-domains*: $A \cap B = \{\}$
defines $\sigma \equiv (\lambda x. \text{if } x \in A \text{ then } \sigma_A\ x \text{ else if } x \in B \text{ then } \sigma_B\ x \text{ else } \text{Var } x)$
shows
 $x \in A \implies \sigma\ x = \sigma_A\ x$
 $x \in B \implies \sigma\ x = \sigma_B\ x$
 $x \notin A \cup B \implies \sigma\ x = \text{Var } x$
 ⟨*proof*⟩

definition *is-grounding-merge* **where**
is-grounding-merge $\gamma\ A\ \gamma_A\ B\ \gamma_B \longleftrightarrow$
 $A \cap B = \{\} \longrightarrow (\forall x \in A. \text{vars-term } (\gamma_A\ x) = \{\}) \longrightarrow (\forall x \in B. \text{vars-term } (\gamma_B\ x) = \{\}) \longrightarrow$
 $(\forall x \in A. \gamma\ x = \gamma_A\ x) \wedge (\forall x \in B. \gamma\ x = \gamma_B\ x)$

lemma *is-grounding-merge-if-mem-then-else*[*simp*]:
fixes $\gamma_A\ \gamma_B\ A\ B$
defines $\gamma \equiv (\lambda x. \text{if } x \in A \text{ then } \gamma_A\ x \text{ else } \gamma_B\ x)$
shows *is-grounding-merge* $\gamma\ A\ \gamma_A\ B\ \gamma_B$
 ⟨*proof*⟩

lemma *is-grounding-merge-restrict-subst-domain-comp*[*simp*]:
fixes $\gamma_A\ \gamma_B\ A\ B$
defines $\gamma \equiv \text{restrict-subst-domain } A\ \gamma_A\ \circ_s\ \gamma_B$

shows *is-grounding-merge* $\gamma A \gamma_A B \gamma_B$
 ⟨*proof*⟩

3.8.2 First Order Terms And Abstract Substitution

no-notation *subst-apply-term* (**infixl** $\langle \cdot \rangle$ 67)

no-notation *subst-compose* (**infixl** $\langle \circ_s \rangle$ 75)

global-interpretation *substitution-ops subst-apply-term Var subst-compose* ⟨*proof*⟩

notation *subst-atm-abbrev* (**infixl** $\langle \cdot a \rangle$ 67)

notation *subst-atm-list* (**infixl** $\langle \cdot al \rangle$ 67)

notation *subst-lit* (**infixl** $\langle \cdot l \rangle$ 67)

notation *subst-cls* (**infixl** $\langle \cdot \rangle$ 67)

notation *subst-cls* (**infixl** $\langle \cdot cs \rangle$ 67)

notation *subst-cls-list* (**infixl** $\langle \cdot cl \rangle$ 67)

notation *subst-cls-lists* (**infixl** $\langle \cdot \cdot cl \rangle$ 67)

notation *comp-subst-abbrev* (**infixl** $\langle \odot \rangle$ 67)

abbreviation *vars-lit* :: (f, v) *Term.term literal* \Rightarrow v set **where**
vars-lit $L \equiv$ *vars-term* (*atm-of* L)

definition *vars-cls* :: (f, v) *term clause* \Rightarrow v set **where**
vars-cls $C =$ *Union* (*set-mset* (*image-mset* *vars-lit* C))

definition *vars-cls* :: (f, v) *term clause set* \Rightarrow v set **where**
vars-cls $N = (\bigcup C \in N. \text{vars-cls } C)$

lemma *vars-cls-empty[simp]*: *vars-cls* $\{\}$ = $\{\}$
 ⟨*proof*⟩

lemma *vars-cls-insert[simp]*: *vars-cls* (*insert* $C N$) = *vars-cls* $C \cup$ *vars-cls* N
 ⟨*proof*⟩

lemma *vars-cls-union[simp]*: *vars-cls* ($CC \cup DD$) = *vars-cls* $CC \cup$ *vars-cls* DD
 ⟨*proof*⟩

lemma *vars-cls-empty[simp]*: *vars-cls* $\{\#\}$ = $\{\}$
 ⟨*proof*⟩

lemma *finite-vars-cls[simp]*: *finite* (*vars-cls* C)
 ⟨*proof*⟩

lemma *vars-cls-plus-iff*: *vars-cls* ($C + D$) = *vars-cls* $C \cup$ *vars-cls* D
 ⟨*proof*⟩

lemma *vars-cls-subset-vars-cls-if-subset-mset*: $C \subseteq\# D \Rightarrow$ *vars-cls* $C \subseteq$ *vars-cls* D
 D

<proof>

lemma *is-ground-atm-iff-vars-empty*: $is\text{-ground-atm } t \longleftrightarrow vars\text{-term } t = \{\}$
<proof>

lemma *is-ground-lit-iff-vars-empty*: $is\text{-ground-lit } L \longleftrightarrow vars\text{-lit } L = \{\}$
<proof>

lemma *is-ground-cls-iff-vars-empty*: $is\text{-ground-cls } C \longleftrightarrow vars\text{-cls } C = \{\}$
<proof>

lemma *is-ground-atm-is-ground-on-var*:
assumes $is\text{-ground-atm } (A \cdot a \ \sigma)$ and $v \in vars\text{-term } A$
shows $is\text{-ground-atm } (\sigma \ v)$
<proof>

lemma *is-ground-lit-is-ground-on-var*:
assumes $ground\text{-lit}: is\text{-ground-lit } (subst\text{-lit } L \ \sigma)$ and $v\text{-in-}L: v \in vars\text{-lit } L$
shows $is\text{-ground-atm } (\sigma \ v)$
<proof>

lemma *is-ground-cls-is-ground-on-var*:
assumes
 $ground\text{-clause}: is\text{-ground-cls } (subst\text{-cls } C \ \sigma)$ and
 $v\text{-in-}C: v \in vars\text{-cls } C$
shows $is\text{-ground-atm } (\sigma \ v)$
<proof>

lemma *vars-atm-subset-subst-domain-if-grounding*:
assumes $is\text{-ground-atm } (t \cdot a \ \gamma)$
shows $vars\text{-term } t \subseteq subst\text{-domain } \gamma$
<proof>

lemma *vars-lit-subset-subst-domain-if-grounding*:
assumes $is\text{-ground-lit } (L \cdot l \ \gamma)$
shows $vars\text{-lit } L \subseteq subst\text{-domain } \gamma$
<proof>

lemma *vars-cls-subset-subst-domain-if-grounding*:
assumes $is\text{-ground-cls } (C \cdot \sigma)$
shows $vars\text{-cls } C \subseteq subst\text{-domain } \sigma$
<proof>

lemma *same-on-vars-lit*:
assumes $\forall v \in vars\text{-lit } L. \sigma \ v = \tau \ v$
shows $subst\text{-lit } L \ \sigma = subst\text{-lit } L \ \tau$
<proof>

lemma *same-on-vars-clause*:

assumes $\forall v \in \text{vars-cls } S. \sigma v = \tau v$

shows $\text{subst-cls } S \sigma = \text{subst-cls } S \tau$

<proof>

lemma *same-on-lits-clause*:

assumes $\forall L \in \# C. \text{subst-lit } L \sigma = \text{subst-lit } L \tau$

shows $\text{subst-cls } C \sigma = \text{subst-cls } C \tau$

<proof>

global-interpretation *substitution* $(\cdot a) \text{ Var} :: - \Rightarrow (f, 'v) \text{ term } (\odot)$

<proof>

lemma *vars-subst-lit-eq-vars-subst-atm*: $\text{vars-lit } (L \cdot l \sigma) = \text{vars-term } (\text{atm-of } L \cdot a \sigma)$

<proof>

lemma *vars-subst-lit-eq*:

$\text{vars-lit } (L \cdot l \sigma) = (\bigcup x \in \text{vars-lit } L. \text{vars-term } (\sigma x))$

<proof>

lemma *vars-subst-cls-eq*:

$\text{vars-cls } (C \cdot \sigma) = (\bigcup x \in \text{vars-cls } C. \text{vars-term } (\sigma x))$

<proof>

lemma *vars-subst-lit-subset*: $\text{vars-lit } (L \cdot l \sigma) \subseteq \text{vars-lit } L - \text{subst-domain } \sigma \cup \text{range-vars } \sigma$

<proof>

lemma *vars-subst-cls-subset*: $\text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } C - \text{subst-domain } \sigma \cup \text{range-vars } \sigma$

<proof>

lemma *vars-subst-cls-subset-weak*: $\text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } C \cup \text{range-vars } \sigma$

<proof>

lemma *vars-cls-plus[simp]*: $\text{vars-cls } (C + D) = \text{vars-cls } C \cup \text{vars-cls } D$

<proof>

lemma *vars-cls-add-mset[simp]*: $\text{vars-cls } (\text{add-mset } L C) = \text{vars-lit } L \cup \text{vars-cls } C$

<proof>

lemma *UN-vars-term-atm-of-cls[simp]*: $(\bigcup T \in \{\text{atm-of ' set-mset } C\}. \bigcup (\text{vars-term ' } T)) = \text{vars-cls } C$

<proof>

lemma *vars-lit-subst-subset-vars-cls-substI[intro]*:

$\text{vars-lit } L \subseteq \text{vars-cls } C \implies \text{vars-lit } (L \cdot l \sigma) \subseteq \text{vars-cls } (C \cdot \sigma)$

<proof>

lemma *vars-subst-cls-subset-vars-cls-subst*:

$\text{vars-cls } C \subseteq \text{vars-cls } D \implies \text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } (D \cdot \sigma)$
 ⟨proof⟩

lemma *vars-cls-subst-subset*:

assumes *range-vars-η*: $\text{range-vars } \eta \subseteq \text{vars-lit } L \cup \text{vars-lit } L'$
shows $\text{vars-cls } (\text{add-mset } L \ D \cdot \eta) \subseteq \text{vars-cls } (\text{add-mset } L' \ (\text{add-mset } L \ D))$
 ⟨proof⟩

definition *disjoint-vars where*

$\text{disjoint-vars } C \ D \longleftrightarrow \text{vars-cls } C \cap \text{vars-cls } D = \{\}$

lemma *disjoint-vars-iff-inter-empty*: $\text{disjoint-vars } C \ D \longleftrightarrow \text{vars-cls } C \cap \text{vars-cls } D = \{\}$

⟨proof⟩

hide-fact *disjoint-vars-def*

lemma *disjoint-vars-sym*: $\text{disjoint-vars } C \ D \longleftrightarrow \text{disjoint-vars } D \ C$

⟨proof⟩

lemma *disjoint-vars-plus-iff*: $\text{disjoint-vars } (C + D) \ E \longleftrightarrow \text{disjoint-vars } C \ E \wedge \text{disjoint-vars } D \ E$

⟨proof⟩

lemma *disjoint-vars-subset-mset*: $\text{disjoint-vars } C \ D \implies E \subseteq\# C \implies \text{disjoint-vars } E \ D$

⟨proof⟩

lemma *disjoint-vars-subst-clsI*:

$\text{disjoint-vars } C \ D \implies \text{range-vars } \sigma \cap \text{vars-cls } D = \{\} \implies \text{disjoint-vars } (C \cdot \sigma) \ D$

⟨proof⟩

lemma *is-renaming-iff*: $\text{is-renaming } \varrho \longleftrightarrow (\forall x. \text{is-Var } (\varrho \ x)) \wedge \text{inj } \varrho$

(**is** ?lhs \longleftrightarrow ?rhs)

⟨proof⟩

lemma *subst-cls-idem-if-disj-vars*: $\text{subst-domain } \sigma \cap \text{vars-cls } C = \{\} \implies C \cdot \sigma = C$

⟨proof⟩

lemma *subst-lit-idem-if-disj-vars*: $\text{subst-domain } \sigma \cap \text{vars-lit } L = \{\} \implies L \cdot l \ \sigma = L$

⟨proof⟩

lemma *subst-lit-restrict-subst-domain*: $\text{vars-lit } L \subseteq V \implies L \cdot l \ \text{restrict-subst-domain } V \ \sigma = L \cdot l \ \sigma$

<proof>

lemma *subst-cls-restrict-subst-domain*: $\text{vars-cls } C \subseteq V \implies C \cdot \text{restrict-subst-domain } V \sigma = C \cdot \sigma$
<proof>

lemma *subst-cls-insert[simp]*: $\text{insert } C U \cdot \text{cs } \eta = \text{insert } (C \cdot \eta) (U \cdot \text{cs } \eta)$
<proof>

lemma *valid-grounding-of-renaming*:
assumes *is-renaming* ϱ
shows $I \models_s \text{grounding-of-cls } (C \cdot \varrho) \longleftrightarrow I \models_s \text{grounding-of-cls } C$
<proof>

lemma *is-unifier-iff-mem-unifiers-Times*:
assumes *fin-AA*: *finite* AA
shows $\text{is-unifier } v \ AA \longleftrightarrow v \in \text{unifiers } (AA \times AA)$
<proof>

lemma *is-mgu-singleton-iff-Unifiers-is-mgu-Times*:
assumes *fin*: *finite* AA
shows $\text{is-mgu } v \ \{AA\} \longleftrightarrow \text{Unifiers.is-mgu } v \ (AA \times AA)$
<proof>

lemma *is-imagu-singleton-iff-Unifiers-is-imagu-Times*:
assumes *fin*: *finite* AA
shows $\text{is-imagu } v \ \{AA\} \longleftrightarrow \text{Unifiers.is-imagu } v \ (AA \times AA)$
<proof>

lemma *unifiers-without-refl*: $\text{unifiers } E = \text{unifiers } \{e \in E. \text{fst } e \neq \text{snd } e\}$
(is ?lhs = ?rhs)
<proof>

lemma *subst-lit-renaming-subst-adapted*:
assumes *ren- ϱ* : *is-renaming* ϱ **and** *vars-L*: $\text{vars-lit } L \subseteq \text{subst-domain } \sigma$
shows $L \cdot l \ \varrho \cdot l \ \text{rename-subst-domain } \varrho \ \sigma = L \cdot l \ \sigma$
<proof>

lemma *subst-renaming-subst-adapted*:
assumes *ren- ϱ* : *is-renaming* ϱ **and** *vars-D*: $\text{vars-cls } D \subseteq \text{subst-domain } \sigma$
shows $D \cdot \varrho \cdot \text{rename-subst-domain } \varrho \ \sigma = D \cdot \sigma$
<proof>

lemma *subst-domain-rename-subst-domain-subset'*:
assumes *is-var- ϱ* : $\forall x. \text{is-Var } (\varrho \ x)$
shows $\text{subst-domain } (\text{rename-subst-domain } \varrho \ \sigma) \subseteq (\bigcup x \in \text{subst-domain } \sigma. \text{vars-term } (\varrho \ x))$
<proof>

lemma *range-vars-eq-empty-if-is-ground*:

is-ground-cls $(C \cdot \gamma) \implies \text{subst-domain } \gamma \subseteq \text{vars-cls } C \implies \text{range-vars } \gamma = \{\}$
 $\langle \text{proof} \rangle$

3.8.3 Minimal, Idempotent Most General Unifier

lemma *is-imgu-if-mgu-eq-Some*:

assumes *mgu*: *Unification.mgu* $t\ u = \text{Some } \mu$

shows *is-imgu* $\mu\ \{\{t, u\}\}$

$\langle \text{proof} \rangle$

primrec *pairs* **where**

pairs $\square = \square \mid$

pairs $(x \# xs) = (x, x) \# \text{map } (\text{Pair } x)\ xs \ @ \ \text{map } (\lambda y. (y, x))\ xs \ @ \ \text{pairs } xs$

lemma *set (pairs [a, b, c, d]) =*

$\{(a, a), (a, b), (a, c), (a, d),$
 $(b, a), (b, b), (b, c), (b, d),$
 $(c, a), (c, b), (c, c), (c, d),$
 $(d, a), (d, b), (d, c), (d, d)\}$

$\langle \text{proof} \rangle$

lemma *set-pairs*: *set (pairs xs) = set xs \times set xs*

$\langle \text{proof} \rangle$

Reflexive and symmetric pairs are not necessary to computing the MGU, but it makes the set of the resulting list equivalent to $\{(x, y). x \in xs \wedge y \in ys\}$, which is necessary for the following properties.

lemma *pair-in-set-pairs*: $a \in \text{set } as \implies b \in \text{set } as \implies (a, b) \in \text{set } (\text{pairs } as)$

$\langle \text{proof} \rangle$

lemma *fst-pair-in-set-if-pair-in-pairs*: $p \in \text{set } (\text{pairs } as) \implies \text{fst } p \in \text{set } as$

$\langle \text{proof} \rangle$

lemma *snd-pair-in-set-if-pair-in-pairs*: $p \in \text{set } (\text{pairs } as) \implies \text{snd } p \in \text{set } as$

$\langle \text{proof} \rangle$

lemma *vars-mset-mset-pairs*:

vars-mset $(\text{mset } (\text{pairs } as)) = (\bigcup b \in \text{set } as. \bigcup a \in \text{set } as. \text{vars-term } a \cup \text{vars-term } b)$

$\langle \text{proof} \rangle$

definition *mgu-sets* **where**

mgu-sets $\mu\ AAA \longleftrightarrow (\exists \text{ass. set } (\text{map set } \text{ass}) = AAA \wedge$

$\text{map-option subst-of } (\text{unify } (\text{concat } (\text{map pairs } \text{ass}))\ \square) = \text{Some } \mu)$

lemma *is-imgu-if-mgu-sets*:

assumes *mgu-AAA*: *mgu-sets* $\mu\ AAA$

shows *is-ingu* μ *AAA*
<proof>

3.8.4 Renaming Extra

context *renaming-apart* **begin**

lemma *inj-Var-comp-renaming*: *finite* $V \implies \text{inj} (Var \circ \text{renaming } V)$
<proof>

lemma *is-renaming-Var-comp-renaming*: *finite* $V \implies \text{Term.is-renaming} (Var \circ \text{renaming } V)$
<proof>

lemma *vars-term-subst-term-Var-comp-renaming-disj*:
assumes *fin-V*: *finite* V
shows *vars-term* $(t \cdot a (Var \circ \text{renaming } V)) \cap V = \{\}$
<proof>

lemma *vars-term-subst-term-Var-comp-renaming-disj'*:
assumes *fin-V*: *finite* $V1$ **and** *sub*: $V2 \subseteq V1$
shows *vars-term* $(t \cdot a (Var \circ \text{renaming } V1)) \cap V2 = \{\}$
<proof>

lemma *vars-lit-subst-renaming-disj*:
assumes *fin-V*: *finite* V
shows *vars-lit* $(L \cdot l (Var \circ \text{renaming } V)) \cap V = \{\}$
<proof>

lemma *vars-cls-subst-renaming-disj*:
assumes *fin-V*: *finite* V
shows *vars-cls* $(C \cdot (Var \circ \text{renaming } V)) \cap V = \{\}$
<proof>

abbreviation *renaming-wrt* :: $(f, -) \text{Term.term clause set} \Rightarrow - \Rightarrow (f, -) \text{Term.term}$
where

renaming-wrt $N \equiv Var \circ \text{renaming} (\text{vars-cls } N)$

lemma *is-renaming-renaming-wrt*: *finite* $N \implies \text{is-renaming} (\text{renaming-wrt } N)$
<proof>

lemma *ex-renaming-to-disjoint-vars*:
fixes $C :: (f, 'a) \text{Term.term clause}$ **and** $N :: (f, 'a) \text{Term.term clause set}$
assumes *fin*: *finite* N
shows $\exists \varrho. \text{is-renaming } \varrho \wedge \text{vars-cls} (C \cdot \varrho) \cap \text{vars-cls } N = \{\}$
<proof>

end

4 SCL State

type-synonym (f, v) *closure* = (f, v) *term clause* \times (f, v) *subst*

type-synonym (f, v) *closure-with-lit* =

(f, v) *term clause* \times (f, v) *term literal* \times (f, v) *subst*

type-synonym (f, v) *trail* = $((f, v)$ *term literal* \times (f, v) *closure-with-lit option*) *list*

type-synonym (f, v) *state* =

(f, v) *trail* \times (f, v) *term clause fset* \times (f, v) *closure option*

Note that, in contrast to Bromberger, Schwarz, and Weidenbach, the level is not part of the state. It would be redundant because it can always be computed from the trail.

abbreviation *initial-state* :: (f, v) *state* **where**

initial-state \equiv $([], \{\}, None)$

definition *state-trail* :: (f, v) *state* \Rightarrow (f, v) *trail* **where**

state-trail $S = fst\ S$

lemma *state-trail-simp*[*simp*]: *state-trail* $(\Gamma, U, u) = \Gamma$

<proof>

definition *state-learned* :: (f, v) *state* \Rightarrow (f, v) *term clause fset* **where**

state-learned $S = fst\ (snd\ S)$

lemma *state-learned-simp*[*simp*]: *state-learned* $(\Gamma, U, u) = U$

<proof>

definition *state-conflict* :: (f, v) *state* \Rightarrow (f, v) *closure option* **where**

state-conflict $S = snd\ (snd\ S)$

lemma *state-conflict-simp*[*simp*]: *state-conflict* $(\Gamma, U, u) = u$

<proof>

lemmas *state-proj-simp* = *state-trail-simp* *state-learned-simp* *state-conflict-simp*

lemma *state-simp*[*simp*]: $(state-trail\ S, state-learned\ S, state-conflict\ S) = S$

