

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	6
3.4	List Extra	6
3.5	Sublist Extra	7
3.6	Multiset Extra	7
3.6.1	Calculus Extra	8
3.7	Clausal Calculus Extra	8
3.7.1	Clausal Calculus Only	8
3.7.2	Clausal Calculus and Abstract Substitution	8
3.8	First Order Terms Extra	9
3.8.1	First Order Terms Only	9
3.8.2	First Order Terms And Abstract Substitution	12

3.8.3	Minimal, Idempotent Most General Unifier	21
3.8.4	Renaming Extra	23
4	SCL State	24
5	SCL(FOL) Calculus	33
5.1	Lemmas About (\prec_B)	34
5.2	Rules	35
5.3	Well-Defined	36
5.4	Some rules are right unique	41
5.5	Miscellaneous Lemmas	41
6	Invariants	48
6.1	Initial Literals Generalize Learned, Trail, and Conflict Literals	48
6.2	Trail Literals Are Ground	54
6.3	Trail Literals Are Defined Only Once	56
6.4	Trail Closures Are False In Subtrails	58
6.5	Trail Literals Were Propagated or Decided	61
6.6	Trail Atoms Are Less Than Bound	68
6.7	Trail Resolved Literals Have Unique Polarity	70
6.8	Trail And Conflict Closures Are Ground	72
6.9	Trail And Conflict Closures Are Ground And False	77
6.10	Learned Clauses Are Non-empty	82
6.11	Backtrack Follows Conflict Resolution	83
6.12	Miscellaneous Lemmas	85
7	Soundness	86
7.1	Sound Trail	86
7.2	Sound State	87
7.3	Initial State Is Sound	87
7.4	SCL Rules Preserve Soundness	87
8	Strategies	95
9	Monotonicity w.r.t. the Bounding Atom	97
9.1	Examples	122
9.2	Miscellaneous Lemmas	123
9.3	Well-Defined	124
9.4	Strict Partial Order	125
9.5	Strict Total (w.r.t. Elements in Trail) Order	126
9.6	Well-Founded	127
9.7	Extension on All Literals	132
9.7.1	Well-Founded	134
9.8	Alternative only for terms	134

10 Reasonable Steps	139
10.1 Invariants	139
10.1.1 No Conflict After Decide	139
10.2 Miscellaneous Lemmas	142
11 Regular Steps	142
11.1 Invariants	143
11.1.1 Almost No Conflict With Trail	143
11.1.2 Backtrack Follows Regular Conflict Resolution	148
11.2 Miscellaneous Lemmas	159
12 Resolve in Regular Runs	164
13 Clause Redundancy	170
14 Trail-Induced Ordering	173
14.1 Miscellaneous Lemmas	173
14.2 Strict Partial Order	174
14.3 Properties	174
15 Dynamic Non-Redundancy	175
16 Static Non-Redundancy	191
16.1 Basic Results	195
16.1.1 Minimal-element characterization of well-foundedness	195
17 Extra Lemmas	200
17.1 Set Extra	200
17.2 Prod Extra	200
17.3 FSet Extra	201
18 Termination	202
18.1 SCL without backtracking terminates	202
18.2 Backtracking can only be done finitely often	210
18.3 Regular SCL terminates	214
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: $\text{finite } V \implies \text{inj } (\text{renaming } V)$

1.1 Interpretation to Prove That Assumptions Are Consistent

experiment begin

definition *renaming-apart-nats* **where**

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

interpretation *renaming-apart-nats*: *renaming-apart* *renaming-apart-nats*

proof *unfold-locales*

show $\bigwedge V x. \text{finite } V \implies \text{renaming-apart-nats } V x \notin V$

unfolding *renaming-apart-nats-def* *Let-def* **by** (*meson* *Max.coboundedI* *Suc-le-lessD* *not-add-less2*)

next

show $\bigwedge V. \text{inj } (\text{renaming-apart-nats } V)$

unfolding *renaming-apart-nats-def* *Let-def* **by** (*rule injI*) *simp*

qed

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*:

$L \cdot l \sigma_L = K \cdot l \sigma_K \implies \text{atm-of } L \cdot a \sigma_L = \text{atm-of } K \cdot a \sigma_K$

by (*cases L*; *cases K*) (*simp-all add: subst-lit-def*)

lemma (*in substitution-ops*) *set-mset-subst-cls-conv*: $\text{set-mset } (C \cdot \sigma) = (\lambda L. L \cdot l \sigma) \text{ ` set-mset } C$

by (*simp add: subst-cls-def*)

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*: $L \notin xs \longleftrightarrow (\forall y \in xs. L \neq y)$
by *auto*

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

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

lemma *not-empty-if-mem*: $x \in X \Longrightarrow X \neq \{\}$
by *blast*

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*: $\bigwedge x \ y \ F. \text{finite } F \Longrightarrow x \neq y \Longrightarrow x \notin F \Longrightarrow y \notin F \Longrightarrow P$
 $(\text{insert } y \ F) \Longrightarrow P \ (\text{insert } x \ (\text{insert } y \ F))$
shows $P \ F$
using $\langle \text{finite } F \rangle$
proof *induct*
show $P \ \{\}$ **by** *fact*
next
fix $x \ F$
assume $F: \text{finite } F$ **and** $P: P \ F$
thus $P \ (\text{insert } x \ F)$
proof (*induction F rule: finite.induct*)
case *emptyI*
show $?case$ **by** (*rule singleton*)
next
case (*insertI F y*)
show $?case$
proof (*cases x = y*)
case *True*
then show $?thesis$

```

    by (simp add: insertI.prem)
next
case x-neq-y: False
show ?thesis
proof (cases x ∈ F ∨ y ∈ F)
  case True
  then show ?thesis
    by (metis insertCI insertI.IH insertI.prem insert-absorb)
next
case False
show ?thesis
proof (rule insert-insert)
  show finite F using insertI by simp
next
  show x ≠ y by (rule x-neq-y)
next
  show x ∉ F using False by simp
next
  show y ∉ F using False by simp
next
  show P (insert y F)
    by (simp add: insertI.prem)
qed
qed
qed
qed
qed

```

3.3 Product Type Extra

lemma *insert-Times*: $\text{insert } a \ A \times B = \text{Pair } a \ 'B \cup A \times B$
 by *blast*

lemma *Times-insert*: $A \times \text{insert } b \ B = (\lambda x. (x, b)) \ 'A \cup A \times B$
 by *blast*

lemma *insert-Times-insert'*:
 $\text{insert } a \ A \times \text{insert } b \ B = \text{insert } (a, b) ((\text{Pair } a \ 'B) \cup ((\lambda x. (x, b)) \ 'A) \cup (A \times B))$
 (is ?lhs = ?rhs)
 unfolding *insert-Times-insert* by *auto*

3.4 List Extra

lemma *lt-lengthD*:
 assumes *i-lt-xs*: $i < \text{length } xs$
 shows $\exists xs1 \ xi \ xs2. xs = xs1 @ xi \# xs2 \wedge \text{length } xs1 = i$
 using *assms*
 by (metis *length-drop*[of - xs, of i] *length-append*[of take i xs drop i xs]
add-diff-cancel-left'[of i] *add-diff-cancel-right*'[of i])

add-diff-cancel-right'[of $\text{length}(\text{take } i \text{ } xs) \text{ } \text{length}(\text{drop } i \text{ } xs)$] *id-take-nth-drop*[of $i \text{ } xs$]
Cons-nth-drop-Suc[of $i \text{ } xs$] *canonically-ordered-monoid-add-class.lessE*[of $i \text{ } \text{length } xs$])

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 –

from *i-lt-xs* **obtain** $xs1 \ xi \ xs'$ **where** $xs = xs1 \ @ \ xi \ \# \ xs'$ **and** $\text{length } xs1 = i$
using *lt-lengthD* **by** *blast*

with *j-lt-xs* **obtain** $xs2 \ xj \ xs3$ **where** $xs = xs1 \ @ \ xi \ \# \ xs2 \ @ \ xj \ \# \ xs3$ **and**
 $\text{length } (xs1 \ @ \ xi \ \# \ xs2) = j$

using *lt-lengthD*

by (*smt* (*verit*, *del-insts*) *append.assoc* *append-Cons* *append-eq-append-conv* *i-lt-j* *list.inject*)

thus *?thesis*

using $\langle \text{length } xs1 = i \rangle$ **by** *blast*

qed

3.5 Sublist Extra

lemma *not-mem-strict-suffix*:

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

unfolding *strict-suffix-def* *suffix-def*

by (*metis* *Cons-eq-append-conv* *Un-iff* *set-append*)

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

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

using *not-mem-strict-suffix*[of $\text{map } f \ xs \ f \ y \ \text{map } f \ ys$, *unfolded* *list.set-map*]

using *map-mono-strict-suffix*[of $- \ \# \ -$, *unfolded* *list.map*]

by *fast*

3.6 Multiset Extra

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

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

unfolding *multp_{DM}-def*

by (*metis* *subset-mset.le-imp-diff-is-add*)

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

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

by (*metis* *multp_{DM}-implies-one-step* *multp_{HO}-imp-multp_{DM}*)

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

by *auto*

lemma *multp-singleton-rightD*:
assumes *multp R M {#x#}* and *transp R*
shows $y \in \# M \implies R y x$
using *multp-implies-one-step[OF <transp R> <multp R M {#x#}>]*
by (*metis add-cancel-left-left set-mset-single single-is-union singletonD*)

3.6.1 Calculus Extra

lemma (*in consequence-relation*) *entails-one-formula*: $N \models U \implies D \in U \implies N \models \{D\}$
using *entail-set-all-formulas* by *blast*

3.7 Clausal Calculus Extra

3.7.1 Clausal Calculus Only

lemma *true-cls-iff-set-mset-eq*: $set-mset C = set-mset D \implies I \models C \longleftrightarrow I \models D$
by (*simp add: true-cls-def*)

lemma *true-cls-if-set-mset-eq*: $(\forall D \in \mathcal{D}. \exists C \in \mathcal{C}. set-mset D = set-mset C) \implies I \models_s \mathcal{C} \implies I \models_s \mathcal{D}$
using *true-cls-iff-set-mset-eq* by (*metis true-cls-def*)

lemma *entails-cls-insert*: $N \models_e insert C U \longleftrightarrow N \models_e \{C\} \wedge N \models_e U$
by *auto*

lemma *Collect-lits-from-atms-conv*: $\{L. P (atm-of L)\} = (\bigcup x \in \{x. P x\}. \{Pos x, Neg x\})$
(*is ?lhs = ?rhs*)
proof (*rule Set.equalityI; rule Set.subsetI*)
fix *L*
show $L \in ?lhs \implies L \in ?rhs$
by (*cases L*) *simp-all*
next
fix *L*
show $L \in ?rhs \implies L \in ?lhs$
by *auto*
qed

3.7.2 Clausal Calculus and Abstract Substitution

lemma (*in substitution*) *is-ground-lit-Pos[simp]*: $is-ground-atm atm \implies is-ground-lit (Pos atm)$
by (*simp add: is-ground-lit-def*)

lemma (*in substitution*) *is-ground-lit-Neg[simp]*: $is-ground-atm atm \implies is-ground-lit (Neg atm)$
by (*simp add: is-ground-lit-def*)

3.8 First Order Terms Extra

3.8.1 First Order Terms Only

lemma *atm-of-eq-uminus-if-lit-eq*: $L = - K \implies \text{atm-of } L = \text{atm-of } K$
by (*cases L*; *cases K*) *simp-all*

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

$\text{range-vars } \sigma \cap \text{subst-domain } \delta = \{\} \implies t \cdot \sigma \cdot \delta = u \cdot \sigma \cdot \delta$

$\text{range-vars } \delta \cap \text{subst-domain } \sigma = \{\} \implies t \cdot \delta \cdot \sigma = u \cdot \delta \cdot \sigma$

proof –

from *u-inter- σ -empty* **have** $u \cdot \sigma \cdot \delta = u \cdot \delta$

by (*simp add: subst-apply-term-ident*)

with $\langle t \cdot \sigma = u \cdot \delta \rangle$ **show** $\text{range-vars } \sigma \cap \text{subst-domain } \delta = \{\} \implies t \cdot \sigma \cdot \delta = u \cdot \sigma \cdot \delta$

unfolding *subst-apply-term-subst-apply-term-eq-subst-apply-term-lhs*[*OF - t-inter- δ -empty*]
by *simp*

from *t-inter- δ -empty* **have** $t \cdot \delta \cdot \sigma = t \cdot \sigma$

by (*simp add: subst-apply-term-ident*)

with $\langle t \cdot \sigma = u \cdot \delta \rangle$ **show** $\text{range-vars } \delta \cap \text{subst-domain } \sigma = \{\} \implies t \cdot \delta \cdot \sigma = u \cdot \delta \cdot \sigma$

unfolding *subst-apply-term-subst-apply-term-eq-subst-apply-term-lhs*[*OF - u-inter- σ -empty*]
by *simp*

qed

lemma *subst-compose-in-unifiersI*:

assumes $t \cdot \sigma = u \cdot \delta$ **and**

$\text{vars-term } t \cap \text{subst-domain } \delta = \{\}$ **and**

$\text{vars-term } u \cap \text{subst-domain } \sigma = \{\}$

shows

$\text{range-vars } \sigma \cap \text{subst-domain } \delta = \{\} \implies \sigma \circ_s \delta \in \text{unifiers } \{(t, u)\}$

$\text{range-vars } \delta \cap \text{subst-domain } \sigma = \{\} \implies \delta \circ_s \sigma \in \text{unifiers } \{(t, u)\}$

using *subst-subst-eq-subst-subst-if-subst-eq-substI*(1)[*OF assms*]

using *subst-subst-eq-subst-subst-if-subst-eq-substI*(2)[*OF assms*]

by (*simp-all add: unifiers-def*)

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

by (*simp add: subst-domain-def*)

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

proof (*unfold is-renaming-def, intro conjI allI*)

fix y **show** *is-Var* ($(\text{Var}(x := \text{Var } x')) y$)

by *simp*

next
show *inj-on* (*Var*($x := \text{Var } x'$)) (*subst-domain* (*Var*($x := \text{Var } x'$)))
apply (*rule inj-onI*)
apply (*simp add: subst-domain-def*)
by *presburger*
qed

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** *vars-term* $t \cap \text{vars-term } u = \{\}$
shows $\exists \mu :: ('f, 'v)\ \text{subst}.$ *Unification.mgu* $t\ u = \text{Some } \mu$
proof –
from *assms* **obtain** $\sigma :: ('f, 'v)\ \text{subst}$ **where** $t \cdot \sigma = u \cdot \sigma$
using *vars-term-disjoint-imp-unifier* **by** *metis*
thus *?thesis*
using *ex-mgu-if-subst-apply-term-eq-subst-apply-term*
by *metis*
qed

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 –
assume $x \in A$
hence *restrict-subst-domain* $A\ \sigma_A\ x = \sigma_A\ x$
by (*simp add: restrict-subst-domain-def*)
moreover **have** $\sigma_A\ x \cdot \sigma_B = \sigma_A\ x$
using *distinct-range*
by (*simp add: <x ∈ A> subst-apply-term-ident*)
ultimately **show** $\sigma\ x = \sigma_A\ x$
by (*simp add: σ-def subst-compose-def*)
next
assume $x \in B$
hence *restrict-subst-domain* $A\ \sigma_A\ x = \text{Var } x$
using *distinct-domains*
by (*metis Int-iff empty-iff restrict-subst-domain-def*)
then **show** $\sigma\ x = \sigma_B\ x$
by (*simp add: σ-def subst-compose-def*)
qed

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 –
show $x \in A \implies \sigma x = \sigma_A x$
by (*simp add: σ -def*)
next
assume $x \in B$
moreover hence $x \notin A$
using *distinct-domains* **by** *auto*
ultimately show $\sigma x = \sigma_B x$
by (*simp add: σ -def*)
next
show $x \notin A \cup B \implies \sigma x = \text{Var } x$
by (*simp add: σ -def*)
qed

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* $\gamma A \gamma_A B \gamma_B$
unfolding *is-grounding-merge-def*
by (*auto simp: γ -def*)

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* $\gamma A \gamma_A B \gamma_B$
unfolding *is-grounding-merge-def*

proof (*intro impI*)

assume *disjoint*: $A \cap B = \{\}$ **and**
ball-A-ground: $\forall x \in A. \text{vars-term } (\gamma_A x) = \{\}$ **and**
ball-B-ground: $\forall x \in B. \text{vars-term } (\gamma_B x) = \{\}$

from *ball-A-ground* **have** $\forall x \in A. \text{vars-term } (\gamma_A x) \cap \text{subst-domain } \gamma_B = \{\}$

by *simp*

thus $(\forall x \in A. \gamma x = \gamma_A x) \wedge (\forall x \in B. \gamma x = \gamma_B x)$

using *restrict-subst-domain-subst-composition*[*OF disjoint, of $\gamma_A \gamma_B$*]

by (*simp-all add: γ -def*)

qed

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* .

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-clss* (**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-clss* :: (*f*, *v*) *term clause set* \Rightarrow *v* set **where**
vars-clss *N* = (\bigcup *C* \in *N*. *vars-cls* *C*)

lemma *vars-clss-empty[simp]*: *vars-clss* {} = {}
by (*simp add: vars-clss-def*)

lemma *vars-clss-insert[simp]*: *vars-clss* (*insert* *C* *N*) = *vars-cls* *C* \cup *vars-clss* *N*
by (*simp add: vars-clss-def*)

lemma *vars-clss-union[simp]*: *vars-clss* (*CC* \cup *DD*) = *vars-clss* *CC* \cup *vars-clss* *DD*
by (*simp add: vars-clss-def*)

lemma *vars-cls-empty[simp]*: *vars-cls* {#} = {}
unfolding *vars-cls-def* **by** *simp*

lemma *finite-vars-cls[simp]*: *finite* (*vars-cls* *C*)
unfolding *vars-cls-def* **by** *simp*

lemma *vars-cls-plus-iff*: *vars-cls* (*C* + *D*) = *vars-cls* *C* \cup *vars-cls* *D*
unfolding *vars-cls-def* **by** *simp*

lemma *vars-cls-subset-vars-cls-if-subset-mset*: *C* $\subseteq_{\#}$ *D* \implies *vars-cls* *C* \subseteq *vars-cls* *D*
by (*auto simp add: vars-cls-def*)

lemma *is-ground-atm-iff-vars-empty*: *is-ground-atm* *t* \iff *vars-term* *t* = {}
by (*metis* (*mono-tags*, *opaque-lifting*) *term.distinct*(1)[*of - undefined undefined*])

$is_ground_atm_def[of\ t]\ subst_apply_term_empty[of\ t]$
 $subst_simps(1)[of\ -\ Fun\ undefined\ undefined]\ equals0I[of\ vars_term\ t]$
 $term_subst_eq[of\ t\ Var]\ equals0D[of\ vars_term\ t]$
 $term_subst_eq_rev[of\ t\ subst\ -\ (Fun\ undefined\ undefined)\ Var])$

lemma *is-ground-lit-iff-vars-empty*: $is_ground_lit\ L \longleftrightarrow vars_lit\ L = \{\}$
by (*simp add: is-ground-atm-iff-vars-empty is-ground-lit-def*)

lemma *is-ground-cls-iff-vars-empty*: $is_ground_cls\ C \longleftrightarrow vars_cls\ C = \{\}$
by (*auto simp: is-ground-cls-def is-ground-lit-iff-vars-empty vars-cls-def*)

lemma *is-ground-atm-is-ground-on-var*:
assumes *is-ground-atm* ($A \cdot a\ \sigma$) **and** $v \in vars_term\ A$
shows *is-ground-atm* ($\sigma\ v$)
using *assms proof* (*induction A*)
case ($Var\ x$)
then show *?case* **by** *auto*
next
case ($Fun\ f\ ts$)
then show *?case* **unfolding** *is-ground-atm-def*
by *auto*
qed

lemma *is-ground-lit-is-ground-on-var*:
assumes *ground-lit*: *is-ground-lit* ($subst_lit\ L\ \sigma$) **and** *v-in-L*: $v \in vars_lit\ L$
shows *is-ground-atm* ($\sigma\ v$)
proof –
let $?A = atm_of\ L$
from *v-in-L* **have** $A-p$: $v \in vars_term\ ?A$
by *auto*
then have *is-ground-atm* ($?A \cdot a\ \sigma$)
using *ground-lit* **unfolding** *is-ground-lit-def* **by** *auto*
then show *?thesis*
using $A-p$ *is-ground-atm-is-ground-on-var* **by** *metis*
qed

lemma *is-ground-cls-is-ground-on-var*:
assumes
ground-clause: *is-ground-cls* ($subst_cls\ C\ \sigma$) **and**
v-in-C: $v \in vars_cls\ C$
shows *is-ground-atm* ($\sigma\ v$)
proof –
from *v-in-C* **obtain** L **where** $L-p$: $L \in \# C\ v \in vars_lit\ L$
unfolding *vars-cls-def* **by** *auto*
then have *is-ground-lit* ($subst_lit\ L\ \sigma$)
using *ground-clause* **unfolding** *is-ground-cls-def subst-cls-def* **by** *auto*
then show *?thesis*
using $L-p$ *is-ground-lit-is-ground-on-var* **by** *metis*
qed

lemma *vars-atm-subset-subst-domain-if-grounding*:
assumes *is-ground-atm* ($t \cdot a \ \gamma$)
shows *vars-term* $t \subseteq$ *subst-domain* γ
using *assms*
by (*metis empty-iff is-ground-atm-iff-vars-empty is-ground-atm-is-ground-on-var subsetI subst-ident-if-not-in-domain term.set-intros(3)*)

lemma *vars-lit-subset-subst-domain-if-grounding*:
assumes *is-ground-lit* ($L \cdot l \ \gamma$)
shows *vars-lit* $L \subseteq$ *subst-domain* γ
using *assms vars-atm-subset-subst-domain-if-grounding*
by (*metis atm-of-subst-lit is-ground-lit-def*)

lemma *vars-cls-subset-subst-domain-if-grounding*:
assumes *is-ground-cls* ($C \cdot \sigma$)
shows *vars-cls* $C \subseteq$ *subst-domain* σ
proof (*rule Set.subsetI*)
fix x **assume** $x \in$ *vars-cls* C
thus $x \in$ *subst-domain* σ
unfolding *subst-domain-def mem-Collect-eq*
by (*metis assms empty-iff is-ground-atm-iff-vars-empty is-ground-cls-is-ground-on-var term.set-intros(3)*)

qed

lemma *same-on-vars-lit*:
assumes $\forall v \in$ *vars-lit* $L. \sigma \ v = \tau \ v$
shows *subst-lit* $L \ \sigma =$ *subst-lit* $L \ \tau$
using *assms*
proof (*induction L*)
case (*Pos* x)
then have $\forall v \in$ *vars-term* $x. \sigma \ v = \tau \ v \implies$ *subst-atm-abbrev* $x \ \sigma =$ *subst-atm-abbrev* $x \ \tau$
using *term-subst-eq* **by** *metis+*
then show *?case*
unfolding *subst-lit-def* **using** *Pos* **by** *auto*

next
case (*Neg* x)
then have $\forall v \in$ *vars-term* $x. \sigma \ v = \tau \ v \implies$ *subst-atm-abbrev* $x \ \sigma =$ *subst-atm-abbrev* $x \ \tau$
using *term-subst-eq* **by** *metis+*
then show *?case*
unfolding *subst-lit-def* **using** *Neg* **by** *auto*

qed

lemma *same-on-vars-clause*:
assumes $\forall v \in$ *vars-cls* $S. \sigma \ v = \tau \ v$

shows $\text{subst-cl}\ S\ \sigma = \text{subst-cl}\ S\ \tau$
by (*smt assms image-eqI image-mset-cong2 mem-simps(9) same-on-vars-lit set-image-mset subst-cl-def vars-cl-def*)

lemma *same-on-lits-clause*:

assumes $\forall L \in \# C. \text{subst-lit}\ L\ \sigma = \text{subst-lit}\ L\ \tau$
shows $\text{subst-cl}\ C\ \sigma = \text{subst-cl}\ C\ \tau$
using *assms unfolding subst-cl-def*
by *simp*

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

proof *unfold-locales*

show $\bigwedge A. A \cdot a\ \text{Var} = A$
by *auto*

next

show $\bigwedge A\ \tau\ \sigma. A \cdot a\ (\tau \odot \sigma) = A \cdot a\ \tau \cdot a\ \sigma$
by *auto*

next

show $\bigwedge \sigma\ \tau. (\bigwedge A. A \cdot a\ \sigma = A \cdot a\ \tau) \Longrightarrow \sigma = \tau$
by (*simp add: subst-term-eqI*)

next

fix $C :: ('f, 'v)\ \text{term}\ \text{clause}$ **and** $\sigma :: ('f, 'v)\ \text{subst}$

assume *is-ground-cl* ($\text{subst-cl}\ C\ \sigma$)

hence *ground-atms- σ* : $\bigwedge v. v \in \text{vars-cl}\ C \Longrightarrow \text{is-ground-atm}\ (\sigma\ v)$

by (*meson is-ground-cl-is-ground-on-var*)

define *some-ground-trm* :: $('f, 'v)\ \text{term}$ **where** *some-ground-trm* = (*Fun unde-fined* \square)

have *ground-trm*: *is-ground-atm* *some-ground-trm*

unfolding *is-ground-atm-def* *some-ground-trm-def* **by** *auto*

define τ **where** $\tau = (\lambda v. \text{if } v \in \text{vars-cl}\ C \text{ then } \sigma\ v \text{ else } \text{some-ground-trm})$

then have $\tau\text{-}\sigma$: $\forall v \in \text{vars-cl}\ C. \sigma\ v = \tau\ v$

unfolding $\tau\text{-def}$ **by** *auto*

have *all-ground- τ* : *is-ground-atm* $(\tau\ v)$ **for** v

proof (*cases* $v \in \text{vars-cl}\ C$)

case *True*

then show *?thesis*

using *ground-atms- σ* $\tau\text{-}\sigma$ **by** *auto*

next

case *False*

then show *?thesis*

unfolding $\tau\text{-def}$ **using** *ground-trm* **by** *auto*

qed

have *is-ground-subst* τ

unfolding *is-ground-subst-def*

proof

fix A

show *is-ground-atm* $(A \cdot a\ \tau)$

```

proof (induction A)
  case (Var v)
  thus ?case using all-ground-τ by auto
next
  case (Fun f As)
  thus ?case using all-ground-τ by (simp add: is-ground-atm-def)
qed
qed
moreover with  $\tau \cdot \sigma$  have  $C \cdot \sigma = C \cdot \tau$ 
  using same-on-vars-clause by auto
ultimately show  $\exists \tau. \text{is-ground-subst } \tau \wedge C \cdot \tau = C \cdot \sigma$ 
  by auto
next
show wfP (strictly-generalizes-atm :: ('f, 'v) term  $\Rightarrow$  -  $\Rightarrow$  -)
  unfolding wfp-def
  by (rule wf-subset[OF wf-subsumes])
  (auto simp: strictly-generalizes-atm-def generalizes-atm-def term-subsumable.subsumes-def
  subsumeseq-term.simps)
qed

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

lemma vars-subst-lit-eq:
   $\text{vars-lit } (L \cdot l \sigma) = (\bigcup x \in \text{vars-lit } L. \text{vars-term } (\sigma x))$ 
  using vars-term-subst-apply-term by (metis atm-of-subst-lit)

lemma vars-subst-cls-eq:
   $\text{vars-cls } (C \cdot \sigma) = (\bigcup x \in \text{vars-cls } C. \text{vars-term } (\sigma x))$ 
  by (simp add: vars-cls-def multiset.set-map UN-UN-flatten subst-cls-def
  vars-subst-lit-eq[symmetric])

lemma vars-subst-lit-subset:  $\text{vars-lit } (L \cdot l \sigma) \subseteq \text{vars-lit } L - \text{subst-domain } \sigma \cup$ 
 $\text{range-vars } \sigma$ 
  using vars-term-subst-apply-term-subset[of atm-of L] by simp

lemma vars-subst-cls-subset:  $\text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } C - \text{subst-domain } \sigma \cup$ 
 $\text{range-vars } \sigma$ 
  unfolding vars-cls-def subst-cls-def
  apply simp
  using vars-subst-lit-subset
  by fastforce

lemma vars-subst-cls-subset-weak:  $\text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } C \cup \text{range-vars } \sigma$ 
  unfolding vars-cls-def subst-cls-def
  apply simp
  using vars-subst-lit-subset
  by fastforce

```

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

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

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

lemma *vars-lit-subst-subset-vars-cls-substI[intro]*:
 $\text{vars-lit } L \subseteq \text{vars-cls } C \implies \text{vars-lit } (L \cdot l \ \sigma) \subseteq \text{vars-cls } (C \cdot \sigma)$
by (*metis subset-Un-eq subst-cls-add-mset vars-cls-add-mset vars-subst-cls-eq*)

lemma *vars-subst-cls-subset-vars-cls-subst*:
 $\text{vars-cls } C \subseteq \text{vars-cls } D \implies \text{vars-cls } (C \cdot \sigma) \subseteq \text{vars-cls } (D \cdot \sigma)$
by (*simp only: vars-subst-cls-eq UN-mono*)

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 –
have $\text{vars-cls } ((\text{add-mset } L \ D) \cdot \eta) \subseteq \text{vars-cls } (\text{add-mset } L \ D) - \text{subst-domain } \eta$
 $\cup \text{range-vars } \eta$
by (*rule vars-subst-cls-subset[of add-mset L D η]*)
also have $\dots \subseteq \text{vars-cls } (\text{add-mset } L \ D) - (\text{vars-lit } L \cup \text{vars-lit } L') \cup \text{vars-lit } L$
 $\cup \text{vars-lit } L'$
using *range-vars- η* **by** *blast*
also have $\dots \subseteq \text{vars-cls } (\text{add-mset } L \ D) \cup \text{vars-lit } L' \cup \text{vars-lit } L$
by *auto*
also have $\dots \subseteq \text{vars-cls } D \cup \text{vars-lit } L' \cup \text{vars-lit } L$
by *auto*
also have $\dots \subseteq \text{vars-cls } (\text{add-mset } L' \ (\text{add-mset } L \ D))$
by *auto*
finally show *?thesis*
by *assumption*
qed

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*: $\text{disjoint-vars } C \ D \longleftrightarrow \text{vars-cls } C \cap \text{vars-cls } D = \{\}$
by (*rule disjoint-vars-def*)

hide-fact *disjoint-vars-def*

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

unfolding *disjoint-vars-iff-inter-empty* **by** *blast*

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

unfolding *disjoint-vars-iff-inter-empty vars-cls-plus-iff*
by (*simp add: Int-Un-distrib2*)

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

by (*metis disjoint-vars-plus-iff subset-mset.diff-add*)

lemma *disjoint-vars-subst-clsI*:

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

unfolding *disjoint-vars-iff-inter-empty*

unfolding *vars-subst-cls-eq*

by (*smt (verit, best) Diff-subset Un-iff disjoint-iff image-cong subsetD vars-subst-cls-eq vars-subst-cls-subset*)

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 (*rule iffI*)

show *?lhs \implies ?rhs*

unfolding *is-renaming-def*

by (*metis is-VarI inj-def[of ϱ] term.inject(1) subst-apply-eq-Var[of ϱ -] eval-subst-def[of Fun ϱ]*)

next

show *?rhs \implies ?lhs*

by (*auto simp: is-renaming-def intro: ex-inverse-of-renaming*)

qed

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

by (*metis (mono-tags, lifting) subst-domain-def[of σ] empty-iff mem-Collect-eq[of $\lambda x. \sigma x \neq \text{Var } x$] Int-iff[of - subst-domain σ vars-cls C] subst-cls-id-subst[of C] same-on-vars-clause[of C σ Var])*)

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

by (*rule subst-cls-idem-if-disj-vars[of - {#L#}], simplified*)

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

by (*simp add: restrict-subst-domain-def same-on-vars-lit subsetD*)

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

by (*simp add: restrict-subst-domain-def same-on-vars-clause subsetD*)

lemma *subst-cls-insert[simp]*: $insert\ C\ U \cdot cs\ \eta = insert\ (C \cdot \eta)\ (U \cdot cs\ \eta)$
by (*simp add: subst-cls-def*)

lemma *valid-grounding-of-renaming*:

assumes *is-renaming* ρ

shows $I \models_s\ grounding-of-cls\ (C \cdot \rho) \longleftrightarrow I \models_s\ grounding-of-cls\ C$

proof –

have $grounding-of-cls\ (C \cdot \rho) = grounding-of-cls\ C$

by (*rule grounding-of-subst-cls-renaming-ident[OF ‹is-renaming ρ ›]*)

thus *?thesis*

by *simp*

qed

lemma *is-unifier-iff-mem-unifiers-Times*:

assumes *fin-AA*: *finite* AA

shows $is-unifier\ v\ AA \longleftrightarrow v \in unifiers\ (AA \times AA)$

proof (*rule iffI*)

assume *unif-v-AA*: $is-unifier\ v\ AA$

show $v \in unifiers\ (AA \times AA)$

unfolding *unifiers-def mem-Collect-eq*

proof (*rule ballI*)

have $card\ (AA \cdot_{set}\ v) \leq 1$

by (*rule unif-v-AA[unfolded is-unifier-def subst-atms-def]*)

fix p **assume** $p \in AA \times AA$

then obtain $a\ b$ **where** *p-def*: $p = (a, b)$ **and** $a \in AA$ **and** $b \in AA$

by *auto*

hence $card\ (AA \cdot_{set}\ v) = 1$

using *fin-AA ‹card (AA ·_{set} v) ≤ 1› antisym-conv2* **by** *fastforce*

hence $a \cdot a\ v = b \cdot a\ v$

using $\langle a \in AA \rangle \langle b \in AA \rangle$ *fin-AA is-unifier-subst-atm-eqI unif-v-AA* **by** *blast*

thus $fst\ p \cdot a\ v = snd\ p \cdot a\ v$

by (*simp add: p-def*)

qed

next

assume *unif-v-AA*: $v \in unifiers\ (AA \times AA)$

show $is-unifier\ v\ AA$

using *fin-AA unif-v-AA*

proof (*induction AA arbitrary: v rule: finite-induct*)

case *empty*

then show *?case*

by (*simp add: is-unifier-def*)

next

case (*insert a AA*)

from *insert.prem*s **have**

v-in: $v \in unifiers\ ((insert\ (a, a)\ (Pair\ a\ 'AA)) \cup (\lambda x. (x, a))\ 'AA) \cup AA \times AA$

unfolding *insert-Times-insert'[of a AA a AA]* **by** *simp*

then show *?case*
by (*smt (verit, del-insts) Set.set-insert Un-insert-left finite.insertI fst-conv image-insert insert.hyps(1) insert-compr is-unifier-alt mem-Collect-eq snd-conv unifiers-def*)
qed
qed

lemma *is-mgu-singleton-iff-Unifiers-is-mgu-Times*:
assumes *fin: finite AA*
shows *is-mgu v {AA} \longleftrightarrow Unifiers.is-mgu v (AA \times AA)*
by (*auto simp: is-mgu-def Unifiers.is-mgu-def is-unifiers-def is-unifier-iff-mem-unifiers-Times[OF fin]*)

lemma *is-imgu-singleton-iff-Unifiers-is-imgu-Times*:
assumes *fin: finite AA*
shows *is-imgu v {AA} \longleftrightarrow Unifiers.is-imgu v (AA \times AA)*
by (*auto simp: is-imgu-def Unifiers.is-imgu-def is-unifiers-def is-unifier-iff-mem-unifiers-Times[OF fin]*)

lemma *unifiers-without-refl: unifiers E = unifiers {e \in E. fst e \neq snd e}*
(is ?lhs = ?rhs)
unfolding *unifiers-def* **by** *fastforce*

lemma *subst-lit-renaming-subst-adapted*:
assumes *ren- ρ : is-renaming ρ and vars-L: vars-lit L \subseteq subst-domain σ*
shows *L \cdot l ρ \cdot l rename-subst-domain ρ σ = L \cdot l σ*
proof –
from *ren- ρ have is-var- ρ : $\forall x. is-Var (\rho x)$ and inj ρ*
by (*simp-all add: is-renaming-iff*)

show *?thesis*
using *vars-L renaming-cancels-rename-subst-domain[OF is-var- ρ <inj ρ]*
by (*cases L*) (*simp-all add: subst-lit-def*)
qed

lemma *subst-renaming-subst-adapted*:
assumes *ren- ρ : is-renaming ρ and vars-D: vars-cls D \subseteq subst-domain σ*
shows *D \cdot ρ \cdot rename-subst-domain ρ σ = D \cdot σ*
unfolding *subst-cls-comp-subst[symmetric]*
proof (*intro same-on-lits-clause ballI*)
fix *L assume L \in # D*
with *vars-D have vars-lit L \subseteq subst-domain σ*
by (*auto dest!: multi-member-split*)
thus *L \cdot l ($\rho \odot$ rename-subst-domain ρ σ) = L \cdot l σ*
unfolding *subst-lit-comp-subst*
by (*rule subst-lit-renaming-subst-adapted[OF ren- ρ]*)
qed

lemma *subst-domain-rename-subst-domain-subset'*:
assumes *is-var-ρ*: $\forall x. \text{is-Var } (\rho x)$
shows *subst-domain* (rename-subst-domain $\rho \sigma$) \subseteq ($\bigcup x \in \text{subst-domain } \sigma.$
vars-term (ρx)
proof (rule *subset-trans*)
show *subst-domain* (rename-subst-domain $\rho \sigma$) \subseteq *the-Var* ' ρ ' *subst-domain* σ
by (rule *subst-domain-rename-subst-domain-subset*[*OF is-var-ρ*])
next
show *the-Var* ' ρ ' *subst-domain* σ \subseteq ($\bigcup_{x \in \text{subst-domain } \sigma} \text{vars-term } (\rho x)$)
unfolding *image-the-Var-image-subst-renaming-eq*[*OF is-var-ρ*] **by** *simp*
qed

lemma *range-vars-eq-empty-iff-is-ground*:
is-ground-cls ($C \cdot \gamma$) \implies *subst-domain* $\gamma \subseteq \text{vars-cls } C \implies \text{range-vars } \gamma = \{\}$
unfolding *range-vars-def UNION-empty-conv subst-range.simps is-ground-cls-iff-vars-empty*
by (*metis* (*no-types, opaque-lifting*) *dual-order.eq-iff*[*of subst-domain* γ *vars-cls*
 C]
is-ground-atm-iff-vars-empty is-ground-cls-iff-vars-empty[*of* $C \cdot \gamma$]
imageE[*of* - γ *vars-cls* C] *vars-cls-subset-subst-domain-iff-grounding*[*of* $C \gamma$]
is-ground-cls-is-ground-on-var[*of* $C \gamma$])

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* $\mu \{\{t, u\}\}$
proof –
have *unifiers* ($\{t, u\} \times \{t, u\}$) = *unifiers* $\{\{t, u\}\}$
by (*auto simp: unifiers-def*)
hence *Unifiers.is-imgu* $\mu \{\{t, u\} \times \{t, u\}\}$
using *mgu-sound*[*OF mgu*]
by (*simp add: Unifiers.is-imgu-def*)
thus *?thesis*
by (*simp add: is-imgu-singleton-iff-Unifiers-is-imgu-Times*)
qed

primrec *pairs* **where**
pairs [] = [] |
pairs ($x \# xs$) = (x, x) # *map* (*Pair* x) xs @ *map* ($\lambda y. (y, x)$) xs @ *pairs* xs

lemma *set* (*pairs* [a, b, c, d]) =
 $\{(a, a), (a, b), (a, c), (a, d),$
 $(b, a), (b, b), (b, c), (b, d),$
 $(c, a), (c, b), (c, c), (c, d),$
 $(d, a), (d, b), (d, c), (d, d)\}$
by *auto*

lemma *set-pairs*: *set* (*pairs* xs) = *set* $xs \times \text{set } xs$

by (*induction xs*) *auto*

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*: $a \in \text{set } as \implies b \in \text{set } as \implies (a, b) \in \text{set } (\text{pairs } as)$
by (*induction as*) *auto*

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

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

lemma *vars-mset-mset-pairs*:
 $\text{vars-mset } (\text{mset } (\text{pairs } as)) = (\bigcup b \in \text{set } as. \bigcup a \in \text{set } as. \text{vars-term } a \cup \text{vars-term } b)$
by (*induction as*) (*auto simp: vars-mset-def*)

definition *mgu-sets* **where**
 $\text{mgu-sets } \mu \text{ AAA} \longleftrightarrow (\exists \text{ ass. set } (\text{map set ass}) = \text{AAA} \wedge$
 $\text{map-option subst-of } (\text{unify } (\text{concat } (\text{map pairs ass})) \ \square) = \text{Some } \mu)$

lemma *is-imgu-if-mgu-sets*:
assumes *mgu-AAA*: $\text{mgu-sets } \mu \text{ AAA}$
shows *is-imgu* $\mu \text{ AAA}$

proof –

from *mgu-AAA* **obtain** *ass xs* **where**

AAA-def: $\text{AAA} = \text{set } (\text{map set ass})$ **and**
unify: $\text{unify } (\text{concat } (\text{map pairs ass})) \ \square = \text{Some } xs$ **and**
subst-of xs $= \mu$

unfolding *mgu-sets-def* **by** *auto*

hence *Unifiers.is-imgu* $\mu (\text{set } (\text{concat } (\text{map pairs ass})))$

using *unify-sound[OF unify]* **by** *simp*

moreover have *unifiers* $(\text{set } (\text{concat } (\text{map pairs ass}))) = \{v. \text{is-unifiers } v \text{ AAA}\}$

unfolding *AAA-def*

proof (*rule Set.equalityI; rule Set.subsetI; unfold mem-Collect-eq*)

fix *x* **assume** *x-in*: $x \in \text{unifiers } (\text{set } (\text{concat } (\text{map pairs ass})))$

show *is-unifiers* $x (\text{set } (\text{map set ass}))$

unfolding *is-unifiers-def*

proof (*rule ballI*)

fix *As* **assume** *As* $\in \text{set } (\text{map set ass})$

hence *finite As* **by** *auto*

from $\langle As \in \text{set } (\text{map set ass}) \rangle$ **obtain** *as* **where**

as-in: $as \in \text{set } ass$ **and** *As-def*: $As = \text{set } as$

by *auto*

show *is-unifier* $x \text{ As}$

```

    unfolding is-unifier-alt[OF ‹finite As›]
  proof (intro ballI)
    fix A B assume A ∈ As B ∈ As
    hence ∃ xs ∈ set ass. (A, B) ∈ set (pairs xs)
      using as-in by (auto simp: As-def intro: pair-in-set-pairs)
    thus A · a x = B · a x
      using x-in[unfolded unifiers-def mem-Collect-eq, rule-format, of (A, B),
simplified]
      by simp
    qed
  qed
next
fix x assume is-unifs-x: is-unifiers x (set (map set ass))
show x ∈ unifiers (set (concat (map pairs ass)))
  unfolding unifiers-def mem-Collect-eq
proof (rule ballI)
  fix p assume p ∈ set (concat (map pairs ass))
  then obtain as where as ∈ set ass and p-in: p ∈ set (pairs as)
    by auto
  hence is-unif-x: is-unifier x (set as)
    using is-unifs-x[unfolded is-unifiers-def] by simp
  moreover have fst p ∈ set as
    by (rule p-in[THEN fst-pair-in-set-if-pair-in-pairs])
  moreover have snd p ∈ set as
    by (rule p-in[THEN snd-pair-in-set-if-pair-in-pairs])
  ultimately show fst p · a x = snd p · a x
    unfolding is-unifier-alt[of set as, simplified]
    by blast
  qed
qed
ultimately show is-imag μ AAA
  unfolding Unifiers.is-imag-def is-imag-def by simp
qed

```

3.8.4 Renaming Extra

context *renaming-apart* **begin**

lemma *inj-Var-comp-renaming*: $\text{finite } V \implies \text{inj } (\text{Var} \circ \text{renaming } V)$
using *inj-compose inj-renaming* **by** (*metis inj-def term.inject(1)*)

lemma *is-renaming-Var-comp-renaming*: $\text{finite } V \implies \text{Term.is-renaming } (\text{Var} \circ \text{renaming } V)$
unfolding *Term.is-renaming-def*
using *inj-Var-comp-renaming* **by** (*metis comp-apply inj-on-subset term.disc(1) top-greatest*)

lemma *vars-term-subst-term-Var-comp-renaming-disj*:
assumes *fin-V*: $\text{finite } V$

shows $\text{vars-term } (t \cdot a \text{ (Var } \circ \text{renaming } V)) \cap V = \{\}$
using $\text{is-renaming-Var-comp-renaming}[OF \text{ fin-V}] \text{renaming-correct}[OF \text{ fin-V}]$
by $(\text{induction } t) \text{auto}$

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

lemma $\text{vars-lit-subst-renaming-disj}$:
assumes $\text{fin-V: finite } V$
shows $\text{vars-lit } (L \cdot l \text{ (Var } \circ \text{renaming } V)) \cap V = \{\}$
using $\text{vars-term-subst-term-Var-comp-renaming-disj}[OF \text{ fin-V}]$ **by** auto

lemma $\text{vars-cls-subst-renaming-disj}$:
assumes $\text{fin-V: finite } V$
shows $\text{vars-cls } (C \cdot (Var \circ \text{renaming } V)) \cap V = \{\}$
unfolding vars-cls-def
apply simp
using $\text{vars-lit-subst-renaming-disj}[OF \text{ fin-V}]$
by $(\text{smt } (\text{verit, best}) \text{UN-iff UN-simps}(10) \text{disjoint-iff multiset.set-map subst-cls-def})$

abbreviation $\text{renaming-wrt} :: ('f, -) \text{Term.term clause set} \Rightarrow - \Rightarrow ('f, -) \text{Term.term}$
where
 $\text{renaming-wrt } N \equiv \text{Var } \circ \text{renaming } (\text{vars-cls } N)$

lemma $\text{is-renaming-renaming-wrt: finite } N \Longrightarrow \text{is-renaming } (\text{renaming-wrt } N)$
by $(\text{simp add: inj-Var-comp-renaming is-renaming-iff vars-cls-def})$

lemma $\text{ex-renaming-to-disjoint-vars}$:
fixes $C :: ('f, 'a) \text{Term.term clause}$ **and** $N :: ('f, 'a) \text{Term.term clause set}$
assumes $\text{fin: finite } N$
shows $\exists \rho. \text{is-renaming } \rho \wedge \text{vars-cls } (C \cdot \rho) \cap \text{vars-cls } N = \{\}$
proof (intro exI conjI)
show $\text{SCL-FOL.is-renaming } (\text{renaming-wrt } N)$
using $\text{fin is-renaming-renaming-wrt}$ **by** metis
next
show $\text{vars-cls } (C \cdot \text{renaming-wrt } N) \cap \text{vars-cls } N = \{\}$
by $(\text{simp add: fin vars-cls-subst-renaming-disj vars-cls-def})$
qed

end

4 SCL State

type-synonym $('f, 'v) \text{closure} = ('f, 'v) \text{term clause} \times ('f, 'v) \text{subst}$
type-synonym $('f, 'v) \text{closure-with-lit} =$
 $('f, 'v) \text{term clause} \times ('f, 'v) \text{term literal} \times ('f, 'v) \text{subst}$
type-synonym $('f, 'v) \text{trail} = (('f, 'v) \text{term literal} \times ('f, 'v) \text{closure-with-lit op-}$

tion) 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 (\square , $\{\square\}$, None)

definition *state-trail* :: (f, v) state \Rightarrow (f, v) trail **where**
state-trail $S = \text{fst } S$

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

definition *state-learned* :: (f, v) state \Rightarrow (f, v) term clause fset **where**
state-learned $S = \text{fst } (\text{snd } S)$

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

definition *state-conflict* :: (f, v) state \Rightarrow (f, v) closure option **where**
state-conflict $S = \text{snd } (\text{snd } S)$

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

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
by (*simp add: state-conflict-def state-learned-def state-trail-def*)

fun *class-of-trail-lit* **where**
class-of-trail-lit $(-, \text{None}) = \{\square\}$ |
class-of-trail-lit $(-, \text{Some } (C, L, -)) = \{\text{add-mset } L \ C\}$

primrec *class-of-trail* :: (f, v) trail \Rightarrow (f, v) term clause fset **where**
class-of-trail $\square = \{\square\}$ |
class-of-trail $(Ln \# \Gamma) = \text{class-of-trail-lit } Ln \ |\cup| \ \text{class-of-trail } \Gamma$

hide-fact *class-of-trail-def*

lemma *class-of-trail-append*: *class-of-trail* $(\Gamma_0 \ @ \ \Gamma_1) = \text{class-of-trail } \Gamma_0 \ |\cup| \ \text{class-of-trail } \Gamma_1$
by (*induction* Γ_0) (*simp-all add: funion-assoc*)

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

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

definition *propagate-lit* **where**

$propagate\text{-lit } L C \gamma = (L \cdot l \gamma, Some (C, L, \gamma))$

abbreviation *trail-propagate* ::

$(f, 'v) \text{ trail} \Rightarrow (f, 'v) \text{ term literal} \Rightarrow (f, 'v) \text{ term clause} \Rightarrow (f, 'v) \text{ subst} \Rightarrow$
 $(f, 'v) \text{ trail}$ **where**

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

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

by (*simp add: propagate-lit-def*)

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

unfolding *suffix-def propagate-lit-def*

by *simp*

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

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

unfolding *propagate-lit-def* **by** *simp*

definition *decide-lit* **where**

$decide\text{-lit } L = (L, None)$

abbreviation *trail-decide* :: $(f, 'v) \text{ trail} \Rightarrow (f, 'v) \text{ term literal} \Rightarrow (f, 'v) \text{ trail}$
where

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

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

by (*simp add: decide-lit-def*)

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

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

by (*simp add: decide-lit-def*)

definition *is-decision-lit*

:: $(f, 'v) \text{ term literal} \times (f, 'v) \text{ closure-with-lit option} \Rightarrow bool$ **where**

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

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

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

lemma

$trail\text{-interp } [] = \{ }$

$trail\text{-interp } ((Pos A, ann) \# \Gamma) = insert A (trail\text{-interp } \Gamma)$

$trail\text{-interp } ((Neg A, ann) \# \Gamma) = trail\text{-interp } \Gamma$

by (*induction* Γ) (*simp-all add: trail-interp-def*)

lemma *trail-interp-eq-Union*:

trail-interp $\Gamma = (\bigcup Ln \in \text{set } \Gamma. \text{ case fst } Ln \text{ of Pos } t \Rightarrow \{t\} \mid \text{Neg } t \Rightarrow \{\})$

unfolding *trail-interp-def* **by** *simp*

definition *trail-true-lit* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ literal} \Rightarrow \text{bool}$ **where**

trail-true-lit $\Gamma L \longleftrightarrow L \in \text{fst } \Gamma \text{ set } \Gamma$

definition *trail-false-lit* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ literal} \Rightarrow \text{bool}$ **where**

trail-false-lit $\Gamma L \longleftrightarrow \neg L \in \text{fst } \Gamma \text{ set } \Gamma$

definition *trail-true-cls* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ clause} \Rightarrow \text{bool}$ **where**

trail-true-cls $\Gamma C \longleftrightarrow (\exists L \in \# C. \text{trail-true-lit } \Gamma L)$

definition *trail-false-cls* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ clause} \Rightarrow \text{bool}$ **where**

trail-false-cls $\Gamma C \longleftrightarrow (\forall L \in \# C. \text{trail-false-lit } \Gamma L)$

definition *trail-true-clss* :: $(\text{'f}, \text{'v}) \text{ trail} \Rightarrow (\text{'f}, \text{'v}) \text{ term clause set} \Rightarrow \text{bool}$ **where**

trail-true-clss $\Gamma N \longleftrightarrow (\forall C \in N. \text{trail-true-cls } \Gamma C)$

definition *trail-defined-lit* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ literal} \Rightarrow \text{bool}$ **where**

trail-defined-lit $\Gamma L \longleftrightarrow (L \in \text{fst } \Gamma \text{ set } \Gamma \vee \neg L \in \text{fst } \Gamma \text{ set } \Gamma)$

definition *trail-defined-cls* :: $(- \text{ literal} \times - \text{ option}) \text{ list} \Rightarrow - \text{ clause} \Rightarrow \text{bool}$ **where**

trail-defined-cls $\Gamma C \longleftrightarrow (\forall L \in \# C. \text{trail-defined-lit } \Gamma L)$

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

trail-defined-lit $\Gamma L \longleftrightarrow \text{trail-true-lit } \Gamma L \vee \text{trail-false-lit } \Gamma L$

unfolding *trail-defined-lit-def* *trail-false-lit-def* *trail-true-lit-def* **by** (*rule refl*)

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

trail-defined-cls $\Gamma C \Longrightarrow \text{trail-true-cls } \Gamma C \vee \text{trail-false-cls } \Gamma C$

unfolding *trail-defined-cls-def* *trail-false-cls-def* *trail-true-cls-def*

unfolding *trail-defined-lit-iff-true-or-false*

by *blast*

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

by (*simp add: trail-false-cls-def*)

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

trail-false-cls $\Gamma (\text{add-mset } L C) \longleftrightarrow \text{trail-false-lit } \Gamma L \wedge \text{trail-false-cls } \Gamma C$

by (*auto simp: trail-false-cls-def*)

lemma *trail-false-cls-plus*:

trail-false-cls $\Gamma (C + D) \longleftrightarrow \text{trail-false-cls } \Gamma C \wedge \text{trail-false-cls } \Gamma D$

by (*auto simp: trail-false-cls-def*)

lemma *not-trail-true-Nil[simp]*:

$\neg \text{trail-true-lit } \square L$

$\neg \text{trail-true-cls } \square C$

$N \neq \{\} \implies \neg \text{trail-true-clss } [] N$
by (*simp-all add: trail-true-lit-def trail-true-cls-def trail-true-clss-def*)

lemma *not-trail-false-Nil*[*simp*]:
 $\neg \text{trail-false-lit } [] L$
 $\text{trail-false-cls } [] C \longleftrightarrow C = \{\#$
by (*simp-all add: trail-false-lit-def trail-false-cls-def*)

lemma *not-trail-defined-lit-Nil*[*simp*]: $\neg \text{trail-defined-lit } [] L$
by (*simp add: trail-defined-lit-iff-true-or-false*)

lemma *trail-defined-lit-if-trail-defined-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-defined-lit } \Gamma' K \implies \text{trail-defined-lit } \Gamma K$
by (*meson image-mono set-mono-suffix subsetD trail-defined-lit-def*)

lemma *trail-defined-cls-if-trail-defined-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-defined-cls } \Gamma' C \implies \text{trail-defined-cls } \Gamma C$
using *trail-defined-cls-def trail-defined-lit-if-trail-defined-suffix* **by** *metis*

lemma *trail-false-lit-if-trail-false-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-false-lit } \Gamma' K \implies \text{trail-false-lit } \Gamma K$
by (*meson image-mono set-mono-suffix subsetD trail-false-lit-def*)

lemma *trail-false-cls-if-trail-false-suffix*:
 $\text{suffix } \Gamma' \Gamma \implies \text{trail-false-cls } \Gamma' C \implies \text{trail-false-cls } \Gamma C$
using *trail-false-cls-def trail-false-lit-if-trail-false-suffix* **by** *metis*

lemma *trail-interp-Cons*: $\text{trail-interp } (Ln \# \Gamma) = \text{trail-interp } [Ln] \cup \text{trail-interp } \Gamma$
unfolding *trail-interp-def* **by** *simp*

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

lemma *true-lit-thick-unionD*: $(I1 \cup I2) \models_l L \implies I1 \models_l L \vee I2 \models_l L$
by *auto*

lemma *subtrail-falseI*:
assumes *tr-false*: $\text{trail-false-cls } ((L, Cl) \# \Gamma) C$ **and** *L-not-in*: $-L \notin \# C$
shows $\text{trail-false-cls } \Gamma C$
unfolding *trail-false-cls-def*
proof (*rule ballI*)
fix *M*
assume *M-in*: $M \in \# C$

from *M-in L-not-in* **have** *M-neq-L*: $M \neq -L$ **by** *auto*

from *M-in tr-false* **have** *tr-false-lit-M*: $\text{trail-false-lit } ((L, Cl) \# \Gamma) M$
unfolding *trail-false-cls-def* **by** *simp*

thus *trail-false-lit* Γ M
unfolding *trail-false-lit-def*
using $M\text{-neq-}L$
by (*cases* L ; *cases* M) (*simp-all add: trail-interp-def trail-false-lit-def*)
qed

lemma *trail-false-cls-ignores-duplicates*:
 $set\text{-mset } C = set\text{-mset } D \implies trail\text{-false-cls } \Gamma C \longleftrightarrow trail\text{-false-cls } \Gamma D$
by (*simp add: trail-false-cls-def*)

lemma *ball-trail-propagate-is-ground-lit*:
assumes $\forall x \in set \Gamma. is\text{-ground-lit } (fst\ x)$ **and** $is\text{-ground-lit } (L \cdot l\ \sigma)$
shows $\forall x \in set (trail\text{-propagate } \Gamma L C\ \sigma). is\text{-ground-lit } (fst\ x)$
unfolding *propagate-lit-def*
using *assms* **by** *simp*

lemma *ball-trail-decide-is-ground-lit*:
assumes $\forall x \in set \Gamma. is\text{-ground-lit } (fst\ x)$ **and** $is\text{-ground-lit } L$
shows $\forall x \in set (trail\text{-decide } \Gamma L). is\text{-ground-lit } (fst\ x)$
using *assms*
by (*simp add: decide-lit-def*)

lemma *trail-false-cls-subst-mgu-before-grounding*:
fixes $\Gamma :: ('f, 'v)\ trail$
assumes *tr-false-cls*: $trail\text{-false-cls } \Gamma ((D + \{\#L, L'\#\}) \cdot \sigma)$ **and**
 $imgu\text{-}\mu$: $is\text{-imgu } \mu \{\{atm\text{-of } L, atm\text{-of } L'\}\}$ **and**
 $unif\text{-}\sigma$: $is\text{-unifiers } \sigma \{\{atm\text{-of } L, atm\text{-of } L'\}\}$
shows $trail\text{-false-cls } \Gamma ((D + \{\#L\#\}) \cdot \mu \cdot \sigma)$
unfolding *trail-false-cls-def*
proof (*rule ballI*)
fix K
assume $K \in \# (D + \{\#L\#\}) \cdot \mu \cdot \sigma$
hence $K \in \# D \cdot \mu \cdot \sigma \vee K = L \cdot l\ \mu \cdot l\ \sigma$ **by** *force*
thus $trail\text{-false-lit } \Gamma K$
proof (*elim disjE*)
show $K \in \# D \cdot \mu \cdot \sigma \implies trail\text{-false-lit } \Gamma K$
using $imgu\text{-}\mu\ unif\text{-}\sigma$
using *tr-false-cls trail-false-cls-def*[*of* $\Gamma D \cdot \sigma + \{\#L, L'\#\} \cdot \sigma$]
 $subst\text{-cls-comp-subst}$ [*of* $D\ \mu\ \sigma$] *SCL-FOL.is-imgu-def*[*of* $\mu \{\{atm\text{-of } L, atm\text{-of } L'\}\}$]
by *force*
next
have $L \cdot l\ \mu \cdot l\ \sigma = L \cdot l\ \sigma$
using $imgu\text{-}\mu\ unif\text{-}\sigma$ **by** (*metis is-imgu-def subst-lit-comp-subst*)
thus $K = L \cdot l\ \mu \cdot l\ \sigma \implies trail\text{-false-lit } \Gamma K$
by (*auto intro: tr-false-cls*[*unfolded trail-false-cls-def, rule-format*])
qed
qed

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

by (*auto simp add: trail-defined-lit-def*)

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

by (*simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set trail-defined-lit-def*)

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

proof (*induction* Γ)

case *Nil*

show *?case* **by** (*simp add: trail-interp-def*)

next

case (*Cons Ln* Γ)

then show *?case*

unfolding *list.set image-insert set-filter-insert-conv trail-interp-Cons'*

by (*simp add: literal.case-eq-if*)

qed

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

using *trail-interp-conv[of* Γ *]* **by** *auto*

inductive *trail-consistent* **where**

Nil[*simp*]: *trail-consistent* $[]$ |

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*:

$\text{trail-consistent } \Gamma \implies \text{distinct } (\text{map } (\text{atm-of } \circ \text{fst}) \Gamma)$

by (*induction* Γ *rule: trail-consistent.induct*)

(*simp-all add: image-comp trail-defined-lit-iff*)

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

by (*induction* Γ) (*auto elim: trail-consistent.cases*)

lemma *trail-consistent-if-suffix*:

$\text{trail-consistent } \Gamma \implies \text{suffix } \Gamma' \Gamma \implies \text{trail-consistent } \Gamma'$

by (*auto simp: suffix-def intro: trail-consistent-appendD*)

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

shows $\text{trail-consistent } \Gamma \implies \text{trail-true-lit } \Gamma L \implies \text{trail-interp } \Gamma \Vdash L$

proof (*induction* Γ *rule: trail-consistent.induct*)

case *Nil*

thus *?case*

by (*simp add: trail-true-lit-def*)

next

case (*Cons* $\Gamma K u$)

show *?case*

```

proof (cases L = K ∨ L = - K)
  case True
  then show ?thesis
  proof (elim disjE)
    assume L = K
    thus ?thesis
  proof (cases L; cases K)
    fix tL tK
    from ⟨L = K⟩ show L = Pos tL ⇒ K = Pos tK ⇒ ?thesis
      by (simp add: trail-interp-def)
    next
      fix tL tK
      from ⟨L = K⟩ show L = Neg tL ⇒ K = Neg tK ⇒ ?thesis
        using Cons.hyps(1)
        by (simp add: trail-defined-lit-iff trail-interp-Cons'
          not-in-trail-interp-if-not-in-trail)
    qed simp-all
  next
    assume L = - K
    then show ?thesis
      by (metis Cons.hyps(1) uminus-not-id[of K] trail-true-lit-def[of (K, u) #
Γ L]
        trail-defined-lit-def[of Γ K] Cons.prem map-of-eq-None-iff[of (K, u) #
Γ L]
        map-of-eq-None-iff[of Γ - K] map-of-Cons-code(2)[of K u Γ L])
    qed
  next
    case False
    with Cons.prem have trail-true-lit Γ L
      by (simp add: trail-true-lit-def)
    with Cons.IH have trail-interp Γ  $\models$  L
      by simp
    with False show ?thesis
      by (cases L; cases K) (simp-all add: trail-interp-def del: true-lit-iff)
    qed
  qed

lemma trail-interp-cls-if-trail-true:
  assumes trail-consistent Γ and trail-true-cls Γ C
  shows trail-interp Γ  $\models$  C
proof -
  from ⟨trail-true-cls Γ C⟩ obtain L where L ∈# C and trail-true-lit Γ L
  by (auto simp: trail-true-cls-def)
  show ?thesis
    unfolding true-cls-def
  proof (rule bezI[OF - ⟨L ∈# C⟩])
    show trail-interp Γ  $\models$  L
      by (rule trail-interp-lit-if-trail-true[OF ⟨trail-consistent Γ⟩ ⟨trail-true-lit Γ L⟩])
    qed

```

qed

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

assumes *trail-consistent* $\Gamma \forall L \in \# C$. *trail-defined-lit* ΓL

shows *trail-true-cls* $\Gamma C \longleftrightarrow$ *trail-interp* $\Gamma \models C$

proof (*rule iffI*)

assume *trail-true-cls* ΓC

thus *trail-interp* $\Gamma \models C$

using *assms(1) trail-interp-cls-if-trail-true* **by** *fast*

next

assume *trail-interp* $\Gamma \models C$

then obtain L **where** $L \in \# C$ **and** *trail-interp* $\Gamma \models_l L$

by (*auto simp: true-cls-def*)

show *trail-true-cls* ΓC

proof (*cases L*)

case (*Pos t*)

hence $t \in$ *trail-interp* Γ

using \langle *trail-interp* $\Gamma \models_l L \rangle$ **by** *simp*

then show *?thesis*

unfolding *trail-true-cls-def*

using $\langle L \in \# C \rangle$ *Pos*

by (*metis assms(1) assms(2) trail-defined-lit-def trail-interp-lit-if-trail-true trail-true-lit-def true-lit-simps(2) uminus-Pos*)

next

case (*Neg t*)

then show *?thesis*

by (*metis* $\langle L \in \# C \rangle \langle$ *trail-interp* $\Gamma \models_l L \rangle$ *assms(1) assms(2) trail-defined-lit-def trail-interp-lit-if-trail-true trail-true-cls-def trail-true-lit-def true-lit-simps(1,2) uminus-Neg*)

qed

qed

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

assumes *trail-consistent* $\Gamma \forall L \in \# C$. *trail-defined-lit* ΓL

shows *trail-false-cls* $\Gamma C \longleftrightarrow \neg$ *trail-interp* $\Gamma \models C$

proof (*rule iffI*)

show *trail-false-cls* $\Gamma C \implies \neg$ *trail-interp* $\Gamma \models C$

by (*metis assms(1) uminus-Neg uminus-Pos true-lit-iff*[*of trail-interp* Γ] *trail-true-lit-def*[*of* $\Gamma - (- Neg -)$] *trail-true-lit-def*[*of* $\Gamma - (- Pos -)$] *trail-false-cls-def*[*of* ΓC] *true-cls-def*[*of trail-interp* ΓC] *trail-false-lit-def*[*of* $\Gamma - Neg -$] *trail-false-lit-def*[*of* $\Gamma - Pos -$] *true-lit-simps(2)*[*of trail-interp* Γ] *true-lit-simps(1)*[*of trail-interp* Γ] *trail-interp-lit-if-trail-true*[*of* $\Gamma Neg -$] *trail-interp-lit-if-trail-true*[*of* $\Gamma Pos -$])

next

show \neg *trail-interp* $\Gamma \models C \implies$ *trail-false-cls* ΓC

using *assms(1) assms(2) trail-defined-cls-def trail-interp-cls-if-trail-true trail-true-or-false-cls-if-defined*

by *blast*
qed

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

Nil[simp]: *trail-closures-false* [] |

Cons:

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

lemma *trail-closures-false-ConsD*: *trail-closures-false* (*Ln* # Γ) \implies *trail-closures-false* Γ

by (*auto elim*: *trail-closures-false.cases*)

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

by (*induction* Γ) (*auto elim*: *trail-closures-false.cases*)

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

assumes $\forall L \in \text{fst } 'set \ \Gamma. \text{is-ground-lit } L$

shows *trail-true-lit* $\Gamma L \implies \text{is-ground-lit } L$

using *assms* **by** (*metis trail-true-lit-def*)

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

assumes $\forall L \in \text{fst } 'set \ \Gamma. \text{is-ground-lit } L$

shows *trail-false-lit* $\Gamma L \implies \text{is-ground-lit } L$

using *assms* **by** (*metis trail-false-lit-def atm-of-uminus is-ground-lit-def*)

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

assumes $\forall L \in \text{fst } 'set \ \Gamma. \text{is-ground-lit } L$

shows *trail-defined-lit* $\Gamma L \implies \text{is-ground-lit } L$

using *assms is-ground-lit-if-true-in-ground-trail is-ground-lit-if-false-in-ground-trail*

unfolding *trail-defined-lit-iff-true-or-false*

by *fast*

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

assumes $\forall L \in \text{fst } 'set \ \Gamma. \text{is-ground-lit } L$

shows *trail-false-clc* $\Gamma C \implies \text{is-ground-clc } C$

unfolding *trail-false-clc-def is-ground-clc-def*

using *assms* **by** (*auto intro*: *is-ground-lit-if-false-in-ground-trail*)

5 SCL(FOL) Calculus

locale *scl-fol-calculus* = *renaming-apart renaming-vars*

for *renaming-vars* :: $'v \text{ set} \Rightarrow 'v \Rightarrow 'v +$

fixes *less-B* :: $(f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term} \Rightarrow \text{bool}$ (**infix** $\langle \prec_B \rangle$ 50)

assumes

finite-less-B: $\bigwedge \beta. \text{finite } \{x. x \prec_B \beta\}$

begin

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

5.1 Lemmas About (\prec_B)

lemma *lits-less-B-conv*: $\{L. \text{atm-of } L \prec_B \beta\} = (\bigcup x \in \{x. x \prec_B \beta\}. \{Pos\ x, Neg\ x\})$
by (rule *Collect-lits-from-atms-conv*)

lemma *lits-eq-conv*: $\{L. \text{atm-of } L = \beta\} = \{Pos\ \beta, Neg\ \beta\}$
by (rule *Collect-lits-from-atms-conv*[of $\lambda x. x = \beta$, *simplified*])

lemma *lits-less-eq-B-conv*:
 $\{L. \text{atm-of } L \prec_B \beta \vee \text{atm-of } L = \beta\} = insert\ (Pos\ \beta)\ (insert\ (Neg\ \beta)\ \{L. \text{atm-of } L \prec_B \beta\})$
unfolding *Collect-disj-eq lits-eq-conv* **by** *simp*

lemma *finite-lits-less-B*: *finite* $\{L. \text{atm-of } L \prec_B \beta\}$
unfolding *lits-less-B-conv*

proof (rule *finite-UN-I*)
show *finite* $\{x. x \prec_B \beta\}$
by (rule *finite-less-B*)

next
show $\bigwedge x. x \in \{x. x \prec_B \beta\} \implies \text{finite}\ \{Pos\ x, Neg\ x\}$
by *simp*

qed

lemma *finite-lits-less-eq-B*: *finite* $\{L. \text{atm-of } L \preceq_B \beta\}$
using *finite-lits-less-B* **by** (*simp add: lits-less-eq-B-conv*)

lemma *Collect-ball-eq-Pow-Collect*: $\{X. \forall x \in X. P\ x\} = Pow\ \{x. P\ x\}$
by *blast*

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

proof –

have 1: $(\forall L \in \# C. P\ L \wedge count\ C\ L = 1) \longleftrightarrow (\exists C'. C = mset\text{-set}\ C' \wedge \text{finite}\ C' \wedge (\forall L \in C'. P\ L))$

for $C\ P$

by (*smt* (*verit*) *count-eq-zero-iff count-mset-set' finite-set-mset finite-set-mset-mset-set multiset-eqI*)

have 2: *finite* $\{C'. \forall L \in C'. \text{atm-of } L \prec_B \beta\}$

unfolding *Collect-ball-eq-Pow-Collect finite-Pow-iff*

by (rule *finite-lits-less-B*)

show *?thesis*

unfolding 1

unfolding *setcompr-eq-image*

apply (*rule finite-imageI*)
using 2 **by** *simp*
qed

5.2 Rules

inductive *propagate* :: (*f*, *v*) *term clause fset* \Rightarrow (*f*, *v*) *term* \Rightarrow (*f*, *v*) *state* \Rightarrow
(*f*, *v*) *state* \Rightarrow *bool* **for** *N* β **where**
propagateI: $C \in | N \cup U \Rightarrow C = \text{add-mset } L \ C' \Rightarrow \text{is-ground-cls } (C \cdot \gamma)$
 \Rightarrow
 $\forall K \in \# C \cdot \gamma. \text{atm-of } K \preceq_B \beta \Rightarrow$
 $C_0 = \{\#K \in \# C'. K \cdot l \ \gamma \neq L \cdot l \ \gamma\} \Rightarrow C_1 = \{\#K \in \# C'. K \cdot l \ \gamma = L \cdot l$
 $\gamma\} \Rightarrow$
trail-false-cls $\Gamma (C_0 \cdot \gamma) \Rightarrow \neg \text{trail-defined-lit } \Gamma (L \cdot l \ \gamma) \Rightarrow$
is-ingu $\mu \{\text{atm-of ' set-mset (add-mset } L \ C_1)\} \Rightarrow$
propagate *N* $\beta (\Gamma, U, \text{None}) (\text{trail-propagate } \Gamma (L \cdot l \ \mu) (C_0 \cdot \mu) \ \gamma, U, \text{None})$

lemma $C \in | N \cup U \Rightarrow C = \text{add-mset } L \ C' \Rightarrow \text{is-ground-cls } (C \cdot \gamma) \Rightarrow$
 $\forall K \in \# C. \text{atm-of } (K \cdot l \ \gamma) \preceq_B \beta \Rightarrow$
 $C_0 = \{\#K \in \# C'. K \cdot l \ \gamma \neq L \cdot l \ \gamma\} \Rightarrow C_1 = \{\#K \in \# C'. K \cdot l \ \gamma = L \cdot l$
 $\gamma\} \Rightarrow$
trail-false-cls $\Gamma (C_0 \cdot \gamma) \Rightarrow \neg \text{trail-defined-lit } \Gamma (L \cdot l \ \gamma) \Rightarrow$
is-ingu $\mu \{\text{atm-of ' set-mset (add-mset } L \ C_1)\} \Rightarrow$
propagate *N* $\beta (\Gamma, U, \text{None}) (\text{trail-propagate } \Gamma (L \cdot l \ \mu) (C_0 \cdot \mu) \ \gamma, U, \text{None})$
apply (*rule propagateI*[of *C N U L C' \gamma \beta - - \Gamma \mu*]; *assumption?*)
by (*metis Melem-subst-cls*)

inductive *decide* :: (*f*, *v*) *term clause fset* \Rightarrow (*f*, *v*) *term* \Rightarrow (*f*, *v*) *state* \Rightarrow
(*f*, *v*) *state* \Rightarrow *bool* **for** *N* β **where**
decideI: *is-ground-lit* $(L \cdot l \ \gamma) \Rightarrow$
 $\neg \text{trail-defined-lit } \Gamma (L \cdot l \ \gamma) \Rightarrow \text{atm-of } L \cdot a \ \gamma \preceq_B \beta \Rightarrow$
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* β **where**
conflictI: $D \in | N \cup U \Rightarrow \text{is-ground-cls } (D \cdot \gamma) \Rightarrow \text{trail-false-cls } \Gamma (D \cdot \gamma)$
 \Rightarrow
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* β **where**
skipI: $-L \notin \# D \cdot \sigma \Rightarrow$
skip *N* $\beta ((L, n) \# \Gamma, U, \text{Some } (D, \sigma)) (\Gamma, U, \text{Some } (D, \sigma))$

lemma $-(fst \ \mathcal{K}) \notin \# D \cdot \sigma \Rightarrow \text{skip } N \ \beta (\mathcal{K} \# \Gamma, U, \text{Some } (D, \sigma)) (\Gamma, U, \text{Some } (D, \sigma))$
by (*metis prod.exhaust-sel skipI*)

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

inductive *resolve* :: (*f*, *v*) term clause fset \Rightarrow (*f*, *v*) term \Rightarrow (*f*, *v*) state \Rightarrow
 (*f*, *v*) state \Rightarrow bool **for** *N* β **where**
resolveI: $\Gamma = \text{trail-propagate } \Gamma' \ K \ D \ \gamma_D \Longrightarrow K \cdot l \ \gamma_D = -(L \cdot l \ \gamma_C) \Longrightarrow$
 $\text{is-renaming } \varrho_C \Longrightarrow \text{is-renaming } \varrho_D \Longrightarrow$
 $\text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C) \cap \text{vars-cls } (\text{add-mset } K \ D \cdot \varrho_D) = \{ \} \Longrightarrow$
 $\text{is-ingu } \mu \{ \{ \text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D \} \} \Longrightarrow$
 $\text{is-grounding-merge } \gamma$
 $(\text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C)) \ (\text{rename-subst-domain } \varrho_C \ \gamma_C)$
 $(\text{vars-cls } (\text{add-mset } K \ D \cdot \varrho_D)) \ (\text{rename-subst-domain } \varrho_D \ \gamma_D) \Longrightarrow$
 $\text{resolve } N \ \beta \ (\Gamma, U, \text{Some } (\text{add-mset } L \ C, \gamma_C)) \ (\Gamma, U, \text{Some } ((C \cdot \varrho_C + D \cdot$
 $\varrho_D) \cdot \mu, \gamma))$

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

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

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* β *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 *resolve* $N' \beta' S S'$
 \neg *backtrack* $N' \beta' S S'$
proof –
from *assms* **obtain** $L C \gamma \Gamma U$ **where**
 S -def: $S = (\Gamma, U, \text{None})$ **and**
 S' -def: $S' = (\text{trail-propagate } \Gamma L C \gamma, U, \text{None})$
by (*auto elim: propagate.cases*)

show \neg *decide* $N' \beta' S S'$
using S -def S' -def
by (*auto simp add: decide-lit-def propagate-lit-def elim: decide.cases*)

show \neg *conflict* $N' \beta' S S'$
using S -def S' -def
by (*auto elim: conflict.cases*)

show \neg *skip* $N' \beta' S S'$
using S -def S' -def
by (*auto elim: skip.cases*)

show \neg *factorize* $N' \beta' S S'$
using S -def S' -def
by (*auto elim: factorize.cases*)

show \neg *resolve* $N' \beta' S S'$
using S -def S' -def
by (*auto elim: resolve.cases*)

show \neg *backtrack* $N' \beta' S S'$
using S -def S' -def
by (*auto elim: backtrack.cases*)

qed

lemma *decide-well-defined*:
assumes *decide* $N \beta S S'$
shows
 \neg *propagate* $N' \beta' S S'$
 \neg *conflict* $N' \beta' S S'$
 \neg *skip* $N' \beta' S S'$
 \neg *factorize* $N' \beta' S S'$
 \neg *resolve* $N' \beta' S S'$
 \neg *backtrack* $N' \beta' S S'$
proof –
from *assms* **obtain** $L \gamma \Gamma U$ **where**
 S -def: $S = (\Gamma, U, \text{None})$ **and**
 S' -def: $S' = (\text{trail-decide } \Gamma (L \cdot l \gamma), U, \text{None})$
by (*auto elim: decide.cases*)

show \neg *propagate* $N' \beta' S S'$

```

using S-def S'-def
by (auto simp add: decide-lit-def propagate-lit-def elim: propagate.cases)

show  $\neg$  conflict  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: conflict.cases)

show  $\neg$  skip  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: skip.cases)

show  $\neg$  factorize  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: factorize.cases)

show  $\neg$  resolve  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: resolve.cases)

show  $\neg$  backtrack  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: backtrack.cases)
qed

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 –
from assms obtain  $C \gamma \Gamma U$  where
  S-def:  $S = (\Gamma, U, \text{None})$  and
  S'-def:  $S' = (\Gamma, U, \text{Some } (C, \gamma))$ 
by (auto elim: conflict.cases)

show  $\neg$  propagate  $N' \beta' S S'$ 
using S-def S'-def
by (auto simp add: decide-lit-def propagate-lit-def elim: propagate.cases)

show  $\neg$  decide  $N' \beta' S S'$ 
using S-def S'-def
by (auto elim: decide.cases)

show  $\neg$  skip  $N' \beta' S S'$ 
using S-def S'-def

```

by (*auto elim: skip.cases*)

show \neg *factorize* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: factorize.cases*)

show \neg *resolve* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: resolve.cases*)

show \neg *backtrack* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: backtrack.cases*)

qed

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 –

from *assms* **obtain** $Ln \Gamma U opt$ **where**
S-def: $S = (Ln \# \Gamma, U, opt)$ **and**
S'-def: $S' = (\Gamma, U, opt)$
by (*auto elim: skip.cases*)

show \neg *propagate* $N' \beta' S S'$
using *S-def S'-def*
by (*auto simp add: decide-lit-def propagate-lit-def elim: propagate.cases*)

show \neg *decide* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: decide.cases*)

show \neg *conflict* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: conflict.cases*)

show \neg *factorize* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: factorize.cases*)

show \neg *resolve* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: resolve.cases*)

show \neg *backtrack* $N' \beta' S S'$
using *S-def S'-def*
by (*auto elim: backtrack.cases*)
qed

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* $N \beta S S'$
using *assms*
by (*auto elim!: propagate.cases decide.cases conflict.cases skip.cases factorize.cases*
resolve.cases backtrack.cases
simp: decide-lit-def propagate-lit-def)

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* $N \beta S S'$
using *assms*
by (*auto elim!: propagate.cases decide.cases conflict.cases skip.cases factorize.cases*
resolve.cases backtrack.cases
simp: decide-lit-def propagate-lit-def)

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 –
from *assms* **obtain** $\Gamma \Gamma'' U C \gamma$ **where**
S-def: $S = (\Gamma, U, \text{Some } (C, \gamma))$ **and**
S'-def: $S' = (\Gamma'', \text{finsert } (C) U, \text{None})$
by (*auto elim: backtrack.cases*)

```

show  $\neg$  propagate  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: propagate.cases)

show  $\neg$  decide  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: decide.cases)

show  $\neg$  conflict  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: conflict.cases)

show  $\neg$  skip  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: skip.cases)

show  $\neg$  factorize  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: factorize.cases)

show  $\neg$  resolve  $N' \beta' S S'$ 
  using S-def S'-def
  by (auto elim: resolve.cases)
qed

```

5.4 Some rules are right unique

```

lemma right-unique-skip: right-unique (skip N  $\beta$ )
proof (rule right-uniqueI)
  fix  $S S' S''$ 
  assume step1: skip N  $\beta$  S S' and step2: skip N  $\beta$  S S''
  show  $S' = S''$ 
  using step1
proof (cases N  $\beta$  S S' rule: skip.cases)
  case hyps1: (skipI L D  $\sigma$  n  $\Gamma$  U)
  show ?thesis
  using step2[unfolded hyps1]
proof (cases N  $\beta$  ((L, n) #  $\Gamma$ , U, Some (D,  $\sigma$ )) S'' rule: skip.cases)
  case skipI
  with hyps1 show ?thesis
  by simp
  qed
qed
qed

```

5.5 Miscellaneous Lemmas

```

lemma conflict-set-after-factorization:
  assumes fact: factorize N  $\beta$  S S' and conflict-S: state-conflict S = Some (C,  $\gamma$ )

```

shows $\exists C' \gamma'. \text{state-conflict } S' = \text{Some } (C', \gamma') \wedge \text{set-mset } (C \cdot \gamma) = \text{set-mset } (C' \cdot \gamma')$

using *fact*

proof (*cases* $N \beta S S'$ *rule: factorize.cases*)

case (*factorizeI* $L \gamma L' \mu \Gamma U D$)

from $\langle L \cdot l \gamma = L' \cdot l \gamma \rangle$ **have** *is-unifier* $\gamma \{ \text{atm-of } L, \text{atm-of } L' \}$

by (*auto intro!*: *is-unifier-alt*[*THEN iffD2*] *intro: subst-atm-of-eqI*)

hence $\mu \odot \gamma = \gamma$

using $\langle \text{is-imgu } \mu \{ \{ \text{atm-of } L, \text{atm-of } L' \} \} \rangle$

by (*simp add: is-imgu-def is-unifiers-def*)

have $L \cdot l \mu \cdot l \gamma = L \cdot l \gamma$

using $\langle \mu \odot \gamma = \gamma \rangle$

by (*metis subst-lit-comp-subst*)

moreover have $D \cdot \mu \cdot \gamma = D \cdot \gamma$

using $\langle \mu \odot \gamma = \gamma \rangle$

by (*metis subst-cls-comp-subst*)

ultimately show *?thesis*

using *conflict-S*[*symmetric*]

unfolding *factorizeI*(1,2)

by (*simp add: \langle L \cdot l \gamma = L' \cdot l \gamma \rangle*)

qed

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

assumes

no-conf: \# S'. conflict N \beta S S' and

could-conf: state-conflict S = None and

C-in: C |\in| N |\cup| state-learned S and

gr-C-\gamma: is-ground-cls (C \cdot \gamma)

shows $\neg \text{trail-false-cls } (\text{state-trail } S) (C \cdot \gamma)$

proof (*rule notI*)

assume *tr-false: trail-false-cls (state-trail S) (C \cdot \gamma)*

from *could-conf* **obtain** ΓU **where** *S-def: S = (\Gamma, U, None)*

by (*metis prod-cases3 state-conflict-simp*)

have *conflict* $N \beta (\Gamma, U, None) (\Gamma, U,$

Some (C, restrict-subst-domain (vars-cls C) \gamma))

proof (*rule conflictI*)

show $C |\in| N |\cup| U$

using *C-in* **by** (*simp add: S-def*)

next

show *is-ground-cls* $(C \cdot \text{restrict-subst-domain } (\text{vars-cls } C) \gamma)$

using *gr-C-\gamma* **by** (*simp add: subst-cls-restrict-subst-domain*)

next

show *trail-false-cls* $\Gamma (C \cdot \text{restrict-subst-domain } (\text{vars-cls } C) \gamma)$

using *tr-false* by (*simp add: S-def subst-cls-restrict-subst-domain*)
 qed
 with *no-conf* show *False*
 by (*simp add: S-def*)
 qed

lemma *scl-mempty-not-in-sate-learned*:
 $scl\ N\ \beta\ S\ S' \implies \{\#\} \notin state\text{-}learned\ S \implies \{\#\} \notin state\text{-}learned\ S'$
unfolding *scl-def*
 by (*elim disjE propagate.cases decide.cases conflict.cases skip.cases factorize.cases*
resolve.cases backtrack.cases) *simp-all*

lemma *conflict-if-mempty-in-initial-clauses-and-no-conflict*:
 assumes $\{\#\} \in N$ and *state-conflict* $S = None$
 shows *conflict* $N\ \beta\ S$ (*state-trail* S , *state-learned* S , *Some* ($\{\#\}$, *Var*))
proof –
 from *assms*(2) obtain $\Gamma\ U$ where *S-def*: $S = (\Gamma, U, None)$
 by (*metis snd-conv state-conflict-def surj-pair*)

show *?thesis*
unfolding *S-def state-trail-simp state-learned-simp*
proof (*rule conflictI*[of $\{\#\}$ $N - Var - -$, *unfolded subst-cls-empty*])
 from *assms*(1) show $\{\#\} \in N \mid \cup U$
 by *simp*
 qed *simp-all*
 qed

lemma *conflict-initial-state-if-mempty-in-intial-clauses*:
 $\{\#\} \in N \implies \text{conflict}\ N\ \beta\ \text{initial-state}\ (\[], \{\#\}, \text{Some}\ (\{\#\}, \text{Var}))$
 using *conflict-if-mempty-in-initial-clauses-and-no-conflict* by *auto*

lemma *conflict-empty-trail*:
 assumes *conf*: *conflict* $N\ \beta\ S\ S'$ and *empty-trail*: *state-trail* $S = []$
 shows $\{\#\} \in N \mid \cup U$ *state-learned* S
 using *conf*
proof (*cases* $N\ \beta\ S\ S'$ *rule: conflict.cases*)
 case (*conflictI* $D\ U\ \gamma\ \Gamma$)
 from *empty-trail* have $\Gamma = []$
unfolding *conflictI*(1,2) by *simp*
 with $\langle \text{trail-false-cls}\ \Gamma\ (D \cdot \gamma) \rangle$ have $D = \{\#\}$
 using *not-trail-false-Nil*(2) *subst-cls-empty-iff* by *blast*
 with $\langle D \in N \mid \cup U \rangle$ show *?thesis*
unfolding *conflictI*(1,2) by *simp*
 qed

lemma *conflict-empty-trail'*:
 assumes $\{\#\} \in N \mid \cup U$
 shows $\exists S'. \text{conflict}\ N\ \beta\ (\[], U, None)\ S'$
 by (*metis assms is-ground-cls-empty not-trail-false-ground-cls-if-no-conflict state-conflict-simp*)

state-learned-simp subst-cls-empty trail-false-cls-mempty)

lemma *empty-in-iff-ex-conflict*: $\{\#\} \mid \in \mid N \mid \cup \mid U \longleftrightarrow (\exists S'. \text{conflict } N \beta (\ [], U, \text{None}) S')$
by (*metis conflict-empty-trail conflict-empty-trail' state-learned-simp state-trail-simp*)

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

assumes *conflict* $N \beta$ *initial-state* S
shows $\exists \gamma. S = (\ [], \{\ \}, \text{Some } (\{\#\}, \gamma))$

using *assms(1)*

proof (*cases rule: conflict.cases*)

case (*conflictI* $D \gamma$)

from $\langle \text{trail-false-cls } \ [] (D \cdot \gamma) \rangle$ **have** $D \cdot \gamma = \{\#\}$

using *not-trail-false-Nil(2)* **by** *blast*

hence $D = \{\#\}$

by *simp*

thus *?thesis*

using $\langle S = (\ [], \{\ \}, \text{Some } (D, \gamma)) \rangle$ **by** *simp*

qed

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

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

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

apply (*rule notI*)

unfolding *scl-def*

apply (*insert assms*)

by (*elim exE disjE propagate.cases decide.cases conflict.cases skip.cases factorize.cases*

resolve.cases backtrack.cases) *simp-all*

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

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

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

proof –

from *D-in* **obtain** $D' \gamma'$ **where**

$D'\text{-in}$: $D' \mid \in \mid N \mid \cup \mid U$ **and** *D-def*: $D = D' \cdot \gamma'$ **and** *gr-D- γ* : *is-ground-cls* $(D' \cdot \gamma')$

unfolding *grounding-of-clss-def* *grounding-of-cls-def*

by (*smt* (*verit*, *ccfv-threshold*) *D-in UN-iff* *grounding-ground mem-Collect-eq union-fset*)

define γ **where**

$\gamma \equiv \text{restrict-subst-domain } (\text{vars-cls } D') \gamma'$

have *conflict* $N \beta (\Gamma, U, \text{None}) (\Gamma, U, \text{Some } (D', \gamma))$

proof (*rule conflictI[OF D'-in]*)

show *is-ground-cls* $(D' \cdot \gamma)$

```

    using gr-D- $\gamma$  by (simp add:  $\gamma$ -def subst-cls-restrict-subst-domain)
  next
    show trail-false-cls  $\Gamma$  ( $D' \cdot \gamma$ )
      using tr-false- $\Gamma$ -D by (simp add: D-def  $\gamma$ -def subst-cls-restrict-subst-domain)
    qed
  thus ?thesis
    by auto
qed

```

```

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 (rule notI, erule exE)
  fix S assume conflict N  $\beta$  ( $\Gamma, U, \text{None}$ ) S
  hence  $\exists S. \text{conflict } N \beta (Ln \# \Gamma, U, \text{None}) S$ 
  proof (cases N  $\beta$  - S rule: conflict.cases)
    case (conflictI D  $\gamma$ )
    have conflict N  $\beta$  ( $Ln \# \Gamma, U, \text{None}$ ) ( $Ln \# \Gamma, U, \text{Some } (D, \gamma)$ )
    proof (rule conflict.conflictI)
      show D  $\in$  N  $\cup$  U
      by (rule conflictI)
    next
      show is-ground-cls (D  $\cdot$   $\gamma$ )
      by (rule conflictI)
    next
      show trail-false-cls ( $Ln \# \Gamma$ ) (D  $\cdot$   $\gamma$ )
      using  $\langle \text{trail-false-cls } \Gamma (D \cdot \gamma) \rangle$ 
      by (simp add: trail-false-cls-def trail-false-lit-def)
    qed
  thus ?thesis
    by metis
  qed
with assms show False
  by argo
qed

```

```

lemma subst-domain-rename-subst-domain-subset-vars-cls-subst-cls:
  assumes  $\forall x. \text{is-Var } (\varrho_C x)$  and
    dom- $\gamma_C$ : subst-domain  $\gamma_C \subseteq \text{vars-cls } (\text{add-mset } L C)$ 
  shows subst-domain (rename-subst-domain  $\varrho_C \gamma_C$ )  $\subseteq$  vars-cls (add-mset L C  $\cdot$ 
 $\varrho_C$ )
proof -
  have subst-domain (rename-subst-domain  $\varrho_C \gamma_C$ )  $\subseteq$  the-Var '  $\varrho_C$  ' subst-domain
 $\gamma_C$ 
  using subst-domain-rename-subst-domain-subset[OF  $\langle \forall x. \text{is-Var } (\varrho_C x) \rangle$ ] by
  simp
  also have ...  $\subseteq$  the-Var '  $\varrho_C$  ' vars-cls (add-mset L C)
  using dom- $\gamma_C$  by auto
  also have ... = ( $\bigcup x \in \text{vars-cls } (\text{add-mset } L C). \text{vars-term } (\varrho_C x)$ )

```

using *image-the-Var-image-subst-renaming-eq*[*OF* $\langle \forall x. \text{is-Var } (\varrho_C x) \rangle$] **by** *simp*
also have $\dots = \text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C)$
using *vars-subst-cls-eq* **by** *metis*
finally show *dom-ren-dom- ϱ_C - γ_C* :
 $\text{subst-domain } (\text{rename-subst-domain } \varrho_C \ \gamma_C) \subseteq \text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C)$
by *assumption*
qed

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* $(\text{add-mset } L \ C \cdot \gamma_C)$ **and**
ground-prop: *is-ground-cls* $(\text{add-mset } K \ D \cdot \gamma_D)$ **and**
dom- γ_D : $\text{subst-domain } \gamma_D \subseteq \text{vars-cls } (\text{add-mset } K \ D)$ **and**
ren- ϱ_C : *is-renaming* ϱ_C **and**
ren- ϱ_D : *is-renaming* ϱ_D **and**
disjoint-vars: $\text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C) \cap \text{vars-cls } (\text{add-mset } K \ D \cdot \varrho_D) =$
 $\{\}$

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 –

have *subst-adapt- ϱ_D - γ_D* :

$$\text{subst-domain } (\text{rename-subst-domain } \varrho_D \ \gamma_D) \cap \text{vars-cls } (\text{add-mset } L \ C \cdot \varrho_C) =$$

$$\{\}$$

using *disjoint-vars ren- ϱ_D dom- γ_D*

subst-domain-rename-subst-domain-subset-vars-cls-subst-cls

by (*metis Int-commute Orderings.order-eq-iff ground-prop is-renaming-iff*
subst-renaming-subst-adapted vars-cls-subset-subst-domain-if-grounding)

show $C \cdot \varrho_C \cdot \gamma = C \cdot \gamma_C$

proof –

have $C \cdot \varrho_C \cdot \text{rename-subst-domain } \varrho_C \ \gamma_C = C \cdot \gamma_C$

proof (*rule subst-renaming-subst-adapted*[*OF ren- ϱ_C*])

show $\text{vars-cls } C \subseteq \text{subst-domain } \gamma_C$

using *vars-cls-subset-subst-domain-if-grounding*[*OF ground-conf*] **by** *simp*

qed

moreover have $C \cdot \varrho_C \cdot \text{rename-subst-domain } \varrho_D \ \gamma_D = C \cdot \varrho_C$

proof (*rule subst-cls-idem-if-disj-vars*)

show $\text{subst-domain } (\text{rename-subst-domain } \varrho_D \ \gamma_D) \cap \text{vars-cls } (C \cdot \varrho_C) = \{\}$

using *subst-adapt- ϱ_D - γ_D* **by** *auto*

qed

ultimately show *?thesis*

unfolding γ -*def* **by** *simp*

qed

show $D \cdot \varrho_D \cdot \gamma = D \cdot \gamma_D$

proof –

have $D \cdot \varrho_D \cdot \text{rename-subst-domain } \varrho_D \gamma_D = D \cdot \gamma_D$

proof (rule subst-renaming-subst-adapted[OF ren- ϱ_D])

show $\text{vars-cls } D \subseteq \text{subst-domain } \gamma_D$

using vars-cls-subset-subst-domain-if-grounding[OF ground-prop] by simp

qed

moreover have $D \cdot \gamma_D \cdot \text{rename-subst-domain } \varrho_C \gamma_C = D \cdot \gamma_D$

using ground-prop by simp

ultimately show ?thesis

unfolding γ -def by simp

qed

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

proof –

have $L \cdot l \varrho_C \cdot l \text{rename-subst-domain } \varrho_C \gamma_C = L \cdot l \gamma_C$

proof (rule subst-lit-renaming-subst-adapted[OF ren- ϱ_C])

show $\text{vars-lit } L \subseteq \text{subst-domain } \gamma_C$

using ground-conf

by (simp add: vars-lit-subset-subst-domain-if-grounding)

qed

moreover have $L \cdot l \varrho_C \cdot l \text{rename-subst-domain } \varrho_D \gamma_D = L \cdot l \varrho_C$

proof (rule subst-lit-idem-if-disj-vars)

show $\text{subst-domain } (\text{rename-subst-domain } \varrho_D \gamma_D) \cap \text{vars-lit } (L \cdot l \varrho_C) = \{\}$

using subst-adapt- ϱ_D - γ_D by auto

qed

ultimately show ?thesis

unfolding γ -def

by (simp add: literal.expand)

qed

moreover show $K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$

proof –

have $\bigwedge \sigma. K \cdot l \gamma_D \cdot l \sigma = K \cdot l \gamma_D$

using ground-prop by (simp add: is-ground-lit-def)

moreover have $K \cdot l \varrho_D \cdot l \text{rename-subst-domain } \varrho_D \gamma_D = K \cdot l \gamma_D$

proof (rule subst-lit-renaming-subst-adapted[OF ren- ϱ_D])

show $\text{vars-lit } K \subseteq \text{subst-domain } \gamma_D$

using ground-prop

by (simp add: vars-lit-subset-subst-domain-if-grounding)

qed

ultimately show ?thesis

unfolding γ -def

by (simp add: literal.expand)

qed

ultimately have $\text{atm-of } L \cdot a \varrho_C \cdot a \gamma = \text{atm-of } K \cdot a \varrho_D \cdot a \gamma$

using $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$

by (*metis atm-of-subst-lit atm-of-uminus*)
 hence *is-unifiers* $\gamma \{\{atm-of\ L \cdot a \varrho_C, atm-of\ K \cdot a \varrho_D\}\}$
 by (*simp add: is-unifiers-def is-unifier-alt*)

 moreover assume *imgu- μ* : *is-imgu* $\mu \{\{atm-of\ L \cdot a \varrho_C, atm-of\ K \cdot a \varrho_D\}\}$

 ultimately show $\mu \odot \gamma = \gamma$
 by (*auto simp: is-imgu-def*)
 qed

6 Invariants

6.1 Initial Literals Generalize Learned, Trail, and Conflict Literals

definition *clss-lits-generalize-clss-lits* where
clss-lits-generalize-clss-lits $N\ U \longleftrightarrow$
 $(\forall L \in \bigcup (set-mset\ 'U). \exists K \in \bigcup (set-mset\ 'N). generalizes-lit\ K\ L)$

lemma *clss-lits-generalize-clss-lits-if-superset*[*simp*]:
 assumes $N2 \subseteq N1$
 shows *clss-lits-generalize-clss-lits* $N1\ N2$
proof (*unfold clss-lits-generalize-clss-lits-def, rule ballI*)
 fix L
 assume *L-in*: $L \in \bigcup (set-mset\ 'N2)$
 show $\exists K \in \bigcup (set-mset\ 'N1). generalizes-lit\ K\ L$
 unfolding *generalizes-lit-def*
proof (*intro bexI exI conjI*)
 show $L \in \bigcup (set-mset\ 'N1)$
 using *L-in* $\langle N2 \subseteq N1 \rangle$ by *blast*
 next
 show $L \cdot l\ Var = L$
 by *simp*
 qed
 qed

lemma *clss-lits-generalize-clss-lits-subset*:
clss-lits-generalize-clss-lits $N\ U1 \implies U2 \subseteq U1 \implies clss-lits-generalize-clss-lits\ N\ U2$
 unfolding *clss-lits-generalize-clss-lits-def* by *blast*

lemma *clss-lits-generalize-clss-lits-insert*:
clss-lits-generalize-clss-lits $N\ (insert\ C\ U) \longleftrightarrow$
 $(\forall L \in \# C. \exists K \in \bigcup (set-mset\ 'N). generalizes-lit\ K\ L) \wedge clss-lits-generalize-clss-lits\ N\ U$
 unfolding *clss-lits-generalize-clss-lits-def* by *blast*

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

class-lits-generalize-class-lits $N1\ N2$ **and**
class-lits-generalize-class-lits $N2\ N3$
shows *class-lits-generalize-class-lits* $N1\ N3$
proof (*unfold class-lits-generalize-class-lits-def*, *rule ballI*)
fix $L3$
assume $L3 \in \bigcup (\text{set-mset } 'N3)$
then obtain $L2\ \sigma2$ **where** $L2 \in \bigcup (\text{set-mset } 'N2)$ **and** $L2 \cdot l\ \sigma2 = L3$
using *assms(2)[unfolded class-lits-generalize-class-lits-def]* *generalizes-lit-def* **by**
meson
then obtain $L1\ \sigma1$ **where** $L1 \in \bigcup (\text{set-mset } 'N1)$ **and** $L1 \cdot l\ \sigma1 = L2$
using *assms(1)[unfolded class-lits-generalize-class-lits-def]* *generalizes-lit-def* **by**
meson

thus $\exists K \in \bigcup (\text{set-mset } 'N1)$. *generalizes-lit* $K\ L3$
unfolding *generalizes-lit-def*
proof (*intro bexI exI conjI*)
show $L1 \cdot l (\sigma1 \odot \sigma2) = L3$
by (*simp add: $L1 \cdot l\ \sigma1 = L2$ $L2 \cdot l\ \sigma2 = L3$*)
qed *simp-all*
qed

lemma *class-lits-generalize-class-lits-subst-class*:
assumes *class-lits-generalize-class-lits* $N1\ N2$
shows *class-lits-generalize-class-lits* $N1\ ((\lambda C. C \cdot \sigma) 'N2)$
unfolding *class-lits-generalize-class-lits-def*
proof (*rule ballI*)
fix L **assume** $L \in \bigcup (\text{set-mset } '(\lambda C. C \cdot \sigma) 'N2)$
then obtain $L2$ **where** $L2 \in \bigcup (\text{set-mset } 'N2)$ **and** $L\text{-def}: L = L2 \cdot l\ \sigma$ **by** *auto*
then obtain $L1\ \sigma1$ **where** $L1\text{-in}: L1 \in \bigcup (\text{set-mset } 'N1)$ **and** $L2\text{-def}: L2 = L1$
 $\cdot l\ \sigma1$
using *assms[unfolded class-lits-generalize-class-lits-def]*
unfolding *generalizes-lit-def* **by** *metis*

show $\exists K \in \bigcup (\text{set-mset } 'N1)$. *generalizes-lit* $K\ L$
unfolding *generalizes-lit-def*
proof (*intro bexI exI*)
show $L1 \in \bigcup (\text{set-mset } 'N1)$
by (*rule L1-in*)
next
show $L1 \cdot l (\sigma1 \odot \sigma) = L$
unfolding $L\text{-def}\ L2\text{-def}$ **by** *simp*
qed
qed

lemma *class-lits-generalize-class-lits-singleton-subst-class*:
class-lits-generalize-class-lits $N\ \{C\} \implies \text{class-lits-generalize-class-lits } N\ \{C \cdot \sigma\}$
by (*rule class-lits-generalize-class-lits-subst-class[of N {C} σ , simplified]*)

lemma *class-lits-generalize-class-lits-subst-class*:

assumes *clss-lits-generalize-clss-lits* N $\{add\text{-}mset\ L1\ (add\text{-}mset\ L2\ C)\}$
shows *clss-lits-generalize-clss-lits* N $\{add\text{-}mset\ (L1 \cdot l\ \sigma)\ (C \cdot \sigma)\}$
proof (*rule clss-lits-generalize-clss-lits-trans*)
show *clss-lits-generalize-clss-lits* N $\{add\text{-}mset\ L1\ (add\text{-}mset\ L2\ C) \cdot \sigma\}$
by (*rule clss-lits-generalize-clss-lits-singleton-subst-cl* $[of\ N - \sigma, OF\ assms]$)
next
show *clss-lits-generalize-clss-lits* $\{add\text{-}mset\ L1\ (add\text{-}mset\ L2\ C) \cdot \sigma\}$
 $\{add\text{-}mset\ (L1 \cdot l\ \sigma)\ (C \cdot \sigma)\}$
apply (*simp add: clss-lits-generalize-clss-lits-def generalizes-lit-def*)
using *subst-lit-id-subst* **by** *blast*
qed

definition *initial-lits-generalize-learned-trail-conflict* **where**
initial-lits-generalize-learned-trail-conflict $N\ S \longleftrightarrow clss\text{-}lits\text{-}generalize\text{-}clss\text{-}lits\ (fset\ N)$

$(fset\ (state\text{-}learned\ S\ |\cup|\ clss\text{-}of\text{-}trail\ (state\text{-}trail\ S)\ |\cup|\ (case\ state\text{-}conflict\ S\ of\ None \Rightarrow \{\|\}\ | Some\ (C, -) \Rightarrow \{C\})))$

lemma *initial-lits-generalize-learned-trail-conflict-initial-state* $[simp]$:
initial-lits-generalize-learned-trail-conflict N *initial-state*
unfolding *initial-lits-generalize-learned-trail-conflict-def* **by** *simp*

lemma *propagate-preserves-initial-lits-generalize-learned-trail-conflict*:
 $propagate\ N\ \beta\ S\ S' \Longrightarrow initial\text{-}lits\text{-}generalize\text{-}learned\text{-}trail\text{-}conflict\ N\ S \Longrightarrow initial\text{-}lits\text{-}generalize\text{-}learned\text{-}trail\text{-}conflict\ N\ S'$

proof (*induction S S' rule: propagate.induct*)
case (*propagateI C U L C' γ C₀ C₁ Γ μ*)

from *propagateI.prem*s **have**

$N\text{-}generalize: clss\text{-}lits\text{-}generalize\text{-}clss\text{-}lits\ (fset\ N)\ (fset\ (U\ |\cup|\ clss\text{-}of\text{-}trail\ \Gamma))$
unfolding *initial-lits-generalize-learned-trail-conflict-def* **by** *simp-all*

from *propagateI.hyps* **have**

$C\text{-}in: C \in | N\ |\cup|\ U$ **and**
 $C\text{-}def: C = add\text{-}mset\ L\ C'$ **and**
 $C_0\text{-}def: C_0 = \{\#K \in \# C'. K \cdot l\ \gamma \neq L \cdot l\ \gamma\#\}$ **by** *simp-all*

have *clss-lits-generalize-clss-lits* $(fset\ N)$
 $(insert\ (add\text{-}mset\ L\ C_0 \cdot \mu)\ (fset\ (U\ |\cup|\ clss\text{-}of\text{-}trail\ \Gamma)))$
unfolding *clss-lits-generalize-clss-lits-insert*

proof (*rule conjI*)

show $\forall L \in \# add\text{-}mset\ L\ C_0 \cdot \mu. \exists K \in \cup (set\text{-}mset\ ' fset\ N). generalizes\text{-}lit\ K$
 L

proof (*rule ballI*)

fix K **assume** $K \in \# add\text{-}mset\ L\ C_0 \cdot \mu$
hence $K = L \cdot l\ \mu \vee (\exists M. M \in \# C_0 \wedge K = M \cdot l\ \mu)$
by *auto*
then obtain K' **where** $K'\text{-}in: K' \in \# C$ **and** $K\text{-}def: K = K' \cdot l\ \mu$
using $C_0\text{-}def$ $C\text{-}def$ **by** *auto*

```

obtain  $D L_D$  where  $D \in N$  and  $L_D \in \# D$  and generalizes-lit  $L_D K'$ 
using  $K'$ -in  $C$ -in  $N$ -generalize[unfolded clss-lits-generalize-clss-lits-def]
by (metis (mono-tags, opaque-lifting) UN-iff funion-iff generalizes-lit-refl)

show  $\exists K' \in \bigcup (\text{set-mset } \text{'fset } N)$ . generalizes-lit  $K' K$ 
proof (rule bxI)
  show generalizes-lit  $L_D K$ 
    using  $\langle \text{generalizes-lit } L_D K' \rangle$ 
    by (metis generalizes-lit-def K-def subst-lit-comp-subst)
  next
    show  $\langle L_D \in \bigcup (\text{set-mset } \text{'fset } N) \rangle$ 
    using  $\langle D \in N \rangle \langle L_D \in \# D \rangle$ 
    by (meson UN-I)
  qed
qed
next
  show clss-lits-generalize-clss-lits (fset  $N$ ) (fset ( $U \mid \cup \mid$  clss-of-trail  $\Gamma$ ))
  by (rule N-generalize)
qed
thus ?case
  by (simp add: initial-lits-generalize-learned-trail-conflict-def propagate-lit-def)
qed

lemma decide-preserves-initial-lits-generalize-learned-trail-conflict:
  decide  $N \beta S S' \implies$  initial-lits-generalize-learned-trail-conflict  $N S \implies$ 
  initial-lits-generalize-learned-trail-conflict  $N S'$ 
proof (induction S S' rule: decide.induct)
  case (decideI L  $\Gamma U$ )
  thus ?case
  by (simp add: decide-lit-def initial-lits-generalize-learned-trail-conflict-def)
qed

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  $N S'$ 
  using assms(1)
proof (cases N  $\beta S S'$  rule: conflict.cases)
  case (conflictI D U  $\gamma \Gamma$ )
  from assms(2) have clss-lits-generalize-clss-lits (fset  $N$ ) (fset ( $U \mid \cup \mid$  clss-of-trail
 $\Gamma$ ))
  unfolding conflictI(1) by (simp add: initial-lits-generalize-learned-trail-conflict-def)
  hence ball-U- $\Gamma$ -generalize:
   $\bigwedge L. L \in \bigcup (\text{set-mset } \text{'fset } (U \mid \cup \mid \text{clss-of-trail } \Gamma)) \implies$ 
   $\exists K \in \bigcup (\text{set-mset } \text{'fset } N)$ . generalizes-lit  $K L$ 
  unfolding clss-lits-generalize-clss-lits-def by blast

have clss-lits-generalize-clss-lits (fset  $N$ ) (insert D (fset ( $U \mid \cup \mid$  clss-of-trail  $\Gamma$ )))
  unfolding clss-lits-generalize-clss-lits-def

```

proof (*rule ballI*)
fix L **assume** $L \in \bigcup (\text{set-mset } \text{'insert } D \text{ (fset (U |}\cup\text{| clss-of-trail } \Gamma\text{))})$
hence $L \in \text{set-mset } D \vee L \in \bigcup (\text{set-mset } \text{'(fset (U |}\cup\text{| clss-of-trail } \Gamma\text{))})$
by *simp*
thus $\exists K \in \bigcup (\text{set-mset } \text{'fset } N\text{). generalizes-lit } K L$
proof (*elim disjE*)
assume $L\text{-in: } L \in \# D$
show *?thesis*
using $\langle D \mid \in \mid N \mid \cup \mid U \rangle$ [*unfolded funion-iff*]
proof (*elim disjE*)
show $D \mid \in \mid N \implies ?thesis$
using $L\text{-in}$
by (*meson UN-I generalizes-lit-refl*)
next
assume $D \mid \in \mid U$
hence $\exists K \in \bigcup (\text{set-mset } \text{'fset } N\text{). generalizes-lit } K L$
using *ball-U- Γ -generalize*[of L] $L\text{-in}$
using *mk-disjoint-finsert* **by** *fastforce*
thus *?thesis*
by *metis*
qed
next
show $L \in \bigcup (\text{set-mset } \text{'fset (U |}\cup\text{| clss-of-trail } \Gamma\text{))} \implies ?thesis$
using *ball-U- Γ -generalize* **by** *simp*
qed
qed
then show *?thesis*
using *assms(2)*
unfolding *conflictI(1,2)*
by (*simp add: initial-lits-generalize-learned-trail-conflict-def*)
qed

lemma *skip-preserves-initial-lits-generalize-learned-trail-conflict*:
 $\text{skip } N \beta S S' \implies \text{initial-lits-generalize-learned-trail-conflict } N S \implies$
 $\text{initial-lits-generalize-learned-trail-conflict } N S'$
proof (*induction S S' rule: skip.induct*)
case (*skipI L D σ Cl Γ U*)
then show *?case*
unfolding *initial-lits-generalize-learned-trail-conflict-def*
unfolding *state-learned-simp state-conflict-simp state-trail-simp option.case prod.case*
by (*auto elim: clss-lits-generalize-clss-lits-subset*)
qed

lemma *factorize-preserves-initial-lits-generalize-learned-trail-conflict*:
 $\text{factorize } N \beta S S' \implies \text{initial-lits-generalize-learned-trail-conflict } N S \implies$
 $\text{initial-lits-generalize-learned-trail-conflict } N S'$
proof (*induction S S' rule: factorize.induct*)
case (*factorizeI L γ L' μ Γ U D*)

moreover have *clss-lits-generalize-clss-lits* (fset N) {add-mset ($L \cdot l \ \mu$) ($D \cdot \mu$)}
using *factorizeI*
unfolding *initial-lits-generalize-learned-trail-conflict-def*
unfolding *state-proj-simp option.case prod.case*
apply (*simp add: clss-lits-generalize-clss-lits-insert generalizes-lit-def*)
by (*smt (verit, best) Melem-subst-cls subst-lit-comp-subst*)
ultimately show ?*case*
unfolding *initial-lits-generalize-learned-trail-conflict-def*
by (*simp add: clss-lits-generalize-clss-lits-insert[of fset N]*)
qed

lemma *resolve-preserves-initial-lits-generalize-learned-trail-conflict*:
resolve N β S S' \implies initial-lits-generalize-learned-trail-conflict N S \implies
initial-lits-generalize-learned-trail-conflict N S'
proof (*induction S S' rule: resolve.induct*)
case (*resolveI Γ Γ' K D δ_D L δ_C ϱ_C ϱ_D C μ γ U*)
moreover have *clss-lits-generalize-clss-lits* (fset N) {(C \cdot ϱ_C + D \cdot ϱ_D) \cdot μ }
proof –
from *resolveI.prem*s **have**
N-lits-sup: clss-lits-generalize-clss-lits (fset N)
(*fset (U \sqcup |clss-of-trail Γ \sqcup | {add-mset L C|})*)
unfolding *initial-lits-generalize-learned-trail-conflict-def* **by** *simp*

have *clss-lits-generalize-clss-lits* (fset N) {C \cdot ϱ_C \cdot μ }
proof –
from *N-lits-sup* **have** *clss-lits-generalize-clss-lits* (fset N) {add-mset L C}
by (*simp add: clss-lits-generalize-clss-lits-insert*)
hence *clss-lits-generalize-clss-lits* (fset N) {C}
by (*simp add: clss-lits-generalize-clss-lits-def*)
thus ?*thesis*
by (*auto intro: clss-lits-generalize-clss-lits-singleton-subst-cls*)
qed

moreover have *clss-lits-generalize-clss-lits* (fset N) {D \cdot ϱ_D \cdot μ }
proof –
from *N-lits-sup* **have** *clss-lits-generalize-clss-lits* (fset N) (*fset (clss-of-trail*
 $\Gamma)$)
by (*rule clss-lits-generalize-clss-lits-subset*) *auto*
hence *clss-lits-generalize-clss-lits* (fset N) {add-mset K D}
unfolding *resolveI.hyps*
by (*simp add: clss-lits-generalize-clss-lits-insert propagate-lit-def*)
hence *clss-lits-generalize-clss-lits* (fset N) {D}
by (*simp add: clss-lits-generalize-clss-lits-def*)
thus ?*thesis*
by (*auto intro: clss-lits-generalize-clss-lits-singleton-subst-cls*)
qed

ultimately show ?*thesis*
by (*auto simp add: clss-lits-generalize-clss-lits-def*)
qed

ultimately show ?*case*

unfolding *initial-lits-generalize-learned-trail-conflict-def*
unfolding *state-trail-simp state-learned-simp state-conflict-simp*
unfolding *option.case prod.case*
by (*metis clss-lits-generalize-clss-lits-insert finsert.rep-eq funion-finsert-right*)
qed

lemma *backtrack-preserves-initial-lits-generalize-learned-trail-conflict:*
backtrack N β S S' ⇒ initial-lits-generalize-learned-trail-conflict N S ⇒
initial-lits-generalize-learned-trail-conflict N S'
proof (*induction S S' rule: backtrack.induct*)
case (*backtrackI Γ Γ' Γ'' L σ D U*)
then show *?case*
unfolding *initial-lits-generalize-learned-trail-conflict-def*
apply (*simp add: clss-of-trail-append*)
apply (*erule clss-lits-generalize-clss-lits-subset*)
by *blast*
qed

lemma *scl-preserves-initial-lits-generalize-learned-trail-conflict:*
assumes *scl N β S S' and initial-lits-generalize-learned-trail-conflict N S*
shows *initial-lits-generalize-learned-trail-conflict N S'*
using *assms unfolding scl-def*
using *propagate-preserves-initial-lits-generalize-learned-trail-conflict*
decide-preserves-initial-lits-generalize-learned-trail-conflict
conflict-preserves-initial-lits-generalize-learned-trail-conflict
skip-preserves-initial-lits-generalize-learned-trail-conflict
factorize-preserves-initial-lits-generalize-learned-trail-conflict
resolve-preserves-initial-lits-generalize-learned-trail-conflict
backtrack-preserves-initial-lits-generalize-learned-trail-conflict
by *metis*