<proof>

fun *clss-of-trail-lit* **where**

clss-of-trail-lit $(-, None) = \{\}\mid$

clss-of-trail-lit $(-, Some\ (C, L, -)) = \{|add-mset\ L\ C|\}$

primrec *clss-of-trail* :: (f, v) *trail* \Rightarrow (f, v) *term clause fset* **where**

clss-of-trail $[] = \{\}\mid$

clss-of-trail $(Ln\ \#\ \Gamma) = clss-of-trail-lit\ Ln\ |\cup|\ clss-of-trail\ \Gamma$

hide-fact *clss-of-trail-def*

lemma *class-of-trail-append*: $\text{class-of-trail } (\Gamma_0 @ \Gamma_1) = \text{class-of-trail } \Gamma_0 \cup \text{class-of-trail } \Gamma_1$

<proof>

fun *class-of-closure* **where**

class-of-closure *None* = $\{\{\}\}$ |

class-of-closure (*Some* (*C*, *-*)) = $\{\{C\}\}$

definition *propagate-lit* **where**

propagate-lit *L C* γ = (*L* · *l* γ , *Some* (*C*, *L*, γ))

abbreviation *trail-propagate* ::

(f, v) *trail* \Rightarrow (f, v) *term literal* \Rightarrow (f, v) *term clause* \Rightarrow (f, v) *subst* \Rightarrow

(f, v) *trail* **where**

trail-propagate $\Gamma L C \gamma \equiv \text{propagate-lit } L C \gamma \# \Gamma$

lemma *fst-propagate-lit[simp]*: $\text{fst } (\text{propagate-lit } L C \sigma) = L \cdot l \sigma$

<proof>

lemma *suffix-trail-propagate[simp]*: $\text{suffix } \Gamma (\text{trail-propagate } \Gamma L C \delta)$

<proof>

lemma *class-of-trail-trail-propagate[simp]*:

$\text{class-of-trail } (\text{trail-propagate } \Gamma L C \gamma) = \text{finsert } (\text{add-mset } L C) (\text{class-of-trail } \Gamma)$

<proof>

definition *decide-lit* **where**

decide-lit *L* = (*L*, *None*)

abbreviation *trail-decide* :: (f, v) *trail* \Rightarrow (f, v) *term literal* \Rightarrow (f, v) *trail*

where

trail-decide $\Gamma L \equiv \text{decide-lit } L \# \Gamma$

lemma *fst-decide-lit[simp]*: $\text{fst } (\text{decide-lit } L) = L$

<proof>

lemma *class-of-trail-trail-decide[simp]*:

$\text{class-of-trail } (\text{trail-decide } \Gamma L) = \text{class-of-trail } \Gamma$

<proof>

definition *is-decision-lit*

:: (f, v) *term literal* \times (f, v) *closure-with-lit option* \Rightarrow *bool* **where**

is-decision-lit *Ln* $\longleftrightarrow \text{snd } Ln = \text{None}$

definition *trail-interp* :: *- list* \Rightarrow *- interp* **where**

trail-interp $\Gamma = \bigcup ((\lambda L. \text{case } L \text{ of } \text{Pos } A \Rightarrow \{A\} \mid \text{Neg } A \Rightarrow \{\}) \text{ 'fst ' set } \Gamma)$

lemma

$trail_interp \ [] = \{\}$
 $trail_interp \ ((Pos \ A, \ ann) \ \# \ \Gamma) = insert \ A \ (trail_interp \ \Gamma)$
 $trail_interp \ ((Neg \ A, \ ann) \ \# \ \Gamma) = trail_interp \ \Gamma$
 $\langle proof \rangle$

lemma *trail-interp-eq-Union*:

$trail_interp \ \Gamma = (\bigcup Ln \in set \ \Gamma. \ case \ fst \ Ln \ of \ Pos \ t \Rightarrow \{t\} \mid Neg \ t \Rightarrow \{\})$
 $\langle proof \rangle$

definition *trail-true-lit* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ literal \Rightarrow bool$ **where**
 $trail_true_lit \ \Gamma \ L \longleftrightarrow L \in fst \ ' \ set \ \Gamma$

definition *trail-false-lit* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ literal \Rightarrow bool$ **where**
 $trail_false_lit \ \Gamma \ L \longleftrightarrow - \ L \in fst \ ' \ set \ \Gamma$

definition *trail-true-cls* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ clause \Rightarrow bool$ **where**
 $trail_true_cls \ \Gamma \ C \longleftrightarrow (\exists L \in \# \ C. \ trail_true_lit \ \Gamma \ L)$

definition *trail-false-cls* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ clause \Rightarrow bool$ **where**
 $trail_false_cls \ \Gamma \ C \longleftrightarrow (\forall L \in \# \ C. \ trail_false_lit \ \Gamma \ L)$

definition *trail-true-clss* :: $(f, v) \ trail \Rightarrow (f, v) \ term \ clause \ set \Rightarrow bool$ **where**
 $trail_true_clss \ \Gamma \ N \longleftrightarrow (\forall C \in N. \ trail_true_cls \ \Gamma \ C)$

definition *trail-defined-lit* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ literal \Rightarrow bool$ **where**
 $trail_defined_lit \ \Gamma \ L \longleftrightarrow (L \in fst \ ' \ set \ \Gamma \vee - \ L \in fst \ ' \ set \ \Gamma)$

definition *trail-defined-cls* :: $(- \ literal \times - \ option) \ list \Rightarrow - \ clause \Rightarrow bool$ **where**
 $trail_defined_cls \ \Gamma \ C \longleftrightarrow (\forall L \in \# \ C. \ trail_defined_lit \ \Gamma \ L)$

lemma *trail-defined-lit-iff-true-or-false*:

$trail_defined_lit \ \Gamma \ L \longleftrightarrow trail_true_lit \ \Gamma \ L \vee trail_false_lit \ \Gamma \ L$
 $\langle proof \rangle$

lemma *trail-true-or-false-cls-if-defined*:

$trail_defined_cls \ \Gamma \ C \Longrightarrow trail_true_cls \ \Gamma \ C \vee trail_false_cls \ \Gamma \ C$
 $\langle proof \rangle$

lemma *trail-false-cls-mempty[simp]*: $trail_false_cls \ \Gamma \ \{\#\}$
 $\langle proof \rangle$

lemma *trail-false-cls-add-mset*:

$trail_false_cls \ \Gamma \ (add_mset \ L \ C) \longleftrightarrow trail_false_lit \ \Gamma \ L \wedge trail_false_cls \ \Gamma \ C$
 $\langle proof \rangle$

lemma *trail-false-cls-plus*:

$trail_false_cls \ \Gamma \ (C + D) \longleftrightarrow trail_false_cls \ \Gamma \ C \wedge trail_false_cls \ \Gamma \ D$
 $\langle proof \rangle$

lemma *not-trail-true-Nil[simp]*:
 $\neg \text{trail-true-lit } [] L$
 $\neg \text{trail-true-cls } [] C$
 $N \neq \{\}$ $\implies \neg \text{trail-true-clss } [] N$
 $\langle \text{proof} \rangle$

lemma *not-trail-false-Nil[simp]*:
 $\neg \text{trail-false-lit } [] L$
 $\text{trail-false-cls } [] C \iff C = \{\#\}$
 $\langle \text{proof} \rangle$

lemma *not-trail-defined-lit-Nil[simp]*: $\neg \text{trail-defined-lit } [] L$
 $\langle \text{proof} \rangle$

lemma *trail-defined-lit-if-trail-defined-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-defined-lit } \Gamma' K \implies \text{trail-defined-lit } \Gamma K$
 $\langle \text{proof} \rangle$

lemma *trail-defined-cls-if-trail-defined-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-defined-cls } \Gamma' C \implies \text{trail-defined-cls } \Gamma C$
 $\langle \text{proof} \rangle$

lemma *trail-false-lit-if-trail-false-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-false-lit } \Gamma' K \implies \text{trail-false-lit } \Gamma K$
 $\langle \text{proof} \rangle$

lemma *trail-false-cls-if-trail-false-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-false-cls } \Gamma' C \implies \text{trail-false-cls } \Gamma C$
 $\langle \text{proof} \rangle$

lemma *trail-interp-Cons*: $\text{trail-interp } (Ln \# \Gamma) = \text{trail-interp } [Ln] \cup \text{trail-interp } \Gamma$
 $\langle \text{proof} \rangle$

lemma *trail-interp-Cons'*: $\text{trail-interp } (Ln \# \Gamma) = (\text{case fst } Ln \text{ of Pos } A \Rightarrow \{A\} \mid$
 $\text{Neg } A \Rightarrow \{\}) \cup \text{trail-interp } \Gamma$
 $\langle \text{proof} \rangle$

lemma *true-lit-thick-unionD*: $(I1 \cup I2) \models_l L \implies I1 \models_l L \vee I2 \models_l L$
 $\langle \text{proof} \rangle$

lemma *subtrail-falseI*:
assumes *tr-false*: $\text{trail-false-cls } ((L, Cl) \# \Gamma) C$ **and** *L-not-in*: $-L \notin \# C$
shows $\text{trail-false-cls } \Gamma C$
 $\langle \text{proof} \rangle$

lemma *trail-false-cls-ignores-duplicates*:
 $\text{set-mset } C = \text{set-mset } D \implies \text{trail-false-cls } \Gamma C \iff \text{trail-false-cls } \Gamma D$
 $\langle \text{proof} \rangle$

lemma *ball-trail-propagate-is-ground-lit*:
assumes $\forall x \in \text{set } \Gamma. \text{is-ground-lit } (\text{fst } x)$ **and** *is-ground-lit* $(L \cdot l \ \sigma)$
shows $\forall x \in \text{set } (\text{trail-propagate } \Gamma \ L \ C \ \sigma). \text{is-ground-lit } (\text{fst } x)$
 $\langle \text{proof} \rangle$

lemma *ball-trail-decide-is-ground-lit*:
assumes $\forall x \in \text{set } \Gamma. \text{is-ground-lit } (\text{fst } x)$ **and** *is-ground-lit* L
shows $\forall x \in \text{set } (\text{trail-decide } \Gamma \ L). \text{is-ground-lit } (\text{fst } x)$
 $\langle \text{proof} \rangle$

lemma *trail-false-cls-subst-mgu-before-grounding*:
fixes $\Gamma :: ('f, 'v) \text{ trail}$
assumes *tr-false-cls*: *trail-false-cls* $\Gamma ((D + \{\#L, L'\#\}) \cdot \sigma)$ **and**
imgu- μ : *is-imgu* $\mu \{\{\text{atm-of } L, \text{atm-of } L'\}\}$ **and**
unif- σ : *is-unifiers* $\sigma \{\{\text{atm-of } L, \text{atm-of } L'\}\}$
shows *trail-false-cls* $\Gamma ((D + \{\#L\#\}) \cdot \mu \cdot \sigma)$
 $\langle \text{proof} \rangle$

lemma *trail-defined-lit-iff-defined-uminus*: *trail-defined-lit* $\Gamma \ L \longleftrightarrow \text{trail-defined-lit } \Gamma \ (-L)$
 $\langle \text{proof} \rangle$

lemma *trail-defined-lit-iff*: *trail-defined-lit* $\Gamma \ L \longleftrightarrow \text{atm-of } L \in \text{atm-of 'fst ' set } \Gamma$
 $\langle \text{proof} \rangle$

lemma *trail-interp-conv*: *trail-interp* $\Gamma = \text{atm-of ' } \{L \in \text{fst ' set } \Gamma. \text{is-pos } L\}$
 $\langle \text{proof} \rangle$

lemma *not-in-trail-interp-if-not-in-trail*: $t \notin \text{atm-of 'fst ' set } \Gamma \implies t \notin \text{trail-interp } \Gamma$
 $\langle \text{proof} \rangle$

inductive *trail-consistent* **where**

Nil[simp]: *trail-consistent* $\square \mid$

Cons: $\neg \text{trail-defined-lit } \Gamma \ L \implies \text{trail-consistent } \Gamma \implies \text{trail-consistent } ((L, u) \# \Gamma)$

lemma *distinct-atm-of-trail-if-trail-consistent*:
trail-consistent $\Gamma \implies \text{distinct } (\text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma)$
 $\langle \text{proof} \rangle$

lemma *trail-consistent-appendD*: *trail-consistent* $(\Gamma @ \Gamma') \implies \text{trail-consistent } \Gamma'$
 $\langle \text{proof} \rangle$

lemma *trail-consistent-if-suffix*:
trail-consistent $\Gamma \implies \text{suffix } \Gamma' \ \Gamma \implies \text{trail-consistent } \Gamma'$
 $\langle \text{proof} \rangle$

lemma *trail-interp-lit-if-trail-true*:

shows $\text{trail-consistent } \Gamma \implies \text{trail-true-lit } \Gamma L \implies \text{trail-interp } \Gamma \models_l L$
<proof>

lemma *trail-interp-cls-if-trail-true*:

assumes *trail-consistent* Γ **and** *trail-true-cls* ΓC
shows $\text{trail-interp } \Gamma \models C$
<proof>

lemma *trail-true-cls-iff-trail-interp-entails*:

assumes *trail-consistent* $\Gamma \forall L \in \# C$. *trail-defined-lit* ΓL
shows $\text{trail-true-cls } \Gamma C \longleftrightarrow \text{trail-interp } \Gamma \models C$
<proof>

lemma *trail-false-cls-iff-not-trail-interp-entails*:

assumes *trail-consistent* $\Gamma \forall L \in \# C$. *trail-defined-lit* ΓL
shows $\text{trail-false-cls } \Gamma C \longleftrightarrow \neg \text{trail-interp } \Gamma \models C$
<proof>

inductive *trail-closures-false* **where**

Nil[simp]: *trail-closures-false* $[] \mid$

Cons:

$(\forall D K \gamma. Kn = \text{propagate-lit } K D \gamma \longrightarrow \text{trail-false-cls } \Gamma (D \cdot \gamma)) \implies$
 $\text{trail-closures-false } \Gamma \implies \text{trail-closures-false } (Kn \# \Gamma)$

lemma *trail-closures-false-ConsD*: $\text{trail-closures-false } (Ln \# \Gamma) \implies \text{trail-closures-false } \Gamma$

<proof>

lemma *trail-closures-false-appendD*: $\text{trail-closures-false } (\Gamma @ \Gamma') \implies \text{trail-closures-false } \Gamma'$

<proof>

lemma *is-ground-lit-if-true-in-ground-trail*:

assumes $\forall L \in \text{fst } \text{'set } \Gamma$. *is-ground-lit* L
shows $\text{trail-true-lit } \Gamma L \implies \text{is-ground-lit } L$
<proof>

lemma *is-ground-lit-if-false-in-ground-trail*:

assumes $\forall L \in \text{fst } \text{'set } \Gamma$. *is-ground-lit* L
shows $\text{trail-false-lit } \Gamma L \implies \text{is-ground-lit } L$
<proof>

lemma *is-ground-lit-if-defined-in-ground-trail*:

assumes $\forall L \in \text{fst } \text{'set } \Gamma$. *is-ground-lit* L
shows $\text{trail-defined-lit } \Gamma L \implies \text{is-ground-lit } L$
<proof>

lemma *is-ground-cls-if-false-in-ground-trail*:

assumes $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{ is-ground-lit } L$
shows $\text{trail-false-cls } \Gamma \ C \implies \text{is-ground-cls } C$
 $\langle \text{proof} \rangle$

5 SCL(FOL) Calculus

locale $\text{scl-fol-calculus} = \text{renaming-apart renaming-vars}$
for $\text{renaming-vars} :: 'v \text{ set} \Rightarrow 'v \Rightarrow 'v +$
fixes $\text{less-B} :: ('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{bool}$ (**infix** $\langle \prec_B \rangle$ 50)
assumes
 $\text{finite-less-B}: \bigwedge \beta. \text{finite } \{x. x \prec_B \beta\}$
begin

abbreviation lesseq-B (**infix** $\langle \preceq_B \rangle$ 50) **where**
 $\text{lesseq-B} \equiv (\prec_B)^{==}$

5.1 Lemmas About $\langle \prec_B \rangle$

lemma $\text{lits-less-B-conv}: \{L. \text{atm-of } L \prec_B \beta\} = (\bigcup x \in \{x. x \prec_B \beta\}. \{\text{Pos } x, \text{Neg } x\})$
 $\langle \text{proof} \rangle$

lemma $\text{lits-eq-conv}: \{L. \text{atm-of } L = \beta\} = \{\text{Pos } \beta, \text{Neg } \beta\}$
 $\langle \text{proof} \rangle$

lemma $\text{lits-less-eq-B-conv}: \{L. \text{atm-of } L \prec_B \beta \vee \text{atm-of } L = \beta\} = \text{insert } (\text{Pos } \beta) (\text{insert } (\text{Neg } \beta) \{L. \text{atm-of } L \prec_B \beta\})$
 $\langle \text{proof} \rangle$

lemma $\text{finite-lits-less-B}: \text{finite } \{L. \text{atm-of } L \prec_B \beta\}$
 $\langle \text{proof} \rangle$

lemma $\text{finite-lits-less-eq-B}: \text{finite } \{L. \text{atm-of } L \preceq_B \beta\}$
 $\langle \text{proof} \rangle$

lemma $\text{Collect-ball-eq-Pow-Collect}: \{X. \forall x \in X. P x\} = \text{Pow } \{x. P x\}$
 $\langle \text{proof} \rangle$

lemma $\text{finite-lit-cls-nodup-less-B}: \text{finite } \{C. \forall L \in \# \ C. \text{atm-of } L \prec_B \beta \wedge \text{count } C \ L = 1\}$
 $\langle \text{proof} \rangle$

5.2 Rules

inductive $\text{propagate} :: ('f, 'v) \text{ term clause fset} \Rightarrow ('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ state} \Rightarrow ('f, 'v) \text{ state} \Rightarrow \text{bool}$ **for** $N \ \beta$ **where**
 $\text{propagateI}: C \ |\in| \ N \ |\cup| \ U \implies C = \text{add-mset } L \ C' \implies \text{is-ground-cls } (C \cdot \gamma)$
 \implies

$\forall K \in \# C \cdot \gamma. \text{atm-of } K \preceq_B \beta \implies$
 $C_0 = \{\#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma\} \implies C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma\} \implies$
 $\text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \implies \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$
 $\text{is-ingu } \mu \{\text{atm-of ' set-mset (add-mset } L C_1)\} \implies$
 $\text{propagate } N \beta (\Gamma, U, \text{None}) (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, \text{None})$

lemma $C \in | N \cup U \implies C = \text{add-mset } L C' \implies \text{is-ground-cls } (C \cdot \gamma) \implies$
 $\forall K \in \# C. \text{atm-of } (K \cdot l \gamma) \preceq_B \beta \implies$
 $C_0 = \{\#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma\} \implies C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma\} \implies$
 $\text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \implies \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$
 $\text{is-ingu } \mu \{\text{atm-of ' set-mset (add-mset } L C_1)\} \implies$
 $\text{propagate } N \beta (\Gamma, U, \text{None}) (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, \text{None})$
<proof>

inductive *decide* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
(*f*, *v*) state \Rightarrow bool **for** $N \beta$ **where**
decideI: $\text{is-ground-lit } (L \cdot l \gamma) \implies$
 $\neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies \text{atm-of } L \cdot a \gamma \preceq_B \beta \implies$
 $\text{decide } N \beta (\Gamma, U, \text{None}) (\text{trail-decide } \Gamma (L \cdot l \gamma), U, \text{None})$

inductive *conflict* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
(*f*, *v*) state \Rightarrow bool **for** $N \beta$ **where**
conflictI: $D \in | N \cup U \implies \text{is-ground-cls } (D \cdot \gamma) \implies \text{trail-false-cls } \Gamma (D \cdot \gamma)$
 \implies
 $\text{conflict } N \beta (\Gamma, U, \text{None}) (\Gamma, U, \text{Some } (D, \gamma))$

inductive *skip* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
(*f*, *v*) state \Rightarrow bool **for** $N \beta$ **where**
skipI: $-L \notin \# D \cdot \sigma \implies$
 $\text{skip } N \beta ((L, n) \# \Gamma, U, \text{Some } (D, \sigma)) (\Gamma, U, \text{Some } (D, \sigma))$

lemma $-(fst \mathcal{K}) \notin \# D \cdot \sigma \implies \text{skip } N \beta (\mathcal{K} \# \Gamma, U, \text{Some } (D, \sigma)) (\Gamma, U, \text{Some } (D, \sigma))$
<proof>

inductive *factorize* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
(*f*, *v*) state \Rightarrow bool **for** $N \beta$ **where**
factorizeI: $L \cdot l \gamma = L' \cdot l \gamma \implies \text{is-ingu } \mu \{\{\text{atm-of } L, \text{atm-of } L'\}\} \implies$
 $\text{factorize } N \beta (\Gamma, U, \text{Some } (\text{add-mset } L' (\text{add-mset } L D), \gamma)) (\Gamma, U, \text{Some } (\text{add-mset } L D \cdot \mu, \gamma))$

inductive *resolve* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
(*f*, *v*) state \Rightarrow bool **for** $N \beta$ **where**
resolveI: $\Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \implies K \cdot l \gamma_D = -(L \cdot l \gamma_C) \implies$
 $\text{is-renaming } \varrho_C \implies \text{is-renaming } \varrho_D \implies$

$$\begin{aligned}
& \text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C) \cap \text{vars-cls } (\text{add-mset } K \ D \cdot \varrho_D) = \{\} \implies \\
& \text{is-ingu } \mu \{ \{ \text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D \} \} \implies \\
& \text{is-grounding-merge } \gamma \\
& \quad (\text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C)) (\text{rename-subst-domain } \varrho_C \ \gamma_C) \\
& \quad (\text{vars-cls } (\text{add-mset } K \ D \cdot \varrho_D)) (\text{rename-subst-domain } \varrho_D \ \gamma_D) \implies \\
& \text{resolve } N \ \beta \ (\Gamma, U, \text{Some } (\text{add-mset } L \ C, \gamma_C)) \ (\Gamma, U, \text{Some } ((C \cdot \varrho_C + D \cdot \\
& \varrho_D) \cdot \mu, \gamma))
\end{aligned}$$

inductive backtrack :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow (*f*, *v*) state \Rightarrow bool **for** $N \ \beta$ **where**
backtrackI: $\Gamma = \text{trail-decide } (\Gamma' @ \Gamma'') \ K \implies K = - (L \cdot l \ \sigma) \implies$
 $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } L \ D \cdot \gamma) \wedge \text{trail-false-cls } \Gamma'' (\text{add-mset } L \ D \cdot \gamma)$
 \implies
backtrack $N \ \beta \ (\Gamma, U, \text{Some } (\text{add-mset } L \ D, \sigma)) \ (\Gamma'', \text{finsert } (\text{add-mset } L \ D) \ U,$
None)

definition scl :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow (*f*, *v*) state \Rightarrow bool **where**
scl $N \ \beta \ S \ S' \longleftrightarrow \text{propagate } N \ \beta \ S \ S' \vee \text{decide } N \ \beta \ S \ S' \vee \text{conflict } N \ \beta \ S \ S' \vee$
skip $N \ \beta \ S \ S' \vee$
factorize $N \ \beta \ S \ S' \vee \text{resolve } N \ \beta \ S \ S' \vee \text{backtrack } N \ \beta \ S \ S'$

Note that, in contrast to Fiori and Weidenbach (CADE 2019), the set N of initial clauses and the ground atom β are parameters of the relation instead of being repeated twice in the states. This is to highlight the fact that they are constant.