6.2 Trail Literals Are Ground

definition *trail-lits-ground where*
trail-lits-ground S ↔ (∀ L ∈ fst ` set (state-trail S). is-ground-lit L)

lemma *trail-lits-ground-initial-state[simp]: trail-lits-ground initial-state*
by (*simp add: trail-lits-ground-def*)

lemma *propagate-preserves-trail-lits-ground:*
assumes *propagate N β S S' and trail-lits-ground S*
shows *trail-lits-ground S'*
using *assms(1)*
proof (*cases N β S S' rule: propagate.cases*)
case (*propagateI C U L C' γ C₀ C₁ Γ μ*)
hence *is-ground-lit (L ·l γ)*
by (*meson Melem-subst-cls is-ground-cls-def mset-subset-eqD mset-subset-eq-add-right union-single-eq-member*)

moreover have $\forall \tau. \text{is-unifiers } \tau \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L \ C_1) \} \longrightarrow \tau =$
 $\mu \odot \tau$
using $\langle \text{is-ingu } \mu \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L \ C_1) \} \rangle$
by $(\text{simp add: is-ingu-def})$

moreover have $\text{is-unifiers } \gamma \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L \ C_1) \}$
by $(\text{auto simp: is-unifiers-def is-unifier-alt } \langle C_1 = \{ \#K \in \# \ C'. \ K \cdot l \ \gamma = L \cdot l \ \gamma \# \} \rangle$
 $\text{intro: subst-atm-of-eqI})$

ultimately have $\text{is-ground-lit } (L \cdot l \ \mu \cdot l \ \gamma)$
by $(\text{metis subst-lit-comp-subst})$

moreover have $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{is-ground-lit } L$
using $\langle \text{trail-lits-ground } S \rangle$ **by** $(\text{simp add: propagateI(1) trail-lits-ground-def})$

ultimately show $?thesis$
by $(\text{simp add: propagateI(2) trail-lits-ground-def propagate-lit-def})$
qed

lemma $\text{decide-preserves-trail-lits-ground:}$
assumes $\text{decide } N \ \beta \ S \ S' \ \mathbf{and} \ \text{trail-lits-ground } S$
shows $\text{trail-lits-ground } S'$
using assms(1)
proof $(\text{cases } N \ \beta \ S \ S' \ \text{rule: decide.cases})$
case $(\text{decideI } L \ \gamma \ \Gamma \ U)$
hence $\text{is-ground-lit } (L \cdot l \ \gamma)$
by metis

moreover have $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{is-ground-lit } L$
using assms(2) **by** $(\text{simp add: decideI(1) trail-lits-ground-def})$

ultimately show $?thesis$
by $(\text{simp add: decideI(2) trail-lits-ground-def decide-lit-def})$
qed

lemma $\text{conflict-preserves-trail-lits-ground:}$
assumes $\text{conflict } N \ \beta \ S \ S' \ \mathbf{and} \ \text{trail-lits-ground } S$
shows $\text{trail-lits-ground } S'$
using assms **by** $(\text{auto simp: trail-lits-ground-def elim!: conflict.cases})$

lemma $\text{skip-preserves-trail-lits-ground:}$
assumes $\text{skip } N \ \beta \ S \ S' \ \mathbf{and} \ \text{trail-lits-ground } S$
shows $\text{trail-lits-ground } S'$
using assms **by** $(\text{auto simp: trail-lits-ground-def elim!: skip.cases})$

lemma $\text{factorize-preserves-trail-lits-ground:}$
assumes $\text{factorize } N \ \beta \ S \ S' \ \mathbf{and} \ \text{trail-lits-ground } S$
shows $\text{trail-lits-ground } S'$

using *assms* **by** (*auto simp: trail-lits-ground-def elim!: factorize.cases*)

lemma *resolve-preserves-trail-lits-ground:*

assumes *resolve* $N \beta S S'$ **and** *trail-lits-ground* S

shows *trail-lits-ground* S'

using *assms* **by** (*auto simp: trail-lits-ground-def elim!: resolve.cases*)

lemma *backtrack-preserves-trail-lits-ground:*

assumes *backtrack* $N \beta S S'$ **and** *trail-lits-ground* S

shows *trail-lits-ground* S'

using *assms* **by** (*auto simp: trail-lits-ground-def decide-lit-def elim!: backtrack.cases*)

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

assumes *scl* $N \beta S S'$ **and** *trail-lits-ground* S

shows *trail-lits-ground* S'

using *assms* **unfolding** *scl-def*

using *propagate-preserves-trail-lits-ground* *decide-preserves-trail-lits-ground*

conflict-preserves-trail-lits-ground *skip-preserves-trail-lits-ground*

factorize-preserves-trail-lits-ground *resolve-preserves-trail-lits-ground*

backtrack-preserves-trail-lits-ground

by *metis*

6.3 Trail Literals Are Defined Only Once

definition *trail-lits-consistent* **where**

trail-lits-consistent $S \longleftrightarrow \text{trail-consistent (state-trail } S)$

lemma *trail-lits-consistent-initial-state*[*simp*]: *trail-lits-consistent initial-state*

by (*simp add: trail-lits-consistent-def*)

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

assumes *propagate* $N \beta S S'$ **and** *invar: trail-lits-consistent* S

shows *trail-lits-consistent* S'

using *assms*(1)

proof (*cases* $N \beta S S'$ *rule: propagate.cases*)

case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)

have $L \cdot l \mu \cdot l \gamma = L \cdot l \gamma$

proof –

have *is-unifiers* $\gamma \{ \text{atm-of } \langle \text{set-mset (add-mset } L C_1) \rangle \}$

by (*smt (verit, del-insts) finite-imageI finite-set-mset image-iff insert-iff*

is-unifier-alt

is-unifiers-def *local.propagateI*(8) *mem-Collect-eq set-mset-add-mset-insert*
set-mset-filter singletonD subst-atm-of-eqI)

hence $\gamma = \mu \odot \gamma$

using $\langle \text{is-imgu } \mu \{ \text{atm-of } \langle \text{set-mset (add-mset } L C_1) \rangle \} \rangle$

by (*simp add: is-imgu-def*)

thus *?thesis*

by (*metis subst-lit-comp-subst*)

qed
hence $\neg \text{trail-defined-lit } \Gamma (L \cdot l \mu \cdot l \gamma)$
using $\langle \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \rangle$ **by** *metis*

moreover from *invar* **have** *trail-consistent* Γ
by (*simp add: propagateI(1) trail-lits-consistent-def*)

ultimately show *?thesis*
by (*auto simp: propagateI(2) propagate-lit-def trail-lits-consistent-def*
intro: trail-consistent.Cons)

qed

lemma *decide-preserves-trail-lits-consistent:*
assumes *decide* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def decide-lit-def elim!: decide.cases*
intro: trail-consistent.Cons)

lemma *conflict-preserves-trail-lits-consistent:*
assumes *conflict* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def elim: conflict.cases*)

lemma *skip-preserves-trail-lits-consistent:*
assumes *skip* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def elim!: skip.cases elim: trail-consistent.cases*)

lemma *factorize-preserves-trail-lits-consistent:*
assumes *factorize* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def elim: factorize.cases*)

lemma *resolve-preserves-trail-lits-consistent:*
assumes *resolve* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def elim: resolve.cases*)

lemma *backtrack-preserves-trail-lits-consistent:*
assumes *backtrack* $N \beta S S'$ **and** *invar: trail-lits-consistent* S
shows *trail-lits-consistent* S'
using *assms*
by (*auto simp: trail-lits-consistent-def decide-lit-def elim!: backtrack.cases*
elim!: trail-consistent-if-suffix intro: suffixI)

lemma *scl-preserves-trail-lits-consistent*:
assumes *scl N β S S' and trail-lits-consistent S*
shows *trail-lits-consistent S'*
using *assms unfolding scl-def*
using *propagate-preserves-trail-lits-consistent decide-preserves-trail-lits-consistent*
conflict-preserves-trail-lits-consistent skip-preserves-trail-lits-consistent
factorize-preserves-trail-lits-consistent resolve-preserves-trail-lits-consistent
backtrack-preserves-trail-lits-consistent
by *metis*

lemma *trail-consistent-iff*: *trail-consistent Γ ↔ (∀Γ' Ln Γ''. Γ = Γ'' @ Ln # Γ' → ¬ trail-defined-lit Γ' (fst Ln))*
proof (*intro iffI allI impI*)
fix *Γ' Ln Γ''*
assume *trail-consistent Γ and Γ = Γ'' @ Ln # Γ'*
thus *¬ trail-defined-lit Γ' (fst Ln)*
proof (*induction Γ arbitrary: Γ'' rule: trail-consistent.induct*)
case *Nil*
thus *?case*
by *simp*
next
case *ind-hyps: (Cons Γ L u)*
thus *?case*
by (*cases Γ'' auto*)
qed
next
assume *∀Γ' Ln Γ''. Γ = Γ'' @ Ln # Γ' → ¬ trail-defined-lit Γ' (fst Ln)*
then show *trail-consistent Γ*
proof (*induction Γ*)
case *Nil*
thus *?case*
by *simp*
next
case (*Cons Ln Γ*)
thus *?case*
by (*cases Ln (simp add: trail-consistent.Cons)*)
qed
qed

6.4 Trail Closures Are False In Subtrails

definition *trail-closures-false'* where
trail-closures-false' S ↔ trail-closures-false (state-trail S)

lemma *trail-closures-false'-initial-state[simp]*: *trail-closures-false' initial-state*
by (*simp add: trail-closures-false'-def*)

lemma *propagate-preserves-trail-closures-false'*:

assumes *step*: propagate $N \beta S S'$ **and** *invar*: trail-closures-false' S
shows trail-closures-false' S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: propagate.cases)
case *step-hyps*: (propagateI $C U L C' \gamma C_0 C_1 \Gamma \mu$)
have *is-unifier* γ (atm-of ' set-mset (add-mset $L C_1$))
unfolding *step-hyps*
by (*auto simp add: is-unifier-alt intro: subst-atm-of-eqI*)
hence $\mu \odot \gamma = \gamma$
using $\langle \text{is-imgu } \mu \{ \text{atm-of ' set-mset (add-mset } L C_1) \} \rangle$
by (*simp add: is-imgu-def is-unifiers-def*)
hence trail-false-cls $\Gamma (C_0 \cdot \mu \cdot \gamma)$
using $\langle \text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \rangle$
by (*metis subst-cls-comp-subst*)
with *invar show ?thesis*
unfolding *step-hyps*(1,2)
by (*simp add: trail-closures-false'-def propagate-lit-def trail-closures-false.Cons*)
qed

lemma *decide-preserves-trail-closures-false'*:
assumes *step*: decide $N \beta S S'$ **and** *invar*: trail-closures-false' S
shows trail-closures-false' S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: decide.cases)
case *step-hyps*: (decideI $L \gamma \Gamma U$)
with *invar show ?thesis*
by (*simp add: trail-closures-false'-def decide-lit-def propagate-lit-def trail-closures-false.Cons*)
qed

lemma *conflict-preserves-trail-closures-false'*:
assumes *step*: conflict $N \beta S S'$ **and** *invar*: trail-closures-false' S
shows trail-closures-false' S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: conflict.cases)
case (conflictI $D U \gamma \Gamma$)
with *invar show ?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *skip-preserves-trail-closures-false'*:
assumes *step*: skip $N \beta S S'$ **and** *invar*: trail-closures-false' S
shows trail-closures-false' S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: skip.cases)
case (skipI $L D \sigma n \Gamma U$)
with *invar show ?thesis*
by (*simp add: trail-closures-false'-def trail-closures-false-ConsD*)
qed

lemma *factorize-preserves-trail-closures-false'*:
assumes *step: factorize* $N \beta S S'$ **and** *invar: trail-closures-false' S*
shows *trail-closures-false' S'*
using *step*
proof (*cases* $N \beta S S'$ *rule: factorize.cases*)
case (*factorizeI* $L \gamma L' \mu \Gamma U D$)
with *invar show ?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *resolve-preserves-trail-closures-false'*:
assumes *step: resolve* $N \beta S S'$ **and** *invar: trail-closures-false' S*
shows *trail-closures-false' S'*
using *step*
proof (*cases* $N \beta S S'$ *rule: resolve.cases*)
case (*resolveI* $\Gamma \Gamma' K D \gamma_D L \gamma_C \varrho_C \varrho_D C \mu \gamma U$)
with *invar show ?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *backtrack-preserves-trail-closures-false'*:
assumes *step: backtrack* $N \beta S S'$ **and** *invar: trail-closures-false' S*
shows *trail-closures-false' S'*
using *step*
proof (*cases* $N \beta S S'$ *rule: backtrack.cases*)
case (*backtrackI* $\Gamma \Gamma' \Gamma'' K L \sigma D U$)
with *invar show ?thesis*
by (*auto simp add: trail-closures-false'-def*
intro: trail-closures-false-ConsD trail-closures-false-appendD)
qed

lemma *scl-preserves-trail-closures-false'*:
assumes *scl* $N \beta S S'$ **and** *trail-closures-false' S*
shows *trail-closures-false' S'*
using *assms unfolding scl-def*
using *propagate-preserves-trail-closures-false' decide-preserves-trail-closures-false'*
conflict-preserves-trail-closures-false' skip-preserves-trail-closures-false'
factorize-preserves-trail-closures-false' resolve-preserves-trail-closures-false'
backtrack-preserves-trail-closures-false'
by *metis*

lemma *trail-closures-false* $\Gamma \longleftrightarrow$
 $(\forall K D \gamma \Gamma' \Gamma''. \Gamma = \Gamma'' @ \text{propagate-lit } K D \gamma \# \Gamma' \longrightarrow \text{trail-false-cls } \Gamma' (D \cdot \gamma))$
proof (*intro iffI allI impI*)
fix $K D \gamma \Gamma' \Gamma''$
assume *trail-closures-false* Γ **and** $\Gamma = \Gamma'' @ \text{trail-propagate } \Gamma' K D \gamma$
thus *trail-false-cls* $\Gamma' (D \cdot \gamma)$

proof (*induction* Γ *arbitrary*: $\Gamma'' \Gamma' K D \gamma$ *rule*: *trail-closures-false.induct*)
case *Nil*
thus *?case by simp*
next
case (*Cons* $u \Gamma L$)
thus *?case*
by (*metis* (*no-types, opaque-lifting*) *Cons-eq-append-conv list.inject*)
qed
next
assume $\forall K D \gamma \Gamma' \Gamma''. \Gamma = \Gamma'' @ \text{trail-propagate } \Gamma' K D \gamma \longrightarrow \text{trail-false-cls } \Gamma'$
 $(D \cdot \gamma)$
thus *trail-closures-false* Γ
by (*induction* Γ) (*simp-all add: trail-closures-false.Cons*)
qed

6.5 Trail Literals Were Propagated or Decided

inductive *trail-propagated-or-decided* **for** $N \beta U$ **where**

Nil[*simp*]: *trail-propagated-or-decided* $N \beta U [] |$

Propagate:

$C | \in | N | \cup | U \implies$
 $C = \text{add-mset } L C' \implies$
 $\text{is-ground-cls } (C \cdot \gamma) \implies$
 $\forall K \in \# C \cdot \gamma. \text{atm-of } K \preceq_B \beta \implies$
 $C_0 = \{\# K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma \#\} \implies$
 $C_1 = \{\# K \in \# C'. K \cdot l \gamma = L \cdot l \gamma \#\} \implies$
 $\text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \implies$
 $\neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$
 $\text{is-imgu } \mu \{\text{atm-of 'set-mset (add-mset } L C_1)\} \implies$
 $\text{trail-propagated-or-decided } N \beta U \Gamma \implies$
 $\text{trail-propagated-or-decided } N \beta U (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma) |$

Decide:

$\text{is-ground-lit } (L \cdot l \gamma) \implies$
 $\neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$
 $\text{atm-of } L \cdot a \gamma \preceq_B \beta \implies$
 $\text{trail-propagated-or-decided } N \beta U \Gamma \implies$
 $\text{trail-propagated-or-decided } N \beta U (\text{trail-decide } \Gamma (L \cdot l \gamma))$

lemma *trail-propagate-or-decide-suffixI*:

assumes *trail-propagated-or-decided* $N \beta U$ *ys* **and** *suffix* $xs ys$

shows *trail-propagated-or-decided* $N \beta U xs$

using *assms*

proof (*induction* ys *arbitrary*: xs *rule*: *trail-propagated-or-decided.induct*)

case *Nil*

hence $xs = []$

by *simp*

thus *?case*

by *simp*

next

```

case (Propagate  $C L C' \gamma C_0 C_1 \Gamma \mu$ )
from Propagate.prems obtain  $zs$  where
  tr-prop-eq: trail-propagate  $\Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma = zs @ xs$ 
  by (auto simp: suffix-def)
show ?case
proof (cases  $zs$ )
  case Nil
    with tr-prop-eq have  $xs = \text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma$ 
    by simp
    then show ?thesis
    by (simp add: trail-propagated-or-decided.Propagate[OF Propagate.hyps])
  next
    case (Cons  $Ln \Gamma'$ )
    with tr-prop-eq have suffix  $xs \Gamma$ 
    by (simp add: suffix-def propagate-lit-def)
    thus ?thesis
    by (rule Propagate.IH)
  qed
next
  case (Decide  $L \gamma \Gamma$ )
  from Decide.prems obtain  $zs$  where
    tr-deci-eq: trail-decide  $\Gamma (L \cdot l \gamma) = zs @ xs$ 
    by (auto simp: suffix-def)
  show ?case
  proof (cases  $zs$ )
    case Nil
      with tr-deci-eq have  $xs = \text{trail-decide } \Gamma (L \cdot l \gamma)$ 
      by simp
      then show ?thesis
      by (simp add: trail-propagated-or-decided.Decide[OF Decide.hyps])
    next
      case (Cons  $Ln \Gamma'$ )
      with tr-deci-eq have suffix  $xs \Gamma$ 
      by (simp add: suffix-def decide-lit-def)
      thus ?thesis
      by (rule Decide.IH)
    qed
  qed

```

definition *trail-propagated-or-decided'* **where**
trail-propagated-or-decided' $N \beta S =$
trail-propagated-or-decided $N \beta (\text{state-learned } S) (\text{state-trail } S)$

lemma *trail-propagated-or-decided-learned-finsert*:
assumes *trail-propagated-or-decided* $N \beta U \Gamma$
shows *trail-propagated-or-decided* $N \beta (\text{finsert } C U) \Gamma$
using *assms*
proof (*induction* Γ *rule: trail-propagated-or-decided.induct*)
case *Nil*

```

show ?case by (simp add: trail-propagated-or-decided.Nil)
next
  case (Propagate D L D'  $\gamma$  D0 D1  $\Gamma$   $\mu$ )

  from Propagate.hyps have D-in: D  $\in$  | N |  $\cup$  | finsert C U
    by simp

  have IH: trail-propagated-or-decided N  $\beta$  (finsert C U)  $\Gamma$ 
    by (rule Propagate.IH)

  show ?case
    using trail-propagated-or-decided.Propagate[OF D-in Propagate.hyps(2,3,4,5,6,7,8,9)
  IH] .
  next
    case (Decide L  $\gamma$   $\Gamma$ )
    then show ?case
      by (simp add: trail-propagated-or-decided.Decide)
  qed

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  $\Gamma_2$ 
  using assms
proof (induction  $\Gamma_1$  @  $\Gamma_2$  arbitrary:  $\Gamma_1$   $\Gamma_2$  rule: trail-propagated-or-decided.induct)
  case Nil
  thus ?case
    by simp
  next
  case (Propagate C L C'  $\gamma$  C0 C1  $\Gamma$   $\mu$ )
  hence tr-prop-eq- $\Gamma_1$ - $\Gamma_2$ :
     $\Gamma_1$  @  $\Gamma_2$  = trail-propagate  $\Gamma$  (L · l  $\mu$ ) (C0 ·  $\mu$ )  $\gamma$ 
    by simp
  thus ?case
    unfolding propagate-lit-def append-eq-Cons-conv
  proof (elim disjE conjE exE)
    assume  $\Gamma_1$  = [] and  $\Gamma_2$ -def:  $\Gamma_2$  = (L · l  $\mu$  · l  $\gamma$ , Some (C0 ·  $\mu$ , L · l  $\mu$ ,  $\gamma$ )) #  $\Gamma$ 
    show ?thesis
      unfolding  $\Gamma_2$ -def
      by (rule trail-propagated-or-decided.Propagate[unfolded propagate-lit-def];
        rule Propagate.hyps)
  next
  fix  $\Gamma_1'$ 
  assume  $\Gamma_1$  = (L · l  $\mu$  · l  $\gamma$ , Some (C0 ·  $\mu$ , L · l  $\mu$ ,  $\gamma$ )) #  $\Gamma_1'$  and  $\Gamma_1'$  @  $\Gamma_2$  =  $\Gamma$ 
  thus ?thesis
    using Propagate.hyps by blast
  qed
next
  case (Decide L  $\gamma$   $\Gamma$ )
  hence  $\Gamma_1$  @  $\Gamma_2$  = trail-decide  $\Gamma$  (L · l  $\gamma$ )

```

```

  by simp
thus ?case
  unfolding decide-lit-def append-eq-Cons-conv
proof (elim disjE conjE exE)
  assume  $\Gamma_1 = []$  and  $\Gamma_2\text{-def}: \Gamma_2 = (L \cdot l \ \gamma, \text{None}) \# \Gamma$ 
  show ?thesis
    unfolding  $\Gamma_2\text{-def}$ 
    by (rule trail-propagated-or-decided.Decide[unfolded decide-lit-def]; rule Decide.hyps)
  next
  fix  $\Gamma_1'$  assume  $\Gamma_1 = (L \cdot l \ \gamma, \text{None}) \# \Gamma_1'$  and  $\Gamma_1' @ \Gamma_2 = \Gamma$ 
  then show ?thesis
    using Decide.hyps by blast
qed
qed

```

lemma *trail-propagated-or-decided-initial-state*[simp]:
trail-propagated-or-decided' N β initial-state
by (*auto simp: trail-propagated-or-decided'-def intro: trail-propagated-or-decided.Nil*)

lemma *propagate-preserves-trail-propagated-or-decided*:
assumes *propagate N β S S'* **and** *trail-propagated-or-decided' N β S*
shows *trail-propagated-or-decided' N β S'*
using *assms(1)*
proof (*cases N β S S' rule: propagate.cases*)
case (*propagateI C U L C' γ C₀ C₁ Γ μ*)

from *propagateI(1) assms(2)* **have** *IH: trail-propagated-or-decided N β U Γ*
by (*simp add: trail-propagated-or-decided'-def*)

```

show ?thesis
  unfolding propagateI(2)
  apply (simp add: trail-propagated-or-decided'-def)
  by (rule trail-propagated-or-decided.Propagate[rotated -1, OF IH])
    (rule propagateI)+
qed

```

lemma *decide-preserves-trail-propagated-or-decided*:
assumes *decide N β S S'* **and** *trail-propagated-or-decided' N β S*
shows *trail-propagated-or-decided' N β S'*
using *assms(1)*
proof (*cases N β S S' rule: decide.cases*)
case (*decideI L γ Γ U*)

from *decideI(1) assms(2)* **have** *IH: trail-propagated-or-decided N β U Γ*
by (*simp add: trail-propagated-or-decided'-def*)

```

show ?thesis
  unfolding decideI(2)
  apply (simp add: trail-propagated-or-decided'-def)

```

by (rule trail-propagated-or-decided.Decide[rotated -1, OF IH])
 (rule decideI)+
 qed

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'* $N \beta S'$
 using *assms* by (auto simp: *trail-propagated-or-decided'-def elim: conflict.cases*)

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'* $N \beta S'$
 using *assms*(1)
proof (*cases* $N \beta S S'$ rule: *skip.cases*)
 case (*skipI* $L D \sigma n \Gamma U$)

from *invar* have *trail-propagated-or-decided* $N \beta U ((L, n) \# \Gamma)$
 unfolding *skipI*(1) by (simp add: *trail-propagated-or-decided'-def*)
 hence *trail-propagated-or-decided* $N \beta U \Gamma$
 by (*cases* $N \beta U (L, n) \# \Gamma$ rule: *trail-propagated-or-decided.cases*)
 (simp-all add: *propagate-lit-def decide-lit-def*)
 thus ?thesis
 unfolding *skipI*(2) by (simp add: *trail-propagated-or-decided'-def*)
 qed

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'* $N \beta S'$
 using *assms* by (auto simp: *trail-propagated-or-decided'-def elim: factorize.cases*)

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'* $N \beta S'$
 using *assms* by (auto simp: *trail-propagated-or-decided'-def elim: resolve.cases*)

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'* $N \beta S'$
 using *assms*(1)
proof (*cases* $N \beta S S'$ rule: *backtrack.cases*)
 case (*backtrackI* $\Gamma \Gamma' \Gamma'' K L \sigma D U$)

have *trail-propagated-or-decided* $N \beta (\text{finsert } (\text{add-mset } L D) U) \Gamma''$
proof (rule *trail-propagated-or-decided-learned-finsert*)
 from *invar* have *trail-propagated-or-decided* $N \beta U (\text{trail-decide } (\Gamma' @ \Gamma'') (-$
 ($L \cdot l \sigma$)))
 unfolding *backtrackI* by (simp add: *trail-propagated-or-decided'-def*)
then show *trail-propagated-or-decided* $N \beta U \Gamma''$
 by (*induction* (*trail-decide* $(\Gamma' @ \Gamma'') (- (L \cdot l \sigma))$)

```

      rule: trail-propagated-or-decided.induct)
    (simp-all add: decide-lit-def propagate-lit-def
      trail-propagated-or-decided-trail-append)
  qed
  thus ?thesis
    unfolding backtrackI by (simp add: trail-propagated-or-decided'-def)
  qed

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'
  using assms unfolding scl-def
  using propagate-preserves-trail-propagated-or-decided decide-preserves-trail-propagated-or-decided
  conflict-preserves-trail-propagated-or-decided skip-preserves-trail-propagated-or-decided
  factorize-preserves-trail-propagated-or-decided resolve-preserves-trail-propagated-or-decided
  backtrack-preserves-trail-propagated-or-decided
  by metis

definition trail-propagated-wf where
  trail-propagated-wf  $\Gamma \longleftrightarrow (\forall (L_\gamma, n) \in \text{set } \Gamma.
    \text{case } n \text{ of}
      \text{None} \Rightarrow \text{True}
    | \text{Some } (-, L, \gamma) \Rightarrow L_\gamma = L \cdot l \ \gamma)$ 

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 (rule iffI)
    show ?lhs  $\implies$  ?rhs
      unfolding trail-propagated-wf-def
      by fastforce
  next
    assume ?rhs
    show ?lhs
      unfolding trail-propagated-wf-def
      proof (rule ballI)
        fix  $\mathcal{K}$  assume  $\mathcal{K} \in \text{set } \Gamma$ 
        show case  $\mathcal{K}$  of (L $_\gamma$ , None)  $\implies$  True | (L $_\gamma$ , Some (x, L,  $\gamma$ ))  $\implies$  L $_\gamma = L \cdot l \ \gamma$ 
          unfolding case-prod-beta
          using  $\langle ?rhs \rangle$ [rule-format, OF  $\langle \mathcal{K} \in \text{set } \Gamma \rangle$ ]
          by (cases snd  $\mathcal{K}$ ) auto
      qed
  qed
  qed

lemma trail-propagated-wf-if-trail-propagated-or-decided:
  trail-propagated-or-decided N U  $\beta$   $\Gamma \implies$  trail-propagated-wf  $\Gamma$ 
  proof (induction  $\Gamma$  rule: trail-propagated-or-decided.induct)
    case Nil

```

```

then show ?case
  by (simp add: trail-propagated-wf-def)
next
  case (Propagate C L C'  $\gamma$  C0 C1  $\Gamma$   $\mu$ )
  then show ?case
    by (simp add: trail-propagated-wf-def propagate-lit-def)
next
  case (Decide L  $\gamma$   $\Gamma$ )
  then show ?case
    by (simp add: trail-propagated-wf-def decide-lit-def)
qed

```

lemma *trail-propagated-wf-if-trail-propagated-or-decided'*:
trail-propagated-or-decided' N β S \implies trail-propagated-wf (state-trail S)
unfolding *trail-propagated-or-decided'-def*
using *trail-propagated-wf-if-trail-propagated-or-decided .*

lemma *trail-propagated-lit-wf-initial-state*:
 $\forall \mathcal{K} \in \text{set (state-trail initial-state)}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
by *simp*

lemma *scl-preserves-trail-propagated-lit-wf*:
assumes *step: scl N β S S' and*
inv: $\forall \mathcal{K} \in \text{set (state-trail S)}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
shows $\forall \mathcal{K} \in \text{set (state-trail S')}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
using *step inv*
unfolding *scl-def*
proof (*elim disjE*)
assume *propagate N β S S'*
then obtain *L C γ where state-trail S' = trail-propagate (state-trail S) L C γ*
by (*auto elim: propagate.cases*)
thus $\forall \mathcal{K} \in \text{set (state-trail S')}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
using *inv by (simp add: propagate-lit-def)*
next
assume *decide N β S S'*
then obtain *L where state-trail S' = trail-decide (state-trail S) L*
by (*auto elim: decide.cases*)
thus $\forall \mathcal{K} \in \text{set (state-trail S')}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
using *inv by (simp add: decide-lit-def)*
next
assume *conflict N β S S'*
hence *state-trail S' = state-trail S*
by (*auto elim: conflict.cases*)
thus $\forall \mathcal{K} \in \text{set (state-trail S')}. \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some (D, K, } \gamma) \longrightarrow \text{fst } \mathcal{K} = K$

```

·l γ
  using inv by argo
next
  assume skip N β S S'
  thus ∀K∈set (state-trail S'). ∀D K γ. snd K = Some (D, K, γ) → fst K = K
·l γ
  using inv skip.simps by fastforce
next
  assume factorize N β S S'
  hence state-trail S' = state-trail S
  by (auto elim: factorize.cases)
  thus ∀K∈set (state-trail S'). ∀D K γ. snd K = Some (D, K, γ) → fst K = K
·l γ
  using inv by argo
next
  assume resolve N β S S'
  hence state-trail S' = state-trail S
  by (auto elim: resolve.cases)
  thus ∀K∈set (state-trail S'). ∀D K γ. snd K = Some (D, K, γ) → fst K = K
·l γ
  using inv by argo
next
  assume backtrack N β S S'
  then obtain Γ where state-trail S = Γ @ state-trail S'
  by (auto elim: backtrack.cases)
  thus ∀K∈set (state-trail S'). ∀D K γ. snd K = Some (D, K, γ) → fst K = K
·l γ
  using inv by simp
qed

```

6.6 Trail Atoms Are Less Than Bound

definition *trail-atoms-lt* where

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

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

by (*simp add: trail-atoms-lt-def*)

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

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

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

using *assms(1)*

proof (*cases* $N \beta S S'$ *rule: propagate.cases*)

case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)

hence *is-ground-lit* ($L \cdot l \gamma$)

by (*meson Melem-subst-cls is-ground-cls-def mset-subset-eqD mset-subset-eq-add-right union-single-eq-member*)

moreover have $\forall \tau. \text{is-unifiers } \tau \{ \text{atm-of 'set-mset (add-mset } L C_1) \} \longrightarrow \tau =$

$\mu \odot \tau$
using $\langle is_imgu \mu \{ atm_of \text{ ' set-mset (add-mset L C}_1) \} \rangle$
by (*simp add: is-imgu-def*)

moreover have $is_unifiers \gamma \{ atm_of \text{ ' set-mset (add-mset L C}_1) \}$
by (*auto simp: is-unifiers-def is-unifier-alt $\langle C_1 = \{ \#K \in \# C' . K \cdot l \gamma = L \cdot l \gamma \# \} \rangle$*)
intro: subst-atm-of-eqI

ultimately have $is_ground_lit (L \cdot l \mu \cdot l \gamma)$
by (*metis subst-lit-comp-subst*)

have $atm_of L \cdot a \mu \cdot a \gamma = atm_of L \cdot a \gamma$
proof –
have $\gamma = \mu \odot \gamma$
using $\langle is_unifiers \gamma \{ atm_of \text{ ' set-mset (add-mset L C}_1) \} \rangle$ *propagateI*
by (*simp add: is-imgu-def*)
thus *?thesis*
by (*metis subst-atm-comp-subst*)

qed

moreover from *propagateI* **have** $atm_of L \cdot a \gamma \preceq_B \beta$
by (*metis add-mset-add-single atm-of-subst-lit subst-cls-single subst-cls-union union-single-eq-member*)

ultimately have $atm_of L \cdot a \mu \cdot a \gamma \preceq_B \beta$
by *simp*
with $\langle trail_atoms_lt \beta S \rangle$ **show** *?thesis*
by (*simp add: trail-atoms-lt-def propagateI(1,2) propagate-lit-def*)

qed

lemma *decide-preserves-trail-atoms-lt:*
assumes $decide N \beta S S'$ **and** $trail_atoms_lt \beta S$
shows $trail_atoms_lt \beta S'$
using *assms* **by** (*auto simp: trail-atoms-lt-def decide-lit-def elim!: decide.cases*)

lemma *conflict-preserves-trail-atoms-lt:*
assumes $conflict N \beta S S'$ **and** $trail_atoms_lt \beta S$
shows $trail_atoms_lt \beta S'$
using *assms* **by** (*auto simp: trail-atoms-lt-def elim!: conflict.cases*)

lemma *skip-preserves-trail-atoms-lt:*
assumes $skip N \beta S S'$ **and** $trail_atoms_lt \beta S$
shows $trail_atoms_lt \beta S'$
using *assms* **by** (*auto simp: trail-atoms-lt-def elim!: skip.cases*)

lemma *factorize-preserves-trail-atoms-lt:*
assumes $factorize N \beta S S'$ **and** $trail_atoms_lt \beta S$
shows $trail_atoms_lt \beta S'$

using *assms* by (auto simp: trail-atoms-lt-def elim!: factorize.cases)

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

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

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

using *assms* by (auto simp: trail-atoms-lt-def elim!: resolve.cases)

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

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

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

using *assms* by (auto simp: trail-atoms-lt-def decide-lit-def elim!: backtrack.cases)

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

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

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

using *assms* **unfolding** *scl-def*

using *propagate-preserves-trail-atoms-lt* *decide-preserves-trail-atoms-lt*

conflict-preserves-trail-atoms-lt *skip-preserves-trail-atoms-lt*

factorize-preserves-trail-atoms-lt *resolve-preserves-trail-atoms-lt*

backtrack-preserves-trail-atoms-lt

by *metis*

6.7 Trail Resolved Literals Have Unique Polarity

definition *trail-resolved-lits-pol* where

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*

by (*simp* add: *trail-resolved-lits-pol-def*)

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'

using *step*

proof (*cases* $N \beta S S'$ rule: *propagate.cases*)

case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)

have *is-unifiers* $\gamma \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L C_1) \}$

unfolding $\langle C_1 = \{ \#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma \# \} \rangle$

by (auto simp add: *is-unifiers-def* *is-unifier-alt* intro: *subst-atm-of-eqI*)

hence $\mu \odot \gamma = \gamma$

using $\langle \text{is-ingu } \mu \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L C_1) \} \rangle$

by (*simp* add: *is-ingu-def*)

hence $L \cdot l \mu \cdot l \gamma = L \cdot l \gamma$ and $C_0 \cdot \mu \cdot \gamma = C_0 \cdot \gamma$

by (*simp*-all del: *subst-lit-comp-subst* *subst-cls-comp-subst*

add: *subst-lit-comp-subst*[*symmetric*] *subst-cls-comp-subst*[*symmetric*])

hence $\neg (L \cdot l \mu \cdot l \gamma) \notin \# C_0 \cdot \mu \cdot \gamma$

using $\langle C_0 = \{ \#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma \# \} \rangle \langle \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \rangle$

$\langle \text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \rangle$
by (*metis trail-defined-lit-iff-defined-uminus trail-defined-lit-iff-true-or-false trail-false-cls-def*)

moreover from invar have $\forall Ln \in \text{set } \Gamma. \forall C L \gamma. \text{snd } Ln = \text{Some } (C, L, \gamma)$
 $\longrightarrow - (L \cdot l \gamma) \notin \# C \cdot \gamma$
unfolding *propagateI(1,2) trail-resolved-lits-pol-def*
by *simp*

ultimately show *?thesis*
unfolding *propagateI(1,2)*
unfolding *trail-resolved-lits-pol-def propagate-lit-def state-proj-simp list.set*
by *fastforce*

qed

lemma *decide-preserves-trail-resolved-lits-pol:*
assumes *step: decide N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*
by (*auto simp: trail-resolved-lits-pol-def decide-lit-def elim: decide.cases*)

lemma *conflict-preserves-trail-resolved-lits-pol:*
assumes *step: conflict N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*
by (*auto simp: trail-resolved-lits-pol-def elim: conflict.cases*)

lemma *skip-preserves-trail-resolved-lits-pol:*
assumes *step: skip N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*
by (*auto simp: trail-resolved-lits-pol-def elim: skip.cases*)

lemma *factorize-preserves-trail-resolved-lits-pol:*
assumes *step: factorize N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*
by (*auto simp: trail-resolved-lits-pol-def elim: factorize.cases*)

lemma *resolve-preserves-trail-resolved-lits-pol:*
assumes *step: resolve N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*
by (*auto simp: trail-resolved-lits-pol-def propagate-lit-def elim!: resolve.cases*)

lemma *backtrack-preserves-trail-resolved-lits-pol:*
assumes *step: backtrack N β S S' and invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms*

by (auto simp: trail-resolved-lits-pol-def decide-lit-def ball-Un elim: backtrack.cases)

lemma *scl-preserves-trail-resolved-lits-pol*:
assumes *scl N β S S' and trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*
using *assms unfolding scl-def*
using *propagate-preserves-trail-resolved-lits-pol decide-preserves-trail-resolved-lits-pol*
conflict-preserves-trail-resolved-lits-pol skip-preserves-trail-resolved-lits-pol
factorize-preserves-trail-resolved-lits-pol resolve-preserves-trail-resolved-lits-pol
backtrack-preserves-trail-resolved-lits-pol
by *metis*

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*
by (*simp add: ground-closures-def*)

lemma *propagate-preserves-ground-closures*:
assumes *step: propagate N β S S' and invar: ground-closures S*
shows *ground-closures S'*
using *step*

proof (*cases N β S S' rule: propagate.cases*)
case (*propagateI C U L C' γ C₀ C₁ Γ μ*)

have *C-def: C = add-mset L (C₀ + C₁)*
using *propagateI(3-) by auto*

have *is-unifiers γ {atm-of 'set-mset (add-mset L C₁)}*
unfolding $\langle C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma \# \}$
by (*auto simp add: is-unifiers-def is-unifier-alt intro: subst-atm-of-eqI*)

hence $\mu \odot \gamma = \gamma$
using $\langle \text{is-ingu } \mu \ \{ \text{atm-of 'set-mset (add-mset L C}_1 \} \rangle$
by (*simp add: is-ingu-def*)

hence $L \cdot l \ \mu \cdot l \ \gamma = L \cdot l \ \gamma$ **and** $C_0 \cdot \mu \cdot \gamma = C_0 \cdot \gamma$
by (*simp-all del: subst-lit-comp-subst subst-cls-comp-subst*
add: subst-lit-comp-subst[symmetric] subst-cls-comp-subst[symmetric])

hence *is-ground-cls (add-mset L C₀ · μ · γ)*

using $\langle \text{is-ground-cls } (C \cdot \gamma) \rangle$

by (*simp add: C-def*)

thus *?thesis*

using *invar*

unfolding *propagateI(1,2)*

by (*simp add: ground-closures-def propagate-lit-def*)

qed

lemma *decide-preserves-ground-closures:*

assumes *step: decide* $N \beta S S'$ **and** *invar: ground-closures* S

shows *ground-closures* S'

using *assms*

by (*cases* $N \beta S S'$ *rule: decide.cases*) (*simp add: ground-closures-def decide-lit-def*)

lemma *conflict-preserves-ground-closures:*

assumes *step: conflict* $N \beta S S'$ **and** *invar: ground-closures* S

shows *ground-closures* S'

using *step*

proof (*cases* $N \beta S S'$ *rule: conflict.cases*)

case (*conflictI* $D U \gamma \Gamma$)

thus *?thesis*

using *invar*

unfolding *conflictI(1,2)*

by (*simp add: ground-closures-def*)

qed

lemma *skip-preserves-ground-closures:*

assumes *step: skip* $N \beta S S'$ **and** *invar: ground-closures* S

shows *ground-closures* S'

using *assms*

by (*cases* $N \beta S S'$ *rule: skip.cases*) (*simp add: ground-closures-def*)

lemma *factorize-preserves-ground-closures:*

assumes *step: factorize* $N \beta S S'$ **and** *invar: ground-closures* S

shows *ground-closures* S'

using *step*

proof (*cases* $N \beta S S'$ *rule: factorize.cases*)

case (*factorizeI* $L \gamma L' \mu \Gamma U D$)

have *is-unifier* $\gamma \{atm\text{-of } L, atm\text{-of } L'\}$

using $\langle L \cdot l \gamma = L' \cdot l \gamma \rangle [THEN\ subst\text{-atm\text{-of}\text{-eqI}]$

by (*simp add: is-unifier-alt*)

hence $\mu \odot \gamma = \gamma$

using $\langle is\text{-imgu } \mu \{\{atm\text{-of } L, atm\text{-of } L'\}\} \rangle$

by (*simp add: is-imgu-def is-unifiers-def*)

have *add-mset* $L D \cdot \mu \cdot \gamma = add\text{-mset } L D \cdot \gamma$

using $\langle \mu \odot \gamma = \gamma \rangle$

by (*metis subst-cls-comp-subst*)

hence *is-ground-cls* (*add-mset* $L D \cdot \mu \cdot \gamma$)

using *factorizeI(3-)* *invar*

unfolding *factorizeI(1,2)*

by (*simp add: ground-closures-def*)

thus *?thesis*

using *invar*

unfolding *factorizeI(1,2)*

by (*simp add: ground-closures-def*)
qed

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-cls* $(C \cdot \varrho_C) \cap$ *vars-cls* $(D \cdot \varrho_D) = \{\}$ **and**
ground-conf: *is-ground-cls* $(C \cdot \gamma_C)$ **and**
ground-prop: *is-ground-cls* $(D \cdot \gamma_D)$ **and**
merge- γ : *is-grounding-merge* γ
(vars-cls $(C \cdot \varrho_C)$) *(rename-subst-domain* ϱ_C $\gamma_C)$
(vars-cls $(D \cdot \varrho_D)$) *(rename-subst-domain* ϱ_D $\gamma_D)$

shows

$\forall L \in \# C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$
 $\forall K \in \# D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$

proof –

have $\forall x \in \text{vars-cls } (C \cdot \varrho_C). \text{vars-term } (\text{rename-subst-domain } \varrho_C \gamma_C x) = \{\}$
using *ground-conf ren- ϱ_C*
by (*metis is-ground-atm-iff-vars-empty is-ground-cls-is-ground-on-var*
subst-renaming-subst-adapted vars-cls-subset-subst-domain-if-grounding)

moreover have $\forall x \in \text{vars-cls } (D \cdot \varrho_D). \text{vars-term } (\text{rename-subst-domain } \varrho_D \gamma_D x) = \{\}$
using *ground-prop ren- ϱ_D*
by (*metis is-ground-atm-iff-vars-empty is-ground-cls-is-ground-on-var*
subst-renaming-subst-adapted vars-cls-subset-subst-domain-if-grounding)

ultimately have

ball-C- ϱ_C -apply- γ : $\forall x \in \text{vars-cls } (C \cdot \varrho_C). \gamma x = \text{rename-subst-domain } \varrho_C \gamma_C x$
and
ball-D- ϱ_D -apply- γ : $\forall x \in \text{vars-cls } (D \cdot \varrho_D). \gamma x = \text{rename-subst-domain } \varrho_D \gamma_D x$
using *disjoint-vars merge- γ*
unfolding *is-grounding-merge-def*
by *simp-all*

show $\forall L \in \# C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$

proof (*rule ballI*)

fix L **assume** *L-in*: $L \in \# C$

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

unfolding *subst-lit-comp-subst[symmetric]*

proof (*intro same-on-vars-lit ballI*)

fix x **assume** $x \in \text{vars-lit } L$

moreover obtain x' **where** $\varrho_C x = \text{Var } x'$

using *ren- ϱ_C*

by (*meson is-Var-def is-renaming-iff*)

ultimately have $x' \in \text{vars-lit } (L \cdot l \varrho_C)$

using *vars-subst-lit-eq* **by** *fastforce*

hence $\gamma x' = \text{rename-subst-domain } \varrho_C \gamma_C x'$

```

    using ball-C- $\varrho_C$ -apply- $\gamma$  L-in multi-member-split by force
  thus ( $\varrho_C \odot \gamma$ )  $x = \gamma_C x$ 
  apply (simp add: subst-compose-def  $\langle \varrho_C x = \text{Var } x' \rangle$ )
  by (metis (no-types, opaque-lifting) L-in Un-iff  $\langle \varrho_C x = \text{Var } x' \rangle \langle x \in \text{vars-lit}$ 
L $\rangle$ 
    ground-conf image-eqI insert-DiffM is-renaming-iff ren- $\varrho_C$  rename-subst-domain-def
    subsetD the-inv-f-f vars-cls-add-mset vars-cls-subset-subst-domain-if-grounding)
  qed
qed

show  $\forall K \in \# D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$ 
proof (rule ballI)
  fix K assume K-in:  $K \in \# D$ 
  show  $K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$ 
  unfolding subst-lit-comp-subst[symmetric]
  proof (intro same-on-vars-lit ballI)
    fix x assume  $x \in \text{vars-lit } K$ 
    moreover obtain  $x'$  where  $\varrho_D x = \text{Var } x'$ 
    using ren- $\varrho_D$ 
    by (meson is-Var-def is-renaming-iff)
    ultimately have  $x' \in \text{vars-lit } (K \cdot l \varrho_D)$ 
    using vars-subst-lit-eq by fastforce
    hence  $\gamma x' = \text{rename-subst-domain } \varrho_D \gamma_D x'$ 
    using ball-D- $\varrho_D$ -apply- $\gamma$  K-in multi-member-split by force
  thus ( $\varrho_D \odot \gamma$ )  $x = \gamma_D x$ 
  apply (simp add: subst-compose-def  $\langle \varrho_D x = \text{Var } x' \rangle$ )
  by (metis (no-types, opaque-lifting) K-in UnII  $\langle \varrho_D x = \text{Var } x' \rangle \langle x \in \text{vars-lit}$ 
K $\rangle$ 
    ground-prop image-eqI is-renaming-iff multi-member-split ren- $\varrho_D$  re-
    name-subst-domain-def
    subset-iff the-inv-f-f vars-cls-add-mset vars-cls-subset-subst-domain-if-grounding)
  qed
qed
qed

lemma resolve-preserves-ground-closures:
  assumes step: resolve N  $\beta$  S S' and invar: ground-closures S
  shows ground-closures S'
  using step
proof (cases N  $\beta$  S S' rule: resolve.cases)
  case (resolveI  $\Gamma \Gamma' K D \gamma_D L \gamma_C \varrho_C \varrho_D C \mu \gamma U$ )
  hence
    ren- $\varrho_C$ : is-renaming  $\varrho_C$  and
    ren- $\varrho_D$ : is-renaming  $\varrho_D$  and
    disjoint-vars: vars-cls (add-mset L C  $\cdot \varrho_C$ )  $\cap$  vars-cls (add-mset K D  $\cdot \varrho_D$ ) =
} and
    merge- $\gamma$ : is-grounding-merge  $\gamma$ 
    (vars-cls (add-mset L C  $\cdot \varrho_C$ )) (rename-subst-domain  $\varrho_C \gamma_C$ )
    (vars-cls (add-mset K D  $\cdot \varrho_D$ )) (rename-subst-domain  $\varrho_D \gamma_D$ )

```

by *simp-all*
hence $\forall x. \text{is-Var } (\varrho_C x)$ **and** $\text{inj } \varrho_C$ **and** $\forall x. \text{is-Var } (\varrho_D x)$ **and** $\text{inj } \varrho_D$
 by (*simp-all add: is-renaming-iff*)

from invar have
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**
min-ground-clo- Γ : $\forall Ln \in \text{set } \Gamma. \forall C L \gamma. \text{snd } Ln = \text{Some } (C, L, \gamma) \longrightarrow$
is-ground-cls (*add-mset* $L C \cdot \gamma$)
unfolding *resolveI*(1,2) $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$
 by (*simp-all add: propagate-lit-def ground-closures-def*)
hence
 $\forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$
 $\forall K \in \# \text{add-mset } K D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$
using *merge-of-renamed-groundings*[*OF ren- ϱ_C ren- ϱ_D disjoint-vars - - merge- γ*]
 by *simp-all*

have *atm-of* $L \cdot a \varrho_C \cdot a \gamma = \text{atm-of } K \cdot a \varrho_D \cdot a \gamma$
using $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$
 $\langle \forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle$ [*rule-format, of L, simplified*]
 $\langle \forall K \in \# \text{add-mset } K D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$ [*rule-format, of K, simplified*]
 by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
hence *is-unifiers* $\gamma \{ \{ \text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D \} \}$
 by (*simp add: is-unifiers-def is-unifier-alt*)
hence $\mu \odot \gamma = \gamma$
using $\langle \text{is-imgu } \mu \{ \{ \text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D \} \} \rangle$
 by (*auto simp: is-imgu-def*)
hence $C \cdot \varrho_C \cdot \mu \cdot \gamma = C \cdot \gamma_C$ **and** $D \cdot \varrho_D \cdot \mu \cdot \gamma = D \cdot \gamma_D$
using $\langle \forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle \langle \forall K \in \# \text{add-mset } K D. K$
 $\cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$
 by (*metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst*
subst-lit-comp-subst)
hence $(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma = C \cdot \gamma_C + D \cdot \gamma_D$
 by (*metis subst-cls-comp-subst subst-cls-union*)
thus *?thesis*
using *ground-conf ground-prop min-ground-clo- Γ*
unfolding *resolveI*
 by (*simp add: ground-closures-def*)

qed

lemma *backtrack-preserves-ground-closures*:
assumes *step*: *backtrack* $N \beta S S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
using *assms*
by (*cases* $N \beta S S'$ *rule: backtrack.cases*)
 (*simp add: ground-closures-def decide-lit-def ball-Un*)

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

shows *ground-closures* S'
using *assms* **unfolding** *scl-def*
using *propagate-preserves-ground-closures* *decide-preserves-ground-closures*
conflict-preserves-ground-closures *skip-preserves-ground-closures*
factorize-preserves-ground-closures *resolve-preserves-ground-closures*
backtrack-preserves-ground-closures
by *metis*

6.9 Trail And Conflict Closures Are Ground And False

definition *ground-false-closures* **where**
ground-false-closures $S \longleftrightarrow \text{ground-closures } S \wedge$
trail-closures-false (*state-trail* S) \wedge
 $(\forall C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-clc } (\text{state-trail } S) (C \cdot \gamma))$

lemma *ground-false-closures-initial-state[simp]*: *ground-false-closures initial-state*
by (*simp* *add: ground-false-closures-def*)

lemma *propagate-preserves-ground-false-closures*:
assumes *step: propagate* $N \beta S S'$ **and** *invar: ground-false-closures* S
shows *ground-false-closures* S'
using *step*
proof (*cases* $N \beta S S'$ *rule: propagate.cases*)
case step-hyps: (propagateI C U L C' γ C₀ C₁ Γ μ)

have *ground-closures* S'
using *invar* *propagate-preserves-ground-closures[OF step]*
by (*metis* *ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail* S')
using *invar* *propagate-preserves-trail-closures-false'[OF step]*
by (*metis* *ground-false-closures-def* *trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-clc } (\text{state-trail } S') (C \cdot \gamma)$
unfolding *step-hyps(1,2)* **by** *simp*

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *decide-preserves-ground-false-closures*:
assumes *step: decide* $N \beta S S'$ **and** *invar: ground-false-closures* S
shows *ground-false-closures* S'
using *step*
proof (*cases* $N \beta S S'$ *rule: decide.cases*)
case step-hyps: (decideI L γ Γ U)

have *ground-closures* S'
using *invar* *decide-preserves-ground-closures*[*OF step*]
by (*metis* *ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail* S')
using *invar* *decide-preserves-trail-closures-false'*[*OF step*]
by (*metis* *ground-false-closures-def* *trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cl } (\text{state-trail } S') (C \cdot \gamma)$
unfolding *step-hyps*(1,2) **by** *simp*

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *conflict-preserves-ground-false-closures*:
assumes *step*: *conflict* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: *conflict.cases*)
case *step-hyps*: (*conflictI* $D U \gamma \Gamma$)

have *ground-closures* S'
using *invar* *conflict-preserves-ground-closures*[*OF step*]
by (*metis* *ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail* S')
using *invar* *conflict-preserves-trail-closures-false'*[*OF step*]
by (*metis* *ground-false-closures-def* *trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cl } (\text{state-trail } S') (C \cdot \gamma)$
unfolding *step-hyps*(1,2)
using *step-hyps*(3-) **by** *simp*

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *skip-preserves-ground-false-closures*:
assumes *step*: *skip* $N \beta S S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
using *step*
proof (*cases* $N \beta S S'$ *rule*: *skip.cases*)
case *step-hyps*: (*skipI* $L D \sigma n \Gamma U$)

have *ground-closures* S'
using *invar* *skip-preserves-ground-closures*[*OF step*]

by (*metis ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail S'*)
using *invar skip-preserves-trail-closures-false'*[*OF step*]
by (*metis ground-false-closures-def trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
using *invar*
unfolding *step-hyps(1,2)*
using $\langle - L \notin \# D \cdot \sigma \rangle$
by (*auto simp add: ground-false-closures-def elim!: subtrail-falseI*)

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *factorize-preserves-ground-false-closures*:
assumes *step: factorize N β S S'* **and** *invar: ground-false-closures S*
shows *ground-false-closures S'*
using *step*
proof (*cases N β S S' rule: factorize.cases*)
case *step-hyps: (factorizeI L γ L' μ Γ U D)*

have *ground-closures S'*
using *invar factorize-preserves-ground-closures[OF step]*
by (*metis ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail S'*)
using *invar factorize-preserves-trail-closures-false'*[*OF step*]
by (*metis ground-false-closures-def trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
using *invar conflict-set-after-factorization[OF step]*
unfolding *step-hyps(1,2) ground-false-closures-def*
by (*auto simp del: set-mset-add-mset-insert dest: trail-false-cls-ignores-duplicates*)

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *resolve-preserves-ground-false-closures*:
assumes *step: resolve N β S S'* **and** *invar: ground-false-closures S*
shows *ground-false-closures S'*
using *step*
proof (*cases N β S S' rule: resolve.cases*)
case *step-hyps: (resolveI Γ Γ' K D γ_D L γ_C ϱ_C ϱ_D C μ γ U)*
hence $\Gamma\text{-def: } \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D$

by *simp*

have *ground-closures* S
 using *invar*
 by (*metis ground-false-closures-def*)
 hence *ground-closures* S'
 using *resolve-preserves-ground-closures*[*OF step*]
 by *metis*

moreover have *trail-closures-false* (*state-trail* S')
 using *invar resolve-preserves-trail-closures-false'*[*OF step*]
 by (*metis ground-false-closures-def trail-closures-false'-def*)

moreover have $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
 proof –
 from $\langle \text{ground-closures } S \rangle$ have
ground-conf: *is-ground-cls* (*add-mset* $L \ C \cdot \gamma_C$) and
ground-prop: *is-ground-cls* (*add-mset* $K \ D \cdot \gamma_D$)
 unfolding *step-hyps*(1,2)
 by (*simp-all add: ground-closures-def Γ -def propagate-lit-def*)
 hence
 $\forall L \in \# \text{add-mset } L \ C. L \cdot l \ \varrho_C \cdot l \ \gamma = L \cdot l \ \gamma_C$
 $\forall K \in \# \text{add-mset } K \ D. K \cdot l \ \varrho_D \cdot l \ \gamma = K \cdot l \ \gamma_D$
 using *merge-of-renamed-groundings step-hyps*(3–)
 by *metis+*

have *atm-of* $L \cdot a \ \varrho_C \cdot a \ \gamma = \text{atm-of } K \cdot a \ \varrho_D \cdot a \ \gamma$
 using $\langle K \cdot l \ \gamma_D = - (L \cdot l \ \gamma_C) \rangle$
 $\langle \forall L \in \# \text{add-mset } L \ C. L \cdot l \ \varrho_C \cdot l \ \gamma = L \cdot l \ \gamma_C \rangle$ [*rule-format, of L, simplified*]
 $\langle \forall K \in \# \text{add-mset } K \ D. K \cdot l \ \varrho_D \cdot l \ \gamma = K \cdot l \ \gamma_D \rangle$ [*rule-format, of K, simplified*]
 by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
 hence *is-unifiers* $\gamma \ \{\{\text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D\}\}$
 by (*simp add: is-unifiers-def is-unifier-alt*)
 hence $\mu \odot \gamma = \gamma$
 using $\langle \text{is-imgu } \mu \ \{\{\text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D\}\} \rangle$
 by (*auto simp: is-imgu-def*)
 hence $C \cdot \varrho_C \cdot \mu \cdot \gamma = C \cdot \gamma_C$ and $D \cdot \varrho_D \cdot \mu \cdot \gamma = D \cdot \gamma_D$
 using $\langle \forall L \in \# \text{add-mset } L \ C. L \cdot l \ \varrho_C \cdot l \ \gamma = L \cdot l \ \gamma_C \rangle \langle \forall K \in \# \text{add-mset } K \ D. K \cdot l \ \varrho_D \cdot l \ \gamma = K \cdot l \ \gamma_D \rangle$
 by (*metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst subst-lit-comp-subst*)+

moreover have *trail-false-cls* $\Gamma (C \cdot \gamma_C)$
 using *invar*
 unfolding *step-hyps*(1,2)
 by (*simp add: ground-false-closures-def trail-false-cls-add-mset*)

moreover have *trail-false-cls* $\Gamma (D \cdot \gamma_D)$

```

proof (rule trail-false-cls-if-trail-false-suffix)
  show suffix  $\Gamma' \Gamma$ 
    unfolding  $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$ 
    by (simp add: suffix-def)
next
  show trail-false-cls  $\Gamma' (D \cdot \gamma_D)$ 
    using invar
    unfolding step-hyps(1,2)  $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$ 
    by (auto simp: ground-false-closures-def propagate-lit-def elim: trail-closures-false.cases)
qed

ultimately show ?thesis
  unfolding step-hyps(1,2)
  by (simp add: trail-false-cls-plus)
qed

ultimately show ?thesis
  unfolding ground-false-closures-def by metis
qed

lemma backtrack-preserves-ground-false-closures:
  assumes step: backtrack  $N \beta S S'$  and invar: ground-false-closures  $S$ 
  shows ground-false-closures  $S'$ 
  using step
proof (cases  $N \beta S S'$  rule: backtrack.cases)
  case step-hyps: (backtrackI  $\Gamma \Gamma' \Gamma'' K L \sigma D U$ )

  have ground-closures  $S'$ 
    using invar backtrack-preserves-ground-closures[OF step]
    by (metis ground-false-closures-def)

  moreover have trail-closures-false (state-trail  $S'$ )
    using invar backtrack-preserves-trail-closures-false'[OF step]
    by (metis ground-false-closures-def trail-closures-false'-def)

  moreover have  $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$ 
    unfolding step-hyps(1,2) by simp

  ultimately show ?thesis
    unfolding ground-false-closures-def by metis
qed

lemma scl-preserves-ground-false-closures:
  assumes scl  $N \beta S S'$  and ground-false-closures  $S$ 
  shows ground-false-closures  $S'$ 
  using assms unfolding scl-def
  using propagate-preserves-ground-false-closures decide-preserves-ground-false-closures
  conflict-preserves-ground-false-closures skip-preserves-ground-false-closures

```

factorize-preserves-ground-false-closures *resolve-preserves-ground-false-closures*
backtrack-preserves-ground-false-closures
 by *metis*

6.10 Learned Clauses Are Non-empty

definition *learned-nonempty* where

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

lemma *learned-nonempty-initial-state[simp]*: *learned-nonempty initial-state*
 by (*simp add: learned-nonempty-def*)

lemma *propagate-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: propagate.cases*) (*simp add: learned-nonempty-def*)

lemma *decide-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: decide.cases*) (*simp add: learned-nonempty-def*)

lemma *conflict-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: conflict.cases*) (*simp add: learned-nonempty-def*)

lemma *skip-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: skip.cases*) (*simp add: learned-nonempty-def*)

lemma *factorize-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: factorize.cases*) (*simp add: learned-nonempty-def*)

lemma *resolve-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: resolve.cases*) (*simp add: learned-nonempty-def*)

lemma *backtrack-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'

using *assms* **by** (*cases rule: backtrack.cases*) (*simp add: learned-nonempty-def*)

lemma *scl-preserves-learned-nonempty*:

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

shows *learned-nonempty* S'
using *assms* **unfolding** *scl-def*
using *propagate-preserves-learned-nonempty* *decide-preserves-learned-nonempty*
conflict-preserves-learned-nonempty *skip-preserves-learned-nonempty*
factorize-preserves-learned-nonempty *resolve-preserves-learned-nonempty*
backtrack-preserves-learned-nonempty
by *metis*

6.11 Backtrack Follows Conflict Resolution

definition *conflict-resolution* **where**

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*
by (*simp* *add*: *conflict-resolution-def*)

lemma *propagate-preserves-conflict-resolution*:

assumes *step*: *propagate* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
using *step* **by** (*auto* *simp*: *conflict-resolution-def* *elim*: *propagate.cases*)

lemma *decide-preserves-conflict-resolution*:

assumes *step*: *decide* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
using *step* **by** (*auto* *simp*: *conflict-resolution-def* *elim*: *decide.cases*)

lemma *conflict-preserves-conflict-resolution*:

assumes *step*: *conflict* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
using *step* **unfolding** *conflict-resolution-def* **by** *blast*

lemma *skip-preserves-conflict-resolution*:

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

proof –

from *step* **have** *state-conflict* $S \neq \text{None}$
by (*auto* *elim*: *skip.cases*)

with *invar* **obtain** $S0 S1$ **where**

conflict $N \beta S0 S1$ **and** $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S1 S$
by (*auto* *simp*: *conflict-resolution-def*)

show *?thesis*

unfolding *conflict-resolution-def*

proof (*intro* *impI* *exI* *conjI*)

show *conflict* $N \beta S0 S1$
by (*rule* $\langle \text{conflict } N \beta S0 S1 \rangle$)

next

```

show (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S'
  using  $\langle$ (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S $\rangle$  step
  by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl sup2CI)
qed
qed

```

```

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'
  using step
proof –
  from step have state-conflict S  $\neq$  None
    by (auto elim: factorize.cases)
  with invar obtain S0 S1 where
    conflict N  $\beta$  S0 S1 and (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S
    by (auto simp: conflict-resolution-def)
  show ?thesis
    unfolding conflict-resolution-def
  proof (intro impI exI conjI)
    show conflict N  $\beta$  S0 S1
      by (rule  $\langle$ conflict N  $\beta$  S0 S1 $\rangle$ )
  next
    show (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S'
      using  $\langle$ (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S $\rangle$  step
      by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl sup2CI)
  qed
qed

```

```

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'
  using step
proof –
  from step have state-conflict S  $\neq$  None
    by (auto elim: resolve.cases)
  with invar obtain S0 S1 where
    conflict N  $\beta$  S0 S1 and (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S
    by (auto simp: conflict-resolution-def)
  show ?thesis
    unfolding conflict-resolution-def
  proof (intro impI exI conjI)
    show conflict N  $\beta$  S0 S1
      by (rule  $\langle$ conflict N  $\beta$  S0 S1 $\rangle$ )
  next
    show (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S'
      using  $\langle$ (skip N  $\beta$   $\sqcup$  factorize N  $\beta$   $\sqcup$  resolve N  $\beta$ )** S1 S $\rangle$  step
      by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl sup2CI)
  qed
qed

```

lemma *backtrack-preserves-conflict-resolution*:
assumes *step*: *backtrack* $N \beta S S'$
shows *conflict-resolution* $N \beta S'$
using *step* **by** (*auto simp: conflict-resolution-def elim: backtrack.cases*)

lemma *scl-preserves-conflict-resolution*:
assumes *scl* $N \beta S S'$ **and** *conflict-resolution* $N \beta S$
shows *conflict-resolution* $N \beta S'$
using *assms* **unfolding** *scl-def*
using *propagate-preserves-conflict-resolution* *decide-preserves-conflict-resolution*
conflict-preserves-conflict-resolution *skip-preserves-conflict-resolution*
factorize-preserves-conflict-resolution *resolve-preserves-conflict-resolution*
backtrack-preserves-conflict-resolution
by *metis*

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 $\{\#\} \mid \in \mid N \vee (\exists S0. \textit{propagate } N \beta S0 S1) \vee (\exists S0. \textit{decide } N \beta S0 S1)$
using *assms*
proof (*cases* $N \beta S1 S2$ *rule: conflict.cases*)
case (*conflictI* $D U \gamma \Gamma$)
with *invars*(2) **have** *trail-propagated-or-decided* $N \beta U \Gamma$
by (*simp add: trail-propagated-or-decided'-def*)
thus *?thesis*
proof (*cases* $N \beta U \Gamma$ *rule: trail-propagated-or-decided.cases*)
case *Nil*
hence $D \cdot \gamma = \{\#\}$
using $\langle \textit{trail-false-cls } \Gamma (D \cdot \gamma) \rangle$ *not-trail-false-Nil*(2) **by** *blast*
hence $D = \{\#\}$
by (*simp add: local.conflictI*(4))
moreover from *invars*(1) **have** $\{\#\} \mid \notin \mid U$
by (*simp add: conflictI*(1) *learned-nonempty-def*)
ultimately have $\{\#\} \mid \in \mid N$
using $\langle D \mid \in \mid N \mid \cup \mid U \rangle$ **by** *simp*
thus *?thesis* **by** *simp*
next
case (*Propagate* $C L C' \gamma_C C_0 C_1 \Gamma' \mu$)
hence $\exists S0. \textit{propagate } N \beta S0 S1$
unfolding *conflictI*(1)
using *propagateI* **by** *blast*
thus *?thesis* **by** *simp*
next
case (*Decide* $L \gamma_L \Gamma'$)
hence $\exists S0. \textit{decide } N \beta S0 S1$
unfolding *conflictI*(1)

using *decideI* **by** *blast*
thus *?thesis* **by** *simp*
qed
qed

lemma *before-backtrack*:

assumes *backt*: *backtrack* $N \beta S_n S_m$ **and**
invar: *conflict-resolution* $N \beta S_n$
shows $\exists S_0 S_1. \text{conflict } N \beta S_0 S_1 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S_1 S_n$
using *backt*
proof (*cases* $N \beta S_n S_m$ *rule*: *backtrack.cases*)
case (*backtrackI* $\Gamma \Gamma' \Gamma'' L \sigma D U$)
thus *?thesis*
using *invar* **by** (*simp add*: *conflict-resolution-def*)
qed

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

trail-false-clc (*state-trail* S) $C \implies \text{trail-atoms-lt } \beta S \implies \forall L \in \# C. \text{atm-of } L \preceq_B \beta$
unfolding *trail-atoms-lt-def*
by (*meson atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set trail-false-clc-def trail-false-lit-def*)

7 Soundness

7.1 Sound Trail

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

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

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* $N []$

by (*simp add*: *sound-trail-def*)

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

by (*meson grounding-of-clss-mono true-clss-mono*)

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

unfolding *sound-trail-def*

by (*meson entails-G-mono less-eq-fset.rep-eq*)

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

by (*simp add*: *sound-trail-def*)

lemma *sound-trail-appendD*: *sound-trail* N $(\Gamma @ \Gamma')$ \implies *sound-trail* N Γ'
by (*induction* Γ) (*auto intro: sound-trail-ConsD*)

lemma *sound-trail-propagate*:

assumes
sound- Γ : *sound-trail* N Γ **and**
N-entails-C-L: *fset* $N \models_{\mathcal{G}e} \{C + \{\#L\#\}\}$
shows *sound-trail* N (*trail-propagate* Γ L C σ)
using *assms*
by (*simp add: sound-trail-def propagate-lit-def*)

lemma *sound-trail-decide*:

sound-trail N $\Gamma \implies$ *sound-trail* N (*trail-decide* Γ L)
by (*simp add: sound-trail-def decide-lit-def*)

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

7.3 Initial State Is Sound

lemma *sound-initial-state*[*simp*]: *sound-state* N β *initial-state*
by (*simp add: sound-state-def trail-atoms-lt-def*)

7.4 SCL Rules Preserve Soundness

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

unfolding *vars-subst-cls-eq*
using *subst-domain-def* **by** *force*

lemma *propagate-preserves-sound-state*:

assumes *step*: *propagate* N β S S' **and** *sound*: *sound-state* N β S
shows *sound-state* N β S'
using *assms*(1)

proof (*cases* N β S S' *rule: propagate.cases*)

case (*propagateI* C U L C' γ C_0 C_1 Γ μ)

hence

S-def: $S = (\Gamma, U, \text{None})$ **and**

S'-def: $S' = (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, \text{None})$ **and**

C-in: $C \mid \in \mid N \mid \cup \mid U$ **and**

C-def: $C = \text{add-mset } L$ C' **and**

gr-C- γ : *is-ground-cls* $(C \cdot \gamma)$ **and**

C₀-def: $C_0 = \{\#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma \#\}$ **and**

C₁-def: $C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma \#\}$ **and**

Γ -false- C_0 - γ : *trail-false-cls* Γ ($C_0 \cdot \gamma$) **and**
 undef- Γ - L - γ : \neg *trail-defined-lit* Γ ($L \cdot l \gamma$) **and**
imgu- μ : *is-imgu* μ {*atm-of* ‘ *set-mset* (*add-mset* L C_1) }
 by *simp-all*

from sound have

sound- Γ : *sound-trail* N Γ **and**
N-entails-U: *fset* $N \models_{\mathcal{G}e}$ *fset* U
unfolding *sound-state-def S-def* **by** *auto*

have *vars- C_0* : *vars-cls* $C_0 \subseteq$ *vars-cls* C'
apply (*simp add: C₀-def*)
by (*metis multiset-partition order-refl sup.coboundedI1 vars-cls-plus-iff*)

have *vars- C_1* : *vars-cls* $C_1 \subseteq$ *vars-cls* C'
apply (*simp add: C₁-def*)
by (*metis multiset-partition order-refl sup.coboundedI1 vars-cls-plus-iff*)

have *fin-atm- C_1* : *finite* (*atm-of* ‘ (*set-mset* C_1))
 by *blast*
hence *is-unifier* γ (*atm-of* ‘ (*set-mset* (*add-mset* L C_1)))
 by (*auto simp add: is-unifier-alt C₁-def intro: subst-atm-of-eqI*)
hence *μ - γ -simp*: $\mu \odot \gamma = \gamma$
using *imgu- μ* [*unfolded is-imgu-def, THEN conjunct2*]
using *is-unifiers-def* **by** *fastforce*

have $L \cdot l \mu \cdot l \gamma = L \cdot l \gamma$
using *μ - γ -simp* **by** (*metis subst-lit-comp-subst*)

have $C_0 \cdot \mu \cdot \gamma = C_0 \cdot \gamma$
using *μ - γ -simp* **by** (*metis subst-cls-comp-subst*)

have *sound-trail* N (*trail-propagate* Γ ($L \cdot l \mu$) ($C_0 \cdot \mu$) γ)
proof (*rule sound-trail-propagate*)
show *sound-trail* N Γ
by (*rule sound- Γ*)

next

have *fset* $N \models_{\mathcal{G}e}$ { C }
using *C-in*[*unfolded funion-iff*] *N-entails-U*
by (*metis* (*no-types, opaque-lifting*) *empty-subsetI*[*of fset -*]
grounding-of-clss-mono[*of {C} fset -*] *insert-subset*[*of C { } fset -*]
true-clss-mono[*of grounding-of-clss {C} grounding-of-clss (fset -)*])
hence *fset* $N \models_{\mathcal{G}e}$ {*add-mset* L ($C_0 + C_1$)}
by (*simp add: C-def C₀-def C₁-def*)
hence *fset* $N \models_{\mathcal{G}e}$ {*add-mset* L ($C_0 + C_1$) $\cdot \mu$ }
by (*metis* (*no-types, opaque-lifting*) *grounding-of-clss-singleton*
subst-cls-eq-grounding-of-cls-subset-eq true-clss-mono)
hence *fset* $N \models_{\mathcal{G}e}$ {*add-mset* L $C_0 \cdot \mu$ }
proof (*rule entails-trans*)

have $\exists C \in \text{grounding-of-clss } \{(C_0 + C_1 + \{\#L\#\}) \cdot \mu\}$. *set-mset* $D = \text{set-mset}$
 C
if D -in: $D \in \text{grounding-of-clss } \{(C_0 + \{\#L\#\}) \cdot \mu\}$ **for** $I D$
proof–
from D -in **obtain** σ **where**
 D -def: $D = \text{add-mset } L C_0 \cdot \mu \cdot \sigma$ **and** gr - σ : *is-ground-subst* σ
by (*auto simp: grounding-of-clss-def*)
show ?thesis
proof (*rule bexI*)
from *ingu*- μ **have** *is-unifier* μ (*atm-of* ‘ *set-mset* (*add-mset* $L C_1$))
by (*simp add: is-ingu-def is-unifiers-def*)
hence $\forall A \in \text{atm-of } \text{‘ set-mset } (\text{add-mset } L C_1)$. $\forall B \in \text{atm-of } \text{‘ set-mset}$
(*add-mset* $L C_1$).
 $A \cdot a \mu = B \cdot a \mu$
using *is-unifier-alt*
by (*metis* (*mono-tags*, *opaque-lifting*) *finite-set-mset*[*of add-mset* $L C_1$]
finite-imageI[*of set-mset* (*add-mset* $L C_1$) *atm-of*])
hence $\forall A \in \text{atm-of } \text{‘ set-mset } C_1$. $A \cdot a \mu = \text{atm-of } L \cdot a \mu$
by (*metis image-insert insert-iff set-mset-add-mset-insert*)
hence $\forall A \in \text{set-mset } C_1$. $A \cdot l \mu = L \cdot l \mu$
unfolding C_1 -def
by (*metis* (*mono-tags*, *lifting*) *subst-lit-is-neg*[*of* - γ] *subst-lit-is-neg*[*of* - μ]
set-mset-filter[*of* $\lambda x. x \cdot l \gamma = L \cdot l \gamma C'$] *atm-of-subst-lit*[*of* - μ]
mem-Collect-eq
image-eqI[*of atm-of* - *atm-of* - *set-mset* $\{\#x \in \# C'. x \cdot l \gamma = L \cdot l \gamma \#\}$]
literal.expand[*of* $L \cdot l \mu - \cdot l \mu$])
hence *set-mset* ((*add-mset* $L C_1$) $\cdot \mu$) = $\{L \cdot l \mu\}$
by *auto*
hence *set-mset* (($C_0 + C_1 + \{\#L\#\}) \cdot \mu$) = *set-mset* (($C_0 + \{\#L\#\}) \cdot$
 μ)
by *auto*
thus *set-mset* $D = \text{set-mset } ((C_0 + C_1 + \{\#L\#\}) \cdot \mu \cdot \sigma)$
unfolding D -def *set-mset-subst-cls-conv*[*of* - σ]
by *simp*
next
show $(C_0 + C_1 + \{\#L\#\}) \cdot \mu \cdot \sigma \in \text{grounding-of-clss } \{(C_0 + C_1 +$
 $\{\#L\#\}) \cdot \mu\}$
using gr - σ **by** (*auto simp: grounding-of-clss-def*)
qed
qed
then show $\{\text{add-mset } L (C_0 + C_1) \cdot \mu\} \models_{\mathcal{G}e} \{\text{add-mset } L C_0 \cdot \mu\}$
by (*auto elim: true-clss-if-set-mset-eq*[*rotated*])
qed
thus *fset* $N \models_{\mathcal{G}e} \{C_0 \cdot \mu + \{\#L \cdot l \mu \#\}$
by (*metis* (*no-types*, *opaque-lifting*) *add-mset-add-single*[*of* $L \cdot l \mu C_0 \cdot \mu$]
grounding-of-clss-singleton[*of add-mset* $L C_0 \cdot \mu$] *subst-cls-add-mset*[*of* L
 $C_0 \mu$])
qed
thus ?thesis

unfolding *S'*-def sound-state-def
using *N*-entails-*U* **by** simp
qed

lemma *decide-preserves-sound-state*:
assumes *step*: *decide* *N* β *S* *S'* **and** *sound*: *sound-state* *N* β *S*
shows *sound-state* *N* β *S'*
using *assms*(1)
proof (*cases* *N* β *S* *S'* *rule*: *decide.cases*)
case (*decideI* *L* γ Γ *U*)
from *decideI*(1) *sound* **have**
sound- Γ : *sound-trail* *N* Γ **and**
N-entails-*U*: *fset* *N* $\models_{\mathcal{G}e}$ *fset* *U*
unfolding *sound-state-def* **by** auto

moreover **have** *sound-trail* *N* (*trail-decide* Γ (*L* \cdot *l* γ))
by (*simp* *add*: *local.decideI*(4) *local.decideI*(5) *sound*- Γ *sound-trail-decide*)

ultimately **show** ?*thesis*
unfolding *decideI* *sound-state-def* **by** simp
qed

lemma *conflict-preserves-sound-state*:
assumes *step*: *conflict* *N* β *S* *S'* **and** *sound*: *sound-state* *N* β *S*
shows *sound-state* *N* β *S'*
using *assms*(1)
proof (*cases* *N* β *S* *S'* *rule*: *conflict.cases*)
case (*conflictI* *D* *U* γ Γ)
hence *D*-in: *D* $|\in|$ *N* $|\cup|$ *U* **by** simp

from *conflictI*(1) *sound* **have**
sound- Γ : *sound-trail* *N* Γ **and**
N-entails-*U*: *fset* *N* $\models_{\mathcal{G}e}$ *fset* *U*
unfolding *sound-state-def* **by** auto

have *N*-entails-*D'*: *fset* *N* $\models_{\mathcal{G}e}$ {*D*}
proof (*intro* *allI* *impI*)
fix *I*
assume *valid-N*: *I* \models_s *grounding-of-clss* (*fset* *N*)
then **show** *I* \models_s *grounding-of-clss* {*D*}
using *D*-in[*unfolded funion-iff*]
unfolding *grounding-of-clss-singleton*
by (*metis* (*mono-tags*, *opaque-lifting*) *N*-entails-*U* *UN-I*[*of* *D* *fset* - - *grounding-of-clss*]
grounding-of-clss-def[*of* *fset* -] *true-clss-def*[*of* *I*])

qed
thus ?*thesis*
unfolding *conflictI*(1,2) *sound-state-def*
using *sound*- Γ *N*-entails-*U* **by** simp

qed

lemma *skip-preserves-sound-state*:

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

shows $sound\text{-}state\ N\ \beta\ S'$

using *step*

proof (*cases* $N\ \beta\ S\ S'$ *rule*: *skip.cases*)

case (*skipI* $L\ D\ \sigma\ Cl\ \Gamma\ U$)

thus *?thesis*

using *sound*

by (*auto simp*: *sound-state-def trail-atoms-lt-def*

intro: *sound-trail-ConsD elim!*: *subtrail-falseI*)

qed

lemma *factorize-preserves-sound-state*:

assumes *step*: $factorize\ N\ \beta\ S\ S'$ **and** *sound*: $sound\text{-}state\ N\ \beta\ S$

shows $sound\text{-}state\ N\ \beta\ S'$

using *assms(1)*

proof (*cases* $N\ \beta\ S\ S'$ *rule*: *factorize.cases*)

case (*factorizeI* $L\ \gamma\ L'\ \mu\ \Gamma\ U\ D$)

from *factorizeI(1)* **sound** **have**

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

N-entails-U: $fset\ N\ \models_{\mathcal{G}e}\ fset\ U$ **and**

N-entails-D-L-L': $fset\ N\ \models_{\mathcal{G}e}\ \{D + \{\#L, L'\#\}\}$

unfolding *sound-state-def* **by** (*simp-all add*: *add-mset-commute*)

from *factorizeI* **have**

imgu- μ : *is-imgu* $\mu\ \{\{atm\text{-}of\ L, atm\text{-}of\ L'\}\}$

by *simp*

from *factorizeI* **have** *L-eq-L'- γ* : $L \cdot l\ \gamma = L' \cdot l\ \gamma$ **by** *simp*

from *L-eq-L'- γ* **have** *unif- γ* : *is-unifier* $\gamma\ \{atm\text{-}of\ L, atm\text{-}of\ L'\}$

by (*auto simp*: *is-unifier-alt intro*: *subst-atm-of-eqI*)

hence *unifs- γ* : *is-unifiers* $\gamma\ \{\{atm\text{-}of\ L, atm\text{-}of\ L'\}\}$

by (*simp add*: *is-unifiers-def*)

from *imgu- μ* **have** *is-unifier* $\mu\ \{atm\text{-}of\ L, atm\text{-}of\ L'\}$

by (*auto simp add*: *is-unifiers-def dest*: *is-imgu-is-mgu*[*THEN is-mgu-is-unifiers*])

hence *L-eq-L'- μ* : $L \cdot l\ \mu = L' \cdot l\ \mu$

apply (*simp add*: *is-unifier-alt*)

by (*metis L-eq-L'- γ atm-of-subst-lit literal.expand subst-lit-is-neg*)

have $fset\ N\ \models_{\mathcal{G}e}\ \{(D + \{\#L\#\}) \cdot \mu\}$

proof (*rule entails-trans*)

show $fset\ N\ \models_{\mathcal{G}e}\ \{D + \{\#L, L'\#\}\}$

by (*rule N-entails-D-L-L'*)

next

have $*$: $\{(D + \{\#L, L'\#\}) \cdot \mu\} = \{D \cdot \mu + \{\#L \cdot l\ \mu, L \cdot l\ \mu\#\}$

by (*simp add: L-eq-L'-μ*)

have $\{D + \{\#L, L'\#\}\} \Vdash_{\mathcal{G}e} \{(D + \{\#L, L'\#\}) \cdot \mu\}$
 using *subst-cls-eq-grounding-of-cls-subset-eq true-clss-mono*
 by (*metis (mono-tags, lifting) grounding-of-clss-singleton*)

moreover have $\{(D + \{\#L, L'\#\}) \cdot \mu\} \Vdash_{\mathcal{G}e} \{(D + \{\#L\#\}) \cdot \mu\}$
apply (*intro allI impI*)
by (*erule true-clss-if-set-mset-eq[rotated]*) (*auto simp add: L-eq-L'-μ grounding-of-cls-def*)

ultimately show $\{D + \{\#L, L'\#\}\} \Vdash_{\mathcal{G}e} \{(D + \{\#L\#\}) \cdot \mu\}$
 by *simp*

qed

thus *?thesis*
unfolding *factorizeI sound-state-def*
using *sound-Γ N-entails-U* **by** *simp*

qed

lemma *resolve-preserves-sound-state:*
assumes *step: resolve N β S S' and sound: sound-state N β S*
shows *sound-state N β S'*
using *assms(1)*

proof (*cases N β S S' rule: resolve.cases*)
case (*resolveI Γ Γ' K D γ_D L γ_C ρ_C ρ_D C μ γ U*)
hence
 Γ-def: $\Gamma = \text{trail-propagate } \Gamma' K D \gamma_D$ **and**
 imgu-μ: $\text{is-imgu } \mu \{\{\text{atm-of } L \cdot a \ \rho_C, \text{atm-of } K \cdot a \ \rho_D\}\}$
by *simp-all*

from *sound* **have**
sound-Γ: sound-trail N Γ and
N-entails-U: fset N $\Vdash_{\mathcal{G}e}$ fset U and
N-entails-conf: fset N $\Vdash_{\mathcal{G}e}$ {add-mset L C}
unfolding *resolveI(1,2) sound-state-def* **by** *simp-all*

from *sound-Γ* **have**
N-entails-prop: fset N $\Vdash_{\mathcal{G}e}$ {add-mset K D}
unfolding *sound-trail-def Γ-def*
by (*simp add: propagate-lit-def*)

moreover have $\text{fset } N \Vdash_{\mathcal{G}e} \{(C \cdot \rho_C + D \cdot \rho_D) \cdot \mu\}$

proof –
have $*$: $\text{fset } N \Vdash_{\mathcal{G}e} \{\text{add-mset } L C \cdot \rho_C \cdot \mu\}$
using *N-entails-conf*
by (*metis (no-types, opaque-lifting) grounding-of-clss-singleton grounding-of-subst-cls-subset true-clss-mono*)

have $**$: $\text{fset } N \Vdash_{\mathcal{G}e} \{\text{add-mset } K D \cdot \rho_D \cdot \mu\}$

```

using N-entails-prop
by (metis (no-types, opaque-lifting) grounding-of-clss-singleton grounding-of-subst-clss-subset
      true-clss-mono)

show fset N  $\models_{\mathcal{G}e}$   $\{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$ 
  unfolding true-clss-def
proof (intro allI impI ballI)
  fix I E
  assume I-entails:  $\forall E \in \text{grounding-of-clss } (fset N). I \models E$  and
    E-in:  $E \in \text{grounding-of-clss } \{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$ 
  then obtain  $\gamma_E$  where
    E-def:  $E = (C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma_E$  and gr- $\gamma_E$ : is-ground-subst  $\gamma_E$ 
    unfolding grounding-of-clss-def grounding-of-clss-def by auto

  have  $I \models_l L \cdot l \varrho_C \cdot l \mu \cdot l \gamma_E \vee (\exists x \in \# C \cdot \varrho_C \cdot \mu \cdot \gamma_E. I \models_l x)$ 
  proof –
    have add-mset  $L C \cdot \varrho_C \cdot \mu \cdot \gamma_E \in \text{grounding-of-clss } \{add-mset L C \cdot \varrho_C \cdot$ 
 $\mu\}$ 
      using gr- $\gamma_E$  unfolding grounding-of-clss-def grounding-of-clss-def by auto
      hence  $\exists K \in \# add-mset L C \cdot \varrho_C \cdot \mu \cdot \gamma_E. I \models_l K$ 
      using *[rule-format, unfolded true-clss-def, OF I-entails]
      by (metis true-clss-def)
      thus ?thesis
      by simp
    qed

  moreover have  $I \models_l K \cdot l \varrho_D \cdot l \mu \cdot l \gamma_E \vee (\exists x \in \# D \cdot \varrho_D \cdot \mu \cdot \gamma_E. I \models_l x)$ 
  proof –
    have add-mset  $K D \cdot \varrho_D \cdot \mu \cdot \gamma_E \in \text{grounding-of-clss } \{add-mset K D \cdot \varrho_D$ 
 $\cdot \mu\}$ 
      using gr- $\gamma_E$  unfolding grounding-of-clss-def grounding-of-clss-def by auto
      hence  $\exists K \in \# add-mset K D \cdot \varrho_D \cdot \mu \cdot \gamma_E. I \models_l K$ 
      using **[rule-format, unfolded true-clss-def, OF I-entails]
      by (metis true-clss-def)
      thus ?thesis
      by simp
    qed

  ultimately show  $I \models E$ 
  proof (elim disjE)
    assume  $I \models_l L \cdot l \varrho_C \cdot l \mu \cdot l \gamma_E$  and  $I \models_l K \cdot l \varrho_D \cdot l \mu \cdot l \gamma_E$ 
    moreover have atm-of  $L \cdot a \varrho_C \cdot a \mu = \text{atm-of } K \cdot a \varrho_D \cdot a \mu$ 
      using imgu- $\mu$ [unfolded is-imgu-def]
      by (meson finite.emptyI finite.insertI insertCI is-unifier-alt is-unifiers-def)
    ultimately have False
      using  $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$ 
      by (cases L; cases K; simp add: uminus-literal-def subst-lit-def)
    thus ?thesis ..
  qed (auto simp: E-def)

```

qed
qed

moreover have *sound-trail* N (*trail-propagate* $\Gamma' K D \gamma_D$)
using Γ -*def* *sound- Γ* **by** *blast*

ultimately show *?thesis*
unfolding *resolveI sound-state-def*
using *N-entails-U* **by** *simp*
qed

lemma *backtrack-preserves-sound-state*:
assumes *step*: *backtrack* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$
shows *sound-state* $N \beta S'$
using *assms(1)*

proof (*cases* $N \beta S S'$ *rule*: *backtrack.cases*)
case (*backtrackI* $\Gamma \Gamma' \Gamma'' K L \sigma D U$)
from *backtrackI(1)* *sound* **have**
sound- Γ : *sound-trail* $N \Gamma$ **and**
N-entails-U: *fset* $N \models_{\mathcal{G}e}$ *fset* U **and**
N-entails-D-L-L': *fset* $N \models_{\mathcal{G}e}$ $\{D + \{\#L\#\}$
unfolding *sound-state-def* **by** *simp-all*

from *backtrackI* **have** Γ -*def*: $\Gamma = \text{trail-decide } (\Gamma' @ \Gamma'') (- (L \cdot l \sigma))$ **by** *simp*

moreover have *sound-trail* $N \Gamma''$
proof –
from *sound- Γ* **have** *sound-trail* $N \Gamma$
by (*rule* *sound-trail-supersetI*) *auto*
then show *?thesis*
by (*auto simp*: Γ -*def* *decide-lit-def intro*: *sound-trail-ConsD sound-trail-appendD*)
qed

moreover have *fset* $N \models_{\mathcal{G}e}$ (*fset* $U \cup \{D + \{\#L\#\}$)
using *N-entails-U N-entails-D-L-L'* **by** (*metis UN-Un grounding-of-clss-def true-clss-union*)

ultimately show *?thesis*
unfolding *backtrackI sound-state-def* **by** *simp*
qed

theorem *scl-preserves-sound-state*:
fixes $N :: ('f, 'v) \text{Term.term clause fset}$
shows *scl* $N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
unfolding *scl-def*
using *propagate-preserves-sound-state decide-preserves-sound-state conflict-preserves-sound-state skip-preserves-sound-state factorize-preserves-sound-state resolve-preserves-sound-state backtrack-preserves-sound-state*
by *metis*

8 Strategies

definition *reasonable-scl* **where**

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

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

unfolding *reasonable-scl-def* *scl-def* **by** *simp*

definition *regular-scl* **where**

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

lemma *reasonable-if-regular*:

regular-scl $N \beta S S' \implies reasonable-scl\ N \beta S S'$

unfolding *regular-scl-def*

proof (*elim disjE conjE*)

assume *conflict* $N \beta S S'$

hence *scl* $N \beta S S'$

by (*simp add: scl-def*)

moreover have *decide* $N \beta S S' \longrightarrow \neg(\exists S''.\ conflict\ N \beta S' S'')$

by (*smt (verit, best) <conflict N beta S S'> conflict.cases option.distinct(1)*)

snd-conv)

ultimately show *reasonable-scl* $N \beta S S'$

by (*simp add: reasonable-scl-def*)

next

assume $\neg(\exists S''.\ conflict\ N \beta S S'')$ **and** *reasonable-scl* $N \beta S S'$

thus *?thesis* **by** *simp*

qed

lemma *scl-if-regular*:

regular-scl $N \beta S S' \implies scl\ N \beta S S'$

using *scl-if-reasonable* *reasonable-if-regular* **by** *simp*

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

lemma *regular-scl* $N \beta S S' \longleftrightarrow reasonable-scl\ N \beta S S' \wedge$

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

(*is ?lhs = ?rhs*)

proof (*rule iffI*)

assume *?lhs*

thus *?rhs*

unfolding *regular-scl-def*

proof (*elim disjE conjE*)

assume *conf*: *conflict* $N \beta S S'$

show *?rhs*

proof (*rule conjI*)

from *conf* **have** $\neg\ decide\ N \beta S S'$

by (*simp add: conflict-well-defined(2)*)

```

    with conf show reasonable-scl  $N \beta S S'$ 
      unfolding reasonable-scl-def scl-def by simp
    next
      show  $(\exists S''. \text{conflict } N \beta S S'') \longrightarrow \text{conflict } N \beta S S'$ 
        using conf by simp
      qed
    next
      assume  $\neg (\exists S''. \text{conflict } N \beta S S'')$  and reasonable-scl  $N \beta S S'$ 
      thus ?rhs
        by simp
      qed
    next
      assume ?rhs
      thus ?lhs
      proof (cases  $\exists S''. \text{conflict } N \beta S S''$ )
        case True
          with <?rhs> have conflict  $N \beta S S'$ 
            by blast
          then show ?thesis
            unfolding regular-scl-def
            by simp
        next
          case False
            then show ?thesis
              unfolding regular-scl-def
              using <?rhs> by simp
      qed
    qed
  qed

```

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**

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

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

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

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

$$\begin{aligned} \text{shortest-backtrack-strategy } \text{regular-scl } N \beta S S' &\implies \text{regular-scl } N \beta S S' \\ \text{by } (\text{simp add: } \text{shortest-backtrack-strategy-def}) \end{aligned}$$

lemma *strategy-restrictions*:

shows

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

$regular\text{-}scl\ N\ \beta\ S\ S' \implies reasonable\text{-}scl\ N\ \beta\ S\ S'$ **and**
 $reasonable\text{-}scl\ N\ \beta\ S\ S' \implies scl\ N\ \beta\ S\ S'$
using *regular-scl-if-shortest-backtrack-strategy reasonable-if-regular scl-if-reasonable*
by *metis+*

primrec *shortest-backtrack* **where**

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

lemma *suffix-shortest-backtrack*: $suffix\ (shortest\text{-}backtrack\ C\ \Gamma)\ \Gamma$
by (*induction* Γ) (*simp-all add: suffix-Cons*)

lemma *ex-conflict-shortest-backtrack*: $ex\text{-}conflict\ C\ (shortest\text{-}backtrack\ C\ \Gamma) \longleftrightarrow$
 $C = \{\#\}$
by (*induction* Γ) (*auto simp add: ex-conflict-def*)

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

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

proof (*induction* Γ)

case *Nil*

then show *?case*

by (*simp add: is-shortest-backtrack-def ex-conflict-def*)

next

case (*Cons* $Kn\ \Gamma$)

then show *?case*

apply (*simp del:*)

apply (*rule conjI*)

apply (*simp add: is-shortest-backtrack-def suffix-Cons*)

by (*meson is-shortest-backtrack-def not-Cons-self2 suffix-ConsD suffix-appendD*
suffix-order.dual-order.antisym suffix-order.dual-order.refl)

qed

9 Monotonicity w.r.t. the Bounding Atom

lemma *scl-monotone-wrt-bound*:

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

shows $scl\ N\ \beta'\ S_0\ S_1$

using *assms(2)[unfolded scl-def]*

proof (*elim disjE*)

assume *propagate* $N\ \beta\ S_0\ S_1$

hence *propagate* $N\ \beta'\ S_0\ S_1$

apply (*cases rule: propagate.cases*)

using *propagateI is-ground-cls-imp-is-ground-lit[unfolded is-ground-lit-def, THEN*
assms(1)]

by *metis*

```

thus ?thesis
  by (simp add: scl-def)
next
assume decide N  $\beta$   $S_0$   $S_1$ 
with assms(1) have decide N  $\beta'$   $S_0$   $S_1$ 
  using decideI decide.cases
  by (metis atm-of-subst-lit is-ground-lit-def[of - · l -])
thus ?thesis
  by (simp add: scl-def)
next
assume conflict N  $\beta$   $S_0$   $S_1$ 
with assms(1) have conflict N  $\beta'$   $S_0$   $S_1$ 
  by (auto intro!: conflictI elim: conflict.cases)
thus ?thesis
  by (simp add: scl-def)
next
assume skip N  $\beta$   $S_0$   $S_1$ 
with assms(1) have skip N  $\beta'$   $S_0$   $S_1$ 
  by (auto intro!: skipI elim: skip.cases)
thus ?thesis
  by (simp add: scl-def)
next
assume factorize N  $\beta$   $S_0$   $S_1$ 
with assms(1) have factorize N  $\beta'$   $S_0$   $S_1$ 
  by (auto intro!: factorizeI elim: factorize.cases)
thus ?thesis
  by (simp add: scl-def)
next
assume resolve N  $\beta$   $S_0$   $S_1$ 
with assms(1) have resolve N  $\beta'$   $S_0$   $S_1$ 
  by (auto intro!: resolveI elim: resolve.cases)
thus ?thesis
  by (simp add: scl-def)
next
assume backtrack N  $\beta$   $S_0$   $S_1$ 
with assms(1) have backtrack N  $\beta'$   $S_0$   $S_1$ 
  by (auto intro!: backtrackI elim: backtrack.cases)
thus ?thesis
  by (simp add: scl-def)
qed

```

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 β S_0 S_1

shows *reasonable-scl* N β' S_0 S_1

unfolding *reasonable-scl-def*

proof (intro conjI impI)

show *scl* N β' S_0 S_1

using *assms* *scl-monotone-wrt-bound* *scl-if-reasonable* **by** *metis*

next
assume $decide\ N\ \beta'\ S_0\ S_1$
with $assms(2)$ **have** $decide\ N\ \beta\ S_0\ S_1$
using $decide\ well\ defined$
by $(simp\ add:\ reasonable\ scl\ def\ scl\ def)$
with $assms(2)$ **have** $\nexists S_2.\ conflict\ N\ \beta\ S_1\ S_2$
by $(simp\ add:\ reasonable\ scl\ def)$
with $assms(1)$ **show** $\nexists S_2.\ conflict\ N\ \beta'\ S_1\ S_2$
by $(simp\ add:\ conflict.\ simps)$
qed

lemma $regular\ scl\ monotone\ wrt\ bound:$
assumes $\bigwedge A.\ is\ ground\ atm\ A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and** $regular\ scl\ N\ \beta$
 $S_0\ S_1$
shows $regular\ scl\ N\ \beta'\ S_0\ S_1$
using $assms(2)[unfolded\ regular\ scl\ def]$
proof $(elim\ disjE\ conjE)$
assume $conflict\ N\ \beta\ S_0\ S_1$
hence $conflict\ N\ \beta'\ S_0\ S_1$
by $(simp\ add:\ conflict.\ simps)$
thus $regular\ scl\ N\ \beta'\ S_0\ S_1$
by $(simp\ add:\ regular\ scl\ def)$
next
assume $\nexists S_1'.\ conflict\ N\ \beta\ S_0\ S_1'$ **and** $reasonable\ scl\ N\ \beta\ S_0\ S_1$
have $\nexists S_1'.\ conflict\ N\ \beta'\ S_0\ S_1'$
using $\langle \nexists S_1'.\ conflict\ N\ \beta\ S_0\ S_1' \rangle$
by $(simp\ add:\ conflict.\ simps)$
moreover **have** $reasonable\ scl\ N\ \beta'\ S_0\ S_1$
using $assms(1)\ \langle reasonable\ scl\ N\ \beta\ S_0\ S_1 \rangle\ reasonable\ scl\ monotone\ wrt\ bound$
by $metis$
ultimately **show** $regular\ scl\ N\ \beta'\ S_0\ S_1$
by $(simp\ add:\ regular\ scl\ def)$
qed

lemma $min\ back\ regular\ scl\ monotone\ wrt\ bound:$
assumes
 $\bigwedge A.\ is\ ground\ atm\ A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and**
 $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta\ S_0\ S_1$
shows $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta'\ S_0\ S_1$
unfolding $shortest\ backtrack\ strategy\ def$
proof $(intro\ conjI\ impI)$
from $assms(2)$ **have** $regular\ scl\ N\ \beta\ S_0\ S_1$
by $(simp\ add:\ shortest\ backtrack\ strategy\ def)$
with $assms(1)$ **show** $regular\ scl\ N\ \beta'\ S_0\ S_1$
using $regular\ scl\ monotone\ wrt\ bound$
by $metis$
next
assume $backtrack\ N\ \beta'\ S_0\ S_1$
with $assms(2)$ **have** $backtrack\ N\ \beta\ S_0\ S_1$

using *backtrack-well-defined*
by (*simp add: shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*)
with *assms(2) show is-shortest-backtrack*
(fst (the (state-conflict S₀))) (state-trail S₀) (state-trail S₁)
by (*simp add: shortest-backtrack-strategy-def*)
qed

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 β S₀ S₁ \implies scl N β' S₀ S₁ and
reasonable-scl N β S₀ S₁ \implies reasonable-scl N β' S₀ S₁ and
regular-scl N β S₀ S₁ \implies regular-scl N β' S₀ S₁ and
shortest-backtrack-strategy regular-scl N β S₀ S₁ \implies
shortest-backtrack-strategy regular-scl N β' S₀ S₁

using *assms*

scl-monotone-wrt-bound
reasonable-scl-monotone-wrt-bound
regular-scl-monotone-wrt-bound
min-back-regular-scl-monotone-wrt-bound

by *metis+*

corollary

assumes

transp-on {A. is-ground-atm A} (\prec_B) and
is-ground-atm β and
is-ground-atm β' and
β \prec_B β'

shows

scl N β S₀ S₁ \implies scl N β' S₀ S₁ and
reasonable-scl N β S₀ S₁ \implies reasonable-scl N β' S₀ S₁ and
regular-scl N β S₀ S₁ \implies regular-scl N β' S₀ S₁ and
shortest-backtrack-strategy regular-scl N β S₀ S₁ \implies
shortest-backtrack-strategy regular-scl N β' S₀ S₁

proof –

from $\langle \text{transp-on } \{A. \text{is-ground-atm } A\} (\prec_B) \rangle$ **have** $\text{transp-on } \{A. \text{is-ground-atm } A\} (\preceq_B)$

by (*metis transp-on-reflcp*)

moreover from $\langle \beta \prec_B \beta' \rangle$ **have** $\beta \preceq_B \beta'$

by *blast*

ultimately have $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$

using $\langle \text{is-ground-atm } \beta \rangle \langle \text{is-ground-atm } \beta' \rangle$

by (*meson CollectI transp-onD*)

thus

scl N β S₀ S₁ \implies scl N β' S₀ S₁ and
reasonable-scl N β S₀ S₁ \implies reasonable-scl N β' S₀ S₁ and
regular-scl N β S₀ S₁ \implies regular-scl N β' S₀ S₁ and

```

    shortest-backtrack-strategy regular-scl N β S0 S1 ⇒
    shortest-backtrack-strategy regular-scl N β' S0 S1
  using monotonicity-wrt-bound
  by metis+
qed

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 β S and conflict-S: state-conflict S = Some
  ({#}, γ)
  shows ¬ satisfiable (grounding-of-cls (fset N))
proof -
  from sound-S conflict-S have fset N ⊧Ge {#{#}}
  unfolding sound-state-def state-conflict-def by auto
  thus ?thesis by simp
qed

lemma propagate-if-conflict-follows-decide:
  assumes
    trail-lt-β: trail-atoms-lt β S2 and
    no-conf:  $\nexists S_1. \text{conflict } N \beta S_0 S_1$  and deci: decide N β S0 S2 and conf: conflict
  N β S2 S3
  shows  $\exists S_4. \text{propagate } N \beta S_0 S_4$ 
proof -
  from deci obtain L γ Γ U where
    S0-def: S0 = (Γ, U, None) and
    S2-def: S2 = (trail-decide Γ (L · l γ), U, None) and
    is-ground-lit (L · l γ) and
    ¬ trail-defined-lit Γ (L · l γ) and
    atm-of L · a γ  $\preceq_B$  β
  by (elim decide.cases) blast

  from conf S2-def obtain D γD where
    S3-def: S3 = (trail-decide Γ (L · l γ), U, Some (D, γD})) and
    D-in: D |∈| N |∪| U and
    ground-D-σ: is-ground-cls (D · γD) and
    tr-Γ-L-false-D: trail-false-cls (trail-decide Γ (L · l γ)) (D · γD)
  by (auto elim: conflict.cases)

  have vars-cls D ⊆ subst-domain γD
  using ground-D-σ vars-cls-subset-subst-domain-if-grounding by blast

```

moreover have $\neg \text{trail-false-cls } \Gamma (D \cdot \gamma_D)$
using *not-trail-false-ground-cls-if-no-conflict*[*OF no-conf*] *D-in*
by (*simp add: S₀-def ground-D- σ*)

ultimately have $-(L \cdot l \gamma) \in\# D \cdot \gamma_D$
using *tr- Γ -L-false-D*
by (*metis subtrail-falseI decide-lit-def*)

then obtain $D' L'$ **where** *D-def*: $D = \text{add-mset } L' D'$ **and** $-(L \cdot l \gamma) = L' \cdot l \gamma_D$
by (*metis Melem-subst-cls multi-member-split*)

define C_0 **where**
 $C_0 = \{\#K \in\# D'. K \cdot l \gamma_D \neq L' \cdot l \gamma_D\# \}$

define C_1 **where**
 $C_1 = \{\#K \in\# D'. K \cdot l \gamma_D = L' \cdot l \gamma_D\# \}$

have *ball-atms-lt- β* : $\forall K \in\# D \cdot \gamma_D. \text{atm-of } K \preceq_B \beta$
proof (*rule ballI*)
fix K **assume** $K \in\# D \cdot \gamma_D$
hence $K = L' \cdot l \gamma_D \vee (K \in\# D' \cdot \gamma_D)$
by (*simp add: D-def*)
thus *atm-of* $K \preceq_B \beta$
proof (*rule disjE*)
assume $K = L' \cdot l \gamma_D$
thus *?thesis*
using $\langle -(L \cdot l \gamma) = L' \cdot l \gamma_D \rangle \langle \text{atm-of } L \cdot a \gamma \preceq_B \beta \rangle$
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
next
have *trail-lt- β'* : $\forall A \in \text{atm-of 'fst ' set (trail-decide } \Gamma (L \cdot l \gamma)). A \preceq_B \beta$
using *trail-lt- β* **by** (*simp add: trail-atoms-lt-def S₂-def*)

assume *K-in*: $K \in\# D' \cdot \gamma_D$
hence *atm-of* $K \in \text{atm-of 'fst ' set (trail-decide } \Gamma (L \cdot l \gamma))$
using *tr- Γ -L-false-D*[*unfolded D-def*]
by (*metis D-def* $\langle K \in\# D \cdot \gamma_D \rangle$ *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set trail-false-cls-def trail-false-lit-def*)
moreover from *trail-lt- β* **have** $\forall A \in \text{atm-of 'fst ' set (trail-decide } \Gamma (L \cdot l \gamma)). A \preceq_B \beta$
by (*simp add: trail-atoms-lt-def S₂-def*)
ultimately show *?thesis*
by *blast*
qed
qed

have *tr-false-C₁*: *trail-false-cls* $\Gamma (C_0 \cdot \gamma_D)$
proof (*rule subtrail-falseI*)

```

show trail-false-cls ((L ·l γ, None) # Γ) (C0 · γD)
  unfolding C0-def trail-false-cls-def
  apply (rule ballI)
  apply (rule tr-Γ-L-false-D[unfolding D-def trail-false-cls-def decide-lit-def,
rule-format])
  by auto
next
show - (L ·l γ) # C0 · γD
  unfolding C0-def
  using ⟨- (L ·l γ) = L' ·l γD⟩ by fastforce
qed

have not-def-L'-ρ-σρ: ¬ trail-defined-lit Γ (L' ·l γD)
  using ⟨¬ trail-defined-lit Γ (L ·l γ)⟩
  by (metis ⟨- (L ·l γ) = L' ·l γD⟩ trail-defined-lit-iff-defined-uminus)

obtain xs where mset xs = add-mset L' C1
  using ex-mset by auto
hence set-xs-conv:
  set xs = set-mset (add-mset L' C1)
  by (metis set-mset-mset)

have unifiers (set (pairs (map atm-of xs))) ≠ {}
proof (rule not-empty-if-mem)
  have set (pairs (map atm-of xs)) =
    atm-of ' set-mset (add-mset L' C1) × atm-of ' set-mset (add-mset L' C1)
  unfolding set-pairs list.set-map set-xs-conv by simp
  also have ... =
    atm-of ' insert L' (set-mset C1) × atm-of ' insert L' (set-mset C1)
  unfolding set-mset-add-mset-insert by simp
  also have ... =
    atm-of ' insert L' {K. K ∈# D' ∧ K ·l γD = L' ·l γD} ×
    atm-of ' insert L' {K. K ∈# D' ∧ K ·l γD = L' ·l γD}
  unfolding C1-def set-mset-filter by simp
  finally have set-pairs-xs-simp: set (pairs (map atm-of xs)) =
    atm-of ' insert L' {K. K ∈# D' ∧ K ·l γD = L' ·l γD} ×
    atm-of ' insert L' {K. K ∈# D' ∧ K ·l γD = L' ·l γD}
  by assumption

show γD ∈ unifiers (set (pairs (map atm-of xs)))
  unfolding unifiers-def mem-Collect-eq
proof (rule ballI)
  fix p assume p-in: p ∈ set (pairs (map atm-of xs))
  then obtain K1 K2 where p-def: p = (atm-of K1, atm-of K2) and
    K1 = L' ∨ K1 ∈ {K. K ∈# D' ∧ K ·l γD = L' ·l γD} and
    K2 = L' ∨ K2 ∈ {K. K ∈# D' ∧ K ·l γD = L' ·l γD}
  by (auto simp: set-pairs-xs-simp)
  hence K1 ·l γD = L' ·l γD ∧ K2 ·l γD = L' ·l γD
  by auto

```

thus $\text{fst } p \cdot a \ \gamma_D = \text{snd } p \cdot a \ \gamma_D$
unfolding $p\text{-def } \text{prod.sel}$
by (*metis atm-of-subst-lit*)
qed
qed
then obtain ys **where**
 $\text{unify-pairs: unify (pairs (map atm-of } xs)) [] = \text{Some } ys$
using $\text{ex-unify-if-unifiers-not-empty}[OF - refl]$ **by** *blast*

define μ **where**
 $\mu = \text{subst-of } ys$

have $\text{imgu-}\mu$: $\text{is-imgu } \mu \ \{\text{atm-of ' set-mset (add-mset } L' \ C_1)\}$
proof (*intro is-imgu-if-mgu-sets[unfolded mgu-sets-def] exI conjI*)
show $\text{set (map set [(map atm-of } xs])} = \{\text{atm-of ' set-mset (add-mset } L' \ C_1)\}$
by (*simp add: set-xs-conv*)
next
show $\text{map-option subst-of (unify (concat (map pairs [map atm-of } xs])) [])} =$
 $\text{Some } \mu$
by (*simp add: unify-pairs } \mu\text{-def}*)
qed

show *?thesis*
using $\text{propagateI}[OF D\text{-in } D\text{-def, of } \gamma_D, \text{unfolded subst-cls-comp-subst subst-lit-comp-subst,}$
 $OF \text{ ground-}D\text{-}\sigma \ \text{ball-atms-lt-}\beta \ C_0\text{-def } C_1\text{-def } \text{tr-false-}C_1 \ \text{not-def-}L'\text{-}\varrho\text{-}\sigma_\varrho$
 $\text{imgu-}\mu]$
unfolding $S_0\text{-def}$ **by** *blast*
qed

theorem *correct-termination*:
fixes $\text{gnd-}N$ **and** $\text{gnd-}N\text{-lt-}\beta$
assumes
 $\text{sound-}S$: $\text{sound-state } N \ \beta \ S$ **and**
 $\text{invars: trail-atoms-lt } \beta \ S \ \text{trail-propagated-wf (state-trail } S) \ \text{trail-lits-consistent}$
 S
 $\text{ground-false-closures } S$ **and**
 $\text{no-new-conflict: } \#S'. \ \text{conflict } N \ \beta \ S \ S'$ **and**
 $\text{no-new-propagate: } \#S'. \ \text{propagate } N \ \beta \ S \ S'$ **and**
 $\text{no-new-decide: } \#S'. \ \text{decide } N \ \beta \ S \ S' \wedge (\#S''. \ \text{conflict } N \ \beta \ S' \ S'')$ **and**
 $\text{no-new-skip: } \#S'. \ \text{skip } N \ \beta \ S \ S'$ **and**
 $\text{no-new-resolve: } \#S'. \ \text{resolve } N \ \beta \ S \ S'$ **and**
 $\text{no-new-backtrack: } \#S'. \ \text{backtrack } N \ \beta \ S \ S' \wedge$
 $\text{is-shortest-backtrack (fst (the (state-conflict } S))) (state-trail } S) (state-trail } S')$

defines
 $\text{gnd-}N \equiv \text{grounding-of-clss (fset } N)$ **and**
 $\text{gnd-}N\text{-lt-}\beta \equiv \{C \in \text{gnd-}N. \ \forall L \in \# \ C. \ \text{atm-of } L \ \preceq_B \ \beta\}$
shows $\neg \text{satisfiable gnd-}N \wedge (\exists \gamma. \ \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable gnd-}N\text{-lt-}\beta \wedge \text{trail-true-clss (state-trail } S) \ \text{gnd-}N\text{-lt-}\beta$

proof –

```

obtain  $\Gamma U u$  where  $S\text{-def}: S = (\Gamma, U, u)$ 
  using prod-cases3 by blast

from sound-S have  $\text{sound-}\Gamma: \text{sound-trail } N \Gamma$ 
  by (simp-all add: S-def sound-state-def)

from  $\langle \text{ground-false-closures } S \rangle$  have  $\text{ground-closures } S$ 
  by (simp add: ground-false-closures-def)

have  $\text{trail-consistent}: \text{trail-consistent } (\text{state-trail } S)$ 
  using  $\langle \text{trail-lits-consistent } S \rangle$  by (simp add: trail-lits-consistent-def)

show ?thesis
proof (cases u)
  case  $u\text{-def}: \text{None}$ 
  hence  $\text{state-conflict } S = \text{None}$ 
  by (simp add: S-def)

  have  $tr\text{-true}: \text{trail-true-cls } \Gamma \text{ gnd-}N\text{-lt-}\beta$ 
    unfolding  $\text{trail-true-cls-def gnd-}N\text{-lt-}\beta\text{-def gnd-}N\text{-def}$ 
  proof (rule ballI, unfold mem-Collect-eq, erule conjE)
    fix  $C$  assume  $C\text{-in}: C \in \text{grounding-of-cls } (\text{fset } N)$  and  $C\text{-lt-}\beta: \forall L \in \# C.$ 
     $\text{atm-of } L \preceq_B \beta$ 

    from  $C\text{-in}$  have  $\text{is-ground-cls } C$ 
      by (rule grounding-ground)

    from  $C\text{-in}$  obtain  $C' \gamma$  where  $C'\text{-in}: C' \in N$  and  $C\text{-def}: C = C' \cdot \gamma$ 
      unfolding  $\text{grounding-of-cls-def grounding-of-cls-def}$ 
      by (smt (verit, ccfv-threshold) UN-iff mem-Collect-eq)

    from no-new-decide have  $\Gamma\text{-defined-}C: \text{trail-defined-cls } \Gamma C$ 
    proof (rule contrapos-np)
      assume  $\neg \text{trail-defined-cls } \Gamma C$ 
      then obtain  $L$  where  $L\text{-in}: L \in \# C$  and  $\neg \text{trail-defined-lit } \Gamma L$ 
        using  $\text{trail-defined-cls-def}$  by blast
      then obtain  $L'$  where  $L'\text{-in}: L' \in \# C'$  and  $L = L' \cdot l \gamma$ 
        using  $C\text{-def Melem-subst-cls}$  by blast

      have  $\text{deci}: \text{decide } N \beta (\Gamma, U, \text{None}) (\text{trail-decide } \Gamma (L' \cdot l \gamma), U, \text{None})$ 
      proof (rule decideI)
        show  $\text{is-ground-lit } (L' \cdot l \gamma)$ 
          using  $L\text{-in } \langle L = L' \cdot l \gamma \rangle \langle \text{is-ground-cls } C \rangle \text{is-ground-cls-def}$  by blast
        next
          show  $\neg \text{trail-defined-lit } \Gamma (L' \cdot l \gamma)$ 
            using  $\langle L = L' \cdot l \gamma \rangle \langle \neg \text{trail-defined-lit } \Gamma L \rangle$  by blast
        next
          show  $\text{atm-of } L' \cdot a \gamma \preceq_B \beta$ 
            using  $\langle L = L' \cdot l \gamma \rangle C\text{-lt-}\beta L\text{-in}$  by fastforce