5.3 Well-Defined

lemma propagate-well-defined:

assumes *propagate* $N \ \beta \ S \ S'$

shows

- $\neg \text{decide } N' \ \beta' \ S \ S'$
- $\neg \text{conflict } N' \ \beta' \ S \ S'$
- $\neg \text{skip } N' \ \beta' \ S \ S'$
- $\neg \text{factorize } N' \ \beta' \ S \ S'$
- $\neg \text{resolve } N' \ \beta' \ S \ S'$
- $\neg \text{backtrack } N' \ \beta' \ S \ S'$

<proof>

lemma decide-well-defined:

assumes *decide* $N \ \beta \ S \ S'$

shows

- $\neg \text{propagate } N' \ \beta' \ S \ S'$
- $\neg \text{conflict } N' \ \beta' \ S \ S'$
- $\neg \text{skip } N' \ \beta' \ S \ S'$
- $\neg \text{factorize } N' \ \beta' \ S \ S'$

\neg *resolve* $N' \beta' S S'$
 \neg *backtrack* $N' \beta' S S'$
 ⟨*proof*⟩

lemma *conflict-well-defined:*
assumes *conflict* $N \beta S S'$
shows

\neg *propagate* $N' \beta' S S'$
 \neg *decide* $N' \beta' S S'$
 \neg *skip* $N' \beta' S S'$
 \neg *factorize* $N' \beta' S S'$
 \neg *resolve* $N' \beta' S S'$
 \neg *backtrack* $N' \beta' S S'$
 ⟨*proof*⟩

lemma *skip-well-defined:*
assumes *skip* $N \beta S S'$
shows

\neg *propagate* $N' \beta' S S'$
 \neg *decide* $N' \beta' S S'$
 \neg *conflict* $N' \beta' S S'$
 \neg *factorize* $N' \beta' S S'$
 \neg *resolve* $N' \beta' S S'$
 \neg *backtrack* $N' \beta' S S'$
 ⟨*proof*⟩

lemma *factorize-well-defined:*
assumes *factorize* $N \beta S S'$
shows

\neg *propagate* $N \beta S S'$
 \neg *decide* $N \beta S S'$
 \neg *conflict* $N \beta S S'$
 \neg *skip* $N \beta S S'$

\neg *backtrack* $N \beta S S'$
 ⟨*proof*⟩

lemma *resolve-well-defined:*
assumes *resolve* $N \beta S S'$
shows

\neg *propagate* $N \beta S S'$
 \neg *decide* $N \beta S S'$
 \neg *conflict* $N \beta S S'$
 \neg *skip* $N \beta S S'$

\neg *backtrack* $N \beta S S'$
 ⟨*proof*⟩

lemma *backtrack-well-defined:*

assumes *backtrack* $N \beta S S'$
shows
 \neg *propagate* $N' \beta' S S'$
 \neg *decide* $N' \beta' S S'$
 \neg *conflict* $N' \beta' S S'$
 \neg *skip* $N' \beta' S S'$
 \neg *factorize* $N' \beta' S S'$
 \neg *resolve* $N' \beta' S S'$
 ⟨*proof*⟩

5.4 Some rules are right unique

lemma *right-unique-skip*: *right-unique* (*skip* $N \beta$)
 ⟨*proof*⟩

5.5 Miscellaneous Lemmas

lemma *conflict-set-after-factorization*:

assumes *fact*: *factorize* $N \beta S S'$ **and** *conflict-S*: *state-conflict* $S = \text{Some } (C, \gamma)$
shows $\exists C' \gamma'. \text{state-conflict } S' = \text{Some } (C', \gamma') \wedge \text{set-mset } (C \cdot \gamma) = \text{set-mset } (C' \cdot \gamma')$
 ⟨*proof*⟩

lemma *not-trail-false-ground-cls-if-no-conflict*:

assumes
no-conf: $\nexists S'. \text{conflict } N \beta S S'$ **and**
could-conf: *state-conflict* $S = \text{None}$ **and**
C-in: $C \in N \cup \text{state-learned } S$ **and**
gr-C- γ : *is-ground-cls* $(C \cdot \gamma)$
shows $\neg \text{trail-false-cls } (\text{state-trail } S) (C \cdot \gamma)$
 ⟨*proof*⟩

lemma *scl-mempty-not-in-sate-learned*:

scl $N \beta S S' \implies \{\#\} \notin \text{state-learned } S \implies \{\#\} \notin \text{state-learned } S'$
 ⟨*proof*⟩

lemma *conflict-if-mempty-in-initial-clauses-and-no-conflict*:

assumes $\{\#\} \in N$ **and** *state-conflict* $S = \text{None}$
shows *conflict* $N \beta S$ (*state-trail* S , *state-learned* S , *Some* ($\{\#\}$, *Var*))
 ⟨*proof*⟩

lemma *conflict-initial-state-if-mempty-in-intial-clauses*:

$\{\#\} \in N \implies \text{conflict } N \beta \text{ initial-state } (\[], \{\#\}, \text{Some } (\{\#\}, \text{Var}))$
 ⟨*proof*⟩

lemma *conflict-empty-trail*:

assumes *conf*: *conflict* $N \beta S S'$ **and** *empty-trail*: *state-trail* $S = \[]$
shows $\{\#\} \in N \cup \text{state-learned } S$
 ⟨*proof*⟩

lemma *conflict-empty-trail'*:

assumes $\{\#\} \mid \in \mid N \mid \cup \mid U$

shows $\exists S'. \text{conflict } N \beta (\ [], U, \text{None}) S'$

<proof>

lemma *mempty-in-iff-ex-conflict*: $\{\#\} \mid \in \mid N \mid \cup \mid U \longleftrightarrow (\exists S'. \text{conflict } N \beta (\ [], U, \text{None}) S')$

<proof>

lemma *conflict-initial-state-only-with-mempty*:

assumes *conflict* $N \beta$ *initial-state* S

shows $\exists \gamma. S = (\ [], \{\ \}, \text{Some } (\{\#\}, \gamma))$

<proof>

lemma *no-more-step-if-conflict-mempty*:

assumes *state-trail* $S = \ []$ *state-conflict* $S = \text{Some } (\{\#\}, \gamma)$

shows $\nexists S'. \text{scl } N \beta S S'$

<proof>

lemma *ex-conflict-if-trail-false-cls*:

assumes *tr-false- Γ -D*: *trail-false-cls* ΓD **and** *D-in*: $D \in \text{grounding-of-cls } (fset N \cup fset U)$

shows $\exists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S'$

<proof>

lemma *no-conflict-tail-trail*:

assumes $\nexists S. \text{conflict } N \beta (Ln \ \# \ \Gamma, U, \text{None}) S$

shows $\nexists S. \text{conflict } N \beta (\Gamma, U, \text{None}) S$

<proof>

lemma *subst-domain-rename-subst-domain-subset-vars-cls-subst-cls*:

assumes $\forall x. \text{is-Var } (\varrho_C x)$ **and**

dom- γ_C : *subst-domain* $\gamma_C \subseteq \text{vars-cls } (add-mset L C)$

shows *subst-domain* $(\text{rename-subst-domain } \varrho_C \gamma_C) \subseteq \text{vars-cls } (add-mset L C \cdot \varrho_C)$

<proof>

lemma *renamed-comp-renamed-simp*:

fixes $\gamma_C \gamma_D$

assumes

$K \cdot l \ \gamma_D = - (L \cdot l \ \gamma_C)$ **and**

ground-conf: *is-ground-cls* $(add-mset L C \cdot \gamma_C)$ **and**

ground-prop: *is-ground-cls* $(add-mset K D \cdot \gamma_D)$ **and**

dom- γ_D : *subst-domain* $\gamma_D \subseteq \text{vars-cls } (add-mset K D)$ **and**

ren- ϱ_C : *is-renaming* ϱ_C **and**

ren- ϱ_D : *is-renaming* ϱ_D **and**

disjoint-vars: $\text{vars-cls } (add-mset L C \cdot \varrho_C) \cap \text{vars-cls } (add-mset K D \cdot \varrho_D) =$

$\{\}$

defines $\gamma \equiv \text{rename-subst-domain } \varrho_D \gamma_D \odot \text{rename-subst-domain } \varrho_C \gamma_C$

shows

subst-renamed-comp-renamed-simp:

$$L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \ C \cdot \varrho_C \cdot \gamma = C \cdot \gamma_C$$

$$K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \ D \cdot \varrho_D \cdot \gamma = D \cdot \gamma_D \text{ and}$$

ingu-comp-renamed-comp-renamed-simp:

$$\text{is-ingu } \mu \ \{\{\text{atm-of } L \cdot a \ \varrho_C, \text{ atm-of } K \cdot a \ \varrho_D\}\} \implies \mu \odot \gamma = \gamma$$

<proof>

6 Invariants

6.1 Initial Literals Generalize Learned, Trail, and Conflict Literals

definition *clss-lits-generalize-clss-lits* **where**

$$\text{clss-lits-generalize-clss-lits } N \ U \longleftrightarrow$$

$$(\forall L \in \bigcup (\text{set-mset } ' U). \exists K \in \bigcup (\text{set-mset } ' N). \text{generalizes-lit } K \ L)$$

lemma *clss-lits-generalize-clss-lits-if-superset*[simp]:

assumes $N2 \subseteq N1$

shows *clss-lits-generalize-clss-lits* $N1 \ N2$

<proof>

lemma *clss-lits-generalize-clss-lits-subset:*

$$\text{clss-lits-generalize-clss-lits } N \ U1 \implies U2 \subseteq U1 \implies \text{clss-lits-generalize-clss-lits } N \ U2$$

<proof>

lemma *clss-lits-generalize-clss-lits-insert:*

$$\text{clss-lits-generalize-clss-lits } N \ (\text{insert } C \ U) \longleftrightarrow$$

$$(\forall L \in \# \ C. \exists K \in \bigcup (\text{set-mset } ' N). \text{generalizes-lit } K \ L) \wedge \text{clss-lits-generalize-clss-lits } N \ U$$

<proof>

lemma *clss-lits-generalize-clss-lits-trans:*

assumes

clss-lits-generalize-clss-lits $N1 \ N2$ **and**

clss-lits-generalize-clss-lits $N2 \ N3$

shows *clss-lits-generalize-clss-lits* $N1 \ N3$

<proof>

lemma *clss-lits-generalize-clss-lits-subst-clss:*

assumes *clss-lits-generalize-clss-lits* $N1 \ N2$

shows *clss-lits-generalize-clss-lits* $N1 \ ((\lambda C. C \cdot \sigma) ' N2)$

<proof>

lemma *clss-lits-generalize-clss-lits-singleton-subst-clss:*

$$\text{clss-lits-generalize-clss-lits } N \ \{C\} \implies \text{clss-lits-generalize-clss-lits } N \ \{C \cdot \sigma\}$$

<proof>

lemma *clss-lits-generalize-clss-lits-subst-clss*:

assumes *clss-lits-generalize-clss-lits* N $\{add-mset\ L1\ (add-mset\ L2\ C)\}$

shows *clss-lits-generalize-clss-lits* N $\{add-mset\ (L1 \cdot l\ \sigma)\ (C \cdot \sigma)\}$

<proof>

definition *initial-lits-generalize-learned-trail-conflict* **where**

initial-lits-generalize-learned-trail-conflict $N\ S \longleftrightarrow$ *clss-lits-generalize-clss-lits* (*fset* N)

(*fset* (*state-learned* S \cup | *clss-of-trail* (*state-trail* S) \cup |

(*case state-conflict* S of *None* \Rightarrow $\{\}\}$ | *Some* ($C, -$) \Rightarrow $\{|C|\}$)))

lemma *initial-lits-generalize-learned-trail-conflict-initial-state[simp]*:

initial-lits-generalize-learned-trail-conflict N *initial-state*

<proof>

lemma *propagate-preserves-initial-lits-generalize-learned-trail-conflict*:

propagate $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *decide-preserves-initial-lits-generalize-learned-trail-conflict*:

decide $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *conflict-preserves-initial-lits-generalize-learned-trail-conflict*:

assumes *conflict* $N\ \beta\ S\ S'$ **and** *initial-lits-generalize-learned-trail-conflict* $N\ S$

shows *initial-lits-generalize-learned-trail-conflict* $N\ S'$

<proof>

lemma *skip-preserves-initial-lits-generalize-learned-trail-conflict*:

skip $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *factorize-preserves-initial-lits-generalize-learned-trail-conflict*:

factorize $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *resolve-preserves-initial-lits-generalize-learned-trail-conflict*:

resolve $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *backtrack-preserves-initial-lits-generalize-learned-trail-conflict*:

backtrack $N\ \beta\ S\ S' \Longrightarrow$ *initial-lits-generalize-learned-trail-conflict* $N\ S \Longrightarrow$

initial-lits-generalize-learned-trail-conflict $N\ S'$

<proof>

lemma *scl-preserves-initial-lits-generalize-learned-trail-conflict*:
assumes *scl* $N \beta S S'$ **and** *initial-lits-generalize-learned-trail-conflict* $N S$
shows *initial-lits-generalize-learned-trail-conflict* $N S'$
 \langle *proof* \rangle

6.2 Trail Literals Are Ground

definition *trail-lits-ground* **where**
 $trail-lits-ground\ S \longleftrightarrow (\forall L \in fst\ 'set\ (state-trail\ S). is-ground-lit\ L)$

lemma *trail-lits-ground-initial-state[simp]*: *trail-lits-ground* *initial-state*
 \langle *proof* \rangle

lemma *propagate-preserves-trail-lits-ground*:
assumes *propagate* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *decide-preserves-trail-lits-ground*:
assumes *decide* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *conflict-preserves-trail-lits-ground*:
assumes *conflict* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *skip-preserves-trail-lits-ground*:
assumes *skip* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *factorize-preserves-trail-lits-ground*:
assumes *factorize* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *resolve-preserves-trail-lits-ground*:
assumes *resolve* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *backtrack-preserves-trail-lits-ground*:
assumes *backtrack* $N \beta S S'$ **and** *trail-lits-ground* S
shows *trail-lits-ground* S'
 \langle *proof* \rangle

lemma *scl-preserves-trail-lits-ground*:
assumes *scl N β S S'* **and** *trail-lits-ground S*
shows *trail-lits-ground S'*
 \langle *proof* \rangle

6.3 Trail Literals Are Defined Only Once

definition *trail-lits-consistent where*
trail-lits-consistent S \longleftrightarrow trail-consistent (state-trail S)

lemma *trail-lits-consistent-initial-state[simp]*: *trail-lits-consistent initial-state*
 \langle *proof* \rangle

lemma *propagate-preserves-trail-lits-consistent*:
assumes *propagate N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *decide-preserves-trail-lits-consistent*:
assumes *decide N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *conflict-preserves-trail-lits-consistent*:
assumes *conflict N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *skip-preserves-trail-lits-consistent*:
assumes *skip N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *factorize-preserves-trail-lits-consistent*:
assumes *factorize N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *resolve-preserves-trail-lits-consistent*:
assumes *resolve N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *backtrack-preserves-trail-lits-consistent*:
assumes *backtrack N β S S'* **and** *invar: trail-lits-consistent S*
shows *trail-lits-consistent S'*
 \langle *proof* \rangle

lemma *scl-preserves-trail-lits-consistent*:

assumes $scl\ N\ \beta\ S\ S'$ **and** *trail-lits-consistent* S
shows *trail-lits-consistent* S'
 $\langle proof \rangle$

lemma *trail-consistent-iff*: *trail-consistent* $\Gamma \longleftrightarrow (\forall \Gamma' Ln\ \Gamma''. \Gamma = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' (fst\ Ln))$
 $\langle proof \rangle$

6.4 Trail Closures Are False In Subtrails

definition *trail-closures-false'* **where**
trail-closures-false' $S \longleftrightarrow \text{trail-closures-false } (state\text{-trail } S)$

lemma *trail-closures-false'-initial-state[simp]*: *trail-closures-false'* *initial-state*
 $\langle proof \rangle$

lemma *propagate-preserves-trail-closures-false'*:
assumes *step*: *propagate* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *decide-preserves-trail-closures-false'*:
assumes *step*: *decide* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *conflict-preserves-trail-closures-false'*:
assumes *step*: *conflict* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *skip-preserves-trail-closures-false'*:
assumes *step*: *skip* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *factorize-preserves-trail-closures-false'*:
assumes *step*: *factorize* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *resolve-preserves-trail-closures-false'*:
assumes *step*: *resolve* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'
 $\langle proof \rangle$

lemma *backtrack-preserves-trail-closures-false'*:
assumes *step*: *backtrack* $N\ \beta\ S\ S'$ **and** *invar*: *trail-closures-false'* S
shows *trail-closures-false'* S'

<proof>

lemma *scl-preserves-trail-closures-false'*:
assumes *scl N β S S' and trail-closures-false' S*
shows *trail-closures-false' S'*
<proof>

lemma *trail-closures-false Γ ↔*
(∀ K D γ Γ' Γ''. Γ = Γ'' @ propagate-lit K D γ # Γ' → trail-false-cls Γ' (D ·
γ))
<proof>

6.5 Trail Literals Were Propagated or Decided

inductive *trail-propagated-or-decided* for *N β U* where

Nil[simp]: trail-propagated-or-decided N β U [] |

Propagate:

C |∈| N |∪| U ⇒
C = add-mset L C' ⇒
is-ground-cls (C · γ) ⇒
∀ K ∈ #C · γ. atm-of K ≼_B β ⇒
C₀ = {#K ∈ #C'. K · l γ ≠ L · l γ#} ⇒
C₁ = {#K ∈ #C'. K · l γ = L · l γ#} ⇒
trail-false-cls Γ (C₀ · γ) ⇒
¬ trail-defined-lit Γ (L · l γ) ⇒
is-imgu μ {atm-of 'set-mset (add-mset L C₁)} ⇒
trail-propagated-or-decided N β U Γ ⇒
trail-propagated-or-decided N β U (trail-propagate Γ (L · l μ) (C₀ · μ) γ) |

Decide:

is-ground-lit (L · l γ) ⇒
¬ trail-defined-lit Γ (L · l γ) ⇒
atm-of L · a γ ≼_B β ⇒
trail-propagated-or-decided N β U Γ ⇒
trail-propagated-or-decided N β U (trail-decide Γ (L · l γ))

lemma *trail-propagate-or-decide-suffixI*:
assumes *trail-propagated-or-decided N β U ys and suffix xs ys*
shows *trail-propagated-or-decided N β U xs*
<proof>

definition *trail-propagated-or-decided'* where

trail-propagated-or-decided' N β S =
trail-propagated-or-decided N β (state-learned S) (state-trail S)

lemma *trail-propagated-or-decided-learned-finsert*:
assumes *trail-propagated-or-decided N β U Γ*
shows *trail-propagated-or-decided N β (finsert C U) Γ*
<proof>

lemma *trail-propagated-or-decided-trail-append*:
assumes *trail-propagated-or-decided* $N \beta U (\Gamma_1 @ \Gamma_2)$
shows *trail-propagated-or-decided* $N \beta U \Gamma_2$
<proof>

lemma *trail-propagated-or-decided-initial-state[simp]*:
trail-propagated-or-decided' $N \beta \text{initial-state}$
<proof>

lemma *propagate-preserves-trail-propagated-or-decided*:
assumes *propagate* $N \beta S S'$ **and** *trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *decide-preserves-trail-propagated-or-decided*:
assumes *decide* $N \beta S S'$ **and** *trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *conflict-preserves-trail-propagated-or-decided*:
assumes *conflict* $N \beta S S'$ **and** *invar: trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *skip-preserves-trail-propagated-or-decided*:
assumes *skip* $N \beta S S'$ **and** *invar: trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *factorize-preserves-trail-propagated-or-decided*:
assumes *factorize* $N \beta S S'$ **and** *invar: trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *resolve-preserves-trail-propagated-or-decided*:
assumes *resolve* $N \beta S S'$ **and** *invar: trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *backtrack-preserves-trail-propagated-or-decided*:
assumes *backtrack* $N \beta S S'$ **and** *invar: trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

lemma *scl-preserves-trail-propagated-or-decided*:
assumes *scl* $N \beta S S'$ **and** *trail-propagated-or-decided'* $N \beta S$
shows *trail-propagated-or-decided'* $N \beta S'$
<proof>

definition *trail-propagated-wf* **where**

trail-propagated-wf $\Gamma \longleftrightarrow (\forall (L_\gamma, n) \in \text{set } \Gamma.$

case n of

None $\Rightarrow \text{True}$

| *Some* $(-, L, \gamma) \Rightarrow L_\gamma = L \cdot l \ \gamma)$

lemma *trail-propagated-wf-iff*:

trail-propagated-wf $\Gamma \longleftrightarrow (\forall Ln \in \text{set } \Gamma. \forall D K \ \gamma. \text{snd } Ln = \text{Some } (D, K, \gamma)$
 $\longrightarrow \text{fst } Ln = K \cdot l \ \gamma)$