```

qed

moreover have $\nexists S''$. *conflict* $N \beta$ (*trail-decide* $\Gamma (L' \cdot l \gamma)$, U , *None*) S''

proof (*rule notI*, *elim exE*)

fix S''

assume *conf*: *conflict* $N \beta$ (*trail-decide* $\Gamma (L' \cdot l \gamma)$, U , *None*) S''

moreover have *trail-atoms-lt* β (*trail-decide* $\Gamma (L' \cdot l \gamma)$, U , *None*)

using *decide-preserves-trail-atoms-lt*[*OF deci*] \langle *trail-atoms-lt* β S \rangle

by (*simp add*: *S-def u-def*)

ultimately have $\exists S_4$. *propagate* $N \beta$ $S S_4$

using *propagate-if-conflict-follows-decide*[*OF - no-new-conflict*]

using *S-def deci u-def* by *blast*

with *no-new-propagate* show *False*

by *metis*

qed

ultimately show $\exists S'$. *decide* $N \beta$ $S S' \wedge (\nexists S''$. *conflict* $N \beta$ $S' S''$)

by (*auto simp* : *S-def u-def*)

qed

show *trail-true-cls* ΓC

using Γ -*defined-C*[*THEN trail-true-or-false-cls-if-defined*]

proof (*elim disjE*)

show *trail-true-cls* $\Gamma C \implies$ *trail-true-cls* ΓC

by *assumption*

next

assume *trail-false-cls* ΓC

define $\varrho :: 'v \Rightarrow ('f, 'v)$ term where

$\varrho =$ *renaming-wrt* (*fset* ($N \mid \cup \mid U \mid \cup \mid$ *cls-of-trail* Γ))

define γ_ϱ where

$\gamma_\varrho =$ *rename-subst-domain* ϱ (*restrict-subst-domain* (*vars-cls* C') γ)

have *conflict* $N \beta$ (Γ , U , *None*) (Γ , U , *Some* (C' , *restrict-subst-domain* (*vars-cls* C') γ))

proof (*rule conflictI*)

show $C' \mid \in \mid N \mid \cup \mid U$

using *C'-in* by *simp*

next

show *is-ground-cls* ($C' \cdot$ *restrict-subst-domain* (*vars-cls* C') γ)

using \langle *is-ground-cls* C \rangle [*unfolded C-def*]

by (*simp add*: *subst-cls-restrict-subst-domain*)

next

show *trail-false-cls* $\Gamma (C' \cdot$ *restrict-subst-domain* (*vars-cls* C') γ)

using \langle *trail-false-cls* ΓC \rangle [*unfolded C-def*]

by (*simp add*: *subst-cls-restrict-subst-domain*)

qed

with *no-new-conflict* have *False*

by (simp add: S-def u-def)
 thus trail-true-cls Γ C ..
 qed
 qed

moreover have satisfiable gnd-N-lt- β
 unfolding true-cls-def gnd-N-lt- β -def
 proof (intro exI ballI, unfold mem-Collect-eq, elim conjE)
 fix C
 have trail-consistent Γ
 using S-def trail-consistent by auto
 show $C \in \text{gnd-N} \implies \forall L \in \# C. \text{atm-of } L \preceq_B \beta \implies \text{trail-interp } \Gamma \Vdash C$
 using tr-true[unfolded gnd-N-lt- β -def]
 using trail-interp-cls-if-trail-true[OF $\langle \text{trail-consistent } \Gamma \rangle$]
 by (simp add: trail-true-cls-def)
 qed

ultimately show ?thesis
 by (simp add: S-def)

next
 case (Some C)
 then obtain C γ_C where u-def: $u = \text{Some } (C, \gamma_C)$ by force

from $\langle \text{ground-false-closures } S \rangle$ have Γ -false- C - γ_C : trail-false-cls Γ $(C \cdot \gamma_C)$
 by (simp add: S-def u-def ground-false-closures-def)

show ?thesis

proof (cases $C = \{\#\}$)
 case True
 hence $\neg \text{satisfiable gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma))$
 using S-def u-def not-satisfiable-if-sound-state-conflict-bottom[OF sound- S]
 by (simp add: gnd-N-def)
 thus ?thesis by simp

next
 case C-not-empty: False
 have False
 proof (cases Γ)
 case Nil
 with Γ -false- C - γ_C show False
 using C-not-empty by simp

next
 case (Cons L_n Γ')
 then obtain K_Γ n where Γ -def: $\Gamma = (K_\Gamma, n) \# \Gamma'$
 by fastforce

show False

proof (cases $\neg K_\Gamma \in \# C \cdot \gamma_C$)
 case True — Literal cannot be skipped
 then obtain C' L where C-def: $C = \text{add-mset } L C'$ and K - γ : $L \cdot l \gamma_C$

```

= - KΓ
  by (metis Melem-subst-cls multi-member-split)
  hence L-eq-uminus-K-γ: KΓ = - (L · l γC)
  by simp

show False
proof (cases n)
  case None — Conflict clause can be backtracked
  hence Γ-def: Γ = trail-decide Γ' (- (L · l γC))
  by (simp add: Γ-def L-eq-uminus-K-γ decide-lit-def)

  from suffix-shortest-backtrack[of add-mset L C' Γ'] obtain Γ'' where
  Γ'-def: Γ' = Γ'' @ shortest-backtrack (add-mset L C') Γ'
  using suffixE by blast

  define S' :: ('f, 'v) state where
  S' = (shortest-backtrack (add-mset L C') Γ', finset (add-mset L C')
  U, None)

  have backtrack N β S S'
  unfolding S-def[unfolded u-def C-def] S'-def
  proof (rule backtrackI[OF - refl])
    show Γ = trail-decide (Γ'' @ shortest-backtrack (add-mset L C') Γ')
    (- (L · l γC))
    using Γ-def Γ'-def by simp
  next
  show ‡γ. is-ground-cls (add-mset L C' · γ) ∧
  trail-false-cls (shortest-backtrack (add-mset L C') Γ') (add-mset L C'
  · γ)
  using ex-conflict-shortest-backtrack[of add-mset L C', simplified]
  by (simp add: ex-conflict-def)
  qed
  moreover have is-shortest-backtrack (fst (the (state-conflict S)))
  (state-trail S) (state-trail S')
  unfolding S-def[unfolded u-def C-def] S'-def
  apply simp
  using is-shortest-backtrack-shortest-backtrack[of add-mset L C', simplified]
  using Γ-def Γ-false-C-γC
  by (metis (no-types, lifting) C-def ‹S = (Γ, U, Some (add-mset L C',
  γC))›
  ‹ground-closures S› ex-conflict-def ground-closures-def is-shortest-backtrack-def
  state-conflict-simp suffix-Cons)

  ultimately show False
  using no-new-backtrack by metis
next
  case Some — Literal can be resolved
  then obtain D K γD where n-def: n = Some (D, K, γD)
  by (metis prod-cases3)

```

```

with ⟨trail-propagated-wf (state-trail S)⟩ have L-def:  $K_\Gamma = K \cdot l \ \gamma_D$ 
  by (simp add:  $\Gamma$ -def S-def trail-propagated-wf-def)
hence  $1: \Gamma = \text{trail-propagate } \Gamma' \ K \ D \ \gamma_D$ 
  using  $\Gamma$ -def n-def
  by (simp add: propagate-lit-def)

from ⟨ground-closures S⟩ have
  ground-conf: is-ground-cls (add-mset  $L \ C' \cdot \gamma_C$ ) and
  ground-prop: is-ground-cls (add-mset  $K \ D \cdot \gamma_D$ )
  unfolding S-def ground-closures-def
  by (simp-all add: 1 C-def u-def ground-closures-def propagate-lit-def)

define  $\varrho :: 'v \Rightarrow ('f, 'v) \text{Term.term}$  where
   $\varrho = \text{renaming-wrt } \{\text{add-mset } K \ D\}$ 

have ren- $\varrho$ : is-renaming  $\varrho$ 
  unfolding  $\varrho$ -def
  by (rule is-renaming-renaming-wrt simp)
hence  $\forall x. \text{is-Var } (\varrho \ x) \ \text{inj } \varrho$ 
  by (simp-all add: is-renaming-iff)

have disjoint-vars:  $\bigwedge C. \text{vars-cls } (C \cdot \varrho) \cap \text{vars-cls } (\text{add-mset } K \ D \cdot \text{Var})$ 
= {
  by (simp add:  $\varrho$ -def vars-cls-subst-renaming-disj)

have  $2: K \cdot l \ \gamma_D = - (L \cdot l \ \gamma_C)$ 
  using  $K\text{-}\gamma \ L\text{-def}$  by fastforce

let  $?\gamma_{D'} = \text{restrict-subst-domain } (\text{vars-cls } (\text{add-mset } K \ D)) \ \gamma_D$ 

have  $K \cdot l \ ?\gamma_{D'} = K \cdot l \ \gamma_D$  and  $D \cdot ?\gamma_{D'} = D \cdot \gamma_D$ 
by (simp-all add: subst-lit-restrict-subst-domain subst-cls-restrict-subst-domain)
hence  $K \cdot l \ ?\gamma_{D'} = - (L \cdot l \ \gamma_C)$  and ground-prop': is-ground-cls (add-mset
 $K \ D \cdot ?\gamma_{D'}$ )
  using  $2$  ground-prop by simp-all

have dom- $\gamma_{D'}$ : subst-domain  $?\gamma_{D'} \subseteq \text{vars-cls } (\text{add-mset } K \ D)$ 
  by simp

let  $?\gamma = \lambda x.$ 
  if  $x \in \text{vars-cls } (\text{add-mset } L \ C' \cdot \varrho)$  then
    rename-subst-domain  $\varrho \ \gamma_C \ x$ 
  else
     $\gamma_D \ x$ 
have  $L \cdot l \ \varrho \cdot l \ ?\gamma = L \cdot l \ \gamma_C$  and  $K \cdot l \ ?\gamma = K \cdot l \ \gamma_D$ 
  using merge-of-renamed-groundings[OF ren- $\varrho$  is-renaming-id-subst
disjoint-vars
  ground-conf ground-prop is-grounding-merge-if-mem-then-else]
  unfolding rename-subst-domain-Var-lhs

```

by *simp-all*

then have $atm\text{-of } L \cdot a \varrho \cdot a \text{ ?}\gamma = atm\text{-of } K \cdot a \text{ ?}\gamma$

by (*smt* (*verit*, *best*) 2 *atm-of-uminus subst-lit-id-subst atm-of-subst-lit*)

then obtain μ **where** $Unification.mgu (atm\text{-of } L \cdot a \varrho) (atm\text{-of } K) =$

Some μ

using *ex-mgu-if-subst-apply-term-eq-subst-apply-term*

by *blast*

hence *imgu- μ* : *is-imgu* μ $\{\{atm\text{-of } L \cdot a \varrho, atm\text{-of } K \cdot a \text{ Var}\}\}$

by (*simp add: is-imgu-if-mgu-eq-Some*)

have $\exists S. resolve\ N\ \beta\ (\Gamma, U, Some\ (add\ mset\ L\ C', \gamma_C))\ S$

using *resolveI[OF 1 2 ren- ϱ is-renaming-id-subst disjoint-vars imgu- μ is-grounding-merge-if-mem-then-else]* ..

with *no-new-resolve* **show** *False*

by (*metis C-def S-def u-def*)

qed

next

case *False* — Literal can be skipped

hence *skip* $N\ \beta\ ((K_\Gamma, n) \# \Gamma', U, Some\ (C, \gamma_C))\ (\Gamma', U, Some\ (C, \gamma_C))$

by (*rule skipI[of K Γ C γ_C N β n Γ' U]*)

with *no-new-skip* **show** *False*

by (*metis S-def Γ -def u-def*)

qed

qed

thus *?thesis* ..

qed

qed

qed

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:

$\bigwedge N\ \beta\ S\ S'. strategy\ N\ \beta\ S\ S' \implies sound\ state\ N\ \beta\ S \implies sound\ state\ N\ \beta\ S'$

$\bigwedge N\ \beta\ S\ S'. strategy\ N\ \beta\ S\ S' \implies trail\ atoms\ lt\ \beta\ S \implies trail\ atoms\ lt\ \beta\ S'$

$\bigwedge N\ \beta\ S\ S'. strategy\ N\ \beta\ S\ S' \implies trail\ propagated\ or\ decided'\ N\ \beta\ S \implies trail\ propagated\ or\ decided'\ N\ \beta\ S'$

$\bigwedge N\ \beta\ S\ S'. strategy\ N\ \beta\ S\ S' \implies trail\ lits\ consistent\ S \implies trail\ lits\ consistent\ S'$

$\bigwedge N\ \beta\ S\ S'. strategy\ N\ \beta\ S\ S' \implies ground\ false\ closures\ S \implies ground\ false\ closures\ S'$

defines

gnd-N \equiv *grounding-of-class* (*fset* N) **and**

$gnd-N-lt-\beta \equiv \{C \in gnd-N. \forall L \in \# C. atm-of L \preceq_B \beta\}$
shows \neg *satisfiable* $gnd-N \wedge (\exists \gamma. state-conflict S = Some (\{\#\}, \gamma)) \vee$
satisfiable $gnd-N-lt-\beta \wedge trail-true-cls (state-trail S) gnd-N-lt-\beta$

proof –

from *no-step* **have** *no-step'*: $\nexists S'. shortest-backtrack-strategy regular-scl N \beta S S'$

proof (*rule contrapos-nn*)

show $\exists S'. shortest-backtrack-strategy regular-scl N \beta S S' \implies \exists S'. strategy N \beta S S'$

using *strategy-restricted-by-min-back* **by** *metis*

qed

show *?thesis*

proof (*rule correct-termination*[*of* $N \beta S$, *folded* $gnd-N-def$, *folded* $gnd-N-lt-\beta-def$])

from *run* **show** *sound-state* $N \beta S$

by (*induction* S *rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(1))

next

from *run* **show** *trail-atoms-lt* βS

by (*induction* S *rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(2))

next

from *run* **have** *trail-propagated-or-decided'* $N \beta S$

by (*induction* S *rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(3))

thus *trail-propagated-wf* (*state-trail* S)

by (*simp add*: *trail-propagated-or-decided'-def*
trail-propagated-wf-if-trail-propagated-or-decided)

next

from *run* **show** *trail-lits-consistent* S

by (*induction* S *rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(4))

next

from *run* **show** *ground-false-closures* S

by (*induction* S *rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(5))

next

from *no-step'* **show** $\nexists S'. conflict N \beta S S'$

unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def*
scl-def

using *backtrack-well-defined*(3) **by** *blast*

next

from *no-step'* **show** $\nexists S'. propagate N \beta S S'$

unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def*
scl-def

using *backtrack-well-defined*(3) *propagate-well-defined*(1) *propagate-well-defined*(6)

by *blast*

next

from *no-step'* **show** $\nexists S'. decide N \beta S S' \wedge (\nexists S''. conflict N \beta S' S'')$

unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def*
scl-def

using *backtrack-well-defined*(2) *backtrack-well-defined*(3) **by** *blast*

next

from *no-step'* **show** $\nexists S'. skip N \beta S S'$

unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*
using *backtrack-well-defined(3) backtrack-well-defined(4) skip-well-defined(2)*
by *blast*
next
from *no-step'* **show** $\nexists S'. \text{resolve } N \beta S S'$
unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*
using *backtrack-well-defined(3) resolve-well-defined(2) resolve-well-defined(5)*
by *blast*
next
from *no-step'* **show** $\nexists S'. \text{backtrack } N \beta S S' \wedge$
is-shortest-backtrack (fst (the (state-conflict S))) (state-trail S) (state-trail S'))
unfolding *shortest-backtrack-strategy-def scl-def regular-scl-def reasonable-scl-def*
using *backtrack-well-defined(2) backtrack-well-defined(3) by blast*
qed
qed

corollary *correct-termination-scl-run:*
fixes *gnd-N and gnd-N-lt-β*
assumes
*run: (scl N β)** initial-state S and*
no-step: ∃ S'. scl N β S S'
defines
gnd-N ≡ grounding-of-cls (fset N) and
gnd-N-lt-β ≡ {C ∈ gnd-N. ∀ L ∈# C. atm-of L ≤_B β}
shows $\neg \text{satisfiable } \text{gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
satisfiable gnd-N-lt-β ∧ trail-true-cls (state-trail S) gnd-N-lt-β
proof (*rule correct-termination-strategy[of - N β, folded gnd-N-def, folded gnd-N-lt-β-def]*)
show *(scl N β)** initial-state S*
by (*rule run*)
next
show $\nexists S'. \text{scl } N \beta S S'$
by (*rule no-step*)
next
show $\bigwedge S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{scl } N \beta S S'$
by (*simp add: regular-scl-if-shortest-backtrack-strategy scl-if-regular*)
next
show $\bigwedge N \beta S S'. \text{scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
using *scl-preserves-sound-state by simp*
next
show $\bigwedge N \beta S S'. \text{scl } N \beta S S' \implies \text{trail-atoms-lt } \beta S \implies \text{trail-atoms-lt } \beta S'$
using *scl-preserves-trail-atoms-lt by simp*
next
show $\bigwedge N \beta S S'. \text{scl } N \beta S S' \implies \text{trail-propagated-or-decided}' N \beta S \implies$
trail-propagated-or-decided}' N \beta S'
using *scl-preserves-trail-propagated-or-decided by simp*
next
show $\bigwedge N \beta S S'. \text{scl } N \beta S S' \implies \text{trail-lits-consistent } S \implies \text{trail-lits-consistent}$

S'
using *scl-preserves-trail-lits-consistent* **by** *simp*
next
show $\bigwedge N \beta S S'. \text{scl } N \beta S S' \implies \text{ground-false-closures } S \implies \text{ground-false-closures } S'$
 S'
using *scl-preserves-ground-false-closures* **by** *simp*
qed

corollary *correct-termination-reasonable-scl-run*:
fixes *gnd-N* **and** *gnd-N-lt- β*
assumes
run: *(reasonable-scl N β)** initial-state S* **and**
no-step: $\nexists S'. \text{reasonable-scl } N \beta S S'$
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 $\neg \text{satisfiable gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable gnd-N-lt-}\beta \wedge \text{trail-true-cls (state-trail } S) \text{ gnd-N-lt-}\beta$
proof (*rule correct-termination-strategy[of - N β , folded gnd-N-def, folded gnd-N-lt- β -def]*)
show *(reasonable-scl N β)** initial-state S*
by (*rule run*)
next
show $\nexists S'. \text{reasonable-scl } N \beta S S'$
by (*rule no-step*)
next
show $\bigwedge S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{reasonable-scl } N \beta S S'$
by (*simp add: reasonable-if-regular regular-scl-if-shortest-backtrack-strategy*)
next
show $\bigwedge N \beta S S'. \text{reasonable-scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
using *scl-preserves-sound-state[OF scl-if-reasonable]* **by** *simp*
next
show $\bigwedge N \beta S S'. \text{reasonable-scl } N \beta S S' \implies \text{trail-atoms-lt } \beta S \implies \text{trail-atoms-lt } \beta S'$
using *scl-preserves-trail-atoms-lt[OF scl-if-reasonable]* **by** *simp*
next
show $\bigwedge N \beta S S'. \text{reasonable-scl } N \beta S S' \implies \text{trail-propagated-or-decided}' N \beta S \implies$
 $\text{trail-propagated-or-decided}' N \beta S'$
using *scl-preserves-trail-propagated-or-decided[OF scl-if-reasonable]* **by** *simp*
next
show $\bigwedge N \beta S S'. \text{reasonable-scl } N \beta S S' \implies \text{trail-lits-consistent } S \implies \text{trail-lits-consistent } S'$
using *scl-preserves-trail-lits-consistent[OF scl-if-reasonable]* **by** *simp*
next
show $\bigwedge N \beta S S'. \text{reasonable-scl } N \beta S S' \implies \text{ground-false-closures } S \implies$
 $\text{ground-false-closures } S'$
using *scl-preserves-ground-false-closures[OF scl-if-reasonable]* **by** *simp*

qed

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

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

assumes

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

no-step: $\nexists S'. regular-scl\ N\ \beta\ S\ S'$

defines

$gnd-N \equiv grounding-of-cls\ (fset\ N)$ **and**

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

shows \neg *satisfiable* $gnd-N \wedge (\exists \gamma.\ state-conflict\ S = Some\ (\{\#\}, \gamma)) \vee$

satisfiable $gnd-N-lt-\beta \wedge trail-true-cls\ (state-trail\ S)\ gnd-N-lt-\beta$

proof (*rule correct-termination-strategy*[*of* - $N\ \beta$, *folded* $gnd-N-def$, *folded* $gnd-N-lt-\beta-def$])

show $(regular-scl\ N\ \beta)^{**}$ *initial-state* S

by (*rule run*)

next

show $\nexists S'. regular-scl\ N\ \beta\ S\ S'$

by (*rule no-step*)

next

show $\bigwedge S\ S'. shortest-backtrack-strategy\ regular-scl\ N\ \beta\ S\ S' \implies regular-scl\ N\ \beta\ S\ S'$

by (*simp add: reasonable-if-regular regular-scl-if-shortest-backtrack-strategy*)

next

show $\bigwedge N\ \beta\ S\ S'. regular-scl\ N\ \beta\ S\ S' \implies sound-state\ N\ \beta\ S \implies sound-state\ N\ \beta\ S'$

using *scl-preserves-sound-state*[*OF scl-if-regular*] **by** *simp*

next

show $\bigwedge N\ \beta\ S\ S'. regular-scl\ N\ \beta\ S\ S' \implies trail-atoms-lt\ \beta\ S \implies trail-atoms-lt\ \beta\ S'$

using *scl-preserves-trail-atoms-lt*[*OF scl-if-regular*] **by** *simp*

next

show $\bigwedge N\ \beta\ S\ S'. regular-scl\ N\ \beta\ S\ S' \implies trail-propagated-or-decided'\ N\ \beta\ S \implies$

$trail-propagated-or-decided'\ N\ \beta\ S'$

using *scl-preserves-trail-propagated-or-decided*[*OF scl-if-regular*] **by** *simp*

next

show $\bigwedge N\ \beta\ S\ S'. regular-scl\ N\ \beta\ S\ S' \implies trail-lits-consistent\ S \implies trail-lits-consistent\ S'$

using *scl-preserves-trail-lits-consistent*[*OF scl-if-regular*] **by** *simp*

next

show $\bigwedge N\ \beta\ S\ S'. regular-scl\ N\ \beta\ S\ S' \implies ground-false-closures\ S \implies ground-false-closures\ S'$

using *scl-preserves-ground-false-closures*[*OF scl-if-regular*] **by** *simp*

qed

corollary *correct-termination-shortest-backtrack-strategy-regular-scl:*

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

assumes

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

no-step: $\nexists S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S'$
defines
gnd-N \equiv *grounding-of-cls* (*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-cls* (*state-trail* *S*) *gnd-N-lt-β*
proof (*rule correct-termination-strategy*[*of* - *N* β , *folded gnd-N-def*, *folded gnd-N-lt-β-def*])
show (*shortest-backtrack-strategy regular-scl* *N* β)** *initial-state* *S*
by (*rule run*)
next
show $\nexists S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S'$
by (*rule no-step*)
next
show $\bigwedge S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{shortest-backtrack-strategy}$
regular-scl *N* β *S* *S'*
by *simp*
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{sound-state}$
N β *S* \implies *sound-state* *N* β *S'*
using *scl-preserves-sound-state*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{trail-atoms-lt}$
β *S* \implies *trail-atoms-lt* β *S'*
using *scl-preserves-trail-atoms-lt*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{trail-propagated-or-decided'}$
N β *S* \implies
trail-propagated-or-decided' *N* β *S'*
using *scl-preserves-trail-propagated-or-decided*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{trail-lits-consistent}$
S \implies *trail-lits-consistent* *S'*
using *scl-preserves-trail-lits-consistent*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{ground-false-closures}$
S \implies *ground-false-closures* *S'*
using *scl-preserves-ground-false-closures*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
qed

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 ∧ (∄ S'. regular-scl N β S S') ∨
    (shortest-backtrack-strategy regular-scl N β)** initial-state S ∧
    (∄ S'. shortest-backtrack-strategy regular-scl N β S S')
defines
  gnd-N ≡ grounding-of-cls (fset N) and
  gnd-N-lt-β ≡ {C ∈ gnd-N. ∀ L ∈ # C. atm-of L ≤B β}
shows ¬ satisfiable gnd-N ∧ (∃ γ. state-conflict S = Some ({#}, γ)) ∨
  satisfiable gnd-N-lt-β ∧ trail-true-cls (state-trail S) gnd-N-lt-β
unfolding gnd-N-def gnd-N-lt-β-def
using assms(1)
  correct-termination-scl-run[of N β S]
  correct-termination-reasonable-scl-run[of N β S]
  correct-termination-regular-scl-run[of N β S]
  correct-termination-shortest-backtrack-strategy-regular-scl[of N β S]
by argo

end

end
theory Trail-Induced-Ordering
imports

  Main

  List-Index.List-Index
begin

lemma wf-if-convertible-to-wf:
  fixes r :: 'a rel and s :: 'b rel and f :: 'a ⇒ 'b
  assumes wf s and convertible: ∧ x y. (x, y) ∈ r ⇒ (f x, f y) ∈ s
  shows wf r
proof (rule wfI-min[of r])
  fix x :: 'a and Q :: 'a set
  assume x ∈ Q
  then obtain y where y ∈ Q and ∧ z. (f z, f y) ∈ s ⇒ z ∉ Q
  by (auto elim: wfE-min[OF wf-inv-image[of s f, OF ‹wf s›], unfolded in-inv-image])
  thus ∃ z ∈ Q. ∀ y. (y, z) ∈ r → y ∉ Q
  by (auto intro: convertible)
qed

lemma wfP-if-convertible-to-wfP: wfP S ⇒ (∧ x y. R x y ⇒ S (f x) (f y)) ⇒
wfP R
using wf-if-convertible-to-wf[to-pred, of S R f] by simp

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 :: - ⇒ nat

```

shows $(\bigwedge x y. R x y \implies f x < f y) \implies \text{wfP } R$
by (rule *wfP-if-convertible-to-wfP*[of ($<$) :: nat \implies nat \implies bool, simplified])

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

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 (rule *transpI*)

show $\bigwedge x y z. \text{trail-less}' Ls x y \implies \text{trail-less}' Ls y z \implies \text{trail-less}' Ls x z$
by (*metis* (*no-types*, *lifting*) *trail-less'-def* *tranclp-trans*)

qed

lemma *trail-less'-Suc*:

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

proof –

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

unfolding *trail-less'-def*

using *assms* **by** *blast*

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

by (*metis* (*mono-tags*, *lifting*) *assms* *trail-less'-def* *tranclp.r-into-trancl*)

ultimately **show** *?thesis*

using *transp-trail-less'*[*THEN* *transpD*] **by** *auto*

qed

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

assumes $Suc\ i < length\ Ls$

shows $trail-less'\ Ls\ (\neg\ (Ls\ !\ Suc\ i))\ (\neg\ (Ls\ !\ i))$

proof –

have $trail-less'\ Ls\ (\neg\ (Ls\ !\ Suc\ i))\ (Ls\ !\ i)$

unfolding *trail-less'-def*

using *assms\ Suc-lessD* **by** *blast*

moreover have $trail-less'\ Ls\ (Ls\ !\ i)\ (\neg\ (Ls\ !\ i))$

unfolding *trail-less'-def*

using *assms\ Suc-lessD* **by** *blast*

ultimately show *?thesis*

using *transp-trail-less'[THEN\ transpD]* **by** *auto*

qed

lemma *trail-less'-id-id*: $j < i \implies i < length\ Ls \implies trail-less'\ Ls\ (Ls\ !\ i)\ (Ls\ !\ j)$

proof (*induction\ i\ arbitrary: j*)

case 0

then show *?case*

by *simp*

next

case (*Suc\ i*)

then show *?case*

using *trail-less'-Suc*

by (*metis\ Suc-lessD\ less-Suc-eq\ transpD\ transp-trail-less'*)

qed

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

$j < i \implies i < length\ Ls \implies trail-less'\ Ls\ (\neg\ (Ls\ !\ i))\ (\neg\ (Ls\ !\ j))$

proof (*induction\ i\ arbitrary: j*)

case 0

then show *?case*

by *simp*

next

case (*Suc\ i*)

then show *?case*

using *trail-less'-comp-Suc-comp*

by (*metis\ Suc-lessD\ less-Suc-eq\ transpD\ transp-trail-less'*)

qed

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

assumes $j < i$ **and** $i < length\ Ls$

shows $trail-less'\ Ls\ (Ls\ !\ i)\ (\neg\ (Ls\ !\ j))$

proof –

have $trail-less'\ Ls\ (Ls\ !\ j)\ (\neg\ (Ls\ !\ j))$

unfolding *trail-less'-def*

using *assms\ dual-order.strict-trans* **by** *blast*

thus *?thesis*

using *trail-less'-id-id*[*OF assms*]
 using *transp-trail-less'*[*THEN transpD*] by *auto*
 qed

lemma *trail-less'-comp-id*:
 assumes $j < i$ and $i < \text{length } Ls$
 shows $\text{trail-less}' Ls (- (Ls ! i)) (Ls ! j)$
proof (*cases i*)
 case 0
 then show *?thesis*
 using *assms(1)* by *blast*
 next
 case (*Suc i'*)
 hence $\text{trail-less}' Ls (- Ls ! i) (Ls ! i')$
 unfolding *trail-less'-def*
 using *Suc-lessD assms(2)* by *blast*
 show *?thesis*
proof (*cases i' = j*)
 case *True*
 then show *?thesis*
 using $\langle \text{trail-less}' Ls (- Ls ! i) (Ls ! i') \rangle$ by *metis*
 next
 case *False*
 hence $\text{trail-less}' Ls (Ls ! i') (Ls ! j)$
 by (*metis Suc Suc-lessD assms(1) assms(2) less-SucE trail-less'-id-id*)
 then show *?thesis*
 using $\langle \text{trail-less}' Ls (- Ls ! i) (Ls ! i') \rangle$
 using *transp-trail-less'*[*THEN transpD*] by *auto*
 qed
 qed

lemma *trail-less-eq-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 $\text{trail-less } Ls = \text{trail-less}' Ls$
proof (*intro ext iffI*)
 fix $L K$
 show $\text{trail-less } Ls L K \implies \text{trail-less}' Ls L K$
 unfolding *trail-less-def*
proof (*elim disjE*)
 assume *trail-less-id-id* $Ls L K$
 thus *?thesis*
 using *trail-less'-id-id* by (*metis trail-less-id-id-def*)
 next

```

show trail-less-comp-id Ls L K  $\implies$  ?thesis
  using trail-less'-comp-id by (metis trail-less-comp-id-def)
next
show trail-less-id-comp Ls L K  $\implies$  ?thesis
  using trail-less'-id-comp
  unfolding trail-less-id-comp-def
  by (metis (mono-tags, lifting) le-eq-less-or-eq trail-less'-def tranclp.r-into-trancl)
next
show trail-less-comp-comp Ls L K  $\implies$  ?thesis
  using trail-less'-comp-comp
  by (metis trail-less-comp-comp-def)
qed
next
fix L K
show trail-less' Ls L K  $\implies$  trail-less Ls L K
  unfolding trail-less'-def
proof (induction K rule: tranclp-induct)
  case (base K)
  then show ?case
  proof (elim exE conjE disjE)
    fix i assume i < length Ls and L = Ls ! i and K = - (Ls ! i)
    hence trail-less-id-comp Ls L K
      by (auto simp: trail-less-id-comp-def)
    thus trail-less Ls L K
      by (simp add: trail-less-def)
  next
    fix i assume Suc i < length Ls and L = - (Ls ! Suc i) and K = Ls ! i
    hence trail-less-comp-id Ls L K
      unfolding trail-less-comp-id-def
      using Suc-lessD by blast
    thus trail-less Ls L K
      by (simp add: trail-less-def)
  qed
next
case (step y z)
from step.hyps(2) show ?case
proof (elim exE conjE disjE)
  fix i assume i < length Ls and y = Ls ! i and z = - (Ls ! i)

  from step.IH[unfolded trail-less-def] show trail-less Ls L z
proof (elim disjE)
  assume trail-less-id-id Ls L y
  hence trail-less-id-comp Ls L z
    unfolding trail-less-id-id-def trail-less-id-comp-def
    by (metis <y = Ls ! i> <z = - Ls ! i> less-or-eq-imp-le)
  thus trail-less Ls L z
    by (simp add: trail-less-def)
next
  assume trail-less-comp-id Ls L y

```

```

hence trail-less-comp-comp Ls L z
  unfolding trail-less-comp-id-def trail-less-comp-comp-def
  by (metis  $\langle y = Ls ! i \rangle \langle z = - Ls ! i \rangle$ )
thus trail-less Ls L z
  by (simp add: trail-less-def)
next
assume trail-less-id-comp Ls L y
hence trail-less-id-comp Ls L z
  unfolding trail-less-id-comp-def
  by (metis  $\langle i < \text{length } Ls \rangle \langle y = Ls ! i \rangle \langle z = - Ls ! i \rangle$  pairwise-distinct)
thus trail-less Ls L z
  by (simp add: trail-less-def)
next
assume trail-less-comp-comp Ls L y
hence trail-less-comp-comp Ls L z
  unfolding trail-less-comp-comp-def
  by (metis  $\langle i < \text{length } Ls \rangle \langle y = Ls ! i \rangle \langle z = - Ls ! i \rangle$  pairwise-distinct)
thus trail-less Ls L z
  by (simp add: trail-less-def)
qed
next
fix i assume Suc i < length Ls and  $y = - Ls ! Suc i$  and  $z = Ls ! i$ 

from step.IH[unfolded trail-less-def] show trail-less Ls L z
proof (elim disjE)
  show trail-less-id-id Ls L y  $\implies$  trail-less Ls L z
    by (metis  $\langle Suc i < \text{length } Ls \rangle \langle y = - Ls ! Suc i \rangle \langle z = Ls ! i \rangle$  not-less-eq
      order-less-imp-not-less pairwise-distinct trail-less-def trail-less-id-id-def)
  next
  show trail-less-comp-id Ls L y  $\implies$  trail-less Ls L z
    by (smt (verit, best)  $\langle Suc i < \text{length } Ls \rangle \langle y = - Ls ! Suc i \rangle \langle z = Ls ! i \rangle$ 
      dual-order.strict-trans lessI pairwise-distinct trail-less-comp-id-def
trail-less-def)
  next
  assume trail-less-id-comp Ls L y
  hence trail-less-id-id Ls L z
  unfolding trail-less-def trail-less-id-comp-def trail-less-id-id-def
  by (metis Suc-le-lessD Suc-lessD  $\langle Suc i < \text{length } Ls \rangle \langle y = - Ls ! Suc i \rangle$ 
 $\langle z = Ls ! i \rangle$ 
pairwise-distinct uminus-uminus-id)
  thus trail-less Ls L z
  by (simp add: trail-less-def)
  next
  assume trail-less-comp-comp Ls L y
  hence trail-less-comp-id Ls L z
  unfolding trail-less-comp-comp-def trail-less-comp-id-def
  by (metis Suc-lessD  $\langle Suc i < \text{length } Ls \rangle \langle y = - Ls ! Suc i \rangle \langle z = Ls ! i \rangle$ 
pairwise-distinct
uminus-uminus-id)

```

```

    thus trail-less Ls L z
      by (simp add: trail-less-def)
  qed
  qed
  qed
  qed

```

9.1 Examples

experiment

```

  fixes L0 L1 L2 :: 'a :: uminus
  begin

```

```

lemma trail-less-id-comp [L2, L1, L0] L2 (− L2)
  unfolding trail-less-id-comp-def
proof (intro exI conjI)
  show (0 :: nat) ≤ 0 by presburger
qed simp-all

```

```

lemma trail-less-comp-id [L2, L1, L0] (− L1) L2
  unfolding trail-less-comp-id-def
proof (intro exI conjI)
  show (0 :: nat) < 1 by presburger
qed simp-all

```

```

lemma trail-less-id-comp [L2, L1, L0] L1 (− L1)
  unfolding trail-less-id-comp-def
proof (intro exI conjI)
  show (1 :: nat) ≤ 1 by presburger
qed simp-all

```

```

lemma trail-less-comp-id [L2, L1, L0] (− L0) L1
  unfolding trail-less-comp-id-def
proof (intro exI conjI)
  show (1 :: nat) < 2 by presburger
qed simp-all

```

```

lemma trail-less-id-comp [L2, L1, L0] L0 (− L0)
  unfolding trail-less-id-comp-def
proof (intro exI conjI)
  show (2 :: nat) ≤ 2 by presburger
qed simp-all

```

```

lemma trail-less-id-id [L2, L1, L0] L1 L2
  unfolding trail-less-id-id-def
proof (intro exI conjI)
  show (0 :: nat) < 1 by presburger
qed simp-all

```

```

lemma trail-less-id-id [L2, L1, L0] L0 L1
  unfolding trail-less-id-id-def
proof (intro exI conjI)
  show (1 :: nat) < 2 by presburger
qed simp-all

lemma trail-less-comp-comp [L2, L1, L0] (– L1) (– L2)
  unfolding trail-less-comp-comp-def
proof (intro exI conjI)
  show (0 :: nat) < 1 by presburger
qed simp-all

lemma trail-less-comp-comp [L2, L1, L0] (– L0) (– L1)
  unfolding trail-less-comp-comp-def
proof (intro exI conjI)
  show (1 :: nat) < 2 by presburger
qed simp-all

end

```

9.2 Miscellaneous Lemmas

```

lemma not-trail-less-Nil: ¬ trail-less [] L K
  unfolding trail-less-def trail-less-id-id-def trail-less-comp-id-def
  trail-less-id-comp-def trail-less-comp-comp-def
  by simp

lemma defined-if-trail-less:
  assumes trail-less Ls L K
  shows  $L \in \text{set } Ls \cup \text{uminus } 'set \ Ls$   $K \in \text{set } Ls \cup \text{uminus } 'set \ Ls$ 
  apply (atomize (full))
  using assms unfolding trail-less-def trail-less-id-id-def trail-less-comp-id-def
  trail-less-id-comp-def trail-less-comp-comp-def
  by (elim disjE exE conjE) simp-all

lemma not-less-if-undefined:
  fixes  $L :: 'a :: \text{uminus}$ 
  assumes
    uminus-uminus-id:  $\bigwedge x :: 'a. - (- x) = x$  and
     $L \notin \text{set } Ls - L \notin \text{set } Ls$ 
  shows ¬ trail-less Ls L K ¬ trail-less Ls K L
  using assms
  unfolding trail-less-def trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def
  trail-less-comp-comp-def
  by auto

lemma defined-conv:
  fixes  $L :: 'a :: \text{uminus}$ 

```

assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $L \in \text{set } Ls \cup \text{uminus } ' \text{set } Ls \longleftrightarrow L \in \text{set } Ls \vee - L \in \text{set } Ls$
by (*auto simp: rev-image-eqI uminus-uminus-id*)

lemma *trail-less-comp-rightI*: $L \in \text{set } Ls \implies \text{trail-less } Ls L (- L)$
by (*metis in-set-conv-nth le-eq-less-or-eq trail-less-def trail-less-id-comp-def*)

lemma *trail-less-comp-leftI*:
fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $- L \in \text{set } Ls \implies \text{trail-less } Ls (- L) L$
by (*rule trail-less-comp-rightI[of - L, unfolded uminus-uminus-id]*)

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
 $\neg \text{trail-less-id-comp } Ls L K$
 $\neg \text{trail-less-comp-id } Ls L K$
 $\neg \text{trail-less-comp-comp } Ls L K$
using *L-le-K*
unfolding *trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def*
trail-less-comp-comp-def
using *pairwise-distinct[rule-format, OF nth-mem nth-mem]*
by *metis+*

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
 $\neg \text{trail-less-id-id } Ls L K$
 $\neg \text{trail-less-comp-id } Ls L K$
 $\neg \text{trail-less-comp-comp } Ls L K$
using *L-le-K*
unfolding *trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def*
trail-less-comp-comp-def
using *pairwise-distinct[rule-format, OF nth-mem nth-mem]*
by *metis+*

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

```

  ¬ trail-less-id-comp Ls L K
  ¬ trail-less-comp-comp Ls L K
using L-le-K
unfolding trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def
  trail-less-comp-comp-def
using pairwise-distinct[rule-format, OF nth-mem nth-mem]
by metis+

```

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
  ¬ trail-less-id-id Ls L K
  ¬ trail-less-id-comp Ls L K
  ¬ trail-less-comp-id Ls L K
using L-le-K
unfolding trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def
  trail-less-comp-comp-def
using pairwise-distinct[rule-format, OF nth-mem nth-mem]
by metis+

```

9.4 Strict Partial Order

lemma *irreflp-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 irreflp (trail-less Ls)
proof (rule irreflpI, rule notI)
fix L :: 'a
assume trail-less Ls L L
then show False
  unfolding trail-less-def
proof (elim disjE)
  show trail-less-id-id Ls L L  $\implies$  False
  unfolding trail-less-id-id-def
  using pairwise-distinct by fastforce
next
  show trail-less-comp-id Ls L L  $\implies$  False
  unfolding trail-less-comp-id-def
  by (metis pairwise-distinct uminus-not-id)
next
  show trail-less-id-comp Ls L L  $\implies$  False
  unfolding trail-less-id-comp-def

```

```

    by (metis pairwise-distinct uminus-not-id)
  next
  show trail-less-comp-comp Ls L L  $\implies$  False
    unfolding trail-less-comp-comp-def
    by (metis pairwise-distinct uminus-uminus-id nat-less-le)
  qed
qed

```

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:
     $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \implies Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$ 
j)
shows transp (trail-less Ls)
proof (rule transpI)
fix L K H :: 'a
show trail-less Ls L K  $\implies$  trail-less Ls K H  $\implies$  trail-less Ls L H
  using pairwise-distinct[rule-format] uminus-not-id uminus-uminus-id
  unfolding trail-less-def trail-less-id-id-def trail-less-comp-id-def
  trail-less-id-comp-def trail-less-comp-comp-def

  by (smt (verit, best) le-eq-less-or-eq order.strict-trans)
qed

```

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:
     $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \implies Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$ 
j)
shows asympt (trail-less Ls)
using irreflp-trail-less[OF assms] transp-trail-less[OF assms]
using asympt-on-iff-irreflp-on-if-transp-on
by auto

```

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 (rule totalp-onI, unfold Un-iff, elim disjE)
fix L K
assume L  $\in$  set Ls and K  $\in$  set Ls and L  $\neq$  K
then obtain i j where i < length Ls Ls ! i = L j < length Ls Ls ! j = K
  unfolding in-set-conv-nth by auto

```

```

thus trail-less  $Ls$   $L$   $K \vee$  trail-less  $Ls$   $K$   $L$ 
  using  $\langle L \neq K \rangle$  less-linear[of  $i$   $j$ ]
  by (meson trail-less-def trail-less-id-id-def)
next
fix  $L$   $K$ 
assume  $L \in$  set  $Ls$  and  $K \in$  uminus ' set  $Ls$  and  $L \neq K$ 
then obtain  $i$   $j$  where  $i <$  length  $Ls$   $Ls ! i = L$   $j <$  length  $Ls - (Ls ! j) = K$ 
  unfolding in-set-conv-nth image-set length-map by auto
thus trail-less  $Ls$   $L$   $K \vee$  trail-less  $Ls$   $K$   $L$ 
  using less-linear[of  $i$   $j$ ]
  by (metis le-eq-less-or-eq trail-less-comp-id-def trail-less-def trail-less-id-comp-def)
next
fix  $L$   $K$ 
assume  $L \in$  uminus ' set  $Ls$  and  $K \in$  set  $Ls$  and  $L \neq K$ 
then obtain  $i$   $j$  where  $i <$  length  $Ls - (Ls ! i) = L$   $j <$  length  $Ls$   $Ls ! j = K$ 
  unfolding in-set-conv-nth image-set length-map by auto
thus trail-less  $Ls$   $L$   $K \vee$  trail-less  $Ls$   $K$   $L$ 
  using less-linear[of  $i$   $j$ ]
  by (metis le-eq-less-or-eq trail-less-comp-id-def trail-less-def trail-less-id-comp-def)
next
fix  $L$   $K$ 
assume  $L \in$  uminus ' set  $Ls$  and  $K \in$  uminus ' set  $Ls$  and  $L \neq K$ 
then obtain  $i$   $j$  where  $i <$  length  $Ls - (Ls ! i) = L$   $j <$  length  $Ls - (Ls ! j) = K$ 
  unfolding in-set-conv-nth image-set length-map by auto
thus trail-less  $Ls$   $L$   $K \vee$  trail-less  $Ls$   $K$   $L$ 
  using  $\langle L \neq K \rangle$  less-linear[of  $i$   $j$ ]
  by (metis trail-less-comp-comp-def trail-less-def)
qed

```

9.6 Well-Founded

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

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:

$\forall i <$ *length* $(L \# Ls). \forall j <$ *length* $(L \# Ls). i \neq j \longrightarrow$

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

shows \neg *trail-less* $(L \# Ls)$ $(- L)$ L

proof (*rule* *notI*, *unfold* *trail-less-def*, *elim* *disjE*)

show *trail-less-id-id* $(L \# Ls)$ $(- L)$ $L \implies$ *False*

unfolding *trail-less-id-id-def*

using *pairwise-distinct* *uminus-not-id* **by** *metis*

next

show *trail-less-comp-id* $(L \# Ls)$ $(- L)$ $L \implies$ *False*

unfolding *trail-less-comp-id-def*

using *pairwise-distinct* *uminus-uminus-id*

```

    by (metis less-not-refl)
next
show trail-less-id-comp (L # Ls) (- L) L  $\implies$  False
  unfolding trail-less-id-comp-def
  using pairwise-distinct uminus-not-id
  by (metis length-pos-if-in-set nth-Cons-0 nth-mem)
next
show trail-less-comp-comp (L # Ls) (- L) L  $\implies$  False
  unfolding trail-less-comp-comp-def
  using pairwise-distinct uminus-not-id uminus-uminus-id by metis
qed

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  $\neg \text{trail-less } Ls L K \neg \text{trail-less } Ls K L$ 
  using undefined[unfolded in-set-conv-nth] uminus-uminus-id
  unfolding trail-less-def trail-less-id-id-def trail-less-comp-id-def
  trail-less-id-comp-def trail-less-comp-comp-def
  by (smt (verit))+

lemma trail-less-ConsD:
  fixes L H K :: 'a :: 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:  $\text{trail-less } (L \# Ls) H K$ 
  shows  $\text{trail-less } Ls H K$ 
  using less-Cons[unfolded trail-less-def]
proof (elim disjE)
  assume trail-less-id-id (L # Ls) H K
  hence trail-less-id-id Ls H K
    unfolding trail-less-id-id-def
    using L-neq-K less-Suc-eq-0-disj by fastforce
  thus ?thesis
    unfolding trail-less-def by simp
next
  assume trail-less-comp-id (L # Ls) H K
  hence trail-less-comp-id Ls H K
    unfolding trail-less-comp-id-def
    using L-neq-K less-Suc-eq-0-disj by fastforce
  thus ?thesis
    unfolding trail-less-def by simp
next
  assume trail-less-id-comp (L # Ls) H K
  hence trail-less-id-comp Ls H K
    unfolding trail-less-id-comp-def
    using L-neq-minus-K uminus-uminus-id less-Suc-eq-0-disj by fastforce

```

```

thus ?thesis
  unfolding trail-less-def by simp
next
assume trail-less-comp-comp (L # Ls) H K
hence trail-less-comp-comp Ls H K
  unfolding trail-less-comp-comp-def
  using L-neq-minus-K uminus-uminus-id less-Suc-eq-0-disj by fastforce
thus ?thesis
  unfolding trail-less-def by simp
qed

lemma trail-subset-empty-or-ex-smallest:
  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  $Q \subseteq \text{set } Ls \cup \text{uminus } \text{'set } Ls \implies Q = \{\} \vee (\exists z \in Q. \forall y. \text{trail-less } Ls y \longrightarrow y \notin Q)$ 
  using pairwise-distinct
proof (induction Ls arbitrary: Q)
  case Nil
  thus ?case by simp
next
  case Cons-ind: (Cons L Ls)
  from Cons-ind.prem1 have pairwise-distinct-L-Ls:
     $\forall i < \text{length } (L \# Ls). \forall j < \text{length } (L \# Ls). i \neq j \longrightarrow (L \# Ls) ! i \neq (L \# Ls) ! j \wedge (L \# Ls) ! i \neq - (L \# Ls) ! j$ 
    by simp
  hence pairwise-distinct-Ls:
     $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$ 
  by (metis distinct.simps(2) distinct-conv-nth length-Cons not-less-eq nth-Cons-Suc)
  show ?case
  proof (cases Q = {})
  case True
  thus ?thesis by simp
  next
  case Q-neq-empty: False
  have Q-minus-subset:  $Q - \{L, - L\} \subseteq \text{set } Ls \cup \text{uminus } \text{'set } Ls$  using Cons-ind.prem1 by auto

  have irreflp-gt-L-Ls: irreflp (trail-less (L # Ls))
  by (rule irreflp-trail-less[OF uminus-not-id uminus-uminus-id pairwise-distinct-L-Ls])

  have  $\exists z \in Q. \forall y. \text{trail-less } (L \# Ls) y z \longrightarrow y \notin Q$ 
  using Cons-ind.IH[OF Q-minus-subset pairwise-distinct-Ls]
  proof (elim disjE bexE)

```

```

assume  $Q - \{L, -L\} = \{\}$ 
with  $Q$ -neq-empty have  $Q \subseteq \{L, -L\}$  by simp
have ?thesis if  $L \in Q$ 
  apply (intro bexI[OF -  $\langle L \in Q \rangle$ ] allI impI)
  apply (erule contrapos-pn)
  apply (drule set-rev-mp[OF -  $\langle Q \subseteq \{L, -L\} \rangle$ ])
  apply simp
  using irreflp-gt-L-Ls[THEN irreflpD, of L]
  using not-trail-less-Cons-id-comp[OF uminus-not-id uminus-uminus-id
    pairwise-distinct-L-Ls]
  by fastforce
moreover have ?thesis if  $L \notin Q$ 
proof -
  from  $\langle L \notin Q \rangle$  have  $Q = \{-L\}$ 
  using  $Q$ -neq-empty  $\langle Q \subseteq \{L, -L\} \rangle$  by auto
  thus ?thesis
  using irreflp-gt-L-Ls[THEN irreflpD, of - L] by auto
qed
ultimately show ?thesis by metis
next
fix K
assume  $K$ -in- $Q$ -minus:  $K \in Q - \{L, -L\}$  and  $\forall y. \text{trail-less } Ls\ y\ K \longrightarrow y \notin Q - \{L, -L\}$ 
from  $K$ -in- $Q$ -minus have  $L \neq K - L \neq K$  by auto
from  $K$ -in- $Q$ -minus have  $L \neq -K$  using  $\langle -L \neq K \rangle$  uminus-uminus-id by
blast
show ?thesis
proof (intro bexI allI impI)
  show  $K \in Q$ 
  using  $K$ -in- $Q$ -minus by simp
next
fix H
assume trail-less (L # Ls) H K
hence trail-less Ls H K
  by (rule trail-less-ConsD[OF uminus-uminus-id  $\langle L \neq K \rangle$   $\langle L \neq -K \rangle$ ])
hence  $H \notin Q - \{L, -L\}$ 
  using  $\langle \forall y. \text{trail-less } Ls\ y\ K \longrightarrow y \notin Q - \{L, -L\} \rangle$  by simp
moreover have  $H \neq L \wedge H \neq -L$ 
  using uminus-uminus-id pairwise-distinct-L-Ls  $\langle \text{trail-less } Ls\ H\ K \rangle$ 
by (metis (no-types, lifting) distinct.simps(2) distinct-conv-nth in-set-conv-nth
  list.set-intros(1,2) not-trail-less-if-undefined(1))
ultimately show  $H \notin Q$ 
  by simp
qed
qed
thus ?thesis by simp
qed
qed

```

```

lemma wfP-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 wfP (trail-less Ls)
  unfolding wfp-eq-minimal
  proof (intro allI impI)
  fix M :: 'a set and L :: 'a
  assume  $L \in M$ 
  show  $\exists z \in M. \forall y. \text{trail-less } Ls \ y \ z \longrightarrow y \notin M$ 
  proof (cases  $M \cap (\text{set } Ls \cup \text{uminus } ' \text{set } Ls) = \{\}$ )
    case True
      with  $\langle L \in M \rangle$  have L-not-in-Ls:  $L \notin \text{set } Ls \wedge - L \notin \text{set } Ls$ 
      unfolding disjoint-iff by (metis UnCI image-eqI uminus-uminus-id)
      then show ?thesis
      proof (intro bexI[OF -  $\langle L \in M \rangle$ ] allI impI)
        fix K
        assume trail-less Ls K L
        hence False
        using L-not-in-Ls not-trail-less-if-undefined[OF - - uminus-uminus-id] by
simp
        thus  $K \notin M ..$ 
      qed
    next
      case False
      hence  $M \cap (\text{set } Ls \cup \text{uminus } ' \text{set } Ls) \subseteq \text{set } Ls \cup \text{uminus } ' \text{set } Ls$ 
      by simp
      with False obtain H where
        H-in:  $H \in M \cap (\text{set } Ls \cup \text{uminus } ' \text{set } Ls)$  and
        all-lt-H-no-in:  $\forall y. \text{trail-less } Ls \ y \ H \longrightarrow y \notin M \cap (\text{set } Ls \cup \text{uminus } ' \text{set } Ls)$ 
        using trail-subset-empty-or-ex-smallest[OF uminus-not-id uminus-uminus-id
pairwise-distinct]
        by meson
        show ?thesis
        proof (rule bexI)
          show  $H \in M$  using H-in by simp
        next
          show  $\forall y. \text{trail-less } Ls \ y \ H \longrightarrow y \notin M$ 
          using all-lt-H-no-in uminus-uminus-id
          by (metis Int-iff Un-iff image-eqI not-trail-less-if-undefined(1))
        qed
      qed
    qed

```

9.7 Extension on All Literals

definition *trail-less-ex* where

```

trail-less-ex lt Ls L K  $\longleftrightarrow$ 
  (if L  $\in$  set Ls  $\vee$   $\neg$  L  $\in$  set Ls then
    if K  $\in$  set Ls  $\vee$   $\neg$  K  $\in$  set Ls then
      trail-less Ls L K
    else
      True
  else
    if K  $\in$  set Ls  $\vee$   $\neg$  K  $\in$  set Ls then
      False
    else
      lt L K)

```

lemma

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

assumes

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

shows $K \in \text{set } Ls \vee \neg K \in \text{set } Ls \implies \text{trail-less-ex } lt \ Ls \ L \ K \longleftrightarrow \text{trail-less } Ls \ L \ K$

using *not-less-if-undefined*[OF *uminus-uminus-id*]

by (*simp add: trail-less-ex-def*)

lemma *trail-less-ex-if-trail-less*:

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

assumes

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

shows $\text{trail-less } Ls \ L \ K \implies \text{trail-less-ex } lt \ Ls \ L \ K$

unfolding *trail-less-ex-def*

using *defined-if-trail-less*[THEN *defined-conv*[OF *uminus-uminus-id*, THEN *iffD1*]]

by *auto*

lemma

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

assumes

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

shows $L \in \text{set } Ls \cup \text{uminus } ' \text{set } Ls \implies K \notin \text{set } Ls \cup \text{uminus } ' \text{set } Ls \implies \text{trail-less-ex } lt \ Ls \ L \ K$

using *defined-conv uminus-uminus-id*

by (*auto simp add: trail-less-ex-def*)

lemma *irreflp-trail-ex-less*:

fixes Ls :: ('a :: uminus) list **and** lt :: 'a \Rightarrow 'a \Rightarrow bool

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**
irreflp-lt: *irreflp lt*
shows *irreflp (trail-less-ex lt Ls)*
unfolding *trail-less-ex-def*
using *irreflp-trail-less[OF uminus-not-id uminus-uminus-id pairwise-distinct] irreflp-lt*
by (*simp add: irreflpD irreflpI*)

lemma *transp-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**
transp-lt: *transp lt*
shows *transp (trail-less-ex lt Ls)*
unfolding *trail-less-ex-def*
using *transp-trail-less[OF uminus-not-id uminus-uminus-id pairwise-distinct] transp-lt*
by (*smt (verit, ccfv-SIG) transp-def*)

lemma *asympt-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**
asympt-lt: *asympt lt*
shows *asympt (trail-less-ex lt Ls)*
unfolding *trail-less-ex-def*
using *asympt-trail-less[OF uminus-not-id uminus-uminus-id pairwise-distinct] asympt-lt*
by (*auto intro: asymptI dest: asymptD*)

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)*
using *totalp-on-trail-less[of Ls]*
using *totalp-on-lt*
unfolding *trail-less-ex-def*
by (*smt (verit, best) Un-iff defined-conv totalp-on-def uminus-uminus-id*)

9.7.1 Well-Founded

lemma *wfP-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**

wfP-lt: $\text{wfP } lt$

shows $\text{wfP } (\text{trail-less-ex } lt \ Ls)$

unfolding *wfp-eq-minimal*

proof (*intro allI impI*)

fix $Q :: 'a \text{ set}$ **and** $x :: 'a$

assume $x \in Q$

show $\exists z \in Q. \forall y. \text{trail-less-ex } lt \ Ls \ y \ z \longrightarrow y \notin Q$

proof (*cases* $Q \cap (\text{set } Ls \cup \text{uminus } ' \text{ set } Ls) = \{\}$)

case *True*

then show *?thesis*

using *wfP-lt[unfolding wfp-eq-minimal, rule-format, OF $\langle x \in Q \rangle$]*

by (*metis (no-types, lifting) defined-conv disjoint-iff trail-less-ex-def uminus-uminus-id*)

next

case *False*

then show *?thesis*

using *trail-subset-empty-or-ex-smallest[OF uminus-not-id uminus-uminus-id pairwise-distinct,*

unfolding wfp-eq-minimal, of $Q \cap (\text{set } Ls \cup \text{uminus } ' \text{ set } Ls)$, simplified]

by (*metis (no-types, lifting) IntD1 IntD2 UnE defined-conv trail-less-ex-def uminus-uminus-id*)

qed

qed

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)

by (*rule transpI*)

(*smt (verit, ccfv-SIG) Suc-lessD assms less-trans-Suc nth-eq-iff-index-eq trail-term-less-def*)

lemma *asympt-trail-term-less*:

assumes *distinct* ts

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

by (*rule asymptI*)

(*metis assms distinct-Ex1 dual-order.strict-trans nth-mem order-less-imp-not-less*)

trail-term-less-def)

lemma *irreflp-trail-term-less*:

assumes *distinct ts*

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

using *assms irreflp-on-if-asymp-on[OF asymp-trail-term-less]* **by** *metis*

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

shows *totalp-on (set ts) (trail-term-less ts)*

by (*rule totalp-onI*) (*metis in-set-conv-nth nat-neq-iff trail-term-less-def*)

lemma *wfP-trail-term-less*:

assumes *distinct ts*

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

proof (*rule wfP-if-convertible-to-nat*)

fix *t1 t2* **assume** *trail-term-less ts t1 t2*

then obtain *i j* **where** *i < length ts* **and** *j < i* **and** *t1 = ts ! i* **and** *t2 = ts ! j*

unfolding *trail-term-less-def* **by** *auto*

then show *index (rev ts) t1 < index (rev ts) t2*

using *assms diff-commute index-nth-id index-rev* **by** *fastforce*

qed

lemma *trail-term-less-Cons-if-mem*:

assumes *y ∈ set xs*

shows *trail-term-less (x # xs) y x*

proof –

from *assms* **obtain** *i* **where** *i < length xs* **and** *xs ! i = y*

by (*meson in-set-conv-nth*)

thus *?thesis*

unfolding *trail-term-less-def*

proof (*intro exI conjI*)

show *Suc i < length (x # xs)*

using *⟨i < length xs⟩* **by** *simp*

next

show *0 < Suc i*

by *simp*

next

show *y = (x # xs) ! Suc i*

using *⟨xs ! i = y⟩* **by** *simp*

next

show *x = (x # xs) ! 0*

by *simp*

qed

qed

end

theory *Initial-Literals-Generalize-Learned-Literals*

imports *SCL-FOL*

begin

global-interpretation *comp-finsert-commute*: *comp-fun-commute finsert*

proof (*unfold-locales*)

show $\bigwedge y x. \text{finsert } y \circ \text{finsert } x = \text{finsert } x \circ \text{finsert } y$

by *auto*

qed

definition *fset-mset* :: 'a *multiset* \Rightarrow 'a *fset*

where *fset-mset* = *fold-mset finsert {||}*

lemma *fset-mset-empty[simp]*: *fset-mset* {#} = {||}

by (*simp add: fset-mset-def*)

lemma *fset-mset-add-mset[simp]*: *fset-mset* (*add-mset* *x* *M*) = *finsert* *x* (*fset-mset* *M*)

by (*simp add: fset-mset-def*)

lemma *fset-fset-mset[simp]*: *fset* (*fset-mset* *M*) = *set-mset* *M*

by (*induction* *M* *rule: multiset-induct*) *simp-all*

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

by (*induction* *M* *rule: multiset-induct*) *simp-all*

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

by *simp*

lemma *fBex-fset-mset-iff[simp]*: $(\exists x \in | \text{fset-mset } M. P x) \iff (\exists x \in \# M. P x)$

by *simp*

lemma *fmember-ffUnion-iff*: $a \in | \text{ffUnion } (f \ |' A) \iff (\exists x \in | A. a \in | f x)$

unfolding *ffUnion.rep-eq* **by** *simp*

lemma *fBex-ffUnion-iff*: $(\exists z \in | \text{ffUnion } (f \ |' A). P z) \iff (\exists x \in | A. \exists z \in | f x. P z)$

unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

lemma *fBall-ffUnion-iff*: $(\forall z \in | \text{ffUnion } (f \ |' A). P z) \iff (\forall x \in | A. \forall z \in | f x. P z)$

unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

abbreviation *grounding-lits-of-cls* **where**

grounding-lits-of-cls *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* (*fset* (*state-learned* *S*)) \subseteq *grounding-lits-of-clss* (*fset* *N*)
 (is ?lhs \subseteq ?rhs)
proof (*rule subsetI*)
from *assms*(1) **have** *N-lits-sup: clss-lits-generalize-clss-lits* (*fset* *N*) (*fset* (*state-learned* *S*))
unfolding *initial-lits-generalize-learned-trail-conflict-def*
using *clss-lits-generalize-clss-lits-subset* **by** *auto*

fix *L*
assume $L \in ?lhs$
then obtain $L' \gamma$ **where**
L-def: $L = L' \cdot l \gamma$ **and**
 $L' \in \bigcup (\text{set-mset } 'fset \text{ (state-learned } S))$ **and**
is-ground-lit ($L' \cdot l \gamma$)
by *auto*
then obtain $L_N \sigma_N$ **where** $L_N \in \bigcup (\text{set-mset } 'fset \text{ } N)$ **and** $L_N \cdot l \sigma_N = L'$
using *N-lits-sup[unfolded clss-lits-generalize-clss-lits-def]*
unfolding *fBex-ffUnion-iff fBall-ffUnion-iff fBex-fset-mset-iff fBall-fset-mset-iff generalizes-lit-def* **by** *meson*
then show $L \in ?rhs$
unfolding *mem-Collect-eq*
using $\langle \text{is-ground-lit } (L' \cdot l \gamma) \rangle$
unfolding *L-def* $\langle L_N \cdot l \sigma_N = L' \rangle$ [*symmetric*]
by (*metis subst-lit-comp-subst*)

qed

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 (*intro Set.equalityI Set.subsetI*)
fix *L*
assume $L \in ?lhs$
then obtain $L' \gamma$ **where** $L = L' \cdot l \gamma$ **and** $L' \in \bigcup (\text{set-mset } 'N)$ **and** *is-ground-lit* ($L' \cdot l \gamma$)
by *auto*

from $\langle L' \in \bigcup (\text{set-mset } 'N) \rangle$ **obtain** *C* **where** $C \in N$ **and** $L' \in \# C$
by *blast*

obtain γ_C **where** *is-ground-cls* ($C \cdot \gamma \cdot \gamma_C$)
using *ex-ground-subst ground-subst-ground-cls* **by** *blast*
hence $L \in \# C \cdot \gamma \cdot \gamma_C$
using $\langle L' \in \# C \rangle \langle L = L' \cdot l \gamma \rangle$
by (*metis Melem-subst-cls* $\langle \text{is-ground-lit } (L' \cdot l \gamma) \rangle$ *is-ground-subst-lit*)
then obtain C' **where** $C \cdot \gamma \cdot \gamma_C = \text{add-mset } L \ C'$
using *multi-member-split* **by** *metis*

moreover have $C \cdot \gamma \cdot \gamma_C \in \text{grounding-of-clss } N$

```

    unfolding grounding-of-clss-def
  proof (rule UN-I)
    show  $C \in N$ 
      using  $\langle C \in N \rangle$  .
  next
    show  $C \cdot \gamma \cdot \gamma_C \in \text{grounding-of-clss } C$ 
      using  $\langle \text{is-ground-clss } (C \cdot \gamma \cdot \gamma_C) \rangle$ 
      by (metis grounding-of-clss-ground grounding-of-subst-clss-subset insert-absorb
insert-subset)
    qed

  ultimately show  $L \in ?rhs$ 
    by auto
  next
  fix  $L$ 
  assume  $L \in ?rhs$ 
  then obtain  $C$  where  $\text{add-mset } L \ C \in \text{grounding-of-clss } N$ 
    by auto
  then obtain  $CC \ \gamma$  where  $CC \in N$  and  $CC \cdot \gamma = \text{add-mset } L \ C$ 
    unfolding grounding-of-clss-def
    by (smt (verit, best) UN-iff grounding-of-clss-def mem-Collect-eq)
  then obtain  $L' \ C'$  where  $CC = \text{add-mset } L' \ C'$  and  $L = L' \cdot l \ \gamma$  and  $C = C'$ 
    by (metis (no-types, lifting) mset-map-invr subst-clss-def)

  show  $L \in ?lhs$ 
  proof (intro CollectI exI conjI)
    show  $L = L' \cdot l \ \gamma$ 
      using  $\langle L = L' \cdot l \ \gamma \rangle$  by simp
  next
  show  $L' \in \bigcup (\text{set-mset } 'N)$ 
    using  $\langle CC \in N \rangle \langle CC = \text{add-mset } L' \ C' \rangle$ 
    by (metis Union-iff image-eqI union-single-eq-member)
  next
  show  $\text{is-ground-lit } (L' \cdot l \ \gamma)$ 
    using  $\langle \text{add-mset } L \ C \in \text{grounding-of-clss } N \rangle \langle L = L' \cdot l \ \gamma \rangle$ 
    by (metis grounding-ground is-ground-clss-add-mset)
  qed
qed

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  $\{L \mid L \ C. \ \text{add-mset } L \ C \in \text{grounding-of-clss } (\text{fset } U)\} \subseteq$ 
     $\{L \mid L \ C. \ \text{add-mset } L \ C \in \text{grounding-of-clss } (\text{fset } N)\}$ 
  using assms grounding-lits-of-learned-subset-grounding-lits-of-initial
  unfolding grounding-lits-of-clss-conv
  by simp

```

```

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*:
 $reasonable-scl\ N\ \beta\ S\ S' \implies sound-state\ N\ \beta\ S \implies sound-state\ N\ \beta\ S'$
using *scl-preserves-sound-state reasonable-scl-def* **by** *blast*

lemma *reasonable-run-sound-state*:
 $(reasonable-scl\ N\ \beta)^{**}\ S\ S' \implies sound-state\ N\ \beta\ S \implies sound-state\ N\ \beta\ S'$
by (*smt (verit, best) reasonable-scl-sound-state rtranclp-induct*)

10.1 Invariants

10.1.1 No Conflict After Decide

inductive *no-conflict-after-decide* **for** $N\ \beta\ U$ **where**
 $Nil[simp]: no-conflict-after-decide\ N\ \beta\ U\ []\ |$
 $Cons: (is-decision-lit\ Ln \longrightarrow (\nexists S'. conflict\ N\ \beta\ (Ln\ \# \Gamma, U, None)\ S')) \implies$
 $no-conflict-after-decide\ N\ \beta\ U\ \Gamma \implies no-conflict-after-decide\ N\ \beta\ U\ (Ln\ \# \Gamma)$

definition *no-conflict-after-decide'* **where**
 $no-conflict-after-decide'\ N\ \beta\ S = no-conflict-after-decide\ N\ \beta\ (state-learned\ S)$
 $(state-trail\ S)$

lemma *no-conflict-after-decide'-initial-state[simp]*: $no-conflict-after-decide'\ N\ \beta\ initial-state$
by (*simp add: no-conflict-after-decide'-def no-conflict-after-decide.Nil*)

lemma *propagate-preserves-no-conflict-after-decide'*:
assumes $propagate\ N\ \beta\ S\ S'$ **and** $no-conflict-after-decide'\ N\ \beta\ S$
shows $no-conflict-after-decide'\ N\ \beta\ S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def propagate-lit-def is-decision-lit-def elim!: propagate.cases intro!: no-conflict-after-decide.Cons*)

lemma *decide-preserves-no-conflict-after-decide'*:
assumes $decide\ N\ \beta\ S\ S'$ **and** $\nexists S''. conflict\ N\ \beta\ S'\ S''$ **and** $no-conflict-after-decide'\ N\ \beta\ S$

shows *no-conflict-after-decide'* $N \beta S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def decide-lit-def is-decision-lit-def*
elim!: decide.cases intro!: no-conflict-after-decide.Cons)

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'* $N \beta S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def elim: conflict.cases*)

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'* $N \beta S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def*
elim!: skip.cases elim: no-conflict-after-decide.cases)

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'* $N \beta S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def elim: factorize.cases*)

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'* $N \beta S'$
using *assms*
by (*auto simp: no-conflict-after-decide'-def elim: resolve.cases*)

lemma *learning-clause-without-conflict-preserves-nex-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 $\nexists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S' \implies \nexists S'. \text{conflict } N \beta (\Gamma, \text{finsert } C U, \text{None}) S'$
proof (*elim contrapos-nn exE*)
fix S'
assume *conflict* $N \beta (\Gamma, \text{finsert } C U, \text{None} :: ('f, 'v) \text{closure option}) S'$
then show $\exists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S'$
proof (*cases* $N \beta (\Gamma, \text{finsert } C U, \text{None} :: ('f, 'v) \text{closure option}) S'$ *rule: conflict.cases*)
case (*conflictI* $D \gamma$)
then show *?thesis*
using *assms conflict.intros* **by** *blast*
qed
qed

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'  $N \beta S'$ 
using step
proof (cases  $N \beta S S'$  rule: backtrack.cases)
case (backtrackI  $\Gamma \Gamma' \Gamma'' K L \sigma D U$ )
have no-conflict-after-decide  $N \beta U (\Gamma' @ \Gamma'')$ 
using invar
unfolding backtrackI(1,2,3) no-conflict-after-decide'-def
by (auto simp: decide-lit-def elim: no-conflict-after-decide.cases)
hence no-conflict-after-decide  $N \beta U \Gamma''$ 
by (induction  $\Gamma'$ ) (auto elim: no-conflict-after-decide.cases)
hence no-conflict-after-decide  $N \beta (\text{finsert } (\text{add-mset } L D) U) \Gamma''$ 
using backtrackI(5)
proof (induction  $\Gamma''$ )
case Nil
show ?case
by (auto intro: no-conflict-after-decide.Nil)
next
case (Cons  $Ln \Gamma''$ )
hence  $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } (Ln \# \Gamma'') (\text{add-mset } L D \cdot \gamma)$ 
by metis
hence  $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } \Gamma'' (\text{add-mset } L D \cdot \gamma)$ 
by (metis (no-types, opaque-lifting) image-insert insert-iff list.set(2) trail-false-cls-def trail-false-lit-def)
hence 1: no-conflict-after-decide  $N \beta (\text{finsert } (\text{add-mset } L D) U) \Gamma''$ 
by (rule Cons.IH)

show ?case
proof (intro no-conflict-after-decide.Cons impI)
assume is-decision-lit  $Ln$ 
with Cons.hyps have  $\nexists S'. \text{conflict } N \beta (Ln \# \Gamma'', U, \text{None}) S'$ 
by simp
then show  $\nexists S'. \text{conflict } N \beta (Ln \# \Gamma'', \text{finsert } (\text{add-mset } L D) U, \text{None}) S'$ 
using learning-clause-without-conflict-preserves-nex-conflict
using  $\langle \nexists \gamma. \text{is-ground-cls } (\text{add-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } (Ln \# \Gamma'') (\text{add-mset } L D \cdot \gamma) \rangle$ 
by blast
next
show no-conflict-after-decide  $N \beta (\text{finsert } (\text{add-mset } L D) U) \Gamma''$ 
using 1 .
qed
qed
thus ?thesis
unfolding backtrackI(1,2) no-conflict-after-decide'-def by simp
qed

```

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'* $N \beta S'$
using *assms* **unfolding** *reasonable-scl-def scl-def*
using *propagate-preserves-no-conflict-after-decide'* *decide-preserves-no-conflict-after-decide'*
conflict-preserves-no-conflict-after-decide' *skip-preserves-no-conflict-after-decide'*
factorize-preserves-no-conflict-after-decide' *resolve-preserves-no-conflict-after-decide'*
backtrack-preserves-no-conflict-after-decide'
by *metis*

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 $\{\#\} \mid \in \mid N \vee (\exists S0. \text{propagate } N \beta S0 S1)$
using *before-conflict*[*OF conf invars*(1,2)]
proof (*elim disjE exE*)
fix $S0$ **assume** *decide* $N \beta S0 S1$
hence *False*
proof (*cases* $N \beta S0 S1$ *rule*: *decide.cases*)
case (*decideI* $L \gamma \Gamma U$)
with *invars*(3) **have** *no-conflict-after-decide* $N \beta U$ (*trail-decide* $\Gamma (L \cdot l \gamma)$)
by (*simp add*: *no-conflict-after-decide'-def*)
hence $\nexists S'. \text{conflict } N \beta (\text{trail-decide } \Gamma (L \cdot l \gamma), U, \text{None}) S'$
by (*rule no-conflict-after-decide.cases*) (*simp-all add*: *decide-lit-def is-decision-lit-def*)
then show *?thesis*
using *conf unfolding decideI*(1,2) **by** *metis*
qed
thus *?thesis ..*
qed *auto*

11 Regular Steps

lemma *regular-scl-if-conflict*[*simp*]: *conflict* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*simp add*: *regular-scl-def*)

lemma *regular-scl-if-skip*[*simp*]: *skip* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*auto simp*: *regular-scl-def reasonable-scl-def scl-def elim*: *conflict.cases skip.cases*)

lemma *regular-scl-if-factorize*[*simp*]: *factorize* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*auto simp*: *regular-scl-def reasonable-scl-def scl-def elim*: *conflict.cases factorize.cases*)

lemma *regular-scl-if-resolve*[*simp*]: *resolve* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*auto simp*: *regular-scl-def reasonable-scl-def scl-def elim*: *conflict.cases resolve.cases*)

lemma *regular-scl-if-backtrack*[*simp*]: *backtrack* $N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*smt (verit) backtrack.cases decide-well-defined*(6) *option.discI regular-scl-def*)

conflict.simps
reasonable-scl-def scl-def state-conflict-simp)

lemma *regular-scl-sound-state*: $\text{regular-scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
by (*rule reasonable-scl-sound-state*[*OF reasonable-if-regular*])

lemma *regular-run-sound-state*:
 $(\text{regular-scl } N \beta)^{**} S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
by (*smt (verit, best) regular-scl-sound-state rtranclp-induct*)

11.1 Invariants

11.1.1 Almost No Conflict With Trail

inductive *no-conflict-with-trail for* $N \beta U$ **where**
Nil: $(\nexists S'. \text{conflict } N \beta ([], U, \text{None}) S') \implies \text{no-conflict-with-trail } N \beta U [] |$
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 $\nexists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S'$
using *assms* **by** (*auto elim: no-conflict-with-trail.cases*)

lemma *nex-conflict-if-no-conflict-with-trail'*:
assumes *no-conflict-with-trail* $N \beta U \Gamma$
shows $\nexists S'. \text{conflict } N \beta ([], U, \text{None}) S'$
using *assms*
by (*induction* Γ *rule: no-conflict-with-trail.induct*) *simp-all*

lemma *no-conflict-after-decide-if-no-conflict-with-trail*:
 $\text{no-conflict-with-trail } N \beta U \Gamma \implies \text{no-conflict-after-decide } N \beta U \Gamma$
by (*induction* Γ *rule: no-conflict-with-trail.induct*)
(simp-all add: no-conflict-after-decide.Cons)

lemma *not-trail-false-cls-if-no-conflict-with-trail*:
 $\text{no-conflict-with-trail } N \beta U \Gamma \implies D \in |N| \cup |U| \implies D \neq \{\#\} \implies \text{is-ground-cls } (D \cdot \gamma) \implies \neg \text{trail-false-cls } \Gamma (D \cdot \gamma)$

proof (*induction* Γ *rule: no-conflict-with-trail.induct*)

case *Nil*

thus *?case* **by** *simp*

next

case (*Cons* $Ln \Gamma$)

hence $\neg \text{trail-false-cls } (Ln \# \Gamma) (D \cdot \gamma)$

by (*metis fst-conv not-trail-false-ground-cls-if-no-conflict state-conflict-simp state-learned-simp state-trail-def*)

thus *?case*

by *simp*

qed

definition *almost-no-conflict-with-trail* **where**

almost-no-conflict-with-trail $N \beta S \longleftrightarrow$
 $\{\#\} \mid \in \mid N \wedge \text{state-trail } S = [] \vee$
no-conflict-with-trail $N \beta (\text{state-learned } S)$
(*case state-trail* S of $[] \Rightarrow [] \mid L_n \# \Gamma \Rightarrow$ *if is-decision-lit* L_n then $L_n \# \Gamma$ 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

no-conflict-with-trail $N \beta (\text{state-learned } S) (\text{state-trail } S)$

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

proof –

from *no-conf* **obtain** ΓU **where** *S-def*: $S = (\Gamma, U, \text{None})$

by (*metis state-simp*)

from $\langle \text{learned-nonempty } S \rangle$ **have** $\{\#\} \mid \notin \mid U$

by (*simp add: S-def learned-nonempty-def*)

show *?thesis*

using *assms(4)*

unfolding *S-def state-proj-simp*

proof (*cases N beta U Gamma rule: no-conflict-with-trail.cases*)

case *Nil*

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

using $\langle \{\#\} \mid \notin \mid N \rangle \langle \{\#\} \mid \notin \mid U \rangle$

by (*auto simp: trail-false-cls-def elim: conflict.cases*)

next

case (*Cons Ln Gamma'*)

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

by (*auto intro: no-conflict-tail-trail*)

qed

qed

lemma *no-conflict-with-trail-if-nex-conflict*:

assumes *no-conf*: $\nexists S'. \text{conflict } N \beta S S'$ *state-conflict* $S = \text{None}$

shows *no-conflict-with-trail* $N \beta (\text{state-learned } S) (\text{state-trail } S)$

proof –

from *no-conf(2)* **obtain** ΓU **where** *S-def*: $S = (\Gamma, U, \text{None})$

by (*metis state-simp*)

show *?thesis*

using *no-conf(1)*

unfolding *S-def state-proj-simp*

proof (*induction Gamma*)

case *Nil*

thus *?case* **by** (*simp add: no-conflict-with-trail.Nil*)

```

next
  case (Cons Ln Γ)
  have  $\nexists a. \text{conflict } N \beta (\Gamma, U, \text{None}) a$ 
    by (rule no-conflict-tail-trail[OF Cons.prem])
  hence no-conflict-with-trail  $N \beta U \Gamma$ 
    by (rule Cons.IH)
  then show ?case
    using Cons.prem
    by (auto intro: no-conflict-with-trail.Cons)
qed
qed

lemma almost-no-conflict-with-trail-if-no-conflict-with-trail:
  no-conflict-with-trail  $N \beta U \Gamma \implies \text{almost-no-conflict-with-trail } N \beta (\Gamma, U, Cl)$ 
  by (cases Γ) (auto simp: almost-no-conflict-with-trail-def elim: no-conflict-with-trail.cases)

lemma almost-no-conflict-with-trail-initial-state[simp]:
  almost-no-conflict-with-trail  $N \beta \text{initial-state}$ 
  by (cases  $\{\#\} \mid \in \mid N$ ) (auto simp: almost-no-conflict-with-trail-def trail-false-cls-def
    elim!: conflict.cases intro: no-conflict-with-trail.Nil)

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  $N \beta S'$ 
  using reg-step[unfolded regular-scl-def]
proof (elim disjE conjE)
  assume conflict  $N \beta S S'$ 
  with step have False
  using conflict-well-defined by blast
  thus ?thesis ..
next
  assume no-conf:  $\nexists S'. \text{conflict } N \beta S S'$  and reasonable-scl  $N \beta S S'$ 
  from step show ?thesis
proof (cases  $N \beta S S'$  rule: propagate.cases)
  case step-hyps: (propagateI C U L C'  $\gamma C_0 C_1 \Gamma \mu$ )
  have no-conflict-with-trail  $N \beta U \Gamma$ 
    by (rule no-conflict-with-trail-if-nex-conflict[OF no-conf,
      unfolded step-hyps state-proj-simp, OF refl])
  thus ?thesis
  unfolding step-hyps(1,2)
  by (simp add: almost-no-conflict-with-trail-def propagate-lit-def is-decision-lit-def)
qed
qed

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 –
  from reg-step have res-step: reasonable-scl  $N \beta S S'$ 

```

```

    by (rule reasonable-if-regular)

from step obtain  $\Gamma$   $U$  where  $S'$ -def:  $S' = (\Gamma, U, None)$ 
  by (auto elim: decide.cases)

have no-conflict-with-trail  $N$   $\beta$  (state-learned  $S'$ ) (state-trail  $S'$ )
proof (rule no-conflict-with-trail-if-nex-conflict)
  show  $\nexists S''$ . conflict  $N$   $\beta$   $S'$   $S''$ 
    using step res-step[unfolded reasonable-scl-def] by argo
next
  show state-conflict  $S' = None$ 
    by (simp add:  $S'$ -def)
qed
thus ?thesis
  unfolding  $S'$ -def
  by (simp add: almost-no-conflict-with-trail-if-no-conflict-with-trail)
qed

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  $N$   $\beta$  ( $\Gamma$ ,  $U$ ,  $Cl2$ )
  by (simp add: almost-no-conflict-with-trail-def)

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 -
  from step obtain  $\Gamma$   $U$   $Cl$  where  $S = (\Gamma, U, None)$  and  $S' = (\Gamma, U, Some Cl)$ 
  by (auto elim: conflict.cases)
  with invar show ?thesis
    using almost-no-conflict-with-trail-conflict-not-relevant by metis
qed

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  $N$   $\beta$   $S'$ 
  using step
proof (cases  $N$   $\beta$   $S$   $S'$  rule: skip.cases)
  case step-hyps: (skipI  $L$   $D$   $\sigma$   $n$   $\Gamma$   $U$ )
  have no-conflict-with-trail  $N$   $\beta$   $U$  (if is-decision-lit ( $L$ ,  $n$ ) then ( $L$ ,  $n$ )  $\#$   $\Gamma$  else  $\Gamma$ )
  using invar unfolding step-hyps(1,2) by (simp add: almost-no-conflict-with-trail-def)
  hence no-conflict-with-trail  $N$   $\beta$   $U$   $\Gamma$ 
  by (cases is-decision-lit ( $L$ ,  $n$ )) (auto elim: no-conflict-with-trail.cases)
  then show ?thesis
    unfolding step-hyps(1,2)
    by (rule almost-no-conflict-with-trail-if-no-conflict-with-trail)
qed

```

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 –
from *step* **obtain** $\Gamma U Cl1 Cl2$ **where** $S = (\Gamma, U, \text{Some } Cl1)$ **and** $S' = (\Gamma, U, \text{Some } Cl2)$
by (*auto elim: factorize.cases*)
with *invar* **show** *?thesis*
using *almost-no-conflict-with-trail-conflict-not-relevant* **by** *metis*
qed

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 –
from *step* **obtain** $\Gamma U Cl1 Cl2$ **where** $S = (\Gamma, U, \text{Some } Cl1)$ **and** $S' = (\Gamma, U, \text{Some } Cl2)$
by (*auto elim: resolve.cases*)
with *invar* **show** *?thesis*
using *almost-no-conflict-with-trail-conflict-not-relevant* **by** *metis*
qed

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* $N \beta S'$
using *step*
proof (*cases* $N \beta S S'$ *rule: backtrack.cases*)
case *step-hyps: (backtrackI* $\Gamma \Gamma' \Gamma'' K L \sigma D U)$
from *invar* **have** *no-conflict-with-trail* $N \beta U ((- (L \cdot l \sigma), \text{None}) \# \Gamma' @ \Gamma'')$
by (*simp add: step-hyps almost-no-conflict-with-trail-def decide-lit-def is-decision-lit-def*)
hence *no-conflict-with-trail* $N \beta U (\Gamma' @ \Gamma'')$
by (*auto elim: no-conflict-with-trail.cases*)
hence *no-conflict-with-trail* $N \beta U \Gamma''$
by (*induction* Γ') (*auto elim: no-conflict-with-trail.cases*)
then **have** *no-conflict-with-trail* $N \beta (\text{finsert } (\text{add-mset } L D) U) \Gamma''$
by (*metis learning-clause-without-conflict-preserves-nex-conflict nex-conflict-if-no-conflict-with-trail no-conflict-with-trail-if-nex-conflict state-conflict-simp state-learned-simp state-trail-simp step-hyps(5)*)
thus *?thesis*
unfolding *step-hyps(1,2)*
by (*rule almost-no-conflict-with-trail-if-no-conflict-with-trail*)
qed

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* $N \beta S'$
using *assms*
using *propagate-preserves-almost-no-conflict-with-trail decide-preserves-almost-no-conflict-with-trail*

conflict-preserves-almost-no-conflict-with-trail skip-preserves-almost-no-conflict-with-trail
factorize-preserves-almost-no-conflict-with-trail resolve-preserves-almost-no-conflict-with-trail
backtrack-preserves-almost-no-conflict-with-trail
by (*metis scl-def reasonable-if-regular scl-if-reasonable*)

11.1.2 Backtrack Follows Regular Conflict Resolution

lemma *before-conflict-in-regular-run:*

assumes

*reg-run: (regular-scl N β)** initial-state S1 and*

conf: conflict N β 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 β S0 S1)

proof –

from *reg-run conf show ?thesis*

proof (*induction S1 arbitrary: S2 rule: rtranclp-induct*)

case *base*

with $\langle \{ \# \} | \notin | N \rangle$ **have** *False*

by (*meson fempty-iff funion-iff mempty-in-iff-ex-conflict*)

thus *?case ..*

next

case (*step S0 S1*)

from *step.hyps(1) have learned-nonempty S0*

by (*induction S0 rule: rtranclp-induct*)

(*simp-all add: scl-preserves-learned-nonempty[OF scl-if-reasonable[OF reasonable-if-regular]]*)

with *step.hyps(2) have learned-nonempty S1*

by (*simp add: scl-preserves-learned-nonempty[OF scl-if-reasonable[OF reasonable-if-regular]]*)

from *step.hyps(1) have trail-propagated-or-decided' N β S0*

by (*induction S0 rule: rtranclp-induct*)

(*simp-all add: scl-preserves-trail-propagated-or-decided[OF scl-if-reasonable[OF reasonable-if-regular]]*)

with *step.hyps(2) have trail-propagated-or-decided' N β S1*

by (*simp add: scl-preserves-trail-propagated-or-decided[OF scl-if-reasonable[OF reasonable-if-regular]]*)

from *step.hyps(1) have almost-no-conflict-with-trail N β S0*

by (*induction S0 rule: rtranclp-induct*)

(*simp-all add: regular-scl-preserves-almost-no-conflict-with-trail*)

with *step.hyps(2) have almost-no-conflict-with-trail N β S1*

by (*simp add: regular-scl-preserves-almost-no-conflict-with-trail*)

show *?case*

proof (*intro exI conjI*)

show (*regular-scl N β)** initial-state S0*

using *step.hyps by simp*

```

next
  show regular-scl N β S0 S1
  using step.hyps by simp
next
from step.premis obtain Γ U C γ where
  S1-def: S1 = (Γ, U, None) and
  S2-def: S2 = (Γ, U, Some (C, γ)) and
  C-in: C |∈| N |∪| U and
  ground-conf: is-ground-cls (C · γ) and
  tr-false-conf: trail-false-cls Γ (C · γ)
  unfolding conflict.simps by auto
with step.hyps have ¬ conflict N β S0 S1 and reasonable-scl N β S0 S1
  unfolding regular-scl-def by (simp-all add: conflict.simps)
with step.premis have scl N β S0 S1 and ¬ decide N β S0 S1
  unfolding reasonable-scl-def by blast+
moreover from step.premis have ¬ backtrack N β S0 S1
proof (cases Γ)
  case Nil
  then show ?thesis
  using ⟨{#} |∉| N⟩ ⟨almost-no-conflict-with-trail N β S1⟩ step.premis
by (auto simp: S1-def almost-no-conflict-with-trail-def elim: no-conflict-with-trail.cases)
next
  case (Cons Ln Γ')
  have C ≠ {#}
  using ⟨{#} |∉| N⟩
  by (metis C-in S1-def ⟨learned-nonempty S1⟩ funionE learned-nonempty-def
state-proj-simp(2))

  from Cons have ¬ is-decision-lit Ln
  using ⟨¬ decide N β S0 S1⟩[unfolded S1-def]
  by (metis (mono-tags, lifting) S1-def ⟨almost-no-conflict-with-trail N β
S1⟩
almost-no-conflict-with-trail-def list.discI list.simps(5)
nex-conflict-if-no-conflict-with-trail state-learned-simp state-trail-simp
step.premis)
  with ⟨{#} |∉| N⟩ have no-conflict-with-trail N β U Γ'
  using ⟨almost-no-conflict-with-trail N β S1⟩
  by (simp add: Cons S1-def almost-no-conflict-with-trail-def)
  with Cons show ?thesis
  unfolding S1-def
  using ⟨{#} |∉| N⟩
  by (smt (verit) S2-def ⟨almost-no-conflict-with-trail N β S0⟩ ⟨learned-nonempty
S1⟩
almost-no-conflict-with-trail-def backtrack.simps conflict.cases finsert-iff
funionE
funion-finsert-right learned-nonempty-def list.case(2) list.sel(3)
list.simps(3)
no-conflict-with-trail.simps not-trail-false-cls-if-no-conflict-with-trail
state-learned-simp state-trail-simp step.premis suffixI decide-lit-def

```

trail-false-cls-if-trail-false-suffix)

qed
ultimately show *propagate* $N \beta S0 S1$
by (*simp add: scl-def S1-def skip.simps conflict.simps factorize.simps re-*
solve.simps)
qed
qed
qed

definition *regular-conflict-resolution where*
regular-conflict-resolution $N \beta S \longleftrightarrow \{\#\} \mid \notin \mid N \longrightarrow$
(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*
by (*simp add: regular-conflict-resolution-def*)

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* $N \beta S'$
proof –
from *step have state-conflict* $S = None$ **and** *state-conflict* $S' = None$
by (*auto elim: propagate.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* $\langle state-conflict\ S' = None \rangle$
unfolding *option.case*
proof (*rule impI*)
assume $\{\#\} \mid \notin \mid N$
with *invar have* (*regular-scl* $N \beta$)^{**} *initial-state* S
unfolding *regular-conflict-resolution-def* $\langle state-conflict\ S = None \rangle$ **by** *simp*
thus (*regular-scl* $N \beta$)^{**} *initial-state* S'
using *reg-step by* (*rule rtranclp.rtrancl-into-rtrancl*)
qed
qed

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* $N \beta S'$
proof –

```

from step have state-conflict  $S = \text{None}$  and state-conflict  $S' = \text{None}$ 
  by (auto elim: decide.cases)

show ?thesis
  unfolding regular-conflict-resolution-def ‹state-conflict  $S' = \text{None}$ ›
  unfolding option.case
proof (rule impI)
  assume  $\{\#\} \notin N$ 
  with invar have (regular-scl  $N \beta$ )** initial-state  $S$ 
    unfolding regular-conflict-resolution-def ‹state-conflict  $S = \text{None}$ › by simp
    thus (regular-scl  $N \beta$ )** initial-state  $S'$ 
    using reg-step by (rule rtranclp.rtrancl-into-rtrancl)
  qed
qed

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  $N \beta S'$ 
proof –
  from step obtain  $C \gamma$  where state-conflict  $S = \text{None}$  and state-conflict  $S' =$ 
Some  $(C, \gamma)$ 
    by (auto elim!: conflict.cases)

  show ?thesis
    unfolding regular-conflict-resolution-def ‹state-conflict  $S' = \text{Some}(C, \gamma)$ ›
    unfolding option.cases
  proof (rule impI)
  assume  $\{\#\} \notin N$ 
  with invar have reg-run: (regular-scl  $N \beta$ )** initial-state  $S$ 
    unfolding regular-conflict-resolution-def ‹state-conflict  $S = \text{None}$ › by simp

  from ‹ $\{\#\} \notin N$ › obtain  $S0$  where
    (regular-scl  $N \beta$ )** initial-state  $S0$  propagate  $N \beta S0 S$  regular-scl  $N \beta S0 S$ 
    using before-conflict-in-regular-run[OF reg-run step] by metis

  with step show  $\exists S0 S1 S2 S3. (regular-scl\ N\ \beta)^{**}\ initial-state\ S0 \wedge$ 
    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  $N \beta$ )**  $S2 S3 \wedge (regular-scl\ N \beta)^{**}\ S2 S3 \wedge$ 
    ( $S3 = S' \vee (\exists S4. resolve\ N \beta S3 S4 \wedge (skip\ N \beta \sqcup factorize\ N \beta \sqcup resolve$ 
     $N \beta)^{**}\ S4 S')$ )
    using regular-scl-if-conflict
    by blast
  qed
qed

lemma
  assumes almost-no-conflict-with-trail  $N \beta S$  and  $\{\#\} \notin N$ 

```

shows *no-conflict-after-decide'* $N \beta S$
proof –
obtain $U \Gamma Cl$ **where** $S\text{-def}: S = (\Gamma, U, Cl)$
by (*metis state-simp*)

show *?thesis*
proof (*cases* Γ)
case *Nil*
thus *?thesis*
by (*simp add: S-def no-conflict-after-decide'-def*)
next
case (*Cons* $Ln \Gamma'$)
with *assms* **have** *no-conf-with-trail:*
no-conflict-with-trail $N \beta U$ (*if is-decision-lit* Ln *then* $Ln \# \Gamma'$ *else* Γ')
by (*simp add: S-def almost-no-conflict-with-trail-def*)

show *?thesis*
using *no-conf-with-trail*
by (*cases is-decision-lit* Ln)
(*simp-all add: S-def Cons no-conflict-after-decide'-def no-conflict-after-decide.Cons no-conflict-after-decide-if-no-conflict-with-trail*)

qed
qed

lemma *mempty-not-in-learned-if-almost-no-conflict-with-trail:*
almost-no-conflict-with-trail $N \beta S \implies \{\#\} \notin N \implies \{\#\} \notin \text{state-learned } S$
unfolding *almost-no-conflict-with-trail-def*
using *nex-conflict-if-no-conflict-with-trail'[folded mempty-in-iff-ex-conflict]*
by *simp*

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* $N \beta S'$
proof –
from *step* **obtain** $C \gamma$ **where**
state-conflict $S = \text{Some } (C, \gamma)$ **and** *state-conflict* $S' = \text{Some } (C, \gamma)$
by (*auto elim!: skip.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S' = \text{Some } (C, \gamma) \rangle$
unfolding *option.cases*
proof (*intro impI*)
assume $\{\#\} \notin N$
with *invar* **obtain** $S0 S1 S2 S3$ **where**
reg-run: $(\text{regular-scl } N \beta)^{**}$ *initial-state* $S0$ **and**
propa: *propagate* $N \beta S0 S1$ *regular-scl* $N \beta S0 S1$ **and**
conft: *conflict* $N \beta S1 S2$ *regular-scl* $N \beta S1 S2$ **and**
facto: $(\text{factorize } N \beta)^{**}$ $S2 S3$ $(\text{regular-scl } N \beta)^{**}$ $S2 S3$ **and**

maybe-reso: $S3 = S \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S4 S)$
unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S = \text{Some } (C, \gamma) \rangle$
unfolding *option.cases*
by *metis*

from *reg-run* **have** $(\text{regular-scl } N \beta)^{**} \text{initial-state } S1$
using $\langle \text{regular-scl } N \beta S0 S1 \rangle$ **by** *simp*
hence $(\text{regular-scl } N \beta)^{**} \text{initial-state } S2$
using $\langle \text{regular-scl } N \beta S1 S2 \rangle$ **by** *simp*
hence $(\text{regular-scl } N \beta)^{**} \text{initial-state } S3$
using $\langle (\text{regular-scl } N \beta)^{**} S2 S3 \rangle$ **by** *simp*

from $\langle (\text{factorize } N \beta)^{**} S2 S3 \rangle$ **have** $\text{state-trail } S3 = \text{state-trail } S2$
by $(\text{induction } S3 \text{ rule: } r\text{tranclp-induct}) (\text{auto elim: } \text{factorize.cases})$
also from $\langle \text{conflict } N \beta S1 S2 \rangle$ **have** $\dots = \text{state-trail } S1$
by $(\text{auto elim: } \text{conflict.cases})$
finally have $\text{state-trail } S3 = \text{state-trail } S1$
by *assumption*

from $\langle (\text{factorize } N \beta)^{**} S2 S3 \rangle$ **have** $\text{state-learned } S3 = \text{state-learned } S2$
proof $(\text{induction } S3 \text{ rule: } r\text{tranclp-induct})$
case *base*
show *?case* **by** *simp*
next
case $(\text{step } y z)$
thus *?case*
by $(\text{elim } \text{factorize.cases}) \text{ simp}$
qed

also from $\langle \text{conflict } N \beta S1 S2 \rangle$ **have** $\dots = \text{state-learned } S1$
by $(\text{auto elim: } \text{conflict.cases})$
finally have $\text{state-learned } S3 = \text{state-learned } S1$
by *assumption*

from $\langle \text{propagate } N \beta S0 S1 \rangle$ **have** $\text{state-learned } S1 = \text{state-learned } S0$
by $(\text{auto elim: } \text{propagate.cases})$

from $\langle \text{propagate } N \beta S0 S1 \rangle$ **obtain** $L C \gamma$ **where**
 $\text{state-trail } S1 = \text{trail-propagate } (\text{state-trail } S0) L C \gamma$
by $(\text{auto elim: } \text{propagate.cases})$

from $\langle (\text{regular-scl } N \beta)^{**} \text{initial-state } S3 \rangle$ **have** *almost-no-conflict-with-trail* $N \beta S3$
using *regular-scl-preserves-almost-no-conflict-with-trail*
by $(\text{induction } S3 \text{ rule: } r\text{tranclp-induct}) \text{ simp-all}$

show $\exists S0 S1 S2 S3. (\text{regular-scl } N \beta)^{**} \text{initial-state } S0 \wedge$
 $\text{propagate } N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$
 $\text{conflict } N \beta S1 S2 \wedge \text{regular-scl } N \beta S1 S2 \wedge$

```

(factorize N β)** S2 S3 ∧ (regular-scl N β)** S2 S3 ∧
(S3 = S' ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S'))
using reg-run propa confl facto
proof (intro impI exI conjI)
  show S3 = S' ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔
resolve N β)** S4 S')
    using maybe-reso
    proof (elim disjE exE conjE)
      fix S4 assume resolve N β S3 S4 and (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S
        with step have ∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔
resolve N β)** S4 S'
          by (meson rtranclp.rtrancl-into-rtrancl sup2CI)
        thus ?thesis ..
      next
        assume S3 = S
        with ⟨almost-no-conflict-with-trail N β S3⟩ ⟨{#} |∉| N⟩
        have no-conf-with-trail: no-conflict-with-trail N β (state-learned S)
          (case state-trail S of [] ⇒ [] | Ln # Γ ⇒ if is-decision-lit Ln then Ln # Γ
else Γ)
          by (simp add: almost-no-conflict-with-trail-def)
        hence {#} |∉| state-learned S
        using nex-conflict-if-no-conflict-with-trail'[folded mempty-in-iff-ex-conflict]
        by simp

from no-conf-with-trail
  have no-conf-with-trail': no-conflict-with-trail N β (state-learned S1)
(state-trail S0)
    using ⟨S3 = S⟩ ⟨state-trail S3 = state-trail S1⟩
    ⟨state-learned S3 = state-learned S1⟩
    ⟨state-trail S1 = trail-propagate (state-trail S0) L C γ⟩
    by (simp add: propagate-lit-def is-decision-lit-def)

have ∃ D γD. state-conflict S2 = Some (D, γD) ∧ - (L · l γ) ∈# D · γD
  using ⟨conflict N β S1 S2⟩
proof (cases N β S1 S2 rule: conflict.cases)
  case (conflictI D U γD Γ)
    hence trail-false-cls (trail-propagate (state-trail S0) L C γ) (D · γD)
      using ⟨state-trail S1 = trail-propagate (state-trail S0) L C γ⟩
      by simp

moreover from no-conf-with-trail' have ¬ trail-false-cls (state-trail S0)
(D · γD)
  unfolding ⟨state-learned S1 = state-learned S0⟩
  proof (rule not-trail-false-cls-if-no-conflict-with-trail)
    show D |∈| N |∪| state-learned S0
      using ⟨state-learned S1 = state-learned S0⟩ local.conflictI(1) lo-
cal.conflictI(3)

```

```

      by fastforce
    next
      have {#} | $\notin$ |  $U$ 
      using ⟨{#} | $\notin$ | state-learned  $S$ ⟩ ⟨ $S3 = S$ ⟩ ⟨state-learned  $S3 = state-learned$ 
 $S1$ ⟩
        unfolding conflictI(1,2)
        by simp
      thus  $D \neq \{#\}$ 
      using ⟨{#} | $\notin$ |  $N$ ⟩ ⟨ $D \in N \mid U$ ⟩
      by auto
    next
      show is-ground-cls ( $D \cdot \gamma_D$ )
      by (rule ⟨is-ground-cls ( $D \cdot \gamma_D$ )⟩)
    qed

    ultimately have  $-(L \cdot l \gamma) \in\# D \cdot \gamma_D$ 
      by (metis subtrail-falseI propagate-lit-def)

    moreover have state-conflict  $S2 = Some (D, \gamma_D)$ 
      unfolding conflictI(1,2) by simp

    ultimately show ?thesis
      by metis
    qed
  then obtain  $D \gamma_D$  where state-conflict  $S2 = Some (D, \gamma_D)$  and  $-(L \cdot l$ 
 $\gamma) \in\# D \cdot \gamma_D$ 
    by metis

  with ⟨(factorize  $N \beta$ )**  $S2 S3$ ⟩
  have  $\exists D' \gamma_{D'}. state-conflict S3 = Some (D', \gamma_{D'}) \wedge -(L \cdot l \gamma) \in\# D' \cdot$ 
 $\gamma_{D'}$ 
  proof (induction  $S3$  arbitrary: rule: rtranclp-induct)
  case base
  thus ?case by simp
  next
  case (step  $y z$ )
  then obtain  $D' \gamma_{D'}$  where state-conflict  $y = Some (D', \gamma_{D'})$  and  $-(L$ 
 $\cdot l \gamma) \in\# D' \cdot \gamma_{D'}$ 
    by auto
  then show ?case
    using step.hyps(2)
    by (metis conflict-set-after-factorization)
  qed
  with step have False
    using ⟨state-trail  $S3 = state-trail S1$ ⟩
    unfolding ⟨ $S3 = S$ ⟩ ⟨state-trail  $S1 = trail-propagate (state-trail S0) L C$ 
 $\gamma$ ⟩
      by (auto simp add: propagate-lit-def elim!: skip.cases)
  thus ?thesis ..

```

qed
 qed
 qed
 qed

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* $N \beta S'$

proof –

from *step* **obtain** $C \gamma C' \gamma'$ **where**
state-conflict $S = \text{Some}(C, \gamma)$ **and** *state-conflict* $S' = \text{Some}(C', \gamma')$
by (*auto elim!*: *factorize.cases*)

show *?thesis*

unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S' = \text{Some}(C', \gamma') \rangle$

unfolding *option.cases*

proof (*intro impI*)

assume $\{\#\} \notin N$

with *invar* **obtain** $S0 S1 S2 S3$ **where**

reg-run: (*regular-scl* $N \beta$)** *initial-state* $S0$ **and**

propa: *propagate* $N \beta S0 S1$ *regular-scl* $N \beta S0 S1$ **and**

confl: *conflict* $N \beta S1 S2$ *regular-scl* $N \beta S1 S2$ **and**

facto: (*factorize* $N \beta$)** $S2 S3$ (*regular-scl* $N \beta$)** $S2 S3$ **and**

maybe-reso: $S3 = S \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta$

$\sqcup \text{resolve } N \beta)$ ** $S4 S)$

unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S = \text{Some}(C, \gamma) \rangle$

unfolding *option.cases*

by *metis*

show $\exists S0 S1 S2 S3. (\text{regular-scl } N \beta)$ ** *initial-state* $S0 \wedge$

propagate $N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$

conflict $N \beta S1 S2 \wedge \text{regular-scl } N \beta S1 S2 \wedge$

(*factorize* $N \beta$)** $S2 S3 \wedge (\text{regular-scl } N \beta)$ ** $S2 S3 \wedge$

$(S3 = S' \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)$ ** $S4 S'))$

using *maybe-reso*

proof (*elim disjE exE conjE*)

assume $S3 = S$

show *?thesis*

using *reg-run propa confl*

proof (*intro exI conjI*)

show (*factorize* $N \beta$)** $S2 S'$

using $\langle (\text{factorize } N \beta)$ ** $S2 S3 \rangle$ *step*

by (*simp add*: $\langle S3 = S \rangle$)

next

show (*regular-scl* $N \beta$)** $S2 S'$

using $\langle (\text{regular-scl } N \beta)$ ** $S2 S3 \rangle$ *reg-step*

by (*simp add*: $\langle S3 = S \rangle$)

```

next
  show  $S' = S' \vee (\exists S_4. \text{resolve } N \beta S' S_4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S_4 S')$ 
    by simp
  qed
next
  fix  $S_4$  assume hyps:  $\text{resolve } N \beta S_3 S_4 (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S_4 S$ 
  show ?thesis
    using reg-run propa confl facto
  proof (intro exI conjI)
    show  $S_3 = S' \vee (\exists S_4. \text{resolve } N \beta S_3 S_4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S_4 S')$ 
      using hyps step
      by (meson rtranclp.rtrancl-into-rtrancl sup2CI)
    qed
  qed
qed
qed

```

lemma *resolve-preserves-regular-conflict-resolution*:

```

assumes step:  $\text{resolve } N \beta S S'$  and reg-step:  $\text{regular-scl } N \beta S S'$  and
  invar:  $\text{regular-conflict-resolution } N \beta S$ 
shows  $\text{regular-conflict-resolution } N \beta S'$ 
proof –
  from step obtain  $C \gamma C' \gamma'$  where
    state-conflict  $S = \text{Some } (C, \gamma)$  and state-conflict  $S' = \text{Some } (C', \gamma')$ 
    by (auto elim!: resolve.cases)

```

show *?thesis*

unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S' = \text{Some } (C', \gamma') \rangle$

unfolding *option.cases*

proof (*intro impI*)

from *step* **have** *state-conflict* $S \neq \text{None}$

by (*auto elim*: *resolve.cases*)

assume $\{\#\} \mid \notin \mid N$

with *invar* **obtain** $S_0 S_1 S_2 S_3$ **where**

reg-run: $(\text{regular-scl } N \beta)^{**} \text{initial-state } S_0$ **and**

propagate $N \beta S_0 S_1$ *regular-scl* $N \beta S_0 S_1$ **and**

conflict $N \beta S_1 S_2$ *regular-scl* $N \beta S_1 S_2$ **and**

$(\text{factorize } N \beta)^{**} S_2 S_3$ $(\text{regular-scl } N \beta)^{**} S_2 S_3$ **and**

maybe-reso: $S_3 = S \vee (\exists S_4. \text{resolve } N \beta S_3 S_4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S_4 S)$

unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S = \text{Some } (C, \gamma) \rangle$

unfolding *option.cases*

by *metis*

then show $\exists S_0 S_1 S_2 S_3. (\text{regular-scl } N \beta)^{**} \text{initial-state } S_0 \wedge$

```

propagate N β S0 S1 ∧ regular-scl N β S0 S1 ∧
conflict N β S1 S2 ∧ regular-scl N β S1 S2 ∧
(factorize N β)** S2 S3 ∧ (regular-scl N β)** S2 S3 ∧
(S3 = S' ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S'))
proof (intro exI conjI)
  show S3 = S' ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔
resolve N β)** S4 S')
  using maybe-reso step
    by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl rtran-
clp.rtrancl-refl
sup2I2)
qed
qed
qed

```

lemma *backtrack-preserves-regular-conflict-resolution:*

```

assumes step: backtrack N β S S' and reg-step: regular-scl N β S S' and
  invar: regular-conflict-resolution N β S
shows regular-conflict-resolution N β S'
proof –
  from step obtain C γ where
    state-conflict S = Some (C, γ) and state-conflict S' = None
  by (auto elim!: backtrack.cases)

```

```

show ?thesis
  unfolding regular-conflict-resolution-def ⟨state-conflict S' = None⟩
  unfolding option.case
proof (rule impI)
  assume {#} |≠| N
  with invar obtain S0 S1 S2 S3 where
    reg-run: (regular-scl N β)** initial-state S0 and
    propa: propagate N β S0 S1 regular-scl N β S0 S1 and
    confl: conflict N β S1 S2 regular-scl N β S1 S2 and
    facto: (factorize N β)** S2 S3 (regular-scl N β)** S2 S3 and
    maybe-reso: S3 = S ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β
⊔ resolve N β)** S4 S)
  unfolding regular-conflict-resolution-def ⟨state-conflict S = Some (C, γ)⟩
  unfolding option.cases
  by metis

```

```

from reg-run propa(2) confl(2) facto(2) have reg-run-S3: (regular-scl N β)**
initial-state S3
by simp

```

```

show (regular-scl N β)** initial-state S'
using maybe-reso
proof (elim disjE exE conjE)
  show S3 = S ⇒ (regular-scl N β)** initial-state S'

```

```

    using reg-run-S3 reg-step by simp
  next
  fix S4 assume hyps: resolve N β S3 S4 (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S
  have (regular-scl N β)** initial-state S4
    using reg-run-S3 regular-scl-if-resolve[OF hyps(1)]
    by (rule rtranclp.rtrancl-into-rtrancl)
  also have (regular-scl N β)** S4 S
    using hyps(2)
    by (rule mono-rtranclp[rule-format, rotated]) auto
  also have (regular-scl N β)** S S'
    using reg-step by simp
  finally show (regular-scl N β)** initial-state S'
    by assumption
qed
qed
qed

```

lemma *regular-scl-preserves-regular-conflict-resolution:*

assumes *reg-step: regular-scl N β S S' and*

invars: regular-conflict-resolution N β S

shows *regular-conflict-resolution N β S'*

using *assms*

using *propagate-preserves-regular-conflict-resolution decide-preserves-regular-conflict-resolution conflict-preserves-regular-conflict-resolution skip-preserves-regular-conflict-resolution factorize-preserves-regular-conflict-resolution resolve-preserves-regular-conflict-resolution backtrack-preserves-regular-conflict-resolution*

by (*metis regular-scl-def reasonable-scl-def scl-def*)

11.2 Miscellaneous Lemmas

lemma *mempty-not-in-initial-clauses-if-non-empty-regular-conflict:*

assumes *state-conflict S = Some (C, γ) and C ≠ {#} and*

invars: almost-no-conflict-with-trail N β S sound-state N β S ground-false-closures S

shows *{#} ∉ N*

proof –

from *assms(1)* **obtain** Γ U **where** *S-def: S = (Γ, U, Some (C, γ))*

by (*metis state-simp*)

from *assms(2)* **obtain** L C' **where** *C-def: C = add-mset L C'*

using *multi-nonempty-split* **by** *metis*

from *invars(3)* **have** *trail-false-cls Γ (C · γ)*

by (*simp add: S-def ground-false-closures-def*)

then obtain Ln Γ' **where** $\Gamma = Ln \# \Gamma'$

by (*metis assms(2) neq-Nil-conv not-trail-false-Nil(2) subst-cls-empty-iff*)

with *invars(1)* **have** *no-conflict-with-trail N β U (if is-decision-lit Ln then Ln # Γ' else Γ')*

by (*simp add: S-def almost-no-conflict-with-trail-def*)
 hence $\nexists S'. \text{conflict } N \beta (\[], U, \text{None}) S'$
 by (*rule nex-conflict-if-no-conflict-with-trail'*)
 hence $\{\#\} \notin N \cup U$
 unfolding *mempty-in-iff-ex-conflict[symmetric]* by *assumption*
 thus *?thesis*
 by *simp*
 qed

lemma *mempty-not-in-initial-clauses-if-regular-run-reaches-non-empty-conflict:*
 assumes (*regular-scl* $N \beta$)** *initial-state* S and *state-conflict* $S = \text{Some } (C, \gamma)$
 and $C \neq \{\#\}$
 shows $\{\#\} \notin N$
proof (*rule notI*)
 from *assms(2)* have *initial-state* $\neq S$ by *fastforce*
 then obtain S' where
 reg-scl-init-S': *regular-scl* $N \beta$ *initial-state* S' and (*regular-scl* $N \beta$)** $S' S$
 by (*metis assms(1) converse-rtranclpE*)

assume $\{\#\} \in N$
 hence *conflict* $N \beta$ *initial-state* $(\[], \{\|\}, \text{Some } (\{\#\}, \text{Var}))$
 by (*rule conflict-initial-state-if-mempty-in-intial-clauses*)
 hence *conf-init: regular-scl* $N \beta$ *initial-state* $(\[], \{\|\}, \text{Some } (\{\#\}, \text{Var}))$
 using *regular-scl-def* by *blast*
 then obtain γ where *S'-def: S' =* $(\[], \{\|\}, \text{Some } (\{\#\}, \gamma))$
 using *reg-scl-init-S'*
 unfolding *regular-scl-def*
 using $\langle \text{conflict } N \beta \text{ initial-state } (\[], \{\|\}, \text{Some } (\{\#\}, \text{Var})) \rangle$
 conflict-initial-state-only-with-mempty
 by *blast*

have $\nexists S'. \text{scl } N \beta (\[], \{\|\}, \text{Some } (\{\#\}, \gamma)) S'$ for γ
 using *no-more-step-if-conflict-mempty* by *simp*
 hence $\nexists S'. \text{regular-scl } N \beta (\[], \{\|\}, \text{Some } (\{\#\}, \gamma)) S'$ for γ
 using *scl-if-reasonable[OF reasonable-if-regular]* by *blast*
 hence $S = S'$
 using $\langle (\text{regular-scl } N \beta)$ ** $S' S \rangle$
 unfolding *S'-def*
 by (*metis converse-rtranclpE*)
 with *assms(2,3)* show *False* by (*simp add: S'-def*)
 qed

lemma *before-regular-backtrack:*
 assumes
 backt: 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 $\exists S0 S1 S2 S3 S4. (\text{regular-scl } N \beta)$ ** *initial-state* $S0 \wedge$
 propagate $N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$

```

    conflict N β S1 S2 ∧ (factorize N β)** S2 S3 ∧ resolve N β S3 S4 ∧
    (skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S
proof –
  from backt obtain L C γ where conflict-S: state-conflict S = Some (add-mset
L C, γ)
  by (auto elim: backtrack.cases)

  have {#} |≠| N
proof (rule mempty-not-in-initial-clauses-if-non-empty-regular-conflict)
  show state-conflict S = Some (add-mset L C, γ)
  by (rule ⟨state-conflict S = Some (add-mset L C, γ)⟩)
next
  show add-mset L C ≠ {#}
  by simp
next
  show almost-no-conflict-with-trail N β S
  by (rule ⟨almost-no-conflict-with-trail N β S⟩)
next
  show sound-state N β S
  by (rule ⟨sound-state N β S⟩)
next
  show ground-false-closures S
  by (rule ⟨ground-false-closures S⟩)
qed

then obtain S0 S1 S2 S3 where
  reg-run: (regular-scl N β)** initial-state S0 and
  propa: propagate N β S0 S1 regular-scl N β S0 S1 and
  confl: conflict N β S1 S2 and
  fact: (factorize N β)** S2 S3 and
  maybe-resolution: S3 = S ∨
  (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S)
using ⟨regular-conflict-resolution N β S⟩ ⟨state-conflict S = Some (add-mset L
C, γ)⟩
  unfolding regular-conflict-resolution-def conflict-S option.case
  by metis

have S3 ≠ S
proof (rule notI)
  from ⟨(factorize N β)** S2 S3⟩ have state-trail S3 = state-trail S2
  by (induction S3 rule: rtranclp-induct) (auto elim: factorize.cases)
  also from ⟨conflict N β S1 S2⟩ have ... = state-trail S1
  by (auto elim: conflict.cases)
  finally have state-trail S3 = state-trail S1
  by assumption
  from ⟨propagate N β S0 S1⟩ obtain L C γ where
  state-trail S1 = trail-propagate (state-trail S0) L C γ
  by (auto elim: propagate.cases)

```

```

from ⟨(factorize N β)** S2 S3⟩ have state-learned S3 = state-learned S2
proof (induction S3 rule: rtranclp-induct)
  case base
  show ?case by simp
next
  case (step y z)
  thus ?case
  by (elim factorize.cases) simp
qed
also from ⟨conflict N β S1 S2⟩ have ... = state-learned S1
  by (auto elim: conflict.cases)
finally have state-learned S3 = state-learned S1
  by assumption

from ⟨propagate N β S0 S1⟩ have state-learned S1 = state-learned S0
  by (auto elim: propagate.cases)

assume S3 = S
hence no-conf-with-trail: no-conflict-with-trail N β (state-learned S0) (state-trail
S0)
  using ⟨almost-no-conflict-with-trail N β S⟩ ⟨{#} |≠| N⟩
  ⟨state-trail S1 = trail-propagate (state-trail S0) L C γ⟩ ⟨state-trail S3 =
state-trail S1⟩
  ⟨state-learned S3 = state-learned S1⟩ ⟨state-learned S1 = state-learned S0⟩
by (simp add: almost-no-conflict-with-trail-def propagate-lit-def is-decision-lit-def)
hence {#} |≠| state-learned S0
  using nex-conflict-if-no-conflict-with-trail'[folded mempty-in-iff-ex-conflict] by
simp

have ∃ D γD. state-conflict S2 = Some (D, γD) ∧ - (L · l γ) ∈# D · γD
  using ⟨conflict N β S1 S2⟩
proof (cases N β S1 S2 rule: conflict.cases)
  case (conflictI D U γD Γ)
  hence trail-false-cls (trail-propagate (state-trail S0) L C γ) (D · γD)
    using ⟨state-trail S1 = trail-propagate (state-trail S0) L C γ⟩
    by simp

moreover from no-conf-with-trail have ¬ trail-false-cls (state-trail S0) (D ·
γD)
proof (rule not-trail-false-cls-if-no-conflict-with-trail)
  show D |∈| N |∪| state-learned S0
  using ⟨state-learned S1 = state-learned S0⟩ local.conflictI(1) local.conflictI(3)
  by fastforce
next
  have {#} |≠| U
  using ⟨{#} |≠| state-learned S0⟩ ⟨S3 = S⟩ ⟨state-learned S3 = state-learned
S1⟩
  ⟨state-learned S1 = state-learned S0⟩
  unfolding conflictI(1,2)

```

```

    by simp
  thus  $D \neq \{\#\}$ 
    using  $\langle \{\#\} \notin N \rangle \langle D \in N \cup U \rangle$ 
    by auto
next
  show is-ground-cls  $(D \cdot \gamma_D)$ 
    by (rule is-ground-cls  $(D \cdot \gamma_D)$ )
qed

ultimately have  $-(L \cdot l \gamma) \in \# D \cdot \gamma_D$ 
  by (metis subtrail-falseI propagate-lit-def)

moreover have state-conflict  $S2 = \text{Some } (D, \gamma_D)$ 
  unfolding conflictI(1,2) by simp

ultimately show ?thesis
  by metis
qed
then obtain  $D \gamma_D$  where state-conflict  $S2 = \text{Some } (D, \gamma_D)$  and  $-(L \cdot l \gamma) \in \# D \cdot \gamma_D$ 
  by metis
with  $\langle \text{factorize } N \beta \rangle^{**} S2 S3$ 
have  $\exists D' \gamma_{D'}. \text{state-conflict } S3 = \text{Some } (D', \gamma_{D'}) \wedge -(L \cdot l \gamma) \in \# D' \cdot \gamma_{D'}$ 
proof (induction  $S3$  arbitrary: rule: rtranclp-induct)
  case base
  thus ?case by simp
next
  case (step  $y z$ )
  then obtain  $D' \gamma_{D'}$  where state-conflict  $y = \text{Some } (D', \gamma_{D'})$  and  $-(L \cdot l \gamma) \in \# D' \cdot \gamma_{D'}$ 
    by auto
  then show ?case
    using step.hyps(2)
    by (metis conflict-set-after-factorization)
qed
with backt  $\langle S3 = S \rangle$  show False
  using  $\langle \text{state-trail } S3 = \text{state-trail } S1 \rangle$ 
  unfolding  $\langle S3 = S \rangle \langle \text{state-trail } S1 = \text{trail-propagate } (\text{state-trail } S0) L C \gamma \rangle$ 
  by (auto simp add: decide-lit-def propagate-lit-def elim!: backtrack.cases)
qed
with maybe-resolution obtain  $S4$  where
  resolve  $N \beta S3 S4$  and  $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S4 S$ 
  by metis
show ?thesis
proof (intro exI conjI)
  show (regular-scl  $N \beta)^{**}$  initial-state  $S0$ 
    by (rule  $\langle \text{regular-scl } N \beta \rangle^{**}$  initial-state  $S0$ )
next
  show propagate  $N \beta S0 S1$ 

```

```

    by (rule ⟨propagate N β S0 S1⟩)
next
  show regular-scl N β S0 S1
    by (rule propa(2))
next
  show conflict N β S1 S2
    by (rule ⟨conflict N β S1 S2⟩)
next
  show (factorize N β)** S2 S3
    by (rule ⟨(factorize N β)** S2 S3⟩)
next
  show resolve N β S3 S4
    by (rule ⟨resolve N β S3 S4⟩)
next
  show (skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S
    by (rule ⟨(skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S⟩)
qed
qed

```

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 $\exists S_3. \text{resolve } N \beta S_2 S_3$

using *propa*

proof (*cases* $N \beta S_0 S_1$ *rule*: *propagate.cases*)

case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)

hence *S₀-def*: $S_0 = (\Gamma, U, \text{None})$

by *simp*

from *conf* **obtain** $\gamma_D D$ **where**

S₂-def: $S_2 = (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, \text{Some } (D, \gamma_D))$ **and**

D-in: $D \mid \in \mid N \mid \cup \mid U$ **and**

gr-D- γ_D : *is-ground-cls* $(D \cdot \gamma_D)$ **and**

tr-false- Γ - L - μ : *trail-false-cls* $(\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma) (D \cdot \gamma_D)$

by (*elim conflict.cases*) (*unfold propagateI(1,2)*, *blast*)

from *no-conf* **have** $\neg \text{trail-false-cls } \Gamma (D \cdot \gamma_D)$

using *gr-D- γ_D* *D-in not-trail-false-ground-cls-if-no-conflict*[*of* $N \beta - D \gamma_D$]

using *S₀-def* **by** *force*

with *tr-false- Γ - L - μ* **have** $-(L \cdot l \mu \cdot l \gamma) \in \# D \cdot \gamma_D$

unfolding *propagate-lit-def* **by** (*metis subtrail-falseI*)

then obtain $D' L'$ **where** *D-def*: $D = \text{add-mset } L' D'$ **and** $1: L \cdot l \mu \cdot l \gamma = -$
 $(L' \cdot l \gamma_D)$

by (*metis Melem-subst-cls multi-member-split uminus-of-uminus-id*)

define ϱ **where**
 $\varrho = \text{renaming-wrt } \{ \text{add-mset } L \ C_0 \cdot \mu \}$

have *is-renaming* ϱ
by (*metis* ϱ -def *finite.emptyI finite.insertI is-renaming-renaming-wrt*)
hence $\forall x. \text{is-Var } (\varrho \ x)$ **and** *inj* ϱ
by (*simp-all add: is-renaming-iff*)

have *disjoint-vars*: $\bigwedge C. \text{vars-cls } (C \cdot \varrho) \cap \text{vars-cls } (\text{add-mset } L \ C_0 \cdot \mu) = \{ \}$
by (*simp add: \varrho-def vars-cls-subst-renaming-disj*)

have $\exists \mu'. \text{Unification.mgu } (\text{atm-of } L' \cdot a \ \varrho) (\text{atm-of } L \cdot a \ \mu) = \text{Some } \mu'$
proof (*rule ex-mgu-if-subst-eq-subst-and-disj-vars*)
have *vars-lit* $L' \subseteq \text{subst-domain } \gamma_D$
using *gr-D-\gamma_D[unfolded D-def]*
by (*simp add: vars-lit-subset-subst-domain-if-grounding is-ground-cls-imp-is-ground-lit*)
hence *atm-of* $L' \cdot a \ \varrho \cdot a \ \text{rename-subst-domain } \varrho \ \gamma_D = \text{atm-of } L' \cdot a \ \gamma_D$
by (*rule renaming-cancels-rename-subst-domain[OF \langle \forall x. \text{is-Var } (\varrho \ x) \rangle \langle \text{inj } \varrho \rangle]*)
then show *atm-of* $L' \cdot a \ \varrho \cdot a \ \text{rename-subst-domain } \varrho \ \gamma_D = \text{atm-of } L \cdot a \ \mu \cdot a \ \gamma$
using 1 **by** (*metis atm-of-subst-lit atm-of-uminus*)
next
show *vars-term* $(\text{atm-of } L' \cdot a \ \varrho) \cap \text{vars-term } (\text{atm-of } L \cdot a \ \mu) = \{ \}$
using *disjoint-vars[of \{ \#L' \# \}]* **by** *auto*
qed
then obtain μ' **where** *imgu-\mu'*: *is-imgu* $\mu' \{ \{ \text{atm-of } L' \cdot a \ \varrho, \text{atm-of } (L \cdot l \ \mu) \} \}$
using *is-imgu-if-mgu-eq-Some* **by** *auto*

let $? \Gamma \text{prop} = \text{trail-propagate } \Gamma (L \cdot l \ \mu) (C_0 \cdot \mu) \ \gamma$
let $? \gamma \text{reso} = \lambda x. \text{if } x \in \text{vars-cls } (\text{add-mset } L' \ D' \cdot \varrho) \text{ then } \text{rename-subst-domain } \varrho \ \gamma_D \ x \text{ else } \gamma \ x$

have *resolve* $N \ \beta$
 $(? \Gamma \text{prop}, U, \text{Some } (\text{add-mset } L' \ D', \gamma_D))$
 $(? \Gamma \text{prop}, U, \text{Some } ((D' \cdot \varrho + C_0 \cdot \mu \cdot \text{Var}) \cdot \mu', ? \gamma \text{reso}))$
proof (*rule resolveI[OF refl]*)
show $L \cdot l \ \mu \cdot l \ \gamma = - (L' \cdot l \ \gamma_D)$
by (*rule 1*)
next
show *is-renaming* ϱ
by (*metis* ϱ -def *finite.emptyI finite.insertI is-renaming-renaming-wrt*)
next
show *vars-cls* $(\text{add-mset } L' \ D' \cdot \varrho) \cap \text{vars-cls } (\text{add-mset } (L \cdot l \ \mu) (C_0 \cdot \mu) \cdot \text{Var}) = \{ \}$
using *disjoint-vars[of add-mset L' D']* **by** *simp*
next
show *is-imgu* $\mu' \{ \{ \text{atm-of } L' \cdot a \ \varrho, \text{atm-of } (L \cdot l \ \mu) \cdot a \ \text{Var} \} \}$
using *imgu-\mu'* **by** *simp*
next

show *is-grounding-merge* ? γ_{reso}
 (vars-cls (add-mset $L' D' \cdot \varrho$)) (rename-subst-domain $\varrho \gamma_D$)
 (vars-cls (add-mset ($L \cdot l \mu$) ($C_0 \cdot \mu$) \cdot Var)) (rename-subst-domain Var γ)
using *is-grounding-merge-if-mem-then-else*
by (*metis rename-subst-domain-Var-lhs*)
qed *simp-all*
thus ?*thesis*
unfolding S_2 -def D -def **by** *metis*
qed

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 $\exists S_4$. resolve $N \beta S_3 S_4$
using *reso*
proof (*cases* $N \beta S_1 S_2$ *rule*: *resolve.cases*)
case (*resolveI* $\Gamma \Gamma' K D \gamma_D L \gamma_C \varrho_C \varrho_D C \mu \gamma U$)

from *fact*[*unfolded resolveI*(1,2)] **obtain** $LL' LL CC \mu_L$ **where**
 S_1 -def: $S_1 = (\Gamma, U, \text{Some} (\text{add-mset } LL' (\text{add-mset } LL CC), \gamma_C))$ **and**
 S_3 -def: $S_3 = (\Gamma, U, \text{Some} (\text{add-mset } LL CC \cdot \mu_L, \gamma_C))$ **and**
 $LL \cdot l \gamma_C = LL' \cdot l \gamma_C$ **and**
imgu- μ_L : *is-imgu* $\mu_L \{\{atm\text{-of } LL, atm\text{-of } LL'\}\}$
by (*auto simp*: $\langle S_1 = (\Gamma, U, \text{Some} (\text{add-mset } L C, \gamma_C)) \rangle$ *elim*: *factorize.cases*)

from *invar* **have**
ground-conf: *is-ground-cls* ($\text{add-mset } L C \cdot \gamma_C$)
unfolding *resolveI*(1,2)
by (*simp-all add*: *ground-closures-def*)

have $\text{add-mset } L C = \text{add-mset } LL' (\text{add-mset } LL CC)$
using *resolveI*(1) S_1 -def **by** *simp*

from *imgu- μ_L* **have** $\mu_L \odot \gamma_C = \gamma_C$
using $\langle LL \cdot l \gamma_C = LL' \cdot l \gamma_C \rangle$
by (*auto simp*: *is-imgu-def is-unifiers-def is-unifier-alt intro!*: *subst-atm-of-eqI*)

have $L \cdot l \mu_L \in \# \text{add-mset } LL CC \cdot \mu_L$
proof (*cases* $L = LL \vee L = LL'$)
case *True*

moreover **have** $LL \cdot l \mu_L = LL' \cdot l \mu_L$
proof –
have *is-unifier* $\mu_L \{\{atm\text{-of } LL, atm\text{-of } LL'\}\}$
using *imgu- μ_L* [*unfolded is-imgu-def*, *THEN conjunct1*, *unfolded is-unifiers-def*,
simplified] .

hence *atm-of* $LL \cdot a \mu_L = atm\text{-of } LL' \cdot a \mu_L$
unfolding *is-unifier-alt*[*of* $\{\{atm\text{-of } LL, atm\text{-of } LL'\}$, *simplified*] ..

hence $atm\text{-of } (LL \cdot l \mu_L) = atm\text{-of } (LL' \cdot l \mu_L)$
unfolding $atm\text{-of-subst-lit}[symmetric]$.

thus *?thesis*
using $\langle LL \cdot l \gamma_C = LL' \cdot l \gamma_C \rangle$
by (*metis* (*no-types*, *opaque-lifting*) *literal.expand subst-lit-is-neg*)
qed

ultimately have $L \cdot l \mu_L = LL \cdot l \mu_L$
by *presburger*
thus *?thesis*
by *simp*
next
case *False*
hence $L \in \# CC$
using $\langle add\text{-mset } L C = add\text{-mset } LL' (add\text{-mset } LL CC) \rangle$
by (*metis insert-iff set-mset-add-mset-insert*)
thus *?thesis*
by *auto*
qed
then obtain CCC **where** $add\text{-mset } LL CC \cdot \mu_L = add\text{-mset } L CCC \cdot \mu_L$
by (*smt* (*verit*, *best*) *Melem-subst-cls mset-add subst-cls-add-mset*)

define $\rho\rho$ **where**
 $\rho\rho = renaming\text{-wrt } \{add\text{-mset } K D\}$

have $ren\text{-}\rho\rho$: *is-renaming* $\rho\rho$
by (*metis* $\rho\rho\text{-def finite.emptyI finite.insertI is-renaming-renaming-wrt}$)

have $disjoint\text{-vars}$: $\bigwedge C. vars\text{-cls } (C \cdot \rho\rho) \cap vars\text{-cls } (add\text{-mset } K D) = \{\}$
by (*simp add:* $\rho\rho\text{-def vars-cls-subst-renaming-disj}$)

have $K \cdot l \gamma_D = - (L \cdot l \mu_L \cdot l \gamma_C)$
proof –
have $L \cdot l \mu_L \cdot l \gamma_C = L \cdot l \gamma_C$
using $\langle \mu_L \odot \gamma_C = \gamma_C \rangle$
by (*metis subst-lit-comp-subst*)
thus *?thesis*
using *resolveI* **by** *simp*
qed

have $\exists \mu. Unification.mgu (atm\text{-of } L \cdot a \mu_L \cdot a \rho\rho) (atm\text{-of } K) = Some \mu$
proof (*rule ex-mgu-if-subst-eq-subst-and-disj-vars*)
show $vars\text{-term } (atm\text{-of } L \cdot a \mu_L \cdot a \rho\rho) \cap vars\text{-lit } K = \{\}$
using $disjoint\text{-vars}[of \{\#L \cdot l \mu_L\#}]$ **by** *auto*
next
have $vars\text{-term } (atm\text{-of } L \cdot a \mu_L) \subseteq subst\text{-domain } \gamma_C$
by (*metis* $\langle \mu_L \odot \gamma_C = \gamma_C \rangle atm\text{-of-subst-lit ground-conf is-ground-cls-add-mset}$)

subst-cls-add-mset subst-lit-comp-subst vars-lit-subset-subst-domain-if-grounding
hence *atm-of L · a μ_L · a ρρ · a rename-subst-domain ρρ γ_C = atm-of L · a μ_L*
· a γ_C
using *ren-ρρ*
by (*simp add: is-renaming-iff renaming-cancels-rename-subst-domain*)
thus *atm-of L · a μ_L · a ρρ · a rename-subst-domain ρρ γ_C = atm-of K · a γ_D*
using $\langle K \cdot l \gamma_D = - (L \cdot l \mu_L \cdot l \gamma_C) \rangle$
by (*metis atm-of-subst-lit atm-of-uminus*)
qed
then obtain $\mu\mu$ **where** *ingmu-μμ: is-ingmu μμ* $\{\{atm-of L \cdot a \mu_L \cdot a \rho\rho, atm-of K\}\}$
using *is-ingmu-if-mgu-eq-Some* **by** *auto*

show *?thesis*
unfolding *S₃-def* $\langle add-mset LL CC \cdot \mu_L = add-mset L CCC \cdot \mu_L \rangle$
using *resolve.resolveI[OF* $\langle \Gamma = trail-propagate \Gamma' K D \gamma_D \rangle$ $\langle K \cdot l \gamma_D = - (L$
· l μ_L · l γ_C) *ren-ρρ*
is-renaming-id-subst, unfolded subst-atm-id-subst subst-cls-id-subst atm-of-subst-lit,
OF disjoint-vars ingmu-μμ is-grounding-merge-if-mem-then-else, of N β U
CCC · μ_L
by *auto*
qed

The following lemma corresponds to Lemma 7 in the paper.

lemma *no-backtrack-after-conflict-if:*
assumes *conf: conflict N β S1 S2* **and** *trail-S2: state-trail S1 = trail-propagate*
 $\Gamma L C \gamma$
shows $\nexists S4. backtrack N \beta S2 S4$
proof –
from *trail-S2 conf* **have** *state-trail S2 = trail-propagate Γ L C γ*
unfolding *conflict.simps* **by** *auto*
then show *?thesis*
unfolding *backtrack.simps propagate-lit-def decide-lit-def*
by *auto*
qed

lemma *skip-state-trail: skip N β S S' ⇒ suffix (state-trail S') (state-trail S)*
by (*auto simp: suffix-def elim: skip.cases*)

lemma *factorize-state-trail: factorize N β S S' ⇒ state-trail S' = state-trail S*
by (*auto elim: factorize.cases*)

lemma *resolve-state-trail: resolve N β S S' ⇒ state-trail S' = state-trail S*
by (*auto elim: resolve.cases*)

lemma *mempty-not-in-initial-clauses-if-run-leads-to-trail:*
assumes
*reg-run: (regular-scl N β)** initial-state S1* **and**
trail-lit: state-trail S1 = Lc # Γ

shows $\{\#\} \mid \notin \mid N$
proof (*rule notI*)
assume $\{\#\} \mid \in \mid N$
then obtain γ **where** *conflict* $N \beta$ *initial-state* $([], \{\mid\}, \text{Some}(\{\#\}, \gamma))$
using *conflict-initial-state-if-empty-in-initial-clauses* **by** *auto*
moreover hence $\nexists S'$. *local.scl* $N \beta$ $([], \{\mid\}, \text{Some}(\{\#\}, \gamma)) S'$ **for** γ
using *no-more-step-if-conflict-mempty* **by** *simp*
ultimately show *False*
using *trail-lit*
by (*metis* (*no-types*, *opaque-lifting*) *conflict-initial-state-only-with-mempty* *converse-rtranclpE*
list.discI *prod.sel(1)* *reasonable-if-regular* *reg-run* *regular-scl-def* *scl-if-reasonable*
state-trail-def)
qed

lemma *conflict-with-literal-gets-resolved*:

assumes
trail-lit: *state-trail* $S1 = Lc \# \Gamma$ **and**
conf: *conflict* $N \beta S1 S2$ **and**
resolution: $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{*} S2 Sn$ **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* $(\text{state-trail } Sn) (\text{state-trail } S1)$
proof –
obtain $S0$ **where** *propa*: *propagate* $N \beta S0 S1$
using *mempty-not-in-init-clss* *before-reasonable-conflict* $[OF \text{conf} \langle \text{learned-nonempty } S1 \rangle$
 $\langle \text{trail-propagated-or-decided}' N \beta S1 \rangle \langle \text{no-conflict-after-decide}' N \beta S1 \rangle]$ **by**
metis

from *trail-lit* *propa* **have** \neg *is-decision-lit* Lc
by (*auto* *simp*: *propagate-lit-def* *is-decision-lit-def* *elim!*: *propagate.cases*)

show *?thesis*

proof (*rule conjI*)

show \neg *is-decision-lit* Lc

by (*rule* $\langle \neg$ *is-decision-lit* $Lc \rangle$)

next

show *strict-suffix* $(\text{state-trail } Sn) (\text{state-trail } S1)$

unfolding *strict-suffix-def*

proof (*rule conjI*)

from *conf* **have** *state-trail* $S2 = \text{state-trail } S1$

by (*auto* *elim*: *conflict.cases*)

moreover from *resolution* **have** *suffix* $(\text{state-trail } Sn) (\text{state-trail } S2)$

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

```

    case base
    thus ?case
      by simp
  next
  case (step y z)
  from step.hyps(2) have suffix (state-trail z) (state-trail y)
    by (auto simp: suffix-def factorize-state-trail resolve-state-trail
      dest: skip-state-trail)
  with step.IH show ?case
    by (auto simp: suffix-def)
  qed
  ultimately show suffix (state-trail Sn) (state-trail S1)
    by simp
  next
  from backtrack <¬ is-decision-lit Lc> show state-trail Sn ≠ state-trail S1
    unfolding trail-lit
    by (auto simp: decide-lit-def is-decision-lit-def elim!: backtrack.cases)
  qed
  qed
  qed

```

13 Clause Redundancy

definition *ground-redundant* where

$ground\text{-}redundant\ lt\ N\ C \longleftrightarrow \{D \in N. lt\ D\ C\} \models_e \{C\}$

definition *redundant* where

$redundant\ lt\ N\ C \longleftrightarrow$
 $(\forall C' \in grounding\text{-}of\text{-}cls\ C. ground\text{-}redundant\ lt\ (grounding\text{-}of\text{-}class\ N)\ C')$

lemma *redundant* $lt\ N\ C \longleftrightarrow (\forall C' \in grounding\text{-}of\text{-}cls\ C. \{D' \in grounding\text{-}of\text{-}class\ N. lt\ D'\ C'\} \models_e \{C'\})$

by (simp add: redundant-def ground-redundant-def)

lemma *ground-redundant-iff*:

$ground\text{-}redundant\ lt\ N\ C \longleftrightarrow (\exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C))$

proof (rule iffI)

assume red: *ground-redundant* $lt\ N\ C$

show $\exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C)$

proof (intro exI conjI)

show $\{D \in N. lt\ D\ C\} \subseteq N$

by simp

next

show $\{D \in N. lt\ D\ C\} \models_e \{C\}$

using red by (simp add: ground-redundant-def)

next

show $\forall D \in \{D \in N. lt\ D\ C\}. lt\ D\ C$

by simp

qed

next
assume $\exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C)$
then show *ground-redundant* *lt* *N* *C*
unfolding *ground-redundant-def*
by (*smt* (*verit*, *ccfv-SIG*) *mem-Collect-eq subset-iff true-clss-mono*)
qed

lemma *ground-redundant-is-ground-standard-redundancy*:
fixes *lt*
defines $Red-F_{\mathcal{G}} \equiv \lambda N. \{C. \text{ground-redundant } lt\ N\ C\}$
shows $Red-F_{\mathcal{G}}\ N = \{C. \exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C)\}$
by (*auto simp*: *Red-F_G-def ground-redundant-iff*)

lemma *redundant-is-standard-redundancy*:
fixes *lt* \mathcal{G}_F \mathcal{G}_{F_s} *Red-F_G* *Red-F*
defines
 $\mathcal{G}_F \equiv \text{grounding-of-clcs}$ **and**
 $\mathcal{G}_{F_s} \equiv \text{grounding-of-clss}$ **and**
 $Red-F_{\mathcal{G}} \equiv \lambda N. \{C. \text{ground-redundant } lt\ N\ C\}$ **and**
 $Red-F \equiv \lambda N. \{C. \text{redundant } lt\ N\ C\}$
shows $Red-F\ N = \{C. \forall D \in \mathcal{G}_F\ C. D \in Red-F_{\mathcal{G}}\ (\mathcal{G}_{F_s}\ N)\}$
using *Red-F-def Red-F_G-def G_{F_s}-def G_F-def redundant-def* **by** *auto*

lemma *ground-redundant-if-strict-subset*:
assumes $D \in N$ **and** $D \subset\# C$
shows *ground-redundant* (*multp_{HO}* *R*) *N* *C*
using *assms*
unfolding *ground-redundant-def*
by (*metis* (*mono-tags*, *lifting*) *CollectI strict-subset-implies-multp_{HO} subset-mset.less-le true-clss-def true-clss-singleton true-clss-subclause*)

lemma *redundant-if-strict-subset*:
assumes $D \in N$ **and** $D \subset\# C$
shows *redundant* (*multp_{HO}* *R*) *N* *C*
unfolding *redundant-def*
proof (*rule ballI*)
fix *C'* **assume** $C' \in \text{grounding-of-clcs } C$
then obtain γ **where** $C' = C \cdot \gamma$ **and** *is-ground-subst* γ
by (*auto simp*: *grounding-of-clcs-def*)

show *ground-redundant* (*multp_{HO}* *R*) (*grounding-of-clss* *N*) *C'*
proof (*rule ground-redundant-if-strict-subset*)
from $\langle D \in N \rangle$ **show** $D \cdot \gamma \in \text{grounding-of-clss } N$
using $\langle \text{is-ground-subst } \gamma \rangle$
by (*auto simp*: *grounding-of-clss-def grounding-of-clcs-def*)
next
from $\langle D \subset\# C \rangle$ **show** $D \cdot \gamma \subset\# C'$
by (*simp add*: $\langle C' = C \cdot \gamma \rangle$ *subst-subset-mono*)
qed

qed

lemma *redundant-if-strict-subsumes*:

assumes $D \cdot \sigma \subset\# C$ **and** $D \in N$
shows *redundant* ($\text{multp}_{HO} R$) $N C$
unfolding *redundant-def*

proof (*rule ballI*)

fix C' **assume** $C' \in \text{grounding-of-clss } C$
then obtain γ **where** $C' = C \cdot \gamma$ **and** *is-ground-subst* γ
by (*auto simp: grounding-of-clss-def*)

show *ground-redundant* ($\text{multp}_{HO} R$) (*grounding-of-clss* N) C'

proof (*rule ground-redundant-if-strict-subset*)

from $\langle D \in N \rangle$ **show** $D \cdot \sigma \cdot \gamma \in \text{grounding-of-clss } N$

using $\langle \text{is-ground-subst } \gamma \rangle$

by (*metis* (*no-types*, *lifting*) *UN-iff ground-subst-ground-clss grounding-of-clss-ground*
grounding-of-clss-def insert-subset subst-clss-comp-subst
subst-clss-eq-grounding-of-clss-subset-eq)

next

from $\langle D \cdot \sigma \subset\# C \rangle$ **show** $D \cdot \sigma \cdot \gamma \subset\# C'$

by (*simp add: $\langle C' = C \cdot \gamma \rangle$ subst-subset-mono*)

qed

qed

lemma *ground-redundant-mono-strong*:

ground-redundant $R N C \implies (\bigwedge x. x \in N \implies R x C \implies S x C) \implies \text{ground-redundant}$
 $S N C$

unfolding *ground-redundant-def*

by (*simp add: true-clss-def*)

lemma *redundant-mono-strong*:

redundant $R N C \implies$

$(\bigwedge x y. x \in \text{grounding-of-clss } N \implies y \in \text{grounding-of-clss } C \implies R x y \implies S x$
 $y) \implies$

redundant $S N C$

by (*auto simp: redundant-def*)

intro: ground-redundant-mono-strong[of R grounding-of-clss N - S])

lemma *redundant-multp-if-redundant-strict-subset*:

redundant $(\subset\#) N C \implies \text{redundant} (\text{multp}_{HO} R) N C$

by (*auto intro: strict-subset-implies-multp_{HO} elim!: redundant-mono-strong*)

lemma *redundant-multp-if-redundant-subset*:

redundant $(\subset\#) N C \implies \text{redundant} (\text{multp} (\text{trail-less-ex } lt \ Ls)) N C$

by (*auto intro: subset-implies-multp elim!: redundant-mono-strong*)

lemma *not-bex-subset-mset-if-not-ground-redundant*:

assumes *is-ground-clss* C **and** *is-ground-clss* N

shows $\neg \text{ground-redundant} (\subset\#) N C \implies \neg (\exists D \in N. D \subset\# C)$

```

using assms unfolding ground-redundant-def
apply (simp add: true-cls-def true-cls-def)
apply (elim exE conjE)
apply (rule ballI)
subgoal premises prems for I D
  using prems(3)[rule-format, of D] prems(1,2,4-)
  apply simp
  by (meson mset-subset-eqD subset-mset.nless-le)
done

```

14 Trail-Induced Ordering

14.1 Miscellaneous Lemmas

lemma *pairwise-distinct-if-trail-consistent*:

```

fixes  $\Gamma$ 
defines  $Ls \equiv (\text{map } \text{fst } \Gamma)$ 
shows trail-consistent  $\Gamma \implies$ 
   $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$ 
unfolding Ls-def
proof (induction  $\Gamma$  rule: trail-consistent.induct)
  case Nil
  show ?case by simp
next
  case (Cons  $\Gamma$  L u)
  from Cons.hyps have L-distinct:
     $\forall x < \text{length } (\text{map } \text{fst } \Gamma). \text{map } \text{fst } \Gamma ! x \neq L$ 
     $\forall x < \text{length } (\text{map } \text{fst } \Gamma). \text{map } \text{fst } \Gamma ! x \neq - L$ 
  unfolding trail-defined-lit-def de-Morgan-disj
  unfolding image-set in-set-conv-nth not-ex de-Morgan-conj disj-not1
  by simp-all
  show ?case
  unfolding list.map prod.sel
  proof (intro allI impI)
  fix  $i j :: \text{nat}$ 
  assume i-lt:  $i < \text{length } (L \# \text{map } \text{fst } \Gamma)$  and j-lt:  $j < \text{length } (L \# \text{map } \text{fst } \Gamma)$ 
and  $i \neq j$ 
  show
     $(L \# \text{map } \text{fst } \Gamma) ! i \neq (L \# \text{map } \text{fst } \Gamma) ! j \wedge$ 
     $(L \# \text{map } \text{fst } \Gamma) ! i \neq - (L \# \text{map } \text{fst } \Gamma) ! j$ 
  proof (cases i)
  case 0
  thus ?thesis
  using L-distinct  $\langle i \neq j \rangle \langle j < \text{length } (L \# \text{map } \text{fst } \Gamma) \rangle$ 
  using gr0-conv-Suc by auto
next
  case (Suc k)
  then show ?thesis
  apply simp

```

using $i\text{-lt } j\text{-lt } \langle i \neq j \rangle L\text{-distinct Cons.IH}[\text{rule-format}]$
using $\text{less-Suc-eq-0-disj}$ **by** force
qed
qed
qed

14.2 Strict Partial Order

lemma $\text{irreflp-trail-less-if-trail-consistant}$:
 $\text{trail-consistent } \Gamma \implies \text{irreflp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$
using $\text{irreflp-trail-less}[OF$
 $\text{Clausal-Logic.uminus-not-id}'$
 $\text{Clausal-Logic.uminus-of-uminus-id}$
 $\text{pairwise-distinct-if-trail-consistent}]$
by assumption

lemma $\text{transp-trail-less-if-trail-consistant}$:
 $\text{trail-consistent } \Gamma \implies \text{transp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$
using $\text{transp-trail-less}[OF$
 $\text{Clausal-Logic.uminus-not-id}'$
 $\text{Clausal-Logic.uminus-of-uminus-id}$
 $\text{pairwise-distinct-if-trail-consistent}]$
by assumption

lemma $\text{asympt-trail-less-if-trail-consistant}$:
 $\text{trail-consistent } \Gamma \implies \text{asympt } (\text{trail-less } (\text{map } \text{fst } \Gamma))$
using $\text{asympt-trail-less}[OF$
 $\text{Clausal-Logic.uminus-not-id}'$
 $\text{Clausal-Logic.uminus-of-uminus-id}$
 $\text{pairwise-distinct-if-trail-consistent}]$
by assumption

14.3 Properties

lemma $\text{trail-defined-lit-if-trail-term-less}$:
assumes $\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma) (\text{atm-of } L) (\text{atm-of } K)$
shows $\text{trail-defined-lit } \Gamma L \text{ trail-defined-lit } \Gamma K$
proof –
from assms **have** $\text{atm-of } L \in \text{set } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma)$ **and** $\text{atm-of } K \in \text{set } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma)$
by $(\text{auto simp: trail-term-less-def})$
hence $\text{atm-of } L \in \text{atm-of } ' \text{fst } ' \text{ set } \Gamma$ **and** $\text{atm-of } K \in \text{atm-of } ' \text{fst } ' \text{ set } \Gamma$
by auto
hence $L \in \text{fst } ' \text{ set } \Gamma \vee - L \in \text{fst } ' \text{ set } \Gamma$ **and** $K \in \text{fst } ' \text{ set } \Gamma \vee - K \in \text{fst } ' \text{ set } \Gamma$
by $(\text{simp-all add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set})$
thus $\text{trail-defined-lit } \Gamma L$ **and** $\text{trail-defined-lit } \Gamma K$
by $(\text{simp-all add: trail-defined-lit-def})$
qed

lemma *trail-defined-cls-if-lt-defined*:
assumes *consistent-Γ*: *trail-consistent* Γ **and**
C-lt-D: $\text{multp}_{HO} (\text{lit-less} (\text{trail-term-less} (\text{map} (\text{atm-of } o \text{ fst}) \Gamma))) C D$ **and**
tr-def-D: *trail-defined-cls* ΓD **and**
lit-less-preserves-term-order: $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{==} (\text{atm-of } L1)$
(*atm-of* $L2$)
shows *trail-defined-cls* ΓC
proof –
from *multp_{HO}-implies-one-step*[*OF C-lt-D*]
obtain $I J K$ **where** *D-def*: $D = I + J$ **and** *C-def*: $C = I + K$ **and** $J \neq \{\#\}$
and
 $*$: $\forall k \in \#K. \exists x \in \#J. \text{lit-less} (\text{trail-term-less} (\text{map} (\text{atm-of } o \text{ fst}) \Gamma)) k x$
by *auto*

show *?thesis*
unfolding *trail-defined-cls-def*
proof (*rule ballI*)
fix L **assume** *L-in*: $L \in \# C$
show *trail-defined-lit* ΓL
proof (*cases* $L \in \# I$)
case *True*
then show *?thesis*
using *tr-def-D D-def*
by (*simp add: trail-defined-cls-def*)
next
case *False*
with *C-def L-in* **have** $L \in \# K$ **by** *fastforce*
then obtain L' **where** *L'-in*: $L' \in \# J$ **and** *lit-less* (*trail-term-less* (*map* (*atm-of*
o fst) Γ)) $L L'$
using $*$ **by** *blast*
hence (*trail-term-less* (*map* (*atm-of* *o fst*) Γ)) $^{==}$ (*atm-of* L) (*atm-of* L')
using *lit-less-preserves-term-order* **by** *metis*
thus *?thesis*
using *trail-defined-lit-if-trail-term-less(1)*
by (*metis* (*mono-tags*, *lifting*) *D-def L'-in atm-of-eq-atm-of sup2E tr-def-D*
trail-defined-cls-def trail-defined-lit-iff-defined-uminus union-iff)
qed
qed
qed

15 Dynamic Non-Redundancy

lemma *regular-run-if-skip-factorize-resolve-run*:
assumes (*skip* $N \beta \sqcup$ *factorize* $N \beta \sqcup$ *resolve* $N \beta$) $^{**} S S'$
shows (*regular-scl* $N \beta$) $^{**} S S'$
using *assms*
proof (*induction* S' *rule: rtranclp-induct*)
case *base*
show *?case* **by** *simp*

```

next
  case (step  $S' S''$ )
  from step.hyps(2) have  $scl\ N\ \beta\ S'\ S''$ 
    unfolding scl-def by blast
  with step.hyps(2) have reasonable-scl  $N\ \beta\ S'\ S''$ 
  using reasonable-scl-def decide-well-defined(4) decide-well-defined(5) skip-well-defined(2)
  by fast
  moreover from step.hyps(2) have  $\neg\ Ex\ (conflict\ N\ \beta\ S')$ 
  apply simp
  by (smt (verit, best) conflict.cases factorize.simps option.distinct(1) resolve.simps
skip.simps
  state-conflict-simp)
  ultimately have regular-scl  $N\ \beta\ S'\ S''$ 
  by (simp add: regular-scl-def)
  with step.IH show ?case
  by simp
qed

```

```

lemma not-trail-true-and-false-lit:
  trail-consistent  $\Gamma \implies \neg\ (trail\ true\ lit\ \Gamma\ L \wedge trail\ false\ lit\ \Gamma\ L)$ 
  apply (simp add: trail-true-lit-def trail-false-lit-def)
  by (metis (no-types, lifting) in-set-conv-nth list.set-map pairwise-distinct-if-trail-consistent
uminus-not-id')

```

```

lemma not-trail-true-and-false-cls:
  trail-consistent  $\Gamma \implies \neg\ (trail\ true\ cls\ \Gamma\ C \wedge trail\ false\ cls\ \Gamma\ C)$ 
  using not-trail-true-and-false-lit
  by (metis trail-false-cls-def trail-true-cls-def)

```

```

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  $R\ L1\ L2 \implies R^{==}\ (atm\ of\ L1)\ (atm\ of\ L2)$ 
  by (cases  $L1$ ; cases  $L2$ ) simp-all

```

```

theorem learned-clauses-in-regular-runs-invars:
  fixes  $\Gamma\ lit-less$ 
  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**

$\text{trail-ord} \equiv \text{multp}_{HO} (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma)))$

shows $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-clss } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \langle \text{grounding-of-clss } (\text{fset } N \cup \text{fset } U) \rangle \wedge$

$C \notin (\text{fset } N \cup \text{fset } U) \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) C)$

proof –

from *conflict* **have** *almost-no-conflict-with-trail* $N \beta S1$

using $\langle \text{almost-no-conflict-with-trail } N \beta S0 \rangle$

by (*rule* *conflict-preserves-almost-no-conflict-with-trail*)

from *conflict* **obtain** $C1 \gamma1$ **where** *conflict-S1*: $\text{state-conflict } S1 = \text{Some } (C1, \gamma1)$

by (*smt* (*verit*, *best*) *conflict.simps* *state-conflict-simp*)

with *backtrack* **obtain** $Cn \gamma n$ **where** *conflict-Sn*: $\text{state-conflict } Sn = \text{Some } (Cn, \gamma n)$ **and** $Cn \neq \{\#\}$

by (*auto elim*: *backtrack.cases*)

with *resolution* *conflict-S1* **have** $C1 \neq \{\#\}$

proof (*induction* Sn *arbitrary*: $C1 \gamma1 Cn \gamma n$ *rule*: *tranclp-induct*)

case (*base* y)

then show $?case$

by (*auto elim*: *skip.cases* *factorize.cases* *resolve.cases*)

next

case (*step* $y z$)

from *step.prem*s *step.hyps* **obtain** $Cy \gamma y$ **where**

conf-y: $\text{state-conflict } y = \text{Some } (Cy, \gamma y)$ **and** $Cy \neq \{\#\}$

by (*auto elim*: *skip.cases* *factorize.cases* *resolve.cases*)

show $?case$

using *step.IH*[*OF* - *conf-y* $\langle Cy \neq \{\#\} \rangle$] *step.prem*s

by *simp*

qed

from *conflict* **have** *sound-S1*: $\text{sound-state } N \beta S1$ **and** *ground-false-closures* $S1$

using *sound-S0* $\langle \text{ground-false-closures } S0 \rangle$

by (*simp-all* *add*: *conflict-preserves-sound-state* *conflict-preserves-ground-false-closures*)

with *resolution* **have** *sound-Sn*: $\text{sound-state } N \beta Sn$ **and** *ground-false-closures* Sn

by (*induction* *rule*: *tranclp-induct*)

(*auto* *intro*:

skip-preserves-sound-state

skip-preserves-ground-false-closures

factorize-preserves-sound-state

factorize-preserves-ground-false-closures)

```

    resolve-preserves-sound-state
    resolve-preserves-ground-false-closures)

from conflict ⟨trail-closures-false' S0⟩ have trail-closures-false' S1
  by (simp add: conflict-preserves-trail-closures-false')

from conflict-Sn sound-Sn have fset N  $\models_{\mathcal{G}e} \{Cn\}$ 
  by (auto simp add: sound-state-def)

from conflict-S1 ⟨ground-false-closures S1⟩ have trail-S1-false-C1-γ1:
  trail-false-cls (state-trail S1) (C1 · γ1)
  by (auto simp add: ground-false-closures-def)

from conflict-Sn ⟨ground-false-closures Sn⟩ have trail-Sn-false-Cn-γn:
  trail-false-cls (state-trail Sn) (Cn · γn)
  by (auto simp add: ground-false-closures-def)

from resolution have suffix (state-trail Sn) (state-trail S1) ∧
  ( $\exists Cn \gamma n. \text{state-conflict } Sn = \text{Some } (Cn, \gamma n) \wedge \text{trail-false-cls } (state-trail S1)$ 
  ( $Cn \cdot \gamma n$ ))
proof (induction Sn rule: tranclp-induct)
  case (base S2)
  thus ?case
  proof (elim sup2E)
    assume skip N β S1 S2
    thus ?thesis
    using conflict-S1 skip.simps suffix-ConsI trail-S1-false-C1-γ1 by fastforce
  next
    assume factorize N β S1 S2
    moreover with ⟨ground-false-closures S1⟩ have ground-false-closures S2
      by (auto intro: factorize-preserves-ground-false-closures)
    ultimately show ?thesis
    by (cases N β S1 S2 rule: factorize.cases) (simp add: ground-false-closures-def)
  next
    assume resolve N β S1 S2
    moreover with ⟨ground-false-closures S1⟩ have ground-false-closures S2
      by (auto intro: resolve-preserves-ground-false-closures)
    ultimately show ?thesis
    by (cases N β S1 S2 rule: resolve.cases) (simp add: ground-false-closures-def)
  qed
next
  case (step Sm Sm')
  from step.hyps(2) have suffix (state-trail Sm') (state-trail Sm)
    by (auto elim!: skip.cases factorize.cases resolve.cases intro: suffix-ConsI)
  with step.IH have suffix (state-trail Sm') (state-trail S1)
    by force

from step.hyps(1) sound-S1 have sound-Sm: sound-state N β Sm
  by (induction rule: tranclp-induct)

```

```

      (auto intro: skip-preserves-sound-state factorize-preserves-sound-state
        resolve-preserves-sound-state)

from step.hyps(1) ⟨trail-closures-false' S1⟩ have trail-closures-false' Sm
  by (induction rule: tranclp-induct)
  (auto intro: skip-preserves-trail-closures-false' factorize-preserves-trail-closures-false'
    resolve-preserves-trail-closures-false')

from step.hyps(1) ⟨ground-false-closures S1⟩ have ground-false-closures Sm
  by (induction rule: tranclp-induct)
  (auto intro: skip-preserves-ground-false-closures factorize-preserves-ground-false-closures
    resolve-preserves-ground-false-closures)

from step.IH obtain Cm γm where
  conflict-Sm: state-conflict Sm = Some (Cm, γm) and
  trail-false-Cm-γm: trail-false-cls (state-trail S1) (Cm · γm)
  using step.prem1 step.hyps(2) ⟨suffix (state-trail Sm') (state-trail Sm)⟩
  ⟨suffix (state-trail Sm') (state-trail S1)⟩
  unfolding suffix-def
  by auto

from step.hyps(2) show ?case
proof (elim sup2E)
  assume skip N β Sm Sm'
  thus ?thesis
    using ⟨suffix (state-trail Sm') (state-trail S1)⟩
    using conflict-Sm skip.simps trail-false-Cm-γm by auto
next
  assume factorize N β Sm Sm'
  thus ?thesis
proof (cases N β Sm Sm' rule: factorize.cases)
  case (factorizeI L γ L' μ Γ U D)
    with conflict-Sm have Cm-def: Cm = add-mset L' (add-mset L D) and
    γm-def: γm = γ
      by simp-all
    have trail-false-cls (state-trail S1) ((D + {#L#}) · μ · γ)
    proof (rule trail-false-cls-subst-mgu-before-grounding[of - - L L'])
      show trail-false-cls (state-trail S1) ((D + {#L, L'#}) · γ)
    by (metis Cm-def γm-def empty-neutral(1) trail-false-Cm-γm union-commute
      union-mset-add-mset-right)
    next
      show is-ingu μ {{atm-of L, atm-of L'}}
      using factorizeI(4) by fastforce
    next
      have is-unifier γ {atm-of L, atm-of L'}
      unfolding is-unifier-alt[of {atm-of L, atm-of L'}, simplified]
      by (metis atm-of-subst-lit factorizeI(3))
    thus is-unifiers γ {{atm-of L, atm-of L'}}
      by (simp add: is-unifiers-def)

```

```

qed
with factorizeI(2) show ?thesis
  using ⟨suffix (state-trail Sm') (state-trail S1)⟩
  by (metis add-mset-add-single state-conflict-simp)
qed
next
assume resolve N β Sm Sm'
thus ?thesis
proof (cases N β Sm Sm' rule: resolve.cases)
case (resolveI Γ Γ' K D γD L γC ρC ρD C μ γ U)
with conflict-Sm have Cm-def: Cm = add-mset L C and γm-def: γm =
 $\gamma_C$ 
  by simp-all
hence tr-false-S1-conf: trail-false-cls (state-trail S1) (add-mset L C · γC)
  using trail-false-Cm-γm by simp

from sound-Sm have sound-trail N Γ
  unfolding resolveI(1,2) sound-state-def
  by simp

from ⟨ground-false-closures Sm⟩ have
  ground-conf: is-ground-cls (add-mset L C · γC) and
  ground-prop: is-ground-cls (add-mset K D · γD)
  unfolding resolveI(1,2) ⟨Γ = trail-propagate Γ' K D γD⟩
  by (simp-all add: ground-false-closures-def ground-closures-def propa-
gate-lit-def)
hence
  ∀ L ∈ #add-mset L C. L · l ρC · l γ = L · l γC
  ∀ K ∈ #add-mset K D. K · l ρD · l γ = K · l γD
  using resolveI merge-of-renamed-groundings by metis+

have atm-of L · a ρC · a γ = atm-of K · a ρD · a γ
  using ⟨K · l γD = - (L · l γC)⟩
  ⟨∀ L ∈ #add-mset L C. L · l ρC · l γ = L · l γC⟩[rule-format, of L, simplified]
  ⟨∀ K ∈ #add-mset K D. K · l ρD · l γ = K · l γD⟩[rule-format, of K,
simplified]
  by (metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit)
hence is-unifiers γ {{atm-of L · a ρC, atm-of K · a ρD}}
  by (simp add: is-unifiers-def is-unifier-alt)
hence μ ⊙ γ = γ
  using ⟨is-ingu μ {{atm-of L · a ρC, atm-of K · a ρD}}⟩
  by (auto simp: is-ingu-def)
hence C · ρC · μ · γ = C · γC and D · ρD · μ · γ = D · γD
  using ⟨∀ L ∈ #add-mset L C. L · l ρC · l γ = L · l γC⟩ ⟨∀ K ∈ #add-mset K
D. K · l ρD · l γ = K · l γD⟩
by (metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst
subst-lit-comp-subst)+
hence (C · ρC + D · ρD) · μ · γ = C · γC + D · γD
  by (metis subst-cls-comp-subst subst-cls-union)

```

```

moreover have trail-false-cls (state-trail S1) (D ·  $\gamma_D$ )
proof (rule trail-false-cls-if-trail-false-suffix)
  show suffix  $\Gamma'$  (state-trail S1)
    using resolveI ⟨suffix (state-trail Sm') (state-trail S1)⟩
    by (metis (no-types, opaque-lifting) state-trail-simp suffix-order.trans
      suffix-trail-propagate)
next
  from ⟨trail-closures-false' Sm'⟩ have trail-closures-false  $\Gamma$ 
    unfolding resolveI(1,2)
    by (simp add: trail-closures-false'-def)
  thus trail-false-cls  $\Gamma'$  (D ·  $\gamma_D$ )
    using resolveI(3-)
    by (auto simp: propagate-lit-def elim: trail-closures-false.cases)
qed

ultimately have trail-false-cls (state-trail S1) ((C ·  $\varrho_C$  + D ·  $\varrho_D$ ) ·  $\mu$  ·  $\gamma$ )
  using tr-false-S1-conf
  by (metis add-mset-add-single subst-cls-union trail-false-cls-plus)
then show ?thesis
  using ⟨suffix (state-trail Sm') (state-trail S1)⟩
  using resolveI(1,2) by simp
qed
qed
qed

with conflict-Sn have
  suffix (state-trail Sn) (state-trail S1) and
  tr-false-S1-Cn- $\gamma_n$ : trail-false-cls (state-trail S1) (Cn ·  $\gamma_n$ )
  by auto

from ⟨ground-false-closures Sn⟩ conflict-Sn have Cn- $\gamma_n$ -in: Cn ·  $\gamma_n$  ∈ ground-
ing-of-cls Cn
  unfolding ground-false-closures-def ground-closures-def
  using grounding-of-cls-ground grounding-of-subst-cls-subset
  by fastforce

from sound-S1 have sound-trail-S1: sound-trail N (state-trail S1)
  by (auto simp add: sound-state-def)

have tr-consistent-S1: trail-consistent (state-trail S1)
  using conflict-preserved-trail-lits-consistent[OF conflict ⟨trail-lits-consistent S0⟩]
  by (simp add: trail-lits-consistent-def)

have  $\forall L \in \#Cn \cdot \gamma_n$ . trail-defined-lit (state-trail S1) L
  using tr-false-S1-Cn- $\gamma_n$  trail-defined-lit-iff-true-or-false trail-false-cls-def by
blast
  hence trail-interp (state-trail S1)  $\models Cn \cdot \gamma_n \longleftrightarrow$  trail-true-cls (state-trail S1)
(Cn ·  $\gamma_n$ )

```

using *tr-consistent-S1 trail-true-cls-iff-trail-interp-entails* **by** *auto*
hence *not-trail-S1-entails-Cn- γ n: \neg trail-interp (state-trail S1) \models $\{Cn \cdot \gamma n\}$*
using *tr-false-S1-Cn- γ n not-trail-true-and-false-cls[OF tr-consistent-S1]* **by**
auto

have *trail-defined-cls (state-trail S1) (Cn · γ n)*
using $\langle \forall L \in \#Cn \cdot \gamma n. \text{trail-defined-lit (state-trail S1) L} \rangle$ *trail-defined-cls-def*
by *blast*

have $\{\#\} \mid \notin \mid N$
by (*rule mempty-not-in-initial-clauses-if-non-empty-regular-conflict[OF conflict-S1 $\langle C1 \neq \{\#\} \rangle$*
 $\langle \text{almost-no-conflict-with-trail } N \beta S1 \rangle$ *sound-S1 $\langle \text{ground-false-closures } S1 \rangle$*)

then obtain *S* **where** *propagate N β S S0*
using *before-reasonable-conflict[OF conflict $\langle \text{learned-nonempty } S0 \rangle$*
 $\langle \text{trail-propagated-or-decided}' N \beta S0 \rangle$ $\langle \text{no-conflict-after-decide}' N \beta S0 \rangle$
by *auto*

have *state-learned S = state-learned S0*
using $\langle \text{propagate } N \beta S S0 \rangle$ **by** (*auto simp add: propagate.simps*)
also from *conflict* **have** $\dots = \text{state-learned } S1$
by (*auto simp add: conflict.simps*)
finally have *state-learned S = state-learned S1*
by *assumption*

have *state-conflict S = None*
using $\langle \text{propagate } N \beta S S0 \rangle$ **by** (*auto simp add: propagate.simps*)

from *conflict* **have** *state-trail S1 = state-trail S0*
by (*smt (verit) conflict.cases state-trail-simp*)

obtain *L C γ* **where** *trail-S0-eq: state-trail S0 = trail-propagate (state-trail S)*
L C γ
using $\langle \text{propagate } N \beta S S0 \rangle$ **unfolding** *propagate.simps* **by** *auto*

with *backtrack* **have** *strict-suffix (state-trail Sn) (state-trail S0)*
using *conflict-with-literal-gets-resolved[OF - conflict resolution[THEN tran-
clp-into-rtranclp] -*
 $\langle \{\#\} \mid \notin \mid N \rangle$ $\langle \text{learned-nonempty } S0 \rangle$ $\langle \text{trail-propagated-or-decided}' N \beta S0 \rangle$
 $\langle \text{no-conflict-after-decide}' N \beta S0 \rangle$
by (*metis (no-types, lifting) propagate-lit-def*)
hence *suffix (state-trail Sn) (state-trail S)*
unfolding *trail-S0-eq propagate-lit-def*
by (*metis suffix-Cons suffix-order.le-less suffix-order.less-irrefl*)

moreover have $\neg \text{trail-defined-lit (state-trail S) (L \cdot l \gamma)}$
proof –
have *trail-consistent (state-trail S0)*
using $\langle \text{state-trail } S1 = \text{state-trail } S0 \rangle$ $\langle \text{trail-consistent (state-trail } S1) \rangle$ **by**