(**is** *?lhs* \longleftrightarrow *?rhs*)

<proof>

lemma *trail-propagated-wf-if-trail-propagated-or-decided*:

trail-propagated-or-decided $N \ U \ \beta \ \Gamma \Longrightarrow \text{trail-propagated-wf } \Gamma$
<proof>

lemma *trail-propagated-wf-if-trail-propagated-or-decided'*:

trail-propagated-or-decided' $N \ \beta \ S \Longrightarrow \text{trail-propagated-wf } (\text{state-trail } S)$
<proof>

lemma *trail-propagated-lit-wf-initial-state*:

$\forall \mathcal{K} \in \text{set } (\text{state-trail initial-state}). \forall D K \ \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} =$
 $= K \cdot l \ \gamma$
<proof>

lemma *scl-preserves-trail-propagated-lit-wf*:

assumes *step*: *scl* $N \ \beta \ S \ S'$ **and**

inv: $\forall \mathcal{K} \in \text{set } (\text{state-trail } S). \forall D K \ \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} =$
 $K \cdot l \ \gamma$

shows $\forall \mathcal{K} \in \text{set } (\text{state-trail } S'). \forall D K \ \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} =$
 $K \cdot l \ \gamma$

<proof>

6.6 Trail Atoms Are Less Than Bound

definition *trail-atoms-lt* **where**

trail-atoms-lt $\beta \ S \longleftrightarrow (\forall A \in \text{atm-of 'fst ' set } (\text{state-trail } S). A \preceq_B \beta)$

lemma *trail-atoms-lt-initial-state[simp]*: *trail-atoms-lt* β *initial-state*

<proof>

lemma *propagate-preserves-trail-atoms-lt*:

assumes *propagate* $N \ \beta \ S \ S'$ **and** *trail-atoms-lt* $\beta \ S$

shows *trail-atoms-lt* $\beta \ S'$

<proof>

lemma *decide-preserves-trail-atoms-lt*:

assumes *decide* $N \ \beta \ S \ S'$ **and** *trail-atoms-lt* $\beta \ S$

shows *trail-atoms-lt* $\beta \ S'$

<proof>

lemma *conflict-preserves-trail-atoms-lt*:
assumes *conflict* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

lemma *skip-preserves-trail-atoms-lt*:
assumes *skip* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

lemma *factorize-preserves-trail-atoms-lt*:
assumes *factorize* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

lemma *resolve-preserves-trail-atoms-lt*:
assumes *resolve* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

lemma *backtrack-preserves-trail-atoms-lt*:
assumes *backtrack* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

lemma *scl-preserves-trail-atoms-lt*:
assumes *scl* $N \beta S S'$ and *trail-atoms-lt* βS
shows *trail-atoms-lt* $\beta S'$
<proof>

6.7 Trail Resolved Literals Have Unique Polarity

definition *trail-resolved-lits-pol* where
 $\text{trail-resolved-lits-pol } S \longleftrightarrow$
 $(\forall Ln \in \text{set } (\text{state-trail } S). \forall C L \gamma. \text{snd } Ln = \text{Some } (C, L, \gamma) \longrightarrow \neg(L \cdot l \gamma) \notin \#$
 $C \cdot \gamma)$

lemma *trail-resolved-lits-pol-initial-state[simp]*: *trail-resolved-lits-pol* *initial-state*
<proof>

lemma *propagate-preserves-trail-resolved-lits-pol*:
assumes *step*: *propagate* $N \beta S S'$ and *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *decide-preserves-trail-resolved-lits-pol*:
assumes *step*: *decide* $N \beta S S'$ and *invar*: *trail-resolved-lits-pol* S

shows *trail-resolved-lits-pol* S'
<proof>

lemma *conflict-preserves-trail-resolved-lits-pol*:

assumes *step*: *conflict* $N \beta S S'$ **and** *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *skip-preserves-trail-resolved-lits-pol*:

assumes *step*: *skip* $N \beta S S'$ **and** *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *factorize-preserves-trail-resolved-lits-pol*:

assumes *step*: *factorize* $N \beta S S'$ **and** *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *resolve-preserves-trail-resolved-lits-pol*:

assumes *step*: *resolve* $N \beta S S'$ **and** *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *backtrack-preserves-trail-resolved-lits-pol*:

assumes *step*: *backtrack* $N \beta S S'$ **and** *invar*: *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

lemma *scl-preserves-trail-resolved-lits-pol*:

assumes *scl* $N \beta S S'$ **and** *trail-resolved-lits-pol* S
shows *trail-resolved-lits-pol* S'
<proof>

6.8 Trail And Conflict Closures Are Ground

definition *ground-closures* **where**

ground-closures $S \longleftrightarrow$
 $(\forall Ln \in \text{set } (\text{state-trail } S). \forall C L \gamma. \text{snd } Ln = \text{Some } (C, L, \gamma) \longrightarrow \text{is-ground-cls}$
 $(\text{add-mset } L C \cdot \gamma)) \wedge$
 $(\forall C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \longrightarrow \text{is-ground-cls } (C \cdot \gamma))$

lemma *ground-closures-initial-state[simp]*: *ground-closures* *initial-state*
<proof>

lemma *propagate-preserves-ground-closures*:

assumes *step*: *propagate* $N \beta S S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
<proof>

lemma *decide-preserves-ground-closures*:
assumes *step*: $decide\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *conflict-preserves-ground-closures*:
assumes *step*: $conflict\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *skip-preserves-ground-closures*:
assumes *step*: $skip\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *factorize-preserves-ground-closures*:
assumes *step*: $factorize\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *merge-of-renamed-groundings*:
assumes
ren- ϱ_C : *is-renaming* ϱ_C **and**
ren- ϱ_D : *is-renaming* ϱ_D **and**
disjoint-vars: $vars-cl\ (C \cdot \varrho_C) \cap vars-cl\ (D \cdot \varrho_D) = \{\}$ **and**
ground-conf: *is-ground-cl* $(C \cdot \gamma_C)$ **and**
ground-prop: *is-ground-cl* $(D \cdot \gamma_D)$ **and**
merge- γ : *is-grounding-merge* γ
 $(vars-cl\ (C \cdot \varrho_C))\ (rename-subst-domain\ \varrho_C\ \gamma_C)$
 $(vars-cl\ (D \cdot \varrho_D))\ (rename-subst-domain\ \varrho_D\ \gamma_D)$
shows
 $\forall L \in \# C. L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$
 $\forall K \in \# D. K \cdot l\ \varrho_D \cdot l\ \gamma = K \cdot l\ \gamma_D$
 $\langle proof \rangle$

lemma *resolve-preserves-ground-closures*:
assumes *step*: $resolve\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *backtrack-preserves-ground-closures*:
assumes *step*: $backtrack\ N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

lemma *scl-preserves-ground-closures*:
assumes $scl\ N\ \beta\ S\ S'$ **and** *ground-closures* S
shows *ground-closures* S'
 $\langle proof \rangle$

6.9 Trail And Conflict Closures Are Ground And False

definition *ground-false-closures* where

$$\begin{aligned} \text{ground-false-closures } S &\longleftrightarrow \text{ground-closures } S \wedge \\ &\text{trail-closures-false } (\text{state-trail } S) \wedge \\ &(\forall C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cl } (\text{state-trail } S) (C \cdot \\ &\gamma)) \end{aligned}$$

lemma *ground-false-closures-initial-state[simp]*: *ground-false-closures initial-state*
<proof>

lemma *propagate-preserves-ground-false-closures*:

assumes *step*: *propagate* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *decide-preserves-ground-false-closures*:

assumes *step*: *decide* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *conflict-preserves-ground-false-closures*:

assumes *step*: *conflict* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *skip-preserves-ground-false-closures*:

assumes *step*: *skip* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *factorize-preserves-ground-false-closures*:

assumes *step*: *factorize* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *resolve-preserves-ground-false-closures*:

assumes *step*: *resolve* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *backtrack-preserves-ground-false-closures*:

assumes *step*: *backtrack* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

lemma *scl-preserves-ground-false-closures*:

assumes *scl* $N \beta S S'$ **and** *ground-false-closures* S
shows *ground-false-closures* S'
<proof>

6.10 Learned Clauses Are Non-empty

definition *learned-nonempty* where

learned-nonempty $S \longleftrightarrow \{\#\} \mid \notin \mid$ *state-learned* S

lemma *learned-nonempty-initial-state[simp]*: *learned-nonempty initial-state*
(*proof*)

lemma *propagate-preserves-learned-nonempty*:

assumes *propagate* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *decide-preserves-learned-nonempty*:

assumes *decide* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *conflict-preserves-learned-nonempty*:

assumes *conflict* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *skip-preserves-learned-nonempty*:

assumes *skip* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *factorize-preserves-learned-nonempty*:

assumes *factorize* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *resolve-preserves-learned-nonempty*:

assumes *resolve* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *backtrack-preserves-learned-nonempty*:

assumes *backtrack* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

lemma *scl-preserves-learned-nonempty*:

assumes *scl* $N \beta S S'$ **and** *learned-nonempty* S

shows *learned-nonempty* S'

(*proof*)

6.11 Backtrack Follows Conflict Resolution

definition *conflict-resolution* where

$$\text{conflict-resolution } N \beta S \longleftrightarrow (\text{state-conflict } S \neq \text{None} \longrightarrow (\exists S0 S1. \text{conflict } N \beta S0 S1 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S1 S))$$

lemma *conflict-resolution-initial-state[simp]*: *conflict-resolution* $N \beta$ *initial-state*
<proof>

lemma *propagate-preserves-conflict-resolution*:

assumes *step*: *propagate* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *decide-preserves-conflict-resolution*:

assumes *step*: *decide* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *conflict-preserves-conflict-resolution*:

assumes *step*: *conflict* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *skip-preserves-conflict-resolution*:

assumes *step*: *skip* $N \beta S S'$ **and** *invar*: *conflict-resolution* $N \beta S$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *factorize-preserves-conflict-resolution*:

assumes *step*: *factorize* $N \beta S S'$ **and** *invar*: *conflict-resolution* $N \beta S$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *resolve-preserves-conflict-resolution*:

assumes *step*: *resolve* $N \beta S S'$ **and** *invar*: *conflict-resolution* $N \beta S$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *backtrack-preserves-conflict-resolution*:

assumes *step*: *backtrack* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
<proof>

lemma *scl-preserves-conflict-resolution*:

assumes *scl* $N \beta S S'$ **and** *conflict-resolution* $N \beta S$
shows *conflict-resolution* $N \beta S'$
<proof>

6.12 Miscellaneous Lemmas

lemma *before-conflict*:

assumes *conflict* $N \beta S1 S2$ **and**

invars: *learned-nonempty* $S1$ *trail-propagated-or-decided'* $N \beta S1$

shows $\{\#\} \mid \in \mid N \vee (\exists S0. \text{propagate } N \beta S0 S1) \vee (\exists S0. \text{decide } N \beta S0 S1)$

<proof>

lemma *before-backtrack*:

assumes *backt*: *backtrack* $N \beta Sn Sm$ **and**

invar: *conflict-resolution* $N \beta Sn$

shows $\exists S0 S1. \text{conflict } N \beta S0 S1 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S1 Sn$

<proof>

lemma *ball-less-B-if-trail-false-and-trail-atoms-lt*:

trail-false-cls (*state-trail* S) $C \implies \text{trail-atoms-lt } \beta S \implies \forall L \in \# C. \text{atm-of } L$

$\preceq_B \beta$

<proof>

7 Soundness

7.1 Sound Trail

abbreviation *entails-G* (**infix** $\langle \models_{\mathcal{G}e} \rangle$ 50) **where**

entails-G $N U \equiv \text{grounding-of-cls } N \models_e \text{grounding-of-cls } U$

definition *sound-trail* **where**

sound-trail $N \Gamma \longleftrightarrow$

$(\forall Ln \in \text{set } \Gamma. \forall D K \gamma. \text{snd } Ln = \text{Some } (D, K, \gamma) \longrightarrow \text{fset } N \models_{\mathcal{G}e} \{\text{add-mset } K D\})$

lemma *sound-trail-Nil[simp]*: *sound-trail* $N \square$

<proof>

lemma *entails-G-mono*: $N \models_{\mathcal{G}e} U \implies N \subseteq NN \implies NN \models_{\mathcal{G}e} U$

<proof>

lemma *sound-trail-supersetI*: *sound-trail* $N \Gamma \implies N \mid \subseteq \mid NN \implies \text{sound-trail } NN \Gamma$

<proof>

lemma *sound-trail-ConsD*: *sound-trail* $N (Ln \# \Gamma) \implies \text{sound-trail } N \Gamma$

<proof>

lemma *sound-trail-appendD*: *sound-trail* $N (\Gamma @ \Gamma') \implies \text{sound-trail } N \Gamma'$

<proof>

lemma *sound-trail-propagate*:

assumes

sound- Γ : *sound-trail* $N \Gamma$ **and**

N-entails-C-L: *fset* $N \models_{\mathcal{G}e} \{C + \{\#L\#\}\}$

shows *sound-trail* N (*trail-propagate* $\Gamma L C \sigma$)

<proof>

lemma *sound-trail-decide*:

sound-trail $N \Gamma \implies \text{sound-trail } N$ (*trail-decide* ΓL)

<proof>

7.2 Sound State

definition *sound-state* :: (*f*, *v*) *term clause* *fset* \Rightarrow (*f*, *v*) *term* \Rightarrow (*f*, *v*) *state*
 \Rightarrow *bool* **where**

sound-state $N \beta S \longleftrightarrow$

$(\exists \Gamma U u. S = (\Gamma, U, u) \wedge \text{sound-trail } N \Gamma \wedge \text{fset } N \models_{\mathcal{G}e} \text{fset } U \wedge$
 $(\text{case } u \text{ of } \text{None} \Rightarrow \text{True} \mid \text{Some } (C, \gamma) \Rightarrow \text{fset } N \models_{\mathcal{G}e} \{C\}))$

7.3 Initial State Is Sound

lemma *sound-initial-state[simp]*: *sound-state* $N \beta$ *initial-state*

<proof>

7.4 SCL Rules Preserve Soundness

lemma *mem-vars-cls-subst-clsD*: $x' \in \text{vars-cls } (C \cdot \varrho) \implies \exists x \in \text{vars-cls } C. x' \in$
vars-term (ϱx)

<proof>

lemma *propagate-preserves-sound-state*:

assumes *step*: *propagate* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$

shows *sound-state* $N \beta S'$

<proof>

lemma *decide-preserves-sound-state*:

assumes *step*: *decide* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$

shows *sound-state* $N \beta S'$

<proof>

lemma *conflict-preserves-sound-state*:

assumes *step*: *conflict* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$

shows *sound-state* $N \beta S'$

<proof>

lemma *skip-preserves-sound-state*:

assumes *step*: *skip* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$

shows *sound-state* $N \beta S'$

<proof>

lemma *factorize-preserves-sound-state*:

assumes *step*: factorize $N \beta S S'$ **and** *sound*: sound-state $N \beta S$
shows sound-state $N \beta S'$
 ⟨*proof*⟩

lemma *resolve-preserves-sound-state*:

assumes *step*: resolve $N \beta S S'$ **and** *sound*: sound-state $N \beta S$
shows sound-state $N \beta S'$
 ⟨*proof*⟩

lemma *backtrack-preserves-sound-state*:

assumes *step*: backtrack $N \beta S S'$ **and** *sound*: sound-state $N \beta S$
shows sound-state $N \beta S'$
 ⟨*proof*⟩

theorem *scl-preserves-sound-state*:

fixes $N :: ('f, 'v) \text{Term.term clause fset}$
shows $\text{scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
 ⟨*proof*⟩

8 Strategies

definition *reasonable-scl where*

$\text{reasonable-scl } N \beta S S' \longleftrightarrow$
 $\text{scl } N \beta S S' \wedge (\text{decide } N \beta S S' \longrightarrow \neg(\exists S''. \text{conflict } N \beta S' S''))$

lemma *scl-if-reasonable*: $\text{reasonable-scl } N \beta S S' \implies \text{scl } N \beta S S'$

⟨*proof*⟩

definition *regular-scl where*

$\text{regular-scl } N \beta S S' \longleftrightarrow$
 $\text{conflict } N \beta S S' \vee \neg(\exists S''. \text{conflict } N \beta S S'') \wedge \text{reasonable-scl } N \beta S S'$

lemma *reasonable-if-regular*:

$\text{regular-scl } N \beta S S' \implies \text{reasonable-scl } N \beta S S'$
 ⟨*proof*⟩

lemma *scl-if-regular*:

$\text{regular-scl } N \beta S S' \implies \text{scl } N \beta S S'$
 ⟨*proof*⟩

The following specification of *regular-scl* is better for the paper as it highlights that it is a restriction of *reasonable-scl*.

lemma $\text{regular-scl } N \beta S S' \longleftrightarrow \text{reasonable-scl } N \beta S S' \wedge$

$((\exists S''. \text{conflict } N \beta S S'') \longrightarrow \text{conflict } N \beta S S')$

(**is** ?lhs = ?rhs)

⟨*proof*⟩

definition *ex-conflict where*

$\text{ex-conflict } C \Gamma \longleftrightarrow (\exists \gamma. \text{is-ground-cls } (C \cdot \gamma) \wedge \text{trail-false-cls } \Gamma (C \cdot \gamma))$

definition *is-shortest-backtrack* **where**

is-shortest-backtrack $C \Gamma \Gamma_0 \longleftrightarrow C \neq \{\#\} \longrightarrow \text{suffix } \Gamma_0 \Gamma \wedge \neg \text{ex-conflict } C \Gamma_0$
 \wedge
 $(\forall Kn. \text{suffix } (Kn \# \Gamma_0) \Gamma \longrightarrow \text{ex-conflict } C (Kn \# \Gamma_0))$

definition *shortest-backtrack-strategy* **where**

shortest-backtrack-strategy $R N \beta S S' \longleftrightarrow R N \beta S S' \wedge (\text{backtrack } N \beta S S' \longrightarrow$
 \longrightarrow
 $\text{is-shortest-backtrack } (\text{fst } (\text{the } (\text{state-conflict } S))) (\text{state-trail } S) (\text{state-trail } S'))$

lemma *regular-scl-if-shortest-backtrack-strategy*:

shortest-backtrack-strategy $\text{regular-scl } N \beta S S' \implies \text{regular-scl } N \beta S S'$
 $\langle \text{proof} \rangle$

lemma *strategy-restrictions*:

shows

shortest-backtrack-strategy $\text{regular-scl } N \beta S S' \implies \text{regular-scl } N \beta S S'$ **and**
 $\text{regular-scl } N \beta S S' \implies \text{reasonable-scl } N \beta S S'$ **and**
 $\text{reasonable-scl } N \beta S S' \implies \text{scl } N \beta S S'$
 $\langle \text{proof} \rangle$

primrec *shortest-backtrack* **where**

shortest-backtrack $C [] = [] \mid$
shortest-backtrack $C (Ln \# \Gamma) =$
 $(\text{if } \text{ex-conflict } C (Ln \# \Gamma) \text{ then}$
 $\text{shortest-backtrack } C \Gamma$
 else
 $Ln \# \Gamma)$

lemma *suffix-shortest-backtrack*: $\text{suffix } (\text{shortest-backtrack } C \Gamma) \Gamma$

$\langle \text{proof} \rangle$

lemma *ex-conflict-shortest-backtrack*: $\text{ex-conflict } C (\text{shortest-backtrack } C \Gamma) \longleftrightarrow$

$C = \{\#\}$
 $\langle \text{proof} \rangle$

lemma *is-shortest-backtrack-shortest-backtrack*:

$C \neq \{\#\} \implies \text{is-shortest-backtrack } C \Gamma (\text{shortest-backtrack } C \Gamma)$
 $\langle \text{proof} \rangle$

9 Monotonicity w.r.t. the Bounding Atom

lemma *scl-monotone-wrt-bound*:

assumes $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and** $\text{scl } N \beta S_0 S_1$
shows $\text{scl } N \beta' S_0 S_1$
 $\langle \text{proof} \rangle$

lemma *reasonable-scl-monotone-wrt-bound*:

assumes $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and** *reasonable-scl* $N \beta S_0 S_1$
shows *reasonable-scl* $N \beta' S_0 S_1$
 ⟨*proof*⟩

lemma *regular-scl-monotone-wrt-bound*:

assumes $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and** *regular-scl* $N \beta S_0 S_1$
shows *regular-scl* $N \beta' S_0 S_1$
 ⟨*proof*⟩

lemma *min-back-regular-scl-monotone-wrt-bound*:

assumes
 $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and**
shortest-backtrack-strategy regular-scl $N \beta S_0 S_1$
shows *shortest-backtrack-strategy regular-scl* $N \beta' S_0 S_1$
 ⟨*proof*⟩

lemma *monotonicity-wrt-bound*:

assumes $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$
shows
scl $N \beta S_0 S_1 \implies \text{scl } N \beta' S_0 S_1$ **and**
reasonable-scl $N \beta S_0 S_1 \implies \text{reasonable-scl } N \beta' S_0 S_1$ **and**
regular-scl $N \beta S_0 S_1 \implies \text{regular-scl } N \beta' S_0 S_1$ **and**
shortest-backtrack-strategy regular-scl $N \beta S_0 S_1 \implies$
shortest-backtrack-strategy regular-scl $N \beta' S_0 S_1$
 ⟨*proof*⟩

corollary

assumes
transp-on $\{A. \text{is-ground-atm } A\} (\prec_B)$ **and**
is-ground-atm β **and**
is-ground-atm β' **and**
 $\beta \prec_B \beta'$
shows
scl $N \beta S_0 S_1 \implies \text{scl } N \beta' S_0 S_1$ **and**
reasonable-scl $N \beta S_0 S_1 \implies \text{reasonable-scl } N \beta' S_0 S_1$ **and**
regular-scl $N \beta S_0 S_1 \implies \text{regular-scl } N \beta' S_0 S_1$ **and**
shortest-backtrack-strategy regular-scl $N \beta S_0 S_1 \implies$
shortest-backtrack-strategy regular-scl $N \beta' S_0 S_1$
 ⟨*proof*⟩

end

end

theory *Correct-Termination*

imports *SCL-FOL*

begin

context *scl-fol-calculus* **begin**

lemma *not-satisfiable-if-sound-state-conflict-bottom*:

assumes *sound-S*: *sound-state* $N \beta S$ **and** *conflict-S*: *state-conflict* $S = \text{Some} (\{\#\}, \gamma)$
shows \neg *satisfiable* (*grounding-of-clss* (*fset* N))
 ⟨*proof*⟩

lemma *propagate-if-conflict-follows-decide*:

assumes
trail-lt-β: *trail-atoms-lt* βS_2 **and**
no-conf: $\nexists S_1$. *conflict* $N \beta S_0 S_1$ **and** *deci*: *decide* $N \beta S_0 S_2$ **and** *conf*: *conflict* $N \beta S_2 S_3$
shows $\exists S_4$. *propagate* $N \beta S_0 S_4$
 ⟨*proof*⟩

theorem *correct-termination*:

fixes *gnd-N* **and** *gnd-N-lt-β*
assumes
sound-S: *sound-state* $N \beta S$ **and**
invars: *trail-atoms-lt* βS *trail-propagated-wf* (*state-trail* S) *trail-lits-consistent* S
ground-false-closures S **and**
no-new-conflict: $\nexists S'$. *conflict* $N \beta S S'$ **and**
no-new-propagate: $\nexists S'$. *propagate* $N \beta S S'$ **and**
no-new-decide: $\nexists S'$. *decide* $N \beta S S' \wedge (\nexists S''$. *conflict* $N \beta S' S'')$ **and**
no-new-skip: $\nexists S'$. *skip* $N \beta S S'$ **and**
no-new-resolve: $\nexists S'$. *resolve* $N \beta S S'$ **and**
no-new-backtrack: $\nexists S'$. *backtrack* $N \beta S S' \wedge$
is-shortest-backtrack (*fst* (*the* (*state-conflict* S))) (*state-trail* S) (*state-trail* S')
defines
gnd-N \equiv *grounding-of-clss* (*fset* N) **and**
gnd-N-lt-β \equiv $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
shows \neg *satisfiable* *gnd-N* $\wedge (\exists \gamma$. *state-conflict* $S = \text{Some} (\{\#\}, \gamma)) \vee$
satisfiable *gnd-N-lt-β* \wedge *trail-true-clss* (*state-trail* S) *gnd-N-lt-β*
 ⟨*proof*⟩

corollary *correct-termination-strategy*:

fixes *gnd-N* **and** *gnd-N-lt-β*
assumes
run: (*strategy* $N \beta$)** *initial-state* S **and**
no-step: $\nexists S'$. *strategy* $N \beta S S'$ **and**
strategy-restricted-by-min-back:
 $\bigwedge S S'$. *shortest-backtrack-strategy* *regular-scl* $N \beta S S' \implies$ *strategy* $N \beta S S'$
and
strategy-preserves-invars:
 $\bigwedge N \beta S S'$. *strategy* $N \beta S S' \implies$ *sound-state* $N \beta S \implies$ *sound-state* $N \beta S'$
 $\bigwedge N \beta S S'$. *strategy* $N \beta S S' \implies$ *trail-atoms-lt* $\beta S \implies$ *trail-atoms-lt* $\beta S'$

$\bigwedge N \beta S S'. \text{strategy } N \beta S S' \implies \text{trail-propagated-or-decided}' N \beta S \implies$
 $\text{trail-propagated-or-decided}' N \beta S'$
 $\bigwedge N \beta S S'. \text{strategy } N \beta S S' \implies \text{trail-lits-consistent } S \implies \text{trail-lits-consistent}$
 S'
 $\bigwedge N \beta S S'. \text{strategy } N \beta S S' \implies \text{ground-false-closures } S \implies \text{ground-false-closures}$
 S'

defines

$gnd-N \equiv \text{grounding-of-clss } (fset N)$ **and**

$gnd-N-lt-\beta \equiv \{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$

shows $\neg \text{satisfiable } gnd-N \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable } gnd-N-lt-\beta \wedge \text{trail-true-clss } (\text{state-trail } S) \text{ } gnd-N-lt-\beta$
 $\langle \text{proof} \rangle$

corollary *correct-termination-scl-run:*

fixes $gnd-N$ **and** $gnd-N-lt-\beta$

assumes

$run: (scl N \beta)^{**}$ *initial-state* S **and**

$no\text{-step}: \# S'. scl N \beta S S'$

defines

$gnd-N \equiv \text{grounding-of-clss } (fset N)$ **and**

$gnd-N-lt-\beta \equiv \{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$

shows $\neg \text{satisfiable } gnd-N \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable } gnd-N-lt-\beta \wedge \text{trail-true-clss } (\text{state-trail } S) \text{ } gnd-N-lt-\beta$
 $\langle \text{proof} \rangle$

corollary *correct-termination-reasonable-scl-run:*

fixes $gnd-N$ **and** $gnd-N-lt-\beta$

assumes

$run: (\text{reasonable-scl } N \beta)^{**}$ *initial-state* S **and**

$no\text{-step}: \# S'. \text{reasonable-scl } N \beta S S'$

defines

$gnd-N \equiv \text{grounding-of-clss } (fset N)$ **and**

$gnd-N-lt-\beta \equiv \{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$

shows $\neg \text{satisfiable } gnd-N \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable } gnd-N-lt-\beta \wedge \text{trail-true-clss } (\text{state-trail } S) \text{ } gnd-N-lt-\beta$
 $\langle \text{proof} \rangle$

corollary *correct-termination-regular-scl-run:*

fixes $gnd-N$ **and** $gnd-N-lt-\beta$

assumes

$run: (\text{regular-scl } N \beta)^{**}$ *initial-state* S **and**

$no\text{-step}: \# S'. \text{regular-scl } N \beta S S'$

defines

$gnd-N \equiv \text{grounding-of-clss } (fset N)$ **and**

$gnd-N-lt-\beta \equiv \{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$

shows $\neg \text{satisfiable } gnd-N \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable } gnd-N-lt-\beta \wedge \text{trail-true-clss } (\text{state-trail } S) \text{ } gnd-N-lt-\beta$
 $\langle \text{proof} \rangle$

corollary *correct-termination-shortest-backtrack-strategy-regular-scl:*
fixes *gnd-N* **and** *gnd-N-lt-β*
assumes
*run: (shortest-backtrack-strategy regular-scl N β)** initial-state S and*
no-step: $\nexists S'$. shortest-backtrack-strategy regular-scl N β S S'
defines
gnd-N \equiv *grounding-of-class (fset N) and*
gnd-N-lt-β \equiv $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
shows \neg *satisfiable gnd-N* \wedge $(\exists \gamma. \text{state-conflict } S = \text{Some} (\{\#\}, \gamma)) \vee$
satisfiable gnd-N-lt-β \wedge *trail-true-class (state-trail S) gnd-N-lt-β*
<proof>

corollary *correct-termination-strategies:*
fixes *gnd-N* **and** *gnd-N-lt-β*
assumes
*(scl N β)** initial-state S* \wedge $(\nexists S'. \text{scl } N \beta S S') \vee$
*(reasonable-scl N β)** initial-state S* \wedge $(\nexists S'. \text{reasonable-scl } N \beta S S') \vee$
*(regular-scl N β)** initial-state S* \wedge $(\nexists S'. \text{regular-scl } N \beta S S') \vee$
*(shortest-backtrack-strategy regular-scl N β)** initial-state S* \wedge
 $(\nexists S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S')$
defines
gnd-N \equiv *grounding-of-class (fset N) and*
gnd-N-lt-β \equiv $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
shows \neg *satisfiable gnd-N* \wedge $(\exists \gamma. \text{state-conflict } S = \text{Some} (\{\#\}, \gamma)) \vee$
satisfiable gnd-N-lt-β \wedge *trail-true-class (state-trail S) gnd-N-lt-β*
<proof>

end

end

theory *Trail-Induced-Ordering*

imports

Main

List-Index.List-Index

begin

lemma *wf-if-convertible-to-wf:*

fixes *r* $:: 'a \text{ rel}$ **and** *s* $:: 'b \text{ rel}$ **and** *f* $:: 'a \Rightarrow 'b$

assumes *wf s* **and** *convertible:* $\bigwedge x y. (x, y) \in r \Longrightarrow (f x, f y) \in s$

shows *wf r*

<proof>

lemma *wfP-if-convertible-to-wfP:* *wfP S* $\Longrightarrow (\bigwedge x y. R x y \Longrightarrow S (f x) (f y)) \Longrightarrow$

wfP R

<proof>

Converting to *nat* is a very common special case that might be found more

easily by Sledgehammer.

lemma *wfP-if-convertible-to-nat*:

fixes $f :: - \Rightarrow \text{nat}$

shows $(\bigwedge x y. R x y \Longrightarrow f x < f y) \Longrightarrow \text{wfP } R$

<proof>

definition *trail-less-id-id* **where**

trail-less-id-id $Ls L K \longleftrightarrow$

$(\exists i < \text{length } Ls. \exists j < \text{length } Ls. i > j \wedge L = Ls ! i \wedge K = Ls ! j)$

definition *trail-less-comp-id* **where**

trail-less-comp-id $Ls L K \longleftrightarrow$

$(\exists i < \text{length } Ls. \exists j < \text{length } Ls. i > j \wedge L = - (Ls ! i) \wedge K = Ls ! j)$

definition *trail-less-id-comp* **where**

trail-less-id-comp $Ls L K \longleftrightarrow$

$(\exists i < \text{length } Ls. \exists j < \text{length } Ls. i \geq j \wedge L = Ls ! i \wedge K = - (Ls ! j))$

definition *trail-less-comp-comp* **where**

trail-less-comp-comp $Ls L K \longleftrightarrow$

$(\exists i < \text{length } Ls. \exists j < \text{length } Ls. i > j \wedge L = - (Ls ! i) \wedge K = - (Ls ! j))$

definition *trail-less* **where**

trail-less $Ls L K \longleftrightarrow \text{trail-less-id-id } Ls L K \vee \text{trail-less-comp-id } Ls L K \vee$

trail-less-id-comp $Ls L K \vee \text{trail-less-comp-comp } Ls L K$

definition *trail-less'* **where**

trail-less' $Ls = (\lambda L K.$

$(\exists i. i < \text{length } Ls \wedge L = Ls ! i \wedge K = - (Ls ! i)) \vee$

$(\exists i. \text{Suc } i < \text{length } Ls \wedge L = - (Ls ! \text{Suc } i) \wedge K = Ls ! i)^{++}$

lemma *transp-trail-less'*: *transp* (*trail-less'* Ls)

<proof>

lemma *trail-less'-Suc*:

assumes $\text{Suc } i < \text{length } Ls$

shows *trail-less'* $Ls (Ls ! \text{Suc } i) (Ls ! i)$

<proof>

lemma *trail-less'-comp-Suc-comp*:

assumes $\text{Suc } i < \text{length } Ls$

shows *trail-less'* $Ls (- (Ls ! \text{Suc } i)) (- (Ls ! i))$

<proof>

lemma *trail-less'-id-id*: $j < i \Longrightarrow i < \text{length } Ls \Longrightarrow \text{trail-less}' Ls (Ls ! i) (Ls ! j)$

<proof>

lemma *trail-less'-comp-comp*:

$j < i \implies i < \text{length } Ls \implies \text{trail-less}' Ls (- (Ls ! i)) (- (Ls ! j))$
<proof>

lemma *trail-less'-id-comp*:

assumes $j < i$ **and** $i < \text{length } Ls$
shows $\text{trail-less}' Ls (Ls ! i) (- (Ls ! j))$
<proof>

lemma *trail-less'-comp-id*:

assumes $j < i$ **and** $i < \text{length } Ls$
shows $\text{trail-less}' Ls (- (Ls ! i)) (Ls ! j)$
<proof>

lemma *trail-less-eq-trail-less'*:

fixes $Ls :: 'a :: \text{uminus}$ *list*
assumes
 uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**
 uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**
 pairwise-distinct:
 $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
shows $\text{trail-less } Ls = \text{trail-less}' Ls$
<proof>

9.1 Examples

experiment

fixes $L0 L1 L2 :: 'a :: \text{uminus}$
begin

lemma *trail-less-id-comp* $[L2, L1, L0] L2 (- L2)$
<proof>

lemma *trail-less-comp-id* $[L2, L1, L0] (- L1) L2$
<proof>

lemma *trail-less-id-comp* $[L2, L1, L0] L1 (- L1)$
<proof>

lemma *trail-less-comp-id* $[L2, L1, L0] (- L0) L1$
<proof>

lemma *trail-less-id-comp* $[L2, L1, L0] L0 (- L0)$
<proof>

lemma *trail-less-id-id* $[L2, L1, L0] L1 L2$

<proof>

lemma *trail-less-id-id* [L2, L1, L0] L0 L1
<proof>

lemma *trail-less-comp-comp* [L2, L1, L0] (- L1) (- L2)
<proof>

lemma *trail-less-comp-comp* [L2, L1, L0] (- L0) (- L1)
<proof>

end

9.2 Miscellaneous Lemmas

lemma *not-trail-less-Nil*: $\neg \text{trail-less } [] L K$
<proof>

lemma *defined-if-trail-less*:
assumes *trail-less* Ls L K
shows $L \in \text{set } Ls \cup \text{uminus } ' \text{set } Ls K \in \text{set } Ls \cup \text{uminus } ' \text{set } Ls$
<proof>

lemma *not-less-if-undefined*:
fixes L :: 'a :: uminus
assumes
 uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**
 $L \notin \text{set } Ls - L \notin \text{set } Ls$
shows $\neg \text{trail-less } Ls L K \neg \text{trail-less } Ls K L$
<proof>

lemma *defined-conv*:
fixes L :: 'a :: uminus
assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $L \in \text{set } Ls \cup \text{uminus } ' \text{set } Ls \longleftrightarrow L \in \text{set } Ls \vee - L \in \text{set } Ls$
<proof>

lemma *trail-less-comp-rightI*: $L \in \text{set } Ls \implies \text{trail-less } Ls L (- L)$
<proof>

lemma *trail-less-comp-leftI*:
fixes Ls :: ('a :: uminus) list
assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $- L \in \text{set } Ls \implies \text{trail-less } Ls (- L) L$
<proof>

9.3 Well-Defined

lemma *trail-less-id-id-well-defined*:

assumes

pairwise-distinct: $\forall x \in \text{set } Ls. \forall y \in \text{set } Ls. x \neq -y$ **and**

L-le-K: *trail-less-id-id* $Ls\ L\ K$

shows

\neg *trail-less-id-comp* $Ls\ L\ K$

\neg *trail-less-comp-id* $Ls\ L\ K$

\neg *trail-less-comp-comp* $Ls\ L\ K$

<proof>

lemma *trail-less-id-comp-well-defined*:

assumes

pairwise-distinct: $\forall x \in \text{set } Ls. \forall y \in \text{set } Ls. x \neq -y$ **and**

L-le-K: *trail-less-id-comp* $Ls\ L\ K$

shows

\neg *trail-less-id-id* $Ls\ L\ K$

\neg *trail-less-comp-id* $Ls\ L\ K$

\neg *trail-less-comp-comp* $Ls\ L\ K$

<proof>

lemma *trail-less-comp-id-well-defined*:

assumes

pairwise-distinct: $\forall x \in \text{set } Ls. \forall y \in \text{set } Ls. x \neq -y$ **and**

L-le-K: *trail-less-comp-id* $Ls\ L\ K$

shows

\neg *trail-less-id-id* $Ls\ L\ K$

\neg *trail-less-id-comp* $Ls\ L\ K$

\neg *trail-less-comp-comp* $Ls\ L\ K$

<proof>

lemma *trail-less-comp-comp-well-defined*:

assumes

pairwise-distinct: $\forall x \in \text{set } Ls. \forall y \in \text{set } Ls. x \neq -y$ **and**

L-le-K: *trail-less-comp-comp* $Ls\ L\ K$

shows

\neg *trail-less-id-id* $Ls\ L\ K$

\neg *trail-less-id-comp* $Ls\ L\ K$

\neg *trail-less-comp-id* $Ls\ L\ K$

<proof>

9.4 Strict Partial Order

lemma *irreflp-trail-less*:

fixes $Ls :: ('a :: \text{uminus}) \text{ list}$

assumes

uminus-not-id: $\bigwedge x :: 'a. -x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. -(-x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq -(Ls !$

$j)$

shows *irreflp* (*trail-less* *Ls*)
 ⟨*proof*⟩

lemma *transp-trail-less*:

fixes *Ls* :: ('a :: *uminus*) *list*

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

j)

shows *transp* (*trail-less* *Ls*)

⟨*proof*⟩

lemma *asympt-trail-less*:

fixes *Ls* :: ('a :: *uminus*) *list*

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

j)

shows *asympt* (*trail-less* *Ls*)

⟨*proof*⟩

9.5 Strict Total (w.r.t. Elements in Trail) Order

lemma *totalp-on-trail-less*:

totalp-on (*set* *Ls* \cup *uminus* ' *set* *Ls*) (*trail-less* *Ls*)

⟨*proof*⟩

9.6 Well-Founded

lemma *not-trail-less-Cons-id-comp*:

fixes *Ls* :: ('a :: *uminus*) *list*

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } (L \# Ls). \forall j < \text{length } (L \# Ls). i \neq j \longrightarrow$

$(L \# Ls) ! i \neq (L \# Ls) ! j \wedge (L \# Ls) ! i \neq - ((L \# Ls) ! j)$

shows $\neg \text{trail-less } (L \# Ls) (- L) L$

⟨*proof*⟩

lemma *not-trail-less-if-undefined*:

fixes *L* :: 'a :: *uminus*

assumes

undefined: $L \notin \text{set } Ls - L \notin \text{set } Ls$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$

shows $\neg \text{trail-less } Ls L K \neg \text{trail-less } Ls K L$

<proof>

lemma *trail-less-ConsD*:

fixes $L H K :: 'a :: \text{uminus}$

assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$ **and**

L-neq-K: $L \neq K$ **and** *L-neq-minus-K*: $L \neq - K$ **and**

less-Cons: *trail-less* $(L \# Ls) H K$

shows *trail-less* $Ls H K$

<proof>

lemma *trail-subset-empty-or-ex-smallest*:

fixes $Ls :: ('a :: \text{uminus}) \text{ list}$

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

$j)$

shows $Q \subseteq \text{set } Ls \cup \text{uminus } \text{' set } Ls \implies Q = \{\} \vee (\exists z \in Q. \forall y. \text{trail-less } Ls y$
 $z \longrightarrow y \notin Q)$

<proof>

lemma *wfP-trail-less*:

fixes $Ls :: ('a :: \text{uminus}) \text{ list}$

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

$j)$

shows *wfP* (*trail-less* Ls)

<proof>

9.7 Extension on All Literals

definition *trail-less-ex* where

trail-less-ex $lt Ls L K \longleftrightarrow$

(if $L \in \text{set } Ls \vee - L \in \text{set } Ls$ then

if $K \in \text{set } Ls \vee - K \in \text{set } Ls$ then

trail-less $Ls L K$

else

True

else

if $K \in \text{set } Ls \vee - K \in \text{set } Ls$ then

False

else

$lt L K$)

lemma

fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes
 $\text{uminus-uminus-id: } \bigwedge x :: 'a. - (- x) = x$
shows $K \in \text{set } Ls \vee - K \in \text{set } Ls \implies \text{trail-less-ex } lt \ Ls \ L \ K \longleftrightarrow \text{trail-less } Ls \ L \ K$
 $\langle \text{proof} \rangle$

lemma *trail-less-ex-if-trail-less*:
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes
 $\text{uminus-uminus-id: } \bigwedge x :: 'a. - (- x) = x$
shows $\text{trail-less } Ls \ L \ K \implies \text{trail-less-ex } lt \ Ls \ L \ K$
 $\langle \text{proof} \rangle$

lemma
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes
 $\text{uminus-uminus-id: } \bigwedge x :: 'a. - (- x) = x$
shows $L \in \text{set } Ls \cup \text{uminus } ' \text{ set } Ls \implies K \notin \text{set } Ls \cup \text{uminus } ' \text{ set } Ls \implies \text{trail-less-ex } lt \ Ls \ L \ K$
 $\langle \text{proof} \rangle$

lemma *irreflp-trail-ex-less*:
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$ **and** $lt :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
assumes
 $\text{uminus-not-id: } \bigwedge x :: 'a. - x \neq x$ **and**
 $\text{uminus-uminus-id: } \bigwedge x :: 'a. - (- x) = x$ **and**
pairwise-distinct:
 $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
and
 $\text{irreflp-lt: } \text{irreflp } lt$
shows $\text{irreflp } (\text{trail-less-ex } lt \ Ls)$
 $\langle \text{proof} \rangle$

lemma *transp-trail-less-ex*:
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes
 $\text{uminus-not-id: } \bigwedge x :: 'a. - x \neq x$ **and**
 $\text{uminus-uminus-id: } \bigwedge x :: 'a. - (- x) = x$ **and**
pairwise-distinct:
 $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
and
 $\text{transp-lt: } \text{transp } lt$
shows $\text{transp } (\text{trail-less-ex } lt \ Ls)$
 $\langle \text{proof} \rangle$

lemma *asympt-trail-less-ex*:
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

j) **and**

asympt-lt: *asympt lt*

shows *asympt* (*trail-less-ex lt Ls*)

<proof>

lemma *totalp-on-trail-less-ex*:

fixes *Ls* :: ('a :: *uminus*) *list*

assumes

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

totalp-on-lt: *totalp-on A lt*

shows *totalp-on* (*A* \cup *set Ls* \cup *uminus ' set Ls*) (*trail-less-ex lt Ls*)

<proof>

9.7.1 Well-Founded

lemma *wfP-trail-less-ex*:

fixes *Ls* :: ('a :: *uminus*) *list*

assumes

uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ **and**

uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$

j) **and**

wfP-lt: *wfP lt*

shows *wfP* (*trail-less-ex lt Ls*)

<proof>

9.8 Alternative only for terms

definition *trail-term-less where*

trail-term-less ts t1 t2 $\longleftrightarrow (\exists i < \text{length } ts. \exists j < i. t1 = ts ! i \wedge t2 = ts ! j)$

lemma *transp-trail-term-less*:

assumes *distinct ts*

shows *transp* (*trail-term-less ts*)

<proof>

lemma *asympt-trail-term-less*:

assumes *distinct ts*

shows *asympt* (*trail-term-less ts*)

<proof>

lemma *irreflp-trail-term-less*:

assumes *distinct ts*

shows *irreflp* (*trail-term-less ts*)

$\langle proof \rangle$

lemma *totalp-on-trail-term-less*:
shows *totalp-on* (set *ts*) (*trail-term-less ts*)
 $\langle proof \rangle$

lemma *wfP-trail-term-less*:
assumes *distinct ts*
shows *wfP* (*trail-term-less ts*)
 $\langle proof \rangle$

lemma *trail-term-less-Cons-if-mem*:
assumes $y \in \text{set } xs$
shows *trail-term-less* ($x \# xs$) $y x$
 $\langle proof \rangle$

end
theory *Initial-Literals-Generalize-Learned-Literals*
imports *SCL-FOL*
begin

global-interpretation *comp-finsert-commute*: *comp-fun-commute finsert*
 $\langle proof \rangle$

definition *fset-mset* :: $'a \text{ multiset} \Rightarrow 'a \text{ fset}$
where *fset-mset* = *fold-mset finsert* $\{\{\}\}$

lemma *fset-mset-mempty[simp]*: *fset-mset* $\{\#\} = \{\{\}\}$
 $\langle proof \rangle$

lemma *fset-mset-add-mset[simp]*: *fset-mset* (*add-mset* $x M$) = *finsert* x (*fset-mset* M)
 $\langle proof \rangle$

lemma *fset-fset-mset[simp]*: *fset* (*fset-mset* M) = *set-mset* M
 $\langle proof \rangle$

lemma *fmember-fset-mset-iff[simp]*: $x \in | \text{fset-mset } M \iff x \in \# M$
 $\langle proof \rangle$

lemma *fBall-fset-mset-iff[simp]*: $(\forall x \in | \text{fset-mset } M. P x) \iff (\forall x \in \# M. P x)$
 $\langle proof \rangle$

lemma *fBex-fset-mset-iff[simp]*: $(\exists x \in | \text{fset-mset } M. P x) \iff (\exists x \in \# M. P x)$
 $\langle proof \rangle$

lemma *fmember-ffUnion-iff*: $a \in | \text{ffUnion } (f \mid A) \iff (\exists x \in | A. a \in | f x)$
 $\langle proof \rangle$

lemma *fBex-ffUnion-iff*: $(\exists z \mid \in \mid \text{ffUnion } (f \mid^{\dagger} A). P z) \longleftrightarrow (\exists x \mid \in \mid A. \exists z \mid \in \mid f x. P z)$
 ⟨proof⟩

lemma *fBall-ffUnion-iff*: $(\forall z \mid \in \mid \text{ffUnion } (f \mid^{\dagger} A). P z) \longleftrightarrow (\forall x \mid \in \mid A. \forall z \mid \in \mid f x. P z)$
 ⟨proof⟩

abbreviation *grounding-lits-of-clss* **where**

grounding-lits-of-clss $N \equiv \{L \cdot l \ \gamma \mid L \ \gamma. L \in \bigcup (\text{set-mset } ' N) \wedge \text{is-ground-lit } (L \cdot l \ \gamma)\}$

context *scl-fol-calculus* **begin**

corollary *grounding-lits-of-learned-subset-grounding-lits-of-initial*:

assumes *initial-lits-generalize-learned-trail-conflict* $N \ S$

shows *grounding-lits-of-clss* $(\text{fset } (\text{state-learned } S)) \subseteq \text{grounding-lits-of-clss } (\text{fset } N)$

(**is** *?lhs* \subseteq *?rhs*)

⟨proof⟩

lemma *grounding-lits-of-clss-conv*:

grounding-lits-of-clss $N = \{L \mid L \ C. \text{add-mset } L \ C \in \text{grounding-of-clss } N\}$

(**is** *?lhs* = *?rhs*)

⟨proof⟩

corollary *groundings-of-learned-subset-groundings-of-initial*:

assumes *initial-lits-generalize-learned-trail-conflict* $N \ S$

defines $U \equiv \text{state-learned } S$

shows $\{L \mid L \ C. \text{add-mset } L \ C \in \text{grounding-of-clss } (\text{fset } U)\} \subseteq$

$\{L \mid L \ C. \text{add-mset } L \ C \in \text{grounding-of-clss } (\text{fset } N)\}$

⟨proof⟩

end

end

theory *Non-Redundancy*

imports

SCL-FOL

Trail-Induced-Ordering

Initial-Literals-Generalize-Learned-Literals

begin

context *scl-fol-calculus* **begin**

10 Reasonable Steps

lemma *reasonable-scl-sound-state*:

reasonable-scl $N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
 ⟨proof⟩

lemma *reasonable-run-sound-state*:

(*reasonable-scl* $N \beta$)^{**} $S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
 ⟨proof⟩

10.1 Invariants

10.1.1 No Conflict After Decide

inductive *no-conflict-after-decide* for $N \beta U$ where

Nil[simp]: *no-conflict-after-decide* $N \beta U$ [] |

Cons: (*is-decision-lit* $Ln \longrightarrow (\nexists S'. \text{conflict } N \beta (Ln \# \Gamma, U, \text{None}) S')$) \implies
no-conflict-after-decide $N \beta U \Gamma \implies \text{no-conflict-after-decide } N \beta U (Ln \# \Gamma)$

definition *no-conflict-after-decide'* where

no-conflict-after-decide' $N \beta S = \text{no-conflict-after-decide } N \beta (\text{state-learned } S)$
 (*state-trail* S)

lemma *no-conflict-after-decide'-initial-state*[simp]: *no-conflict-after-decide'* $N \beta$ *initial-state*

⟨proof⟩

lemma *propagate-preserves-no-conflict-after-decide'*:

assumes *propagate* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$

shows *no-conflict-after-decide'* $N \beta S'$

⟨proof⟩

lemma *decide-preserves-no-conflict-after-decide'*:

assumes *decide* $N \beta S S'$ **and** $\nexists S''. \text{conflict } N \beta S' S''$ **and** *no-conflict-after-decide'* $N \beta S$

shows *no-conflict-after-decide'* $N \beta S'$

⟨proof⟩

lemma *conflict-preserves-no-conflict-after-decide'*:

assumes *conflict* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$

shows *no-conflict-after-decide'* $N \beta S'$

⟨proof⟩

lemma *skip-preserves-no-conflict-after-decide'*:

assumes *skip* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$

shows *no-conflict-after-decide'* $N \beta S'$

⟨proof⟩

lemma *factorize-preserves-no-conflict-after-decide'*:

assumes *factorize* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$

shows *no-conflict-after-decide'* $N \beta S'$

⟨proof⟩

lemma *resolve-preserves-no-conflict-after-decide'*:
assumes *resolve* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$
shows *no-conflict-after-decide'* $N \beta S'$
 \langle *proof* \rangle

lemma *learning-clause-without-conflict-preserves-no-conflict*:
fixes $N :: ('f, 'v) \text{Term.term clause fset}$
assumes $\nexists \gamma. \text{is-ground-cls } (C \cdot \gamma) \wedge \text{trail-false-cls } \Gamma (C \cdot \gamma)$
shows $\nexists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S' \implies \nexists S'. \text{conflict } N \beta (\Gamma, \text{finsert } C U, \text{None}) S'$
 \langle *proof* \rangle

lemma *backtrack-preserves-no-conflict-after-decide'*:
assumes *step: backtrack* $N \beta S S'$ **and** *invar: no-conflict-after-decide'* $N \beta S$
shows *no-conflict-after-decide'* $N \beta S'$
 \langle *proof* \rangle

lemma *reasonable-scl-preserves-no-conflict-after-decide'*:
assumes *reasonable-scl* $N \beta S S'$ **and** *no-conflict-after-decide'* $N \beta S$
shows *no-conflict-after-decide'* $N \beta S'$
 \langle *proof* \rangle

10.2 Miscellaneous Lemmas

lemma *before-reasonable-conflict*:
assumes *conf: conflict* $N \beta S1 S2$ **and**
invars: learned-nonempty $S1$ *trail-propagated-or-decided'* $N \beta S1$
no-conflict-after-decide' $N \beta S1$
shows $\{\#\} \in N \vee (\exists S0. \text{propagate } N \beta S0 S1)$
 \langle *proof* \rangle

11 Regular Steps

lemma *regular-scl-if-conflict[simp]*: *conflict* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
 \langle *proof* \rangle

lemma *regular-scl-if-skip[simp]*: *skip* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
 \langle *proof* \rangle

lemma *regular-scl-if-factorize[simp]*: *factorize* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
 \langle *proof* \rangle

lemma *regular-scl-if-resolve[simp]*: *resolve* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
 \langle *proof* \rangle

lemma *regular-scl-if-backtrack[simp]*: *backtrack* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
 \langle *proof* \rangle

lemma *regular-scl-sound-state*: *regular-scl* $N \beta S S' \implies \text{sound-state } N \beta S \implies$

sound-state $N \beta S'$
 ⟨proof⟩

lemma *regular-run-sound-state*:

(*regular-scl* $N \beta$)^{**} $S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
 ⟨proof⟩

11.1 Invariants

11.1.1 Almost No Conflict With Trail

inductive *no-conflict-with-trail* **for** $N \beta U$ **where**

Nil: $(\nexists S'. \text{conflict } N \beta ([], U, \text{None}) S') \implies \text{no-conflict-with-trail } N \beta U [] \mid$
Cons: $(\nexists S'. \text{conflict } N \beta (Ln \# \Gamma, U, \text{None}) S') \implies$
 $\text{no-conflict-with-trail } N \beta U \Gamma \implies \text{no-conflict-with-trail } N \beta U (Ln \# \Gamma)$

lemma *nex-conflict-if-no-conflict-with-trail*:

assumes *no-conflict-with-trail* $N \beta U \Gamma$
shows $\nexists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S'$
 ⟨proof⟩

lemma *nex-conflict-if-no-conflict-with-trail'*:

assumes *no-conflict-with-trail* $N \beta U \Gamma$
shows $\nexists S'. \text{conflict } N \beta ([], U, \text{None}) S'$
 ⟨proof⟩

lemma *no-conflict-after-decide-if-no-conflict-with-trail*:

no-conflict-with-trail $N \beta U \Gamma \implies \text{no-conflict-after-decide } N \beta U \Gamma$
 ⟨proof⟩

lemma *not-trail-false-cls-if-no-conflict-with-trail*:

no-conflict-with-trail $N \beta U \Gamma \implies D \mid \in \mid N \mid \cup \mid U \implies D \neq \{\#\} \implies \text{is-ground-cls}$
 $(D \cdot \gamma) \implies$
 $\neg \text{trail-false-cls } \Gamma (D \cdot \gamma)$
 ⟨proof⟩

definition *almost-no-conflict-with-trail* **where**

almost-no-conflict-with-trail $N \beta S \longleftrightarrow$
 $\{\#\} \mid \in \mid N \wedge \text{state-trail } S = [] \vee$
 $\text{no-conflict-with-trail } N \beta (\text{state-learned } S)$
 (case *state-trail* S of $[] \Rightarrow [] \mid Ln \# \Gamma \Rightarrow$ if *is-decision-lit* Ln then $Ln \# \Gamma$ else Γ)

lemma *nex-conflict-if-no-conflict-with-trail''*:

assumes *no-conf*: *state-conflict* $S = \text{None}$ **and** $\{\#\} \mid \notin \mid N$ **and** *learned-nonempty* S
 $\text{no-conflict-with-trail } N \beta (\text{state-learned } S) (\text{state-trail } S)$
shows $\nexists S'. \text{conflict } N \beta S S'$
 ⟨proof⟩

lemma *no-conflict-with-trail-if-no-conflict*:

assumes *no-conf*: $\nexists S'. \text{conflict } N \beta S S' \text{ state-conflict } S = \text{None}$

shows *no-conflict-with-trail* $N \beta (\text{state-learned } S) (\text{state-trail } S)$

<proof>

lemma *almost-no-conflict-with-trail-if-no-conflict-with-trail*:

no-conflict-with-trail $N \beta U \Gamma \implies \text{almost-no-conflict-with-trail } N \beta (\Gamma, U, Cl)$

<proof>

lemma *almost-no-conflict-with-trail-initial-state[simp]*:

almost-no-conflict-with-trail $N \beta \text{initial-state}$

<proof>

lemma *propagate-preserves-almost-no-conflict-with-trail*:

assumes *step*: *propagate* $N \beta S S'$ **and** *reg-step*: *regular-scl* $N \beta S S'$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *decide-preserves-almost-no-conflict-with-trail*:

assumes *step*: *decide* $N \beta S S'$ **and** *reg-step*: *regular-scl* $N \beta S S'$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *almost-no-conflict-with-trail-conflict-not-relevant*:

almost-no-conflict-with-trail $N \beta (\Gamma, U, Cl1) \longleftrightarrow$

almost-no-conflict-with-trail $N \beta (\Gamma, U, Cl2)$

<proof>

lemma *conflict-preserves-almost-no-conflict-with-trail*:

assumes *step*: *conflict* $N \beta S S'$ **and** *invar*: *almost-no-conflict-with-trail* $N \beta S$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *skip-preserves-almost-no-conflict-with-trail*:

assumes *step*: *skip* $N \beta S S'$ **and** *invar*: *almost-no-conflict-with-trail* $N \beta S$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *factorize-preserves-almost-no-conflict-with-trail*:

assumes *step*: *factorize* $N \beta S S'$ **and** *invar*: *almost-no-conflict-with-trail* $N \beta S$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *resolve-preserves-almost-no-conflict-with-trail*:

assumes *step*: *resolve* $N \beta S S'$ **and** *invar*: *almost-no-conflict-with-trail* $N \beta S$

shows *almost-no-conflict-with-trail* $N \beta S'$

<proof>

lemma *backtrack-preserves-almost-no-conflict-with-trail*:

assumes *step: backtrack* $N \beta S S'$ **and** *invar: almost-no-conflict-with-trail* $N \beta S$
shows *almost-no-conflict-with-trail* $N \beta S'$
 $\langle \text{proof} \rangle$

lemma *regular-scl-preserves-almost-no-conflict-with-trail:*
assumes *regular-scl* $N \beta S S'$ **and** *almost-no-conflict-with-trail* $N \beta S$
shows *almost-no-conflict-with-trail* $N \beta S'$
 $\langle \text{proof} \rangle$

11.1.2 Backtrack Follows Regular Conflict Resolution

lemma *before-conflict-in-regular-run:*
assumes
reg-run: (regular-scl $N \beta)$ ** *initial-state* $S1$ **and**
conf: conflict $N \beta S1 S2$ **and**
 $\{\#\} \notin N$
shows $\exists S0. (regular-scl\ N\ \beta)$ ** *initial-state* $S0 \wedge regular-scl\ N\ \beta\ S0\ S1 \wedge$
(propagate $N \beta S0 S1)$
 $\langle \text{proof} \rangle$

definition *regular-conflict-resolution where*
regular-conflict-resolution $N \beta S \longleftrightarrow \{\#\} \notin N \longrightarrow$
(case state-conflict S of
None $\Rightarrow (regular-scl\ N\ \beta)$ ** *initial-state* $S \mid$
Some $- \Rightarrow (\exists S0\ S1\ S2\ S3. (regular-scl\ N\ \beta)$ ** *initial-state* $S0 \wedge$
propagate $N \beta S0 S1 \wedge regular-scl\ N\ \beta\ S0\ S1 \wedge$
conflict $N \beta S1 S2 \wedge regular-scl\ N\ \beta\ S1\ S2 \wedge$
(factorize $N \beta)$ ** $S2\ S3 \wedge (regular-scl\ N\ \beta)$ ** $S2\ S3 \wedge$
 $(S3 = S \vee (\exists S4. resolve\ N\ \beta\ S3\ S4 \wedge (skip\ N\ \beta \sqcup factorize\ N\ \beta \sqcup resolve$
 $N\ \beta)$ ** $S4\ S)))$

lemma *regular-conflict-resolution-initial-state[simp]:*
regular-conflict-resolution $N \beta$ *initial-state*
 $\langle \text{proof} \rangle$

lemma *propagate-preserves-regular-conflict-resolution:*
assumes *step: propagate* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 $\langle \text{proof} \rangle$

lemma *decide-preserves-regular-conflict-resolution:*
assumes *step: decide* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 $\langle \text{proof} \rangle$

lemma *conflict-preserves-regular-conflict-resolution:*

assumes *step: conflict* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

lemma
assumes *almost-no-conflict-with-trail* $N \beta S$ **and** $\{\#\} \notin N$
shows *no-conflict-after-decide'* $N \beta S$
 \langle *proof* \rangle

lemma *mempty-not-in-learned-if-almost-no-conflict-with-trail:*
almost-no-conflict-with-trail $N \beta S \implies \{\#\} \notin N \implies \{\#\} \notin \text{state-learned } S$
 \langle *proof* \rangle

lemma *skip-preserves-regular-conflict-resolution:*
assumes *step: skip* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

lemma *factorize-preserves-regular-conflict-resolution:*
assumes *step: factorize* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

lemma *resolve-preserves-regular-conflict-resolution:*
assumes *step: resolve* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

lemma *backtrack-preserves-regular-conflict-resolution:*
assumes *step: backtrack* $N \beta S S'$ **and** *reg-step: regular-scl* $N \beta S S'$ **and**
invar: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

lemma *regular-scl-preserves-regular-conflict-resolution:*
assumes *reg-step: regular-scl* $N \beta S S'$ **and**
invars: regular-conflict-resolution $N \beta S$
shows *regular-conflict-resolution* $N \beta S'$
 \langle *proof* \rangle

11.2 Miscellaneous Lemmas

lemma *mempty-not-in-initial-clauses-if-non-empty-regular-conflict:*
assumes *state-conflict* $S = \text{Some } (C, \gamma)$ **and** $C \neq \{\#\}$ **and**
invars: almost-no-conflict-with-trail $N \beta S$ *sound-state* $N \beta S$ *ground-false-closures*

S
shows $\{\#\} \mid \notin \mid N$
 $\langle \text{proof} \rangle$

lemma *mempty-not-in-initial-clauses-if-regular-run-reaches-non-empty-conflict*:
assumes $(\text{regular-scl } N \ \beta)^{**}$ *initial-state* S **and** *state-conflict* $S = \text{Some } (C, \gamma)$
and $C \neq \{\#\}$
shows $\{\#\} \mid \notin \mid N$
 $\langle \text{proof} \rangle$

lemma *before-regular-backtrack*:
assumes
backt: $\text{backtrack } N \ \beta \ S \ S'$ **and**
invars: *sound-state* $N \ \beta \ S$ *almost-no-conflict-with-trail* $N \ \beta \ S$
regular-conflict-resolution $N \ \beta \ S$ *ground-false-closures* S
shows $\exists S_0 \ S_1 \ S_2 \ S_3 \ S_4. (\text{regular-scl } N \ \beta)^{**}$ *initial-state* $S_0 \wedge$
propagate $N \ \beta \ S_0 \ S_1 \wedge$ *regular-scl* $N \ \beta \ S_0 \ S_1 \wedge$
conflict $N \ \beta \ S_1 \ S_2 \wedge$ $(\text{factorize } N \ \beta)^{**}$ $S_2 \ S_3 \wedge$ *resolve* $N \ \beta \ S_3 \ S_4 \wedge$
 $(\text{skip } N \ \beta \sqcup \text{factorize } N \ \beta \sqcup \text{resolve } N \ \beta)^{**}$ $S_4 \ S$
 $\langle \text{proof} \rangle$

12 Resolve in Regular Runs

lemma *resolve-if-conflict-follows-propagate*:
assumes
no-conf: $\nexists S_1. \text{conflict } N \ \beta \ S_0 \ S_1$ **and**
propa: *propagate* $N \ \beta \ S_0 \ S_1$ **and**
conf: *conflict* $N \ \beta \ S_1 \ S_2$
shows $\exists S_3. \text{resolve } N \ \beta \ S_2 \ S_3$
 $\langle \text{proof} \rangle$

lemma *factorize-preserves-resolvability*:
assumes *reso*: *resolve* $N \ \beta \ S_1 \ S_2$ **and** *fact*: *factorize* $N \ \beta \ S_1 \ S_3$ **and**
invar: *ground-closures* S_1
shows $\exists S_4. \text{resolve } N \ \beta \ S_3 \ S_4$
 $\langle \text{proof} \rangle$

The following lemma corresponds to Lemma 7 in the paper.

lemma *no-backtrack-after-conflict-if*:
assumes *conf*: *conflict* $N \ \beta \ S_1 \ S_2$ **and** *trail-S2*: *state-trail* $S_1 = \text{trail-propagate}$
 $\Gamma \ L \ C \ \gamma$
shows $\nexists S_4. \text{backtrack } N \ \beta \ S_2 \ S_4$
 $\langle \text{proof} \rangle$

lemma *skip-state-trail*: *skip* $N \ \beta \ S \ S' \implies \text{suffix } (\text{state-trail } S') (\text{state-trail } S)$
 $\langle \text{proof} \rangle$

lemma *factorize-state-trail*: *factorize* $N \ \beta \ S \ S' \implies \text{state-trail } S' = \text{state-trail } S$
 $\langle \text{proof} \rangle$

lemma *resolve-state-trail*: $resolve\ N\ \beta\ S\ S' \implies state_trail\ S' = state_trail\ S$
 ⟨proof⟩

lemma *mempty-not-in-initial-clauses-if-run-leads-to-trail*:

assumes

reg-run: $(regular_scl\ N\ \beta)^{**}$ *initial-state* $S1$ **and**

trail-lit: $state_trail\ S1 = Lc\ \#\ \Gamma$

shows $\{\#\} \mid \notin \mid N$

⟨proof⟩

lemma *conflict-with-literal-gets-resolved*:

assumes

trail-lit: $state_trail\ S1 = Lc\ \#\ \Gamma$ **and**

conf: $conflict\ N\ \beta\ S1\ S2$ **and**

resolution: $(skip\ N\ \beta \sqcup factorize\ N\ \beta \sqcup resolve\ N\ \beta)^{**}$ $S2\ Sn$ **and**

backtrack: $\exists Sn'. backtrack\ N\ \beta\ Sn\ Sn'$ **and**

mempty-not-in-init-clss: $\{\#\} \mid \notin \mid N$ **and**

invars: $learned_nonempty\ S1\ trail_propagated_or_decided'\ N\ \beta\ S1\ no_conflict_after_decide'$
 $N\ \beta\ S1$

shows $\neg is_decision_lit\ Lc \wedge strict_suffix\ (state_trail\ Sn)\ (state_trail\ S1)$

⟨proof⟩

13 Clause Redundancy

definition *ground-redundant* **where**

$ground_redundant\ lt\ N\ C \longleftrightarrow \{D \in N. lt\ D\ C\} \models_e \{C\}$

definition *redundant* **where**

$redundant\ lt\ N\ C \longleftrightarrow$

$(\forall C' \in grounding_of_cls\ C. ground_redundant\ lt\ (grounding_of_class\ N)\ C')$

lemma $redundant\ lt\ N\ C \longleftrightarrow (\forall C' \in grounding_of_cls\ C. \{D' \in grounding_of_cls\ N. lt\ D'\ C'\} \models_e \{C'\})$

⟨proof⟩

lemma *ground-redundant-iff*:

$ground_redundant\ lt\ N\ C \longleftrightarrow (\exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C))$

⟨proof⟩

lemma *ground-redundant-is-ground-standard-redundancy*:

fixes lt

defines $Red_F_G \equiv \lambda N. \{C. ground_redundant\ lt\ N\ C\}$

shows $Red_F_G\ N = \{C. \exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C)\}$

⟨proof⟩

lemma *redundant-is-standard-redundancy*:

fixes $lt \mathcal{G}_F \mathcal{G}_{Fs} Red-F_G Red-F$
defines
 $\mathcal{G}_F \equiv \text{grounding-of-clcs}$ **and**
 $\mathcal{G}_{Fs} \equiv \text{grounding-of-clss}$ **and**
 $Red-F_G \equiv \lambda N. \{C. \text{ground-redundant } lt \ N \ C\}$ **and**
 $Red-F \equiv \lambda N. \{C. \text{redundant } lt \ N \ C\}$
shows $Red-F \ N = \{C. \forall D \in \mathcal{G}_F \ C. D \in Red-F_G \ (\mathcal{G}_{Fs} \ N)\}$
 $\langle \text{proof} \rangle$

lemma *ground-redundant-if-strict-subset:*
assumes $D \in N$ **and** $D \subset\# \ C$
shows $\text{ground-redundant} \ (multp_{HO} \ R) \ N \ C$
 $\langle \text{proof} \rangle$

lemma *redundant-if-strict-subset:*
assumes $D \in N$ **and** $D \subset\# \ C$
shows $\text{redundant} \ (multp_{HO} \ R) \ N \ C$
 $\langle \text{proof} \rangle$

lemma *redundant-if-strict-subsumes:*
assumes $D \cdot \sigma \subset\# \ C$ **and** $D \in N$
shows $\text{redundant} \ (multp_{HO} \ R) \ N \ C$
 $\langle \text{proof} \rangle$

lemma *ground-redundant-mono-strong:*
 $\text{ground-redundant} \ R \ N \ C \implies (\bigwedge x. x \in N \implies R \ x \ C \implies S \ x \ C) \implies \text{ground-redundant} \ S \ N \ C$
 $\langle \text{proof} \rangle$

lemma *redundant-mono-strong:*
 $\text{redundant} \ R \ N \ C \implies$
 $(\bigwedge x \ y. x \in \text{grounding-of-clss} \ N \implies y \in \text{grounding-of-clcs} \ C \implies R \ x \ y \implies S \ x$
 $y) \implies$
 $\text{redundant} \ S \ N \ C$
 $\langle \text{proof} \rangle$

lemma *redundant-multp-if-redundant-strict-subset:*
 $\text{redundant} \ (\subset\#) \ N \ C \implies \text{redundant} \ (multp_{HO} \ R) \ N \ C$
 $\langle \text{proof} \rangle$

lemma *redundant-multp-if-redundant-subset:*
 $\text{redundant} \ (\subset\#) \ N \ C \implies \text{redundant} \ (multp \ (\text{trail-less-ex } lt \ Ls)) \ N \ C$
 $\langle \text{proof} \rangle$

lemma *not-bex-subset-mset-if-not-ground-redundant:*
assumes *is-ground-clcs* C **and** *is-ground-clss* N
shows $\neg \text{ground-redundant} \ (\subset\#) \ N \ C \implies \neg (\exists D \in N. D \subset\# \ C)$
 $\langle \text{proof} \rangle$

14 Trail-Induced Ordering

14.1 Miscellaneous Lemmas

lemma *pairwise-distinct-if-trail-consistent*:

fixes Γ

defines $Ls \equiv (\text{map } \text{fst } \Gamma)$

shows $\text{trail-consistent } \Gamma \implies$

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$

$\langle \text{proof} \rangle$

14.2 Strict Partial Order

lemma *irreflp-trail-less-if-trail-consistent*:

$\text{trail-consistent } \Gamma \implies \text{irreflp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$

$\langle \text{proof} \rangle$

lemma *transp-trail-less-if-trail-consistent*:

$\text{trail-consistent } \Gamma \implies \text{transp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$

$\langle \text{proof} \rangle$

lemma *asymp-trail-less-if-trail-consistent*:

$\text{trail-consistent } \Gamma \implies \text{asymp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$

$\langle \text{proof} \rangle$

14.3 Properties

lemma *trail-defined-lit-if-trail-term-less*:

assumes $\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst } \Gamma)) (\text{atm-of } L) (\text{atm-of } K)$

shows $\text{trail-defined-lit } \Gamma L \text{ trail-defined-lit } \Gamma K$

$\langle \text{proof} \rangle$

lemma *trail-defined-cls-if-lt-defined*:

assumes $\text{consistent-}\Gamma$: $\text{trail-consistent } \Gamma$ **and**

$C\text{-lt-}D$: $\text{multp}_{HO} (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst } \Gamma)))) C D$ **and**

$\text{tr-def-}D$: $\text{trail-defined-cls } \Gamma D$ **and**

$\text{lit-less-preserves-term-order}$: $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{==} (\text{atm-of } L1)$
($\text{atm-of } L2$)

shows $\text{trail-defined-cls } \Gamma C$

$\langle \text{proof} \rangle$

15 Dynamic Non-Redundancy

lemma *regular-run-if-skip-factorize-resolve-run*:

assumes $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S S'$

shows $(\text{regular-scl } N \beta)^{**} S S'$

$\langle \text{proof} \rangle$

lemma *not-trail-true-and-false-lit*:

$\text{trail-consistent } \Gamma \implies \neg (\text{trail-true-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma L)$

<proof>

lemma *not-trail-true-and-false-cls*:

trail-consistent $\Gamma \implies \neg (\text{trail-true-cls } \Gamma \ C \wedge \text{trail-false-cls } \Gamma \ C)$

<proof>

fun *standard-lit-less* **where**

standard-lit-less $R \ (Pos \ t1) \ (Pos \ t2) = R \ t1 \ t2 \ |$

standard-lit-less $R \ (Pos \ t1) \ (Neg \ t2) = R^{==} \ t1 \ t2 \ |$

standard-lit-less $R \ (Neg \ t1) \ (Pos \ t2) = R \ t1 \ t2 \ |$

standard-lit-less $R \ (Neg \ t1) \ (Neg \ t2) = R \ t1 \ t2$

lemma *standard-lit-less-preserves-term-less*:

shows *standard-lit-less* $R \ L1 \ L2 \implies R^{==} \ (\text{atm-of } L1) \ (\text{atm-of } L2)$

<proof>

theorem *learned-clauses-in-regular-runs-invars*:

fixes Γ *lit-less*

assumes

sound-S0: *sound-state* $N \ \beta \ S0$ **and**

invars: *learned-nonempty* $S0$ *trail-propagated-or-decided'* $N \ \beta \ S0$

no-conflict-after-decide' $N \ \beta \ S0$ *almost-no-conflict-with-trail* $N \ \beta \ S0$

trail-lits-consistent $S0$ *trail-closures-false'* $S0$ *ground-false-closures* $S0$ **and**

conflict: *conflict* $N \ \beta \ S0 \ S1$ **and**

resolution: (*skip* $N \ \beta \sqcup$ *factorize* $N \ \beta \sqcup$ *resolve* $N \ \beta$)⁺⁺ $S1 \ Sn$ **and**

backtrack: *backtrack* $N \ \beta \ Sn \ Sn'$ **and**

lit-less-preserves-term-order: $\bigwedge R \ L1 \ L2. \text{lit-less } R \ L1 \ L2 \implies R^{==} \ (\text{atm-of } L1)$

(*atm-of* $L2$)

defines

$\Gamma \equiv \text{state-trail } S1$ **and**

$U \equiv \text{state-learned } S1$ **and**

trail-ord $\equiv \text{multp}_{HO} \ (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o \ \text{fst}) \ \Gamma)))$

shows $(\exists C \ \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \text{'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin (\text{fset } N \cup \text{fset } U) \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) \ C)$

<proof>

theorem *dynamic-non-redundancy-regular-scl*:

fixes Γ

assumes

regular-run: (*regular-scl* $N \ \beta$)^{**} *initial-state* $S0$ **and**

conflict: *conflict* $N \ \beta \ S0 \ S1$ **and**

resolution: (*skip* $N \ \beta \sqcup$ *factorize* $N \ \beta \sqcup$ *resolve* $N \ \beta$)⁺⁺ $S1 \ Sn$ **and**

backtrack: *backtrack* $N \ \beta \ Sn \ Sn'$ **and**

lit-less-preserves-term-order: $\bigwedge R \ L1 \ L2. \text{lit-less } R \ L1 \ L2 \implies R^{==} \ (\text{atm-of } L1)$

(*atm-of* $L2$)

defines

$\Gamma \equiv \text{state-trail } S1 \text{ and}$

$U \equiv \text{state-learned } S1 \text{ and}$

$\text{trail-ord} \equiv \text{multp}_{HO} (\text{lit-less} (\text{trail-term-less} (\text{map} (\text{atm-of } o \text{ fst}) \Gamma)))$

shows $(\text{regular-scl } N \beta)^{**} \text{ initial-state } Sn' \wedge$

$(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } ' \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin \text{fset } N \cup \text{fset } U \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord} (\text{fset } N \cup \text{fset } U) C$

$\langle \text{proof} \rangle$

theorem *dynamic-non-redundancy-projectable-strategy:*

fixes

$S1 :: ('f, 'v) \text{ state and}$

$\text{lit-less} :: (('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{bool}) \Rightarrow$

$('f, 'v) \text{ term literal} \Rightarrow ('f, 'v) \text{ term literal} \Rightarrow \text{bool and}$

$\text{strategy and strategy-init and proj}$

defines

$\Gamma \equiv \text{state-trail } S1 \text{ and}$

$U \equiv \text{state-learned } S1$

defines

$\text{trail-ord} \equiv \text{multp}_{HO} (\text{lit-less} (\text{trail-term-less} (\text{map} (\text{atm-of } o \text{ fst}) \Gamma)))$

assumes

$\text{run: strategy}^{**} \text{ strategy-init } S0 \text{ and}$

$\text{conflict: conflict } N \beta (\text{proj } S0) S1 \text{ and}$

$\text{resolution: } (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{++} S1 Sn \text{ and}$

$\text{backtrack: backtrack } N \beta Sn Sn' \text{ and}$

$\text{strategy-restricts-regular-scl:}$

$\bigwedge S S'. \text{strategy}^{**} \text{ strategy-init } S \Longrightarrow \text{strategy } S S' \Longrightarrow \text{regular-scl } N \beta (\text{proj } S) (\text{proj } S')$ **and**

$\text{initial-state: proj strategy-init} = \text{initial-state and}$

$\text{lit-less-preserves-term-order: } \bigwedge R L1 L2. \text{lit-less } R L1 L2 \Longrightarrow R^{==} (\text{atm-of } L1)$
 $(\text{atm-of } L2)$

shows $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } ' \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin \text{fset } N \cup \text{fset } U \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord} (\text{fset } N \cup \text{fset } U) C$

$\langle \text{proof} \rangle$

corollary *dynamic-non-redundancy-strategy:*

fixes Γ

assumes

$\text{run: strategy}^{**} \text{ initial-state } S0 \text{ and}$

$\text{conflict: conflict } N \beta S0 S1 \text{ and}$

$\text{resolution: } (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{++} S1 Sn \text{ and}$

backtrack: $\text{backtrack } N \beta \text{ } S_n \text{ } S_n' \text{ and}$
strategy-imp-regular-scl: $\bigwedge S S'. \text{ strategy } S S' \implies \text{ regular-scl } N \beta S S' \text{ and}$
lit-less-preserves-term-order: $\bigwedge R L1 L2. \text{ lit-less } R L1 L2 \implies R^{==} \text{ (atm-of } L1)$
(atm-of $L2$)

defines

$\Gamma \equiv \text{state-trail } S1 \text{ and}$

$U \equiv \text{state-learned } S1 \text{ and}$

$\text{trail-ord} \equiv \text{multp}_{HO} (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma)))$

shows $(\exists C \gamma. \text{state-conflict } S_n = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \text{ 'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin \text{fset } N \cup \text{fset } U \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) C$

$\langle \text{proof} \rangle$

16 Static Non-Redundancy

lemma *before-regular-backtrack'*:

assumes

run: $(\text{regular-scl } N \beta)^{**} \text{ initial-state } S \text{ and}$

step: $\text{backtrack } N \beta S S'$

shows $\exists S0 S1 S2 S3 S4. (\text{regular-scl } N \beta)^{**} \text{ initial-state } S0 \wedge$

$\text{propagate } N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$

$\text{conflict } N \beta S1 S2 \wedge (\text{factorize } N \beta)^{**} S2 S3 \wedge \text{resolve } N \beta S3 S4 \wedge$

$(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S4 S$

$\langle \text{proof} \rangle$

theorem *static-non-subsumption-regular-scl*:

assumes

run: $(\text{regular-scl } N \beta)^{**} \text{ initial-state } S \text{ and}$

step: $\text{backtrack } N \beta S S'$

defines

$U \equiv \text{state-learned } S$

shows $\exists C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \wedge \neg (\exists D \in \text{fset } N \cup \text{fset } U. \text{subsumes } D C)$

$\langle \text{proof} \rangle$

corollary *static-non-subsumption-projectable-strategy*:

fixes *strategy* **and** *strategy-init* **and** *proj*

assumes

run: $\text{strategy}^{**} \text{ strategy-init } S \text{ and}$

step: $\text{backtrack } N \beta (\text{proj } S) S' \text{ and}$

strategy-restricts-regular-scl:

$\bigwedge S S'. \text{strategy}^{**} \text{ strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta (\text{proj } S) (\text{proj } S') \text{ and}$

initial-state: $\text{proj strategy-init} = \text{initial-state}$

defines

$U \equiv \text{state-learned } (\text{proj } S)$

shows $\exists C \gamma. \text{state-conflict } (\text{proj } S) = \text{Some } (C, \gamma) \wedge \neg (\exists D |\in| N |\cup| U. \text{subsumes } D C)$
<proof>

corollary *static-non-subsumption-strategy*:

assumes

run: *strategy** initial-state S and*

step: *backtrack N β S S' and*

strategy-imp-regular-scl: $\bigwedge S S'. \text{strategy } S S' \implies \text{regular-scl } N \beta S S'$

defines

$U \equiv \text{state-learned } S$

shows $\exists C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \wedge \neg (\exists D |\in| N |\cup| U. \text{subsumes } D C)$

<proof>

end

end

theory *Wellfounded-Extra*

imports

Main

Ordered-Resolution-Prover.Lazy-List-Chain

begin

16.1 Basic Results

16.1.1 Minimal-element characterization of well-foundedness

lemma *minimal-if-wf-on*:

assumes *wf*: *wf-on A R and B ⊆ A and B ≠ {}*

shows $\exists z \in B. \forall y. (y, z) \in R \longrightarrow y \notin B$

<proof>

lemma *wfE-min'*:

$wf R \implies Q \neq \{\} \implies (\bigwedge z. z \in Q \implies (\bigwedge y. (y, z) \in R \implies y \notin Q) \implies \text{thesis})$

<proof>

lemma *wf-on-if-minimal*:

assumes $\bigwedge B. B \subseteq A \implies B \neq \{\} \implies \exists z \in B. \forall y. (y, z) \in R \longrightarrow y \notin B$

shows *wf-on A R*

<proof>

definition *inv-imagep-on* :: *'a set ⇒ ('b ⇒ 'b ⇒ bool) ⇒ ('a ⇒ 'b) ⇒ 'a ⇒ 'a ⇒ bool where*

inv-imagep-on A R f = (λx y. x ∈ A ∧ y ∈ A ∧ R (f x) (f y))

lemma *wfp-on-inv-imagep*:

assumes *wf*: *wfp-on (f ' A) R*

shows *wfp-on A (inv-imagep R f)*

<proof>

definition *lex-prodp* **where**

$lex-prodp\ R_A\ R_B\ x\ y \longleftrightarrow R_A\ (fst\ x)\ (fst\ y) \vee fst\ x = fst\ y \wedge R_B\ (snd\ x)\ (snd\ y)$

lemma *lex-prodp-lex-prod-iff*[*pred-set-conv*]:

$lex-prodp\ R_A\ R_B\ x\ y \longleftrightarrow (x, y) \in lex-prod\ \{(x, y). R_A\ x\ y\}\ \{(x, y). R_B\ x\ y\}$

<proof>

lemma *lex-prod-lex-prodp-iff*:

$lex-prod\ \{(x, y). R_A\ x\ y\}\ \{(x, y). R_B\ x\ y\} = \{(x, y). lex-prodp\ R_A\ R_B\ x\ y\}$

<proof>

lemma *wfp-on-lex-prodp*: $wfp-on\ A\ R_A \implies wfp-on\ B\ R_B \implies wfp-on\ (A \times B)$

(*lex-prodp* $R_A\ R_B$)

<proof>

corollary *wfp-lex-prodp*: $wfp\ R_A \implies wfp\ R_B \implies wfp\ (lex-prodp\ R_A\ R_B)$

<proof>

lemma *wfp-on-sup-if-convertible-to-wfp*:

includes *lattice-syntax*

assumes

wf-S: $wfp-on\ A\ S$ **and**

wf-Q: $wfp-on\ (f\ 'A)\ Q$ **and**

convertible-R: $\bigwedge x\ y. x \in A \implies y \in A \implies R\ x\ y \implies Q\ (f\ x)\ (f\ y)$ **and**

convertible-S: $\bigwedge x\ y. x \in A \implies y \in A \implies S\ x\ y \implies Q\ (f\ x)\ (f\ y) \vee f\ x = f\ y$

shows $wfp-on\ A\ (R \sqcup S)$

<proof>

lemma *chain-lnth-rtranclp*:

assumes

chain: *Lazy-List-Chain.chain* $R\ xs$ **and**

len: $enat\ j < llength\ xs$

shows $R^{**}\ (lhd\ xs)\ (lnth\ xs\ j)$

<proof>

lemma *chain-conj-rtranclpI*:

fixes $xs :: 'a\ llist$

assumes *Lazy-List-Chain.chain* $(\lambda x\ y. R\ x\ y)\ (LCons\ init\ xs)$

shows *Lazy-List-Chain.chain* $(\lambda x\ y. R\ x\ y \wedge R^{**}\ init\ x)\ (LCons\ init\ xs)$

<proof>

lemma *rtranclp-implies-ex-lfinite-chain*:

assumes *run*: $R^{**}\ x_0\ x$

shows $\exists xs. lfinite\ xs \wedge chain\ (\lambda y\ z. R\ y\ z \wedge R^{**}\ x_0\ y)\ (LCons\ x_0\ xs) \wedge llast\ (LCons\ x_0\ xs) = x$

<proof>

lemma *chain-conj-rtranclpD*:

fixes $xs :: 'a\ list$

assumes $inf: \neg\ lfinite\ xs$ **and** $chain: chain\ (\lambda y\ z.\ R\ y\ z\ \wedge\ R^{**}\ x_0\ y)\ xs$

shows $\exists\ ys.\ lfinite\ ys\ \wedge\ chain\ (\lambda y\ z.\ R\ y\ z\ \wedge\ R^{**}\ x_0\ y)\ (lappend\ ys\ xs) \wedge\ lhd\ (lappend\ ys\ xs) = x_0$

<proof>

lemma *wfp-on-rtranclp-conversep-iff-no-infinite-down-chain-llist*:

fixes $R\ x_0$

shows $wfp\text{-on}\ \{x.\ R^{**}\ x_0\ x\}\ R^{-1-1} \longleftrightarrow (\nexists\ xs.\ \neg\ lfinite\ xs \wedge\ Lazy\text{-List}\text{-Chain.}\ chain\ R\ (LCons\ x_0\ xs))$

<proof>

end

theory *Termination*

imports

SCL-FOL

Non-Redundancy

Wellfounded-Extra

HOL-Library.Monad-Syntax

begin

17 Extra Lemmas

17.1 Set Extra

lemma *minus-psubset-minusI*:

assumes $C \subset B$ **and** $B \subseteq A$

shows $(A - B \subset A - C)$

<proof>

17.2 Prod Extra

lemma *lex-prod-lex-prodp-eq*:

$lex\text{-prod}\ \{(x,\ y).\ RA\ x\ y\}\ \{(x,\ y).\ RB\ x\ y\} = \{(x,\ y).\ lex\text{-prodp}\ RA\ RB\ x\ y\}$

<proof>

lemma *reflp-on-lex-prodp*:

assumes $reflp\text{-on}\ A\ RA$

shows $reflp\text{-on}\ (A \times B)\ (lex\text{-prodp}\ RA\ RB)$

<proof>

lemma *transp-lex-prodp*:

assumes $transp\ RA$ **and** $transp\ RB$

shows $transp\ (lex\text{-prodp}\ RA\ RB)$

<proof>

lemma *asympt-lex-prodp*:

assumes $asympt\ RA$ **and** $asympt\ RB$

shows *asympt* (*lex-prodp* *RA RB*)
 ⟨*proof*⟩

lemma *totalp-on-lex-prodp*:
assumes *totalp-on A RA* **and** *totalp-on B RB*
shows *totalp-on (A × B) (lex-prodp RA RB)*
 ⟨*proof*⟩

17.3 FSet Extra

lemma *finsert-Abs-fset*: *finite A* \implies *finsert a (Abs-fset A) = Abs-fset (insert a A)*
 ⟨*proof*⟩

lemma *minus-pfsubset-minusI*:
assumes *C |⊂| B* **and** *B |⊆| A*
shows *(A |−| B |⊂| A |−| C)*
 ⟨*proof*⟩

lemma *Abs-fset-minus*: *finite A* \implies *finite B* \implies *Abs-fset (A − B) = Abs-fset A*
|−| Abs-fset B
 ⟨*proof*⟩

lemma *fminus-conv*: *A |⊂| B* \longleftrightarrow *fset A ⊂ fset B* \wedge *finite (fset A)* \wedge *finite (fset B)*
 ⟨*proof*⟩

18 Termination

context *scl-fol-calculus* **begin**

18.1 SCL without backtracking terminates

definition *M-prop-deci* :: $- \Rightarrow - \Rightarrow (-, -)$ *Term.term literal fset* **where**
M-prop-deci $\beta \Gamma = \text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\} \text{ |−| } (\text{fst } |^{\uparrow} \text{fset-of-list } \Gamma)$

primrec *M-skip-fact-reso* **where**
M-skip-fact-reso $\square C = \square \text{ |}$
M-skip-fact-reso $(Ln \# \Gamma) C =$
 (let $n = \text{count } C \text{ } (- \text{ (fst } Ln))$ in
 (case *snd Ln* of *None* $\Rightarrow 0$ | *Some -* $\Rightarrow n$) $\#$
M-skip-fact-reso $\Gamma (C + (\text{case } \text{snd } Ln \text{ of } \text{None} \Rightarrow \{\#\} \text{ | } \text{Some } (D, -, \gamma) \Rightarrow$
repeat-mset $n (D \cdot \gamma))))$

fun *M-skip-fact-reso'* **where**
M-skip-fact-reso' $C \square = \square \text{ |}$
M-skip-fact-reso' $C ((-, \text{None}) \# \Gamma) = 0 \# \text{M-skip-fact-reso}' C \Gamma \text{ |}$
M-skip-fact-reso' $C ((K, \text{Some } (D, -, \gamma)) \# \Gamma) =$
 (let $n = \text{count } C \text{ } (- K)$ in $n \# \text{M-skip-fact-reso}' (C + \text{repeat-mset } n (D \cdot \gamma))$
 $\Gamma)$

lemma $\mathcal{M}\text{-skip-fact-reso } \Gamma C = \mathcal{M}\text{-skip-fact-reso}' C \Gamma$

$\langle \text{proof} \rangle$

lemma $\mathcal{M}\text{-skip-fact-reso}' C (\text{decide-lit } K \# \Gamma) = 0 \# \mathcal{M}\text{-skip-fact-reso}' C \Gamma$

$\langle \text{proof} \rangle$

lemma $\mathcal{M}\text{-skip-fact-reso}' C (\text{propagate-lit } K D \gamma \# \Gamma) =$

$(\text{let } n = \text{count } C \text{ } (- (K \cdot l \gamma)) \text{ in } n \# \mathcal{M}\text{-skip-fact-reso}' (C + \text{repeat-mset } n (D \cdot \gamma)) \Gamma)$

$\langle \text{proof} \rangle$

fun $\mathcal{M} :: - \Rightarrow ('f, 'v) \text{ state} \Rightarrow$

$\text{bool} \times ('f, 'v) \text{ Term.term literal fset} \times \text{nat list} \times \text{nat}$ **where**

$\mathcal{M} \beta (\Gamma, U, \text{None}) = (\text{True}, \mathcal{M}\text{-prop-deci } \beta \Gamma, [], 0) \mid$

$\mathcal{M} \beta (\Gamma, U, \text{Some } (C, \gamma)) = (\text{False}, \{\|\}, \mathcal{M}\text{-skip-fact-reso } \Gamma (C \cdot \gamma), \text{size } C)$

lemma $\text{length-}\mathcal{M}\text{-skip-fact-reso}[\text{simp}]: \text{length } (\mathcal{M}\text{-skip-fact-reso } \Gamma C) = \text{length } \Gamma$

$\langle \text{proof} \rangle$

lemma $\mathcal{M}\text{-skip-fact-reso-add-mset}:$

$(\mathcal{M}\text{-skip-fact-reso } \Gamma C, \mathcal{M}\text{-skip-fact-reso } \Gamma (\text{add-mset } L C)) \in (\text{List.lenlex } \{(x, y). x < y\})^=$

$\langle \text{proof} \rangle$

lemma $\text{termination-scl-without-back-invars}:$

fixes $N \beta$

defines

$\text{scl-without-backtrack} \equiv \text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta \sqcup$

$\text{factorize } N \beta \sqcup \text{resolve } N \beta$ **and**

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$

$\text{initial-lits-generalize-learned-trail-conflict } N \sqcap \text{ground-closures}$

shows $\text{wfp-on } \{S. \text{invars } S\} \text{scl-without-backtrack}^{-1-1}$

$\langle \text{proof} \rangle$

corollary $\text{termination-scl-without-back}:$

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**

$\beta :: ('f, 'v) \text{ Term.term}$

defines

$\text{scl-without-backtrack} \equiv \text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta \sqcup$

$\text{factorize } N \beta \sqcup \text{resolve } N \beta$ **and**

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$

$\text{initial-lits-generalize-learned-trail-conflict } N \sqcap \text{ground-closures}$

shows $\text{wfp-on } \{S. \text{scl-without-backtrack}^{**} \text{initial-state } S\} \text{scl-without-backtrack}^{-1-1}$

$\langle \text{proof} \rangle$

corollary *termination-strategy-without-back*:

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**

$\beta :: ('f, 'v) \text{ Term.term}$

defines

$scl\text{-without-backtrack} \equiv \text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta \sqcup$

$\text{factorize } N \beta \sqcup \text{resolve } N \beta$

assumes *strategy-stronger*: $\bigwedge S S'. \text{strategy } S S' \implies scl\text{-without-backtrack } S S'$

shows *wfp-on* $\{S. \text{strategy}^{**} \text{ initial-state } S\} \text{strategy}^{-1-1}$

<proof>

18.2 Backtracking can only be done finitely often

definition *fclss-no-dup* $:: ('f, 'v) \text{ Term.term} \Rightarrow ('f, 'v) \text{ Term.term literal fset fset}$

where

$fclss\text{-no-dup } \beta = fPow (\text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\})$

lemma *image-fset-fset-fPow-eq*: $fset ' fset (fPow A) = Pow (fset A)$

<proof>

lemma

assumes $\forall L \in \# C. \text{count } C L = 1$

shows $\exists C'. C = \text{mset-set } C'$

<proof>

lemma *fmember-fclss-no-dup-if*:

assumes $\forall L \in | C. \text{atm-of } L \preceq_B \beta$

shows $C \in | fclss\text{-no-dup } \beta$

<proof>

definition *M-back* $:: - \Rightarrow ('f, 'v) \text{ state} \Rightarrow ('f, 'v) \text{ Term.term literal fset fset}$

where

$M\text{-back } \beta S = \text{Abs-fset } (fset (fclss\text{-no-dup } \beta) -$

$\text{Abs-fset ' set-mset ' grounding-of-clss } (fset (\text{state-learned } S)))$

lemma *M-back-after-regular-backtrack*:

assumes

regular-run: $(\text{regular-scl } N \beta)^{**} \text{ initial-state } S0$ **and**

conflict: $\text{conflict } N \beta S0 S1$ **and**

resolution: $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{++} S1 Sn$ **and**

backtrack: $\text{backtrack } N \beta Sn Sn'$

defines $U \equiv \text{state-learned } Sn$

shows

$\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset ' grounding-of-clss } (fset N \cup fset U)$ **and**

$M\text{-back } \beta Sn' \sqsubset | M\text{-back } \beta Sn$

<proof>

18.3 Regular SCL terminates

theorem *termination-regular-scl-invars:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**
 $\beta :: ('f, 'v) \text{ Term.term}$

defines

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$
 $\text{initial-lits-generalize-learned-trail-conflict } N \sqcap \text{ground-closures} \sqcap \text{ground-false-closures}$

\sqcap

$\text{sound-state } N \beta \sqcap \text{almost-no-conflict-with-trail } N \beta \sqcap \text{regular-conflict-resolution}$

$N \beta$

shows

$\text{wfp-on } \{S. \text{invars } S\} (\text{regular-scl } N \beta)^{-1-1}$

<proof>

corollary *termination-regular-scl:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**
 $\beta :: ('f, 'v) \text{ Term.term}$

defines

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$
 $\text{initial-lits-generalize-learned-trail-conflict } N \sqcap \text{ground-closures} \sqcap \text{ground-false-closures}$

\sqcap

$\text{sound-state } N \beta \sqcap \text{almost-no-conflict-with-trail } N \beta \sqcap \text{regular-conflict-resolution}$

$N \beta$

shows $\text{wfp-on } \{S. (\text{regular-scl } N \beta)^{**} \text{initial-state } S\} (\text{regular-scl } N \beta)^{-1-1}$

<proof>

corollary *termination-projectable-strategy:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**
 $\beta :: ('f, 'v) \text{ Term.term}$ **and**

strategy **and** strategy-init **and** proj

assumes $\text{strategy-restricts-regular-scl:}$

$\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta (\text{proj } S)$

$(\text{proj } S')$ **and**

$\text{initial-state: proj strategy-init} = \text{initial-state}$

shows $\text{wfp-on } \{S. \text{strategy}^{**} \text{strategy-init } S\} \text{strategy}^{-1-1}$

<proof>

corollary *termination-strategy:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**
 $\beta :: ('f, 'v) \text{ Term.term}$

assumes $\text{strategy-restricts-regular-scl: } \bigwedge S S'. \text{strategy } S S' \implies \text{regular-scl } N \beta$
 $S S'$

shows $\text{wfp-on } \{S. \text{strategy}^{**} \text{initial-state } S\} \text{strategy}^{-1-1}$

<proof>

end

end

theory *Completeness*

imports

Correct-Termination

Termination

Functional-Ordered-Resolution-Prover.IsaFoR-Term

begin

lemma (**in** *scl-fol-calculus*) *regular-scl-run-derives-contradiction-if-unsat*:

fixes $N \beta \text{gnd-}N$

defines

$\text{gnd-}N \equiv \text{grounding-of-clss } (fset \ N) \ \mathbf{and}$

$\text{gnd-}N\text{-lt-}\beta \equiv \{C \in \text{gnd-}N. \forall L \in \# \ C. \text{atm-of } L \preceq_B \beta\}$

assumes

$\text{unsat}: \neg \text{satisfiable } \text{gnd-}N\text{-lt-}\beta \ \mathbf{and}$

$\text{run}: (\text{regular-scl } N \ \beta)^{**} \text{ initial-state } S \ \mathbf{and}$

$\text{no-more-step}: \nexists S'. \text{regular-scl } N \ \beta \ S \ S'$

shows $\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)$

<proof>

theorem (**in** *scl-fol-calculus*)

fixes $N \beta \text{gnd-}N$

defines

$\text{gnd-}N \equiv \text{grounding-of-clss } (fset \ N) \ \mathbf{and}$

$\text{gnd-}N\text{-lt-}\beta \equiv \{C \in \text{gnd-}N. \forall L \in \# \ C. \text{atm-of } L \preceq_B \beta\}$

assumes $\text{unsat}: \neg \text{satisfiable } \text{gnd-}N\text{-lt-}\beta$

shows $\exists S. (\text{regular-scl } N \ \beta)^{**} \text{ initial-state } S \wedge$

$(\nexists S'. \text{regular-scl } N \ \beta \ S \ S') \wedge$

$(\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma))$

<proof>

lemma (**in** *scl-fol-calculus*) *no-infinite-down-chain*:

$\nexists Ss. \neg \text{ifinite } Ss \wedge \text{Lazy-List-Chain.chain } (\lambda S \ S'. \text{regular-scl } N \ \beta \ S \ S') \ (\text{LCons initial-state } Ss)$

<proof>

theorem (**in** *scl-fol-calculus*) *completeness-wrt-bound*:

fixes $N \beta \text{gnd-}N$

defines

$\text{gnd-}N \equiv \text{grounding-of-clss } (fset \ N) \ \mathbf{and}$

$\text{gnd-}N\text{-lt-}\beta \equiv \{C \in \text{gnd-}N. \forall L \in \# \ C. \text{atm-of } L \preceq_B \beta\}$

assumes $\text{unsat}: \neg \text{satisfiable } \text{gnd-}N\text{-lt-}\beta$

shows

$\nexists Ss. \neg \text{ifinite } Ss \wedge \text{Lazy-List-Chain.chain } (\lambda S \ S'. \text{regular-scl } N \ \beta \ S \ S')$

$(\text{LCons initial-state } Ss) \ \mathbf{and}$

$\forall S. (\text{regular-scl } N \ \beta)^{**} \text{ initial-state } S \longrightarrow (\nexists S'. \text{regular-scl } N \ \beta \ S \ S') \longrightarrow$

$(\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma))$

<proof>

locale *compact-scl* =

scl-fol-calculus renaming-vars ($<$) :: ($'f$:: *weighted*, $'v$) *term* \Rightarrow ($'f$, $'v$) *term* \Rightarrow *bool*

for *renaming-vars* :: $'v$ *set* \Rightarrow $'v \Rightarrow 'v$

begin

theorem *ex-bound-if-unsat*:

fixes N :: ($'f$, $'v$) *term clause fset*

defines

$gnd-N \equiv$ *grounding-of-cls* ($fset$ N)

assumes *unsat*: \neg *satisfiable* $gnd-N$

shows $\exists \beta. \neg$ *satisfiable* $\{C \in gnd-N. \forall L \in \# C. atm-of L \leq \beta\}$

<proof>

end

end

theory *Invariants*

imports *SCL-FOL*

begin

The following lemma restate existing invariants in a compact, paper-friendly way.

lemma (in *scl-fol-calculus*) *scl-state-invariants*:

shows

inv-trail-lits-ground:

trail-lits-ground *initial-state*

$scl N \beta S S' \Longrightarrow trail-lits-ground S \Longrightarrow trail-lits-ground S'$ **and**

inv-trail-atoms-lt:

trail-atoms-lt β *initial-state*

$scl N \beta S S' \Longrightarrow trail-atoms-lt \beta S \Longrightarrow trail-atoms-lt \beta S'$ **and**

inv-undefined-trail-lits:

$\forall \Gamma' Ln \Gamma''. state-trail initial-state = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg trail-defined-lit \Gamma'$
(*fst Ln*)

$scl N \beta S S' \Longrightarrow$

$(\forall \Gamma' Ln \Gamma''. state-trail S = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg trail-defined-lit \Gamma' (fst Ln))$

\Longrightarrow

$(\forall \Gamma' Ln \Gamma''. state-trail S' = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg trail-defined-lit \Gamma' (fst Ln))$ **and**

inv-ground-closures:

ground-closures *initial-state*

$scl N \beta S S' \Longrightarrow ground-closures S \Longrightarrow ground-closures S'$ **and**

inv-ground-false-closures:

ground-false-closures *initial-state*

$scl N \beta S S' \Longrightarrow ground-false-closures S \Longrightarrow ground-false-closures S'$ **and**

inv-trail-propagated-lits-wf:

$\forall \mathcal{K} \in \text{set}(\text{state-trail initial-state}). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some}(D, K, \gamma) \longrightarrow \text{fst}$
 $\mathcal{K} = K \cdot l \gamma$
 $\text{scl } N \beta S S' \Longrightarrow$
 $(\forall \mathcal{K} \in \text{set}(\text{state-trail } S). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some}(D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} = K$
 $\cdot l \gamma) \Longrightarrow$
 $(\forall \mathcal{K} \in \text{set}(\text{state-trail } S'). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some}(D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} = K$
 $\cdot l \gamma)$ **and**
inv-trail-resolved-lits-pol:
trail-resolved-lits-pol initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{trail-resolved-lits-pol } S \Longrightarrow \text{trail-resolved-lits-pol } S'$ **and**
inv-initial-lits-generalize-learned-trail-conflict:
initial-lits-generalize-learned-trail-conflict N initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{initial-lits-generalize-learned-trail-conflict } N S \Longrightarrow \text{ini-$
 $\text{tial-lits-generalize-learned-trail-conflict } N S'$ **and**
inv-sound-state:
sound-state N β initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{sound-state } N \beta S \Longrightarrow \text{sound-state } N \beta S'$
 ⟨proof⟩

end

References

- [1] M. Bromberger, S. Schwarz, and C. Weidenbach. SCL(FOL) revisited, 2023.
- [2] A. Fiori and C. Weidenbach. SCL clause learning from simple models. In P. Fontaine, editor, *Automated Deduction – CADE 27*, volume 11716 of *Lecture Notes in Artificial Intelligence*, pages 233–249, Natal, Brazil, 2019. Springer.