```

simp
  thus ?thesis
    by (smt (verit, best) Pair-inject list.distinct(1) list.inject trail-S0-eq
        trail-consistent.cases propagate-lit-def)
qed

ultimately have  $\neg$  trail-defined-lit (state-trail Sn) (L · l  $\gamma$ )
  using trail-defined-lit-if-trail-defined-suffix by metis

moreover have trail-false-cls (state-trail Sn) (Cn ·  $\gamma$ n)
  using ⟨ground-false-closures Sn⟩ conflict-Sn by (auto simp add: ground-false-closures-def)

ultimately have L · l  $\gamma$   $\notin$ # Cn ·  $\gamma$ n  $\wedge$   $\neg$  (L · l  $\gamma$ )  $\notin$ # Cn ·  $\gamma$ n
  unfolding trail-false-cls-def trail-false-lit-def trail-defined-lit-def
  by (metis uminus-of-uminus-id)

have no-conf-at-S:  $\nexists$  S'. conflict N  $\beta$  S S'
proof (rule nex-conflict-if-no-conflict-with-trail'')
  show state-conflict S = None
    using ⟨propagate N  $\beta$  S S0⟩ by (auto elim: propagate.cases)
next
  show {#} | $\notin$ | N
    by (rule ⟨{#} | $\notin$ | N⟩)
next
  show learned-nonempty S
    using ⟨learned-nonempty S0⟩ ⟨state-learned S = state-learned S0⟩
    by (simp add: learned-nonempty-def)
next
  show no-conflict-with-trail N  $\beta$  (state-learned S) (state-trail S)
    using ⟨almost-no-conflict-with-trail N  $\beta$  S0⟩
    using ⟨propagate N  $\beta$  S S0⟩
    by (auto simp: almost-no-conflict-with-trail-def ⟨state-learned S = state-learned
S0⟩
        propagate-lit-def is-decision-lit-def elim!: propagate.cases)
qed

have conf-at-S-if:  $\exists$  S'. conflict N  $\beta$  S S'
  if D-in: D  $\in$  grounding-of-cls (fset N  $\cup$  fset U) and
  tr-false-D: trail-false-cls (state-trail S) D
  for D
  using ex-conflict-if-trail-false-cls[OF tr-false-D D-in]
  unfolding U-def ⟨state-learned S = state-learned S1⟩[symmetric]
  ⟨state-conflict S = None⟩[symmetric]
  by simp

have not-gr-red-Cn- $\gamma$ n:
   $\neg$  ground-redundant trail-ord (grounding-of-cls (fset N  $\cup$  fset U)) (Cn ·  $\gamma$ n)
proof (rule notI)
  define gnds-le-Cn- $\gamma$ n where

```

$gnds-le-Cn-\gamma n = \{D \in \text{grounding-of-clss } (fset\ N \cup fset\ U). \text{trail-ord } D\ (Cn \cdot \gamma n)\}$

assume *ground-redundant trail-ord* (*grounding-of-clss* (*fset* $N \cup fset\ U$)) ($Cn \cdot \gamma n$)

hence $gnds-le-Cn-\gamma n \models_e \{Cn \cdot \gamma n\}$

unfolding *ground-redundant-def* $gnds-le-Cn-\gamma n-def$ **by** *simp*

hence $\neg \text{trail-interp } (state-trail\ S1) \models_s gnds-le-Cn-\gamma n$

using *not-trail-S1-entails-Cn-\gamma n* **by** *auto*

then obtain D **where** $D-in: D \in gnds-le-Cn-\gamma n$ **and** $\neg \text{trail-interp } (state-trail\ S1) \models D$

by (*auto simp: true-clss-def*)

from $D-in$ **have**

$D-in: D \in \text{grounding-of-clss } (fset\ N \cup fset\ U)$ **and**

$D-le-Cn-\gamma n: \text{trail-ord } D\ (Cn \cdot \gamma n)$

unfolding $gnds-le-Cn-\gamma n-def$ **by** *simp-all*

from $D-le-Cn-\gamma n$ **have** *trail-defined-cls* (*state-trail* $S1$) D

using $\langle \text{trail-defined-cls } (state-trail\ S1)\ (Cn \cdot \gamma n) \rangle$

using *trail-defined-cls-if-lt-defined*

using $\Gamma-def\ lit-less-preserves-term-order\ tr-consistent-S1\ \text{trail-ord-def}$

by *metis*

hence *trail-false-cls* (*state-trail* $S1$) D

using $\langle \neg \text{trail-interp } (state-trail\ S1) \models D \rangle$

using $\langle \text{trail-consistent } (state-trail\ S1) \rangle\ \text{trail-interp-cls-if-trail-true}$

trail-true-or-false-cls-if-defined **by** *blast*

have $L \cdot l\ \gamma \notin\# D \wedge \neg (L \cdot l\ \gamma) \notin\# D$

proof $-$

from $D-le-Cn-\gamma n$ **have** $D-lt-Cn-\gamma n'$:

$\text{multp}_{HO} (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } \circ \text{fst}) (state-trail\ S1))))\ D$
($Cn \cdot \gamma n$)

unfolding *trail-ord-def* $\Gamma-def$.

have $\forall K \in\# Cn \cdot \gamma n. \neg K \in \text{fst } 'set\ (state-trail\ S1)$

by (*rule* $\langle \text{trail-false-cls } (state-trail\ S1)\ (Cn \cdot \gamma n) \rangle$ [*unfolded trail-false-cls-def trail-false-lit-def*])

hence $\forall K \in\# Cn \cdot \gamma n. \neg K \in \text{insert } (L \cdot l\ \gamma)\ (\text{fst } 'set\ (state-trail\ S))$

unfolding $\langle state-trail\ S1 = state-trail\ S0 \rangle$

$\langle state-trail\ S0 = \text{trail-propagate } (state-trail\ S)\ L\ C\ \gamma \rangle$

propagate-lit-def list.set image-insert prod.sel

by *simp*

hence $*$: $\forall K \in\# Cn \cdot \gamma n. \neg K \in \text{fst } 'set\ (state-trail\ S)$

by (*metis* $\langle L \cdot l\ \gamma \notin\# Cn \cdot \gamma n \wedge \neg (L \cdot l\ \gamma) \notin\# Cn \cdot \gamma n \rangle$ *insert-iff uminus-lit-swap*)

have $**$: $\forall K \in\# Cn \cdot \gamma n. \text{trail-term-less } (\text{map } (\text{atm-of } \circ \text{fst}) (state-trail\ S1))$
(*atm-of* K) (*atm-of* $(L \cdot l\ \gamma)$)

```

unfolding ⟨state-trail S1 = state-trail S0⟩
  ⟨state-trail S0 = trail-propagate (state-trail S) L C γ⟩
  propagate-lit-def comp-def prod.sel list.map
proof (rule ballI)
fix K assume K ∈# Cn · γn
with * have - K ∈ fst ‘ set (state-trail S)
  by metis
hence atm-of K ∈ set (map (λx. atm-of (fst x)) (state-trail S))
by (metis (no-types, lifting) atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
  comp-eq-dest-lhs image-comp image-cong list.set-map)
thus trail-term-less (atm-of (L ·l γ) # map (λx. atm-of (fst x)) (state-trail
S))
  (atm-of K) (atm-of (L ·l γ))
  using trail-term-less-Cons-if-mem by metis
qed

have ¬ (L ·l γ ∈# D ∨ - (L ·l γ) ∈# D)
proof (rule notI)
  obtain I J K where
    Cn · γn = I + J and D-def: D = I + K and J ≠ {#} and
    ∀ k∈#K. ∃ x∈#J. lit-less (trail-term-less (map (atm-of ∘ fst) (state-trail
S1))) k x
  using multpHO-implies-one-step[OF D-lt-Cn-γn]
  by auto
assume L ·l γ ∈# D ∨ - (L ·l γ) ∈# D
then show False
  unfolding D-def Multiset.union-iff
proof (elim disjE)
  show L ·l γ ∈# I ⇒ False
    using ⟨L ·l γ ∉# Cn · γn ∧ - (L ·l γ) ∉# Cn · γn⟩ ⟨Cn · γn = I +
J⟩ by simp
  next
    show - (L ·l γ) ∈# I ⇒ False
      using ⟨L ·l γ ∉# Cn · γn ∧ - (L ·l γ) ∉# Cn · γn⟩ ⟨Cn · γn = I +
J⟩ by simp
  next
    show L ·l γ ∈# K ⇒ False
      using ⟨L ·l γ ∉# Cn · γn ∧ - (L ·l γ) ∉# Cn · γn⟩ [THEN conjunct1]
      **[unfolded ⟨Cn · γn = I + J⟩ ⟨∀ k∈#K. ∃ x∈#J. lit-less (trail-term-less
(map (atm-of ∘ fst) (state-trail S1))) k x⟩
      by (metis (no-types, lifting) D-def Un-insert-right
        ⟨¬ trail-interp (state-trail S1) ⊨ D⟩
        ⟨state-trail S0 = trail-propagate (state-trail S) L C γ⟩
        ⟨state-trail S1 = state-trail S0⟩ ⟨trail-consistent (state-trail S1)⟩
image-insert
        insert-iff list.set(2) mk-disjoint-insert prod.sel(1) set-mset-union
        trail-interp-cls-if-trail-true propagate-lit-def trail-true-cls-def
        trail-true-lit-def)
  next

```

```

assume  $-(L \cdot l \gamma) \in \# K$ 
then obtain  $j$  where
   $j$ -in:  $j \in \# J$  and
     $uminus-L-\gamma$ -lt- $j$ : lit-less (trail-term-less (map (atm-of  $\circ$  fst) (state-trail
S1))) ( $-(L \cdot l \gamma)$ )  $j$ 
    using  $\langle \forall k \in \# K. \exists x \in \# J. \textit{lit-less} (\textit{trail-term-less} (\textit{map} (\textit{atm-of} \circ \textit{fst})$ 
(state-trail S1)))  $k$   $x$   $\rangle$ 
    by blast

from  $j$ -in have
  trail-term-less (map (atm-of  $\circ$  fst) (state-trail S1)) (atm-of  $j$ ) (atm-of ( $L$ 
 $\cdot l \gamma$ ))
  using **
  by (auto simp:  $\langle Cn \cdot \gamma n = I + J \rangle$ )

moreover from  $uminus-L-\gamma$ -lt- $j$  have trail-term-less (map (atm-of  $\circ$  fst)
(state-trail S1)) (atm-of ( $L \cdot l \gamma$ )) (atm-of  $j$ )
  using calculation lit-less-preserves-term-order by fastforce

moreover from tr-consistent-S1 have distinct (map (atm-of  $\circ$  fst)
(state-trail S1))
  using distinct-atm-of-trail-if-trail-consistent by metis

ultimately show False
  using asyp-trail-term-less[THEN asympD]
  by metis
qed
qed
thus ?thesis by simp
qed
hence trail-false-cls (state-trail S) D
  using D-in  $\langle \textit{trail-false-cls} (\textit{state-trail} \textit{S1}) \textit{D} \rangle$ 
  unfolding  $\langle \textit{state-trail} \textit{S1} = \textit{state-trail} \textit{S0} \rangle$ 
   $\langle \textit{state-trail} \textit{S0} = \textit{trail-propagate} (\textit{state-trail} \textit{S}) \textit{L} \textit{C} \gamma \rangle$ 
  by (simp add: propagate-lit-def subtrail-falseI)
thus False
  using no-conf-at-S conf-at-S-if[OF D-in] by metis
qed
hence  $\neg$  redundant trail-ord (fset N  $\cup$  fset U) Cn
  unfolding redundant-def
  using Cn-gamma-n-in by auto

moreover have  $Cn \cdot \gamma n \notin$  grounding-of-cls (fset N  $\cup$  fset U)
proof  $-$ 
  have is-ground-cls ( $Cn \cdot \gamma n$ )
    using Cn-gamma-n-in is-ground-cls-if-in-grounding-of-cls by blast

moreover have is-ground-cls (grounding-of-cls (fset N  $\cup$  fset U))
  by simp

```

ultimately have $\neg (\{D \in \text{grounding-of-clss } (fset\ N \cup fset\ U). \text{trail-ord } D (Cn \cdot \gamma n)\} \models_e \{Cn \cdot \gamma n\})$
using *not-gr-red-Cn- γn*
by (*simp add: ground-redundant-def*)
thus *?thesis*
using $\langle \text{suffix } (state\text{-trail } S_n) (state\text{-trail } S) \rangle \text{ conf-at-}S\text{-if no-conf-at-}S$
 $\text{trail-}S_n\text{-false-}Cn\text{-}\gamma n \text{ trail-false-cl-iff-trail-false-suffix}$ **by** *blast*
qed

moreover have $set\text{-mset } (Cn \cdot \gamma n) \notin set\text{-mset } \text{'grounding-of-clss } (fset\ N \cup fset\ U)$

proof (*rule notI*)

assume $set\text{-mset } (Cn \cdot \gamma n) \in set\text{-mset } \text{'grounding-of-clss } (fset\ N \cup fset\ U)$

then obtain D **where**

$D\text{-in: } D \in \text{grounding-of-clss } (fset\ N \cup fset\ U)$ **and**

$set\text{-mset-eq-}D\text{-}Cn\text{-}\gamma n: set\text{-mset } D = set\text{-mset } (Cn \cdot \gamma n)$

by (*auto simp add: image-iff*)

have $\neg \text{trail-interp } (state\text{-trail } S1) \models D$

unfolding $true\text{-cls-iff-set-mset-eq}[OF\ set\text{-mset-eq-}D\text{-}Cn\text{-}\gamma n]$

using *not-trail-S1-entails-Cn- γn*

by *simp*

have $trail\text{-defined-cl-cls } (state\text{-trail } S1) D$

using $\langle \forall L \in \# Cn \cdot \gamma n. \text{trail-defined-lit } (state\text{-trail } S1) L \rangle$

unfolding $set\text{-mset-eq-}D\text{-}Cn\text{-}\gamma n[symmetric]$

by (*simp add: trail-defined-cl-cls-def*)

hence $trail\text{-false-cl-cls } (state\text{-trail } S1) D$

using $\langle \neg \text{trail-interp } (state\text{-trail } S1) \models D \rangle$

using *tr-consistent-S1 trail-interp-cl-iff-trail-true trail-true-or-false-cl-iff-defined*

by *blast*

have $L \cdot l \gamma \notin \# D \wedge \neg (L \cdot l \gamma) \notin \# D$

using $\langle L \cdot l \gamma \notin \# Cn \cdot \gamma n \wedge \neg (L \cdot l \gamma) \notin \# Cn \cdot \gamma n \rangle$

unfolding $set\text{-mset-eq-}D\text{-}Cn\text{-}\gamma n[symmetric]$

by *assumption*

hence $trail\text{-false-cl-cls } (state\text{-trail } S) D$

using $D\text{-in } \langle \text{trail-false-cl-cls } (state\text{-trail } S1) D \rangle$

unfolding $\langle state\text{-trail } S1 = state\text{-trail } S0 \rangle$

$\langle state\text{-trail } S0 = \text{trail-propagate } (state\text{-trail } S) L C \gamma \rangle$

by (*simp add: propagate-lit-def subtrail-falseI*)

thus *False*

using *no-conf-at-S conf-at-S-iff[OF D-in]* **by** *metis*

qed

moreover have $\neg (\exists D \in fset\ N \cup fset\ U. \exists \sigma. D \cdot \sigma = Cn)$

by (*metis (no-types, lifting) Cn- γn -in Set.set-insert UnCI calculation(2)*)

grounding-of-clss-insert grounding-of-subst-cl-cls-subset subsetD)

moreover hence $Cn \notin \text{fset } N \cup \text{fset } U$
 using *subst-cls-id-subst* by *blast*

ultimately show *?thesis*
 using *conflict-Sn* by *simp*

qed

theorem *dynamic-non-redundancy-regular-scl:*

fixes Γ

assumes

regular-run: $(\text{regular-scl } N \beta)^{**}$ *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: $\text{backtrack } N \beta Sn Sn'$ and

lit-less-preserved-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 $(\text{regular-scl } N \beta)^{**}$ *initial-state* $Sn' \wedge$

$(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } ' \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin \text{fset } N \cup \text{fset } U \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) C$)

proof –

have *sound-state* $N \beta$ *initial-state*

by (*rule sound-initial-state*)

with *regular-run* **have** *sound-S0*: *sound-state* $N \beta S0$

by (*rule regular-run-sound-state*)

from *regular-run* **have** *learned-nonempty* $S0$

by (*induction* $S0$ *rule*: *rtranclp-induct*)

(*auto intro*: *scl-preserved-learned-nonempty* *reasonable-if-regular scl-if-reasonable*)

from *regular-run* **have** *trail-propagated-or-decided'* $N \beta S0$

by (*induction* $S0$ *rule*: *rtranclp-induct*)

(*auto intro*: *scl-preserved-trail-propagated-or-decided*

reasonable-if-regular scl-if-reasonable)

from *regular-run* **have** *no-conflict-after-decide'* $N \beta S0$

by (*induction* $S0$ *rule*: *rtranclp-induct*)

(*auto intro*: *reasonable-scl-preserved-no-conflict-after-decide'* *reasonable-if-regular*)

from *regular-run* **have** *almost-no-conflict-with-trail* $N \beta S0$

by (*induction* $S0$ *rule*: *rtranclp-induct*)

```

    (simp-all add: regular-scl-preserves-almost-no-conflict-with-trail)

from regular-run have trail-lits-consistent  $S0$ 
  by (induction  $S0$  rule: rtranclp-induct)
  (auto intro: scl-preserves-trail-lits-consistent reasonable-if-regular scl-if-reasonable)

from regular-run have trail-lits-consistent  $S0$ 
  by (induction  $S0$  rule: rtranclp-induct)
  (auto intro: scl-preserves-trail-lits-consistent reasonable-if-regular scl-if-reasonable)

from regular-run have trail-closures-false'  $S0$ 
  by (induction  $S0$  rule: rtranclp-induct)
  (auto intro: scl-preserves-trail-closures-false' reasonable-if-regular scl-if-reasonable)

from regular-run have ground-false-closures  $S0$ 
  by (induction  $S0$  rule: rtranclp-induct)
  (auto intro: scl-preserves-ground-false-closures reasonable-if-regular scl-if-reasonable)

from regular-run conflict have (regular-scl  $N \beta$ )** initial-state  $S1$ 
  by (meson regular-scl-def rtranclp.simps)
also from resolution have reg-run- $S1$ - $Sn$ : (regular-scl  $N \beta$ )** ...  $Sn$ 
  using regular-run-if-skip-factorize-resolve-run tranclp-into-rtranclp by metis
also have (regular-scl  $N \beta$ )** ...  $Sn'$ 
proof (rule r-into-rtranclp)
  from backtrack have scl  $N \beta Sn Sn'$ 
    by (simp add: scl-def)
  with backtrack have reasonable-scl  $N \beta Sn Sn'$ 
    using reasonable-scl-def decide-well-defined(6) by blast
  with backtrack show regular-scl  $N \beta Sn Sn'$ 
    unfolding regular-scl-def
    by (smt (verit) conflict.simps option.simps(3) backtrack.cases state-conflict-simp)
qed
finally have (regular-scl  $N \beta$ )** initial-state  $Sn'$  by assumption
thus ?thesis
  using learned-clauses-in-regular-runs-invars[OF sound- $S0$  <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  $S0$ >
    conflict resolution backtrack]
  using lit-less-preserves-term-order
  using U-def  $\Gamma$ -def trail-ord-def by presburger
qed

theorem dynamic-non-redundancy-projectable-strategy:
fixes
   $S1$  :: ( $f$ ,  $v$ ) state and
  lit-less :: (( $f$ ,  $v$ ) term  $\Rightarrow$  ( $f$ ,  $v$ ) term  $\Rightarrow$  bool)  $\Rightarrow$ 
  ( $f$ ,  $v$ ) term literal  $\Rightarrow$  ( $f$ ,  $v$ ) term literal  $\Rightarrow$  bool and

```

strategy **and** *strategy-init* **and** *proj*
defines
 $\Gamma \equiv \text{state-trail } S1$ **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
run: *strategy*** *strategy-init* *S0* **and**
conflict: *conflict* *N* β (*proj* *S0*) *S1* **and**
resolution: (*skip* *N* β \sqcup *factorize* *N* β \sqcup *resolve* *N* β)⁺⁺ *S1* *Sn* **and**
backtrack: *backtrack* *N* β *Sn* *Sn'* **and**
strategy-restricts-regular-scl:
 $\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta$ (*proj*
S) (*proj* *S'*) **and**
initial-state: *proj* *strategy-init* = *initial-state* **and**
lit-less-preserves-term-order: $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{==}$ (*atm-of* *L1*)
(*atm-of* *L2*)
shows $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$
 $C \cdot \gamma \notin \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$
 $\text{set-mset } (C \cdot \gamma) \notin \text{set-mset 'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$
 $C \notin \text{fset } N \cup \text{fset } U \wedge$
 $\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$
 $\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) C)$
proof –
from *backtrack* **have** *backtrack'*: *backtrack* *N* β *Sn* *Sn'*
by (*simp* *add*: *shortest-backtrack-strategy-def*)

have $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$
 $C \cdot \gamma \notin \text{grounding-of-cls} (\text{fset } N \cup \text{fset } U) \wedge$
 $\text{set-mset } (C \cdot \gamma) \notin \text{set-mset 'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$
 $C \notin \text{fset } N \cup \text{fset } U \wedge$
 $\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$
 $\neg \text{redundant } (\text{multp}_{HO} (\text{lit-less}$
 $\quad (\text{trail-term-less} (\text{map} (\text{atm-of } o \text{ fst}) (\text{state-trail } S1))))))$
 $(\text{fset } N \cup \text{fset } U) C)$
unfolding *U-def*
proof (*rule* *dynamic-non-redundancy-regular-scl*[*THEN* *conjunct2*])
show (*regular-scl* *N* β)** *initial-state* (*proj* *S0*)
using *run*
proof (*induction* *S0* *rule*: *rtranclp-induct*)
case *base*
thus ?*case*
unfolding *initial-state* **by** *simp*
next
case (*step* *y* *z*)
thus ?*case*
using *strategy-restricts-regular-scl*
by (*meson* *rtranclp.simps*)
qed

```

next
  from assms show conflict  $N \beta$  (proj  $S0$ )  $S1$ 
  by simp
next
  from assms show (skip  $N \beta \sqcup$  factorize  $N \beta \sqcup$  resolve  $N \beta$ )++  $S1 Sn$ 
  by simp
next
  from assms show backtrack  $N \beta Sn Sn'$ 
  by (simp add: shortest-backtrack-strategy-def)
next
  from assms show  $\bigwedge R L1 L2. \textit{lit-less } R L1 L2 \implies R^{==} (\textit{atm-of } L1) (\textit{atm-of } L2)$ 
  by simp
qed
thus ?thesis
  by (auto simp add: trail-ord-def  $\Gamma$ -def)
qed

```

corollary *dynamic-non-redundancy-strategy:*

```

fixes  $\Gamma$ 
assumes
  run: strategy** 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
  strategy-imp-regular-scl:  $\bigwedge S S'. \textit{strategy } S S' \implies \textit{regular-scl } N \beta S S'$  and
  lit-less-preserves-term-order:  $\bigwedge R L1 L2. \textit{lit-less } R L1 L2 \implies R^{==} (\textit{atm-of } L1)$ 
(atm-of  $L2$ )
defines
   $\Gamma \equiv \textit{state-trail } S1$  and
   $U \equiv \textit{state-learned } S1$  and
  trail-ord  $\equiv \textit{multp}_{HO} (\textit{lit-less } (\textit{trail-term-less } (\textit{map } (\textit{atm-of } o \textit{fst}) \Gamma)))$ 
shows  $(\exists C \gamma. \textit{state-conflict } Sn = \textit{Some } (C, \gamma) \wedge$ 
   $C \cdot \gamma \notin \textit{grounding-of-class } (\textit{fset } N \cup \textit{fset } U) \wedge$ 
   $\textit{set-mset } (C \cdot \gamma) \notin \textit{set-mset } ' \textit{grounding-of-class } (\textit{fset } N \cup \textit{fset } U) \wedge$ 
   $C \notin \textit{fset } N \cup \textit{fset } U \wedge$ 
   $\neg (\exists D \in \textit{fset } N \cup \textit{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$ 
   $\neg \textit{redundant trail-ord } (\textit{fset } N \cup \textit{fset } U) C)$ 
using dynamic-non-redundancy-projectable-strategy[of strategy initial-state - - -
 $\lambda x. x$ ]
using assms by blast

```

16 Static Non-Redundancy

lemma *before-regular-backtrack'*:

```

assumes
  run: (regular-scl  $N \beta$ )** initial-state  $S$  and
  step: backtrack  $N \beta S S'$ 
shows  $\exists S0 S1 S2 S3 S4. (\textit{regular-scl } N \beta)$ ** initial-state  $S0 \wedge$ 

```

$propagate\ N\ \beta\ S0\ S1 \wedge regular\text{-}scl\ N\ \beta\ S0\ S1 \wedge$
 $conflict\ N\ \beta\ S1\ S2 \wedge (factorize\ N\ \beta)^{**}\ S2\ S3 \wedge resolve\ N\ \beta\ S3\ S4 \wedge$
 $(skip\ N\ \beta \sqcup factorize\ N\ \beta \sqcup resolve\ N\ \beta)^{**}\ S4\ S$

proof –

from *run* **have** *sound-state* $N\ \beta\ S$
by (*induction* S *rule*: *rtranclp-induct*)
(simp-all add: scl-preserves-sound-state[OF scl-if-regular])

moreover from *run* **have** *almost-no-conflict-with-trail* $N\ \beta\ S$
by (*induction* S *rule*: *rtranclp-induct*)
(simp-all add: regular-scl-preserves-almost-no-conflict-with-trail)

moreover from *run* **have** *regular-conflict-resolution* $N\ \beta\ S$
by (*induction* S *rule*: *rtranclp-induct*)
(simp-all add: regular-scl-preserves-regular-conflict-resolution)

moreover from *run* **have** *ground-false-closures* S
by (*induction* S *rule*: *rtranclp-induct*)
(simp-all add: scl-preserves-ground-false-closures[OF scl-if-regular])

ultimately obtain $S0\ S1\ S2\ S3\ S4$ **where**
run-S0: $(regular\text{-}scl\ N\ \beta)^{**}$ *initial-state* $S0$ **and**
propa: $propagate\ N\ \beta\ S0\ S1\ regular\text{-}scl\ N\ \beta\ S0\ S1$ **and**
conft: $conflict\ N\ \beta\ S1\ S2$ **and**
facto: $(factorize\ N\ \beta)^{**}\ S2\ S3$ **and**
resol: $resolve\ N\ \beta\ S3\ S4$ **and**
reg-res: $(skip\ N\ \beta \sqcup factorize\ N\ \beta \sqcup resolve\ N\ \beta)^{**}\ S4\ S$
using *before-regular-backtrack*[*OF step*] **by** *blast*

thus *?thesis*
by *metis*

qed

theorem *static-non-subsumption-regular-scl*:

assumes

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

defines

$U \equiv state\text{-}learned\ S$

shows $\exists C\ \gamma. state\text{-}conflict\ S = Some\ (C, \gamma) \wedge \neg (\exists D\ |\in|\ N\ |\cup|\ U. subsumes\ D\ C)$

proof –

from *before-regular-backtrack'*[*OF run step*] **obtain** $S0\ S1\ S2\ S3\ S4$ **where**
run-S0: $(regular\text{-}scl\ N\ \beta)^{**}$ *initial-state* $S0$ **and**
propa: $propagate\ N\ \beta\ S0\ S1\ regular\text{-}scl\ N\ \beta\ S0\ S1$ **and**
conft: $conflict\ N\ \beta\ S1\ S2$ **and**
facto: $(factorize\ N\ \beta)^{**}\ S2\ S3$ **and**
resol: $resolve\ N\ \beta\ S3\ S4$ **and**
reg-res: $(skip\ N\ \beta \sqcup factorize\ N\ \beta \sqcup resolve\ N\ \beta)^{**}\ S4\ S$

using *before-regular-backtrack*[*OF step*] **by** *blast*

have (*regular-scl* $N \beta$)** *initial-state* $S1$
using *run-S0 propa*(2) **by** *simp*

moreover have *reg-res'*: (*skip* $N \beta \sqcup$ *factorize* $N \beta \sqcup$ *resolve* $N \beta$)⁺⁺ $S2 S$

proof –

have (*skip* $N \beta \sqcup$ *factorize* $N \beta \sqcup$ *resolve* $N \beta$)** $S2 S3$
using *facto*
by (*rule mono-rtranclp*[*rule-format, rotated*]) *simp*
also have (*skip* $N \beta \sqcup$ *factorize* $N \beta \sqcup$ *resolve* $N \beta$)⁺⁺ $S3 S4$
using *resol* **by** *auto*
finally show *?thesis*
using *reg-res* **by** *simp*

qed

ultimately obtain $C \gamma lt$ **where**

reg-run: (*regular-scl* $N \beta$)** *initial-state* S' **and**
conf: *state-conflict* $S = \text{Some } (C, \gamma)$ **and**
not-gen: $\neg (\exists D \in \text{fset } N \cup \text{fset } (\text{state-learned } S2). \exists \sigma. D \cdot \sigma = C)$ **and**
not-red: $\neg \text{redundant } (\text{multp}_{HO} (\text{standard-lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } \circ \text{fst}) (\text{state-trail } S2))))))$
 $(\text{fset } N \cup \text{fset } (\text{state-learned } S2)) C$

using *dynamic-non-redundancy-regular-scl*[*OF - confl - step, of standard-lit-less*]
using *standard-lit-less-preserves-term-less*
by *metis+*

from *not-red* **have** $\neg (\exists D \in \text{fset } N \cup \text{fset } (\text{state-learned } S2). \exists \sigma. D \cdot \sigma \subset \# C)$
using *redundant-if-strict-subsumes*
by (*metis union-fset*)

with *not-gen* **have** $\neg (\exists D \in \text{fset } N \cup \text{fset } (\text{state-learned } S2). \exists \sigma. D \cdot \sigma \subseteq \# C)$
using *subset-mset.order-iff-strict* **by** *blast*

hence *not-sub*: $\neg (\exists D \in \text{fset } N \cup \text{fset } (\text{state-learned } S2). \text{subsumes } D C)$
by (*simp add: subsumes-def*)

from *reg-res'* **have** *learned-S2*: *state-learned* $S2 = \text{state-learned } S$

proof (*induction* S)

case (*base* y)
thus *?case*
by (*auto elim: skip.cases factorize.cases resolve.cases*)

next

case (*step* $y z$)
from *step.hyps* **have** *state-learned* $y = \text{state-learned } z$
by (*auto elim: skip.cases factorize.cases resolve.cases*)
with *step.IH* **show** *?case*
by *simp*

qed

show *?thesis*

unfolding *U-def*
using *conf not-sub[unfolded learned-S2]*
by *simp*
qed

corollary *static-non-subsumption-projectable-strategy:*
fixes *strategy and strategy-init and proj*
assumes
*run: strategy** strategy-init S and*
step: backtrack N β (proj S) S' and
strategy-restricts-regular-scl:
 $\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta (\text{proj } S) (\text{proj } S')$ **and**
initial-state: proj strategy-init = initial-state
defines
 $U \equiv \text{state-learned } (\text{proj } S)$
shows $\exists C \gamma. \text{state-conflict } (\text{proj } S) = \text{Some } (C, \gamma) \wedge \neg (\exists D |\in| N |\cup| U. \text{subsumes } D C)$
unfolding *U-def*
proof (*rule static-non-subsumption-regular-scl*)
show (*regular-scl N β*)** *initial-state (proj S)*
using *run*
proof (*induction S rule: rtranclp-induct*)
case *base*
thus *?case*
unfolding *initial-state by simp*
next
case (*step y z*)
thus *?case*
using *strategy-restricts-regular-scl*
by (*meson rtranclp.simps*)
qed
next
from *step show backtrack N β (proj S) S'*
by *simp*
qed

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 $\exists C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \wedge \neg (\exists D |\in| N |\cup| U. \text{subsumes } D C)$
unfolding *U-def*
using *static-non-subsumption-projectable-strategy[of strategy initial-state - - - λx. x]*

```

    using assms by blast

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  $B \subseteq A$  and  $B \neq \{\}$ 
  shows  $\exists z \in B. \forall y. (y, z) \in R \longrightarrow y \notin B$ 
  using wf-on-iff-ex-minimal[THEN iffD1, rule-format, OF assms] .

```

lemma *wfE-min'*:

```

   $wf\ R \implies Q \neq \{\} \implies (\bigwedge z. z \in Q \implies (\bigwedge y. (y, z) \in R \implies y \notin Q) \implies thesis)$ 
 $\implies thesis$ 
  using wfE-min[of R - Q] by blast

```

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
  using wf-on-iff-ex-minimal[THEN iffD2, rule-format, OF assms] .

```

definition *inv-imagep-on* :: *'a* *set* \Rightarrow (*'b* \Rightarrow *'b* \Rightarrow *bool*) \Rightarrow (*'a* \Rightarrow *'b*) \Rightarrow *'a* \Rightarrow *'a* \Rightarrow *bool* **where**

```

  inv-imagep-on A R f =  $(\lambda x\ y. x \in A \wedge y \in A \wedge R\ (f\ x)\ (f\ y))$ 

```

lemma *wfp-on-inv-imagep*:

```

  assumes wf: wfp-on (f ' A) R
  shows wfp-on A (inv-imagep R f)
  unfolding wfp-on-iff-ex-minimal
proof (intro allI impI)
  fix B assume  $B \subseteq A$  and  $B \neq \{\}$ 
  hence  $\exists z \in f\ 'B. \forall y. R\ y\ z \longrightarrow y \notin f\ 'B$ 
    using wf[unfolded wfp-on-iff-ex-minimal, rule-format, of f ' B] by blast
  thus  $\exists z \in B. \forall y. \text{inv-imagep}\ R\ f\ y\ z \longrightarrow y \notin B$ 
    unfolding inv-imagep-def
    by (metis image-iff)
qed

```

definition *lex-prodp* **where**

```

  lex-prodp RA RB 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*]:
 $lex-prodp\ R_A\ R_B\ x\ y \longleftrightarrow (x, y) \in lex-prod\ \{(x, y). R_A\ x\ y\}\ \{(x, y). R_B\ x\ y\}$
unfolding *lex-prodp-def lex-prod-def* **by** *auto*

lemma *lex-prod-lex-prodp-iff*:
 $lex-prod\ \{(x, y). R_A\ x\ y\}\ \{(x, y). R_B\ x\ y\} = \{(x, y). lex-prodp\ R_A\ R_B\ x\ y\}$
unfolding *lex-prodp-def lex-prod-def* **by** *auto*

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*)
using *wf-on-lex-prod*[*of A - B -, to-pred, unfolded lex-prod-lex-prodp-iff, to-pred*]
.

corollary *wfp-lex-prodp*: $wfp\ R_A \implies wfp\ R_B \implies wfp\ (lex-prodp\ R_A\ R_B)$
using *wfp-on-lex-prodp*[*of UNIV - UNIV, simplified*].

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 (*rule wfp-on-if-convertible-to-wfp-on*)
show $wfp-on\ ((\lambda x. (f\ x, x))\ 'A)\ (lex-prodp\ Q\ S)$
proof (*rule wfp-on-subset*)
show $wfp-on\ (f\ 'A \times A)\ (lex-prodp\ Q\ S)$
by (*rule wfp-on-lex-prodp*[*OF wf-Q wf-S*])
next
show $(\lambda x. (f\ x, x))\ 'A \subseteq f\ 'A \times A$
by *auto*
qed

next
fix $x\ y$
show $x \in A \implies y \in A \implies (R \sqcup S)\ x\ y \implies lex-prodp\ Q\ S\ (f\ x, x)\ (f\ y, y)$
using *convertible-R convertible-S*
by (*auto simp add: lex-prodp-def*)
qed

lemma *chain-lnth-rtranclp*:
assumes
chain: *Lazy-List-Chain.chain* $R\ xs$ **and**
len: $enat\ j < llength\ xs$
shows $R^{**}\ (lhd\ xs)\ (lnth\ xs\ j)$
using *len*
proof (*induction j*)
case 0
from *chain* **obtain** $x\ xs'$ **where** $xs = LCons\ x\ xs'$

```

    by (auto elim: chain.cases)
  thus ?case
    by simp
next
  case (Suc j)
  then show ?case
    by (metis Suc-ile-eq chain chain-lnth-rel less-le-not-le rtranclp.simps)
qed

```

```

lemma chain-conj-rtranclpI:
  fixes xs :: 'a llist
  assumes Lazy-List-Chain.chain (λx y. R x y) (LCons init xs)
  shows Lazy-List-Chain.chain (λx y. R x y ∧ R** init x) (LCons init xs)
proof (intro lnth-rel-chain allI impI conjI)
  show ¬ lnull (LCons init xs)
    by simp
next
  fix j
  assume hyp: enat (j + 1) < llength (LCons init xs)

  from hyp show R (lnth (LCons init xs) j) (lnth (LCons init xs) (j + 1))
    using assms[THEN chain-lnth-rel, of j] by simp

  from hyp show R** init (lnth (LCons init xs) j)
    using assms[THEN chain-lnth-rtranclp, of j]
    by (simp add: Suc-ile-eq)
qed

```

```

lemma rtranclp-implies-ex-lfinite-chain:
  assumes run: R** x₀ x
  shows ∃ xs. lfinite xs ∧ chain (λy z. R y z ∧ R** x₀ y) (LCons x₀ xs) ∧ llast
(LCons x₀ xs) = x
  using run
proof (induction x rule: rtranclp-induct)
  case base
  then show ?case
    by (meson chain.chain-singleton lfinite-code(1) llast-singleton)
next
  case (step y z)
  from step.IH obtain xs where
    lfinite xs and chain (λy z. R y z ∧ R** x₀ y) (LCons x₀ xs) and llast (LCons
x₀ xs) = y
    by auto
  let ?xs = lappend xs (LCons z LNil)
  show ?case
  proof (intro exI conjI)
    show lfinite ?xs
      using ⟨lfinite xs⟩ by simp
  next

```

```

show chain ( $\lambda y z. R y z \wedge R^{**} x_0 y$ ) (LCons  $x_0$  ?xs)
  using ⟨chain ( $\lambda y z. R y z \wedge R^{**} x_0 y$ ) (LCons  $x_0$  xs)⟩ ⟨llast (LCons  $x_0$  xs)
= y⟩
  chain.chain-singleton chain-lappend step.hyps(1) step.hyps(2)
  by fastforce
next
show llast (LCons  $x_0$  ?xs) = z
  by (simp add: ⟨lfinite xs⟩ llast-LCons)
qed
qed

```

lemma chain-conj-rtranclD:

```

fixes xs :: 'a llist
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) =  $x_0$ 
  using chain
proof (cases  $\lambda y z. R y z \wedge R^{**} x_0 y$  xs rule: chain.cases)
  case (chain-singleton x)
  with inf have False
  by simp
  thus ?thesis ..
next
case (chain-cons xs' x)
hence  $R^{**} x_0 x$ 
  by auto
thus ?thesis
proof (cases R  $x_0$  x rule: rtrancl.cases)
  case rtrancl-refl
  then show ?thesis
  using chain local.chain-cons(1) by force
next
case (rtrancl-into-rtrancl  $x_n$ )
then obtain ys where
  lfin-ys: lfinite ys and
  chain-ys: chain ( $\lambda y z. R y z \wedge R^{**} x_0 y$ ) (LCons  $x_0$  ys) and
  llast-ys: llast (LCons  $x_0$  ys) =  $x_n$ 
  by (auto dest: rtrancl-implies-ex-lfinite-chain)
show ?thesis
proof (intro exI conjI)
  show lfinite (LCons  $x_0$  ys)
  using lfin-ys
  by simp
next
have R (llast (LCons  $x_0$  ys)) (lhd xs)
  using rtrancl-into-rtrancl(2) chain-cons(1) llast-ys
  by simp
moreover have  $R^{**} x_0$  (llast (LCons  $x_0$  ys))
  using rtrancl-into-rtrancl(1,2)

```

using *lappend-code(2)*[of x_0 ys xs]
lhd-LCons[of x_0 (*lappend* ys xs)] *local.chain-cons(1)*
using *llast-ys*
by *fastforce*
ultimately show *chain* ($\lambda y z. R y z \wedge R^{**} x_0 y$) (*lappend* (*LCons* x_0 ys) xs)
using *chain-lappend[OF chain-ys chain]*
by *metis*
next
show *lhd* (*lappend* (*LCons* x_0 ys) xs) = x_0
by *simp*
qed
qed
qed

lemma *wfp-on-rtranclp-conversep-iff-no-infinite-down-chain-llist*:

fixes $R x_0$
shows *wfp-on* $\{x. R^{**} x_0 x\} R^{-1-1} \longleftrightarrow (\nexists xs. \neg \text{lfinite } xs \wedge \text{Lazy-List-Chain.chain } R (LCons x_0 xs))$
proof (*rule iffI*)
assume *wfp-on* $\{x. R^{**} x_0 x\} R^{-1-1}$
hence *wfp* ($\lambda z y. R^{-1-1} z y \wedge z \in \{x. R^{**} x_0 x\} \wedge y \in \{x. R^{**} x_0 x\}$)
using *wfp-on-iff-wfp* **by** *blast*
hence *wfp* ($\lambda z y. R y z \wedge R^{**} x_0 y$)
by (*auto elim: wfp-on-mono-strong*)
hence $\nexists xs. \neg \text{lfinite } xs \wedge \text{Lazy-List-Chain.chain } (\lambda y z. R y z \wedge R^{**} x_0 y) xs$
unfolding *wfP-iff-no-infinite-down-chain-llist*
by (*metis (no-types, lifting) Lazy-List-Chain.chain-mono conversepI*)
hence $\nexists xs. \neg \text{lfinite } xs \wedge \text{Lazy-List-Chain.chain } (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons x_0 xs)$
by (*meson lfinite-LCons*)
thus $\nexists xs. \neg \text{lfinite } xs \wedge \text{Lazy-List-Chain.chain } R (LCons x_0 xs)$
using *chain-conj-rtranclpI*
by *fastforce*
next
assume $\nexists xs. \neg \text{lfinite } xs \wedge \text{Lazy-List-Chain.chain } R (LCons x_0 xs)$
hence *no-inf-chain*: $\nexists xs. \neg \text{lfinite } xs \wedge \text{chain } (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons x_0 xs)$
by (*metis (mono-tags, lifting) Lazy-List-Chain.chain-mono*)
have $\nexists xs. \neg \text{lfinite } xs \wedge \text{chain } (\lambda y z. R y z \wedge R^{**} x_0 y) xs$
proof (*rule notI, elim exE conjE*)
fix xs **assume** $\neg \text{lfinite } xs$ **and** *chain* ($\lambda y z. R y z \wedge R^{**} x_0 y$) xs
then obtain ys **where**
lfinite ys **and** *chain* ($\lambda y z. R y z \wedge R^{**} x_0 y$) (*lappend* ys xs) **and** *lhd* (*lappend* ys xs) = x_0
by (*auto dest: chain-conj-rtranclpD*)
hence $\exists xs. \neg \text{lfinite } xs \wedge \text{chain } (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons x_0 xs)$
proof (*intro exI conjI*)
show $\neg \text{lfinite } (ltl (lappend ys xs))$
using $\langle \neg \text{lfinite } xs \rangle$ *lfinite-lappend lfinite-ltl*

```

    by blast
  next
  show chain ( $\lambda y z. R y z \wedge R^{**} x_0 y$ ) (LCons  $x_0$  (ltl (lappend ys xs)))
    using ⟨chain ( $\lambda y z. R y z \wedge R^{**} x_0 y$ ) (lappend ys xs)⟩ ⟨lhd (lappend ys xs)⟩
=  $x_0$ ⟩
    chain-not-lnull lhd-LCons-ltl
    by fastforce
  qed
  with no-inf-chain show False
    by argo
  qed
  hence Wellfounded.wfP ( $\lambda z y. R y z \wedge y \in \{x. R^{**} x_0 x\}$ )
    unfolding wfP-iff-no-infinite-down-chain-llist
    using Lazy-List-Chain.chain-mono by fastforce
  hence wfp ( $\lambda z y. R^{-1-1} z y \wedge z \in \{x. R^{**} x_0 x\} \wedge y \in \{x. R^{**} x_0 x\}$ )
    by (auto elim: wfp-on-mono-strong)
  thus wfp-on  $\{x. R^{**} x_0 x\} R^{-1-1}$ 
    unfolding wfp-on-iff-wfp[of  $\{x. R^{**} x_0 x\} R^{-1-1}$ ] by argo
  qed

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 (rule Set.psubsetI)
  show  $A - B \subseteq A - C$ 
    using assms(1) by blast
next
  show  $A - B \neq A - C$ 
    using assms by blast
qed

```

17.2 Prod Extra

```

lemma lex-prod-lex-prodp-eq:
  lex-prod  $\{(x, y). RA x y\} \{(x, y). RB x y\} = \{(x, y). lex-prodp RA RB x y\}$ 
  unfolding lex-prodp-def lex-prod-def

```

by *auto*

lemma *reflp-on-lex-prodp*:
 assumes *reflp-on A RA*
 shows *reflp-on (A × B) (lex-prodp RA RB)*
proof (*rule reflp-onI*)
 fix *x* **assume** *x ∈ A × B*
 hence *fst x ∈ A*
 by *auto*
 thus *lex-prodp RA RB x x*
 by (*simp add: lex-prodp-def <reflp-on A RA>[THEN reflp-onD]*)
qed

lemma *transp-lex-prodp*:
 assumes *transp RA and transp RB*
 shows *transp (lex-prodp RA RB)*
proof (*rule transpI*)
 fix *x y z* **assume** *lex-prodp RA RB x y and lex-prodp RA RB y z*
 thus *lex-prodp RA RB x z*
 by (*auto simp add: lex-prodp-def <transp RA>[THEN transpD, of fst x fst y fst z]*
 <transp RB>[THEN transpD, of snd x snd y snd z])
qed

lemma *asympt-lex-prodp*:
 assumes *asympt RA and asympt RB*
 shows *asympt (lex-prodp RA RB)*
proof (*rule asymptI*)
 fix *x y* **assume** *lex-prodp RA RB x y*
 thus \neg *lex-prodp RA RB y x*
 using *assms* **by** (*metis (full-types, opaque-lifting) asymptD lex-prodp-def*)
qed

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 (*rule totalp-onI*)
 fix *x y* **assume** *x ∈ A × B and y ∈ A × B and x ≠ y*
 then show *lex-prodp RA RB x y ∨ lex-prodp RA RB y x*
 using *assms*
 by (*metis (full-types) lex-prodp-def mem-Times-iff prod-eq-iff totalp-on-def*)
qed

17.3 FSet Extra

lemma *finsert-Abs-fset*: *finite A ⇒ finsert a (Abs-fset A) = Abs-fset (insert a A)*
 by (*simp add: eq-onp-same-args finsert.abs-eq*)

lemma *minus-pfssubset-minusI*:

assumes $C \mid\subset\mid B$ **and** $B \mid\subseteq\mid A$
shows $(A \mid-\mid B \mid\subset\mid A \mid-\mid C)$
proof (*rule FSet.pfssubsetI*)
show $A \mid-\mid B \mid\subseteq\mid A \mid-\mid C$
using *assms(1)* **by** *blast*
next
show $A \mid-\mid B \neq A \mid-\mid C$
using *assms* **by** *blast*
qed

lemma *Abs-fset-minus*: $\text{finite } A \implies \text{finite } B \implies \text{Abs-fset } (A - B) = \text{Abs-fset } A \mid-\mid \text{Abs-fset } B$
by (*metis Abs-fset-inverse fset-inverse mem-Collect-eq minus-fset*)

lemma *fminus-conv*: $A \mid\subset\mid B \iff \text{fset } A \subset \text{fset } B \wedge \text{finite } (\text{fset } A) \wedge \text{finite } (\text{fset } B)$
by (*simp add: less-eq-fset.rep-eq less-le-not-le*)

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 $\beta \Gamma = \text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\} \mid-\mid (\text{fst} \mid\uparrow\mid \text{fset-of-list } \Gamma)$

primrec *M-skip-fact-reso* **where**

M-skip-fact-reso $\square C = \square \mid$
M-skip-fact-reso $(Ln \# \Gamma) C =$
(let $n = \text{count } C \text{ } (- \text{ } (\text{fst } Ln))$ *in*
(case $\text{snd } Ln$ *of* $\text{None} \Rightarrow 0 \mid \text{Some } - \Rightarrow n$) $\#$
M-skip-fact-reso $\Gamma (C + (\text{case } \text{snd } Ln \text{ of } \text{None} \Rightarrow \{\#\} \mid \text{Some } (D, -, \gamma) \Rightarrow \text{repeat-mset } n (D \cdot \gamma)))$)

fun *M-skip-fact-reso'* **where**

M-skip-fact-reso' $C \square = \square \mid$
M-skip-fact-reso' $C ((-, \text{None}) \# \Gamma) = 0 \# \text{M-skip-fact-reso}' C \Gamma \mid$
M-skip-fact-reso' $C ((K, \text{Some } (D, -, \gamma)) \# \Gamma) =$
(let $n = \text{count } C \text{ } (- K)$ *in* $n \# \text{M-skip-fact-reso}' (C + \text{repeat-mset } n (D \cdot \gamma))$)
 Γ)

lemma *M-skip-fact-reso* $\Gamma C = \text{M-skip-fact-reso}' C \Gamma$

proof (*induction* Γ *arbitrary: C*)

case *Nil*

show *?case*

by *simp*

next

case (*Cons* $Kn \Gamma$)

```

then show ?case
  apply (cases Kn)
  apply (cases snd Kn)
  by (auto simp add: Let-def)
qed

```

```

lemma  $\mathcal{M}$ -skip-fact-reso' C (decide-lit K #  $\Gamma$ ) = 0 #  $\mathcal{M}$ -skip-fact-reso' C  $\Gamma$ 
by (simp add: decide-lit-def)

```

```

lemma  $\mathcal{M}$ -skip-fact-reso' C (propagate-lit K D  $\gamma$  #  $\Gamma$ ) =
  (let n = count C (- (K · l  $\gamma$ )) in n #  $\mathcal{M}$ -skip-fact-reso' (C + repeat-mset n (D
  ·  $\gamma$ ))  $\Gamma$ )
by (simp add: propagate-lit-def)

```

```

fun  $\mathcal{M}$  :: -  $\Rightarrow$  ('f, 'v) state  $\Rightarrow$ 
  bool  $\times$  ('f, 'v) Term.term literal fset  $\times$  nat list  $\times$  nat where
   $\mathcal{M}$   $\beta$  ( $\Gamma$ , U, None) = (True,  $\mathcal{M}$ -prop-deci  $\beta$   $\Gamma$ , [], 0) |
   $\mathcal{M}$   $\beta$  ( $\Gamma$ , U, Some (C,  $\gamma$ )) = (False, {||},  $\mathcal{M}$ -skip-fact-reso  $\Gamma$  (C ·  $\gamma$ ), size C)

```

```

lemma length- $\mathcal{M}$ -skip-fact-reso[simp]: length ( $\mathcal{M}$ -skip-fact-reso  $\Gamma$  C) = length  $\Gamma$ 
by (induction  $\Gamma$  arbitrary: C) (simp-all add: Let-def)

```

```

lemma  $\mathcal{M}$ -skip-fact-reso-add-mset:
  ( $\mathcal{M}$ -skip-fact-reso  $\Gamma$  C,  $\mathcal{M}$ -skip-fact-reso  $\Gamma$  (add-mset L C))  $\in$  (List.lenlex {(x,
  y). x < y})=
proof (induction  $\Gamma$  arbitrary: C)
  case Nil
  show ?case by simp
next
  case (Cons Ln  $\Gamma$ )
  show ?case
  proof (cases snd Ln)
    case None
    then show ?thesis
    using Cons.IH[of C]
    by (simp add: Cons-lenlex-iff)
  next
  case (Some cl)
  show ?thesis
  proof (cases L = - fst Ln)
    case True
    then show ?thesis
    by (simp add: Let-def Some Cons-lenlex-iff)
  next
  case False
  then show ?thesis
  using Cons.IH
  by (auto simp add: Let-def Some Cons-lenlex-iff)
qed

```

qed
qed

lemma *termination-scl-without-back-invars*:

fixes $N \beta$

defines

$scl\text{-without-backtrack} \equiv propagate\ N\ \beta \sqcup decide\ N\ \beta \sqcup conflict\ N\ \beta \sqcup skip\ N\ \beta \sqcup$

$factorize\ N\ \beta \sqcup resolve\ N\ \beta$ **and**

$invars \equiv trail\text{-atoms-}lt\ \beta \sqcap trail\text{-resolved-lits-pol} \sqcap trail\text{-lits-ground} \sqcap$

$initial\text{-lits-generalize-learned-trail-conflict}\ N \sqcap ground\text{-closures}$

shows $wfp\text{-on}\ \{S. invars\ S\}\ scl\text{-without-backtrack}^{-1-1}$

proof –

let $?less =$

$lex\text{-prodp}\ ((<) :: bool \Rightarrow bool \Rightarrow bool)$

$(lex\text{-prodp}\ (|\subset|))$

$(lex\text{-prodp}\ (\lambda x\ y. (x, y) \in List.lenlex\ \{(x :: - :: wellorder, y). x < y\}))$

$((<) :: nat \Rightarrow nat \Rightarrow bool))$

show $wfp\text{-on}\ \{S. invars\ S\}\ scl\text{-without-backtrack}^{-1-1}$

proof (*rule wfp-on-if-convertible-to-wfp-on*)

fix $S' S :: ('f, 'v)\ state$

assume $S' \in \{S. invars\ S\}$ **and** $S \in \{S. invars\ S\}$ **and** *step: scl-without-backtrack*⁻¹⁻¹
 $S' S$

hence

$trail\text{-atoms-}lt\ \beta\ S$ **and**

$trail\text{-resolved-lits-pol}\ S$ **and**

$trail\text{-lits-ground}\ S$ **and**

$initial\text{-lits-generalize-learned-trail-conflict}\ N\ S$ **and**

$ground\text{-closures}\ S$

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

by (*simp-all add: invars-def*)

from *step* **show** $?less\ (\mathcal{M}\ \beta\ S')\ (\mathcal{M}\ \beta\ S)$

unfolding *conversep-iff scl-without-backtrack-def sup-apply sup-bool-def*

proof (*elim disjE*)

assume $decide\ N\ \beta\ S\ S'$

thus $?less\ (\mathcal{M}\ \beta\ S')\ (\mathcal{M}\ \beta\ S)$

proof (*cases N beta S S' rule: decide.cases*)

case (*decideI L gamma Gamma U*)

have $\mathcal{M}\text{-prop-deci}\ \beta\ ((L \cdot l\ \gamma, None) \# \Gamma) |\subset| \mathcal{M}\text{-prop-deci}\ \beta\ \Gamma$

unfolding $\mathcal{M}\text{-prop-deci-def fset-of-list-simps fimage-finsert prod.sel}$

proof (*rule minus-pfsubset-minusI*)

show $fst\ |\uparrow\ fset\text{-of-list}\ \Gamma\ |\subset|\ finsert\ (L \cdot l\ \gamma)\ (fst\ |\uparrow\ fset\text{-of-list}\ \Gamma)$

using $\langle \neg trail\text{-defined-lit}\ \Gamma\ (L \cdot l\ \gamma) \rangle [unfolding\ trail\text{-defined-lit-def}]$

by (*metis (no-types, lifting) finsertCI fset-of-list-elem fset-of-list-map*

fsubset-finsertI list.set-map nless-le)

next

have $L \cdot l\ \gamma \in \{L. atm\text{-of}\ L \preceq_B\ \beta\}$

```

    using ‹atm-of L · a γ ≼B β›
    by simp
  moreover have fst ‹set Γ ⊆ {L. atm-of L ≼B β}›
    using ‹trail-atoms-lt β S›
    by (auto simp: trail-atoms-lt-def decideI(1))
  ultimately have insert (L · l γ) (fst ‹set Γ ⊆ {L. atm-of L ≼B β}›)
    by simp
  then show finsert (L · l γ) (fst |q fset-of-list Γ) |⊆| Abs-fset {L. atm-of L
≼B β}
    using finite-lits-less-eq-B
    by (simp add: less-eq-fset.rep-eq Abs-fset-inverse fset-of-list.rep-eq)
  qed
  then show ?thesis
    unfolding decideI(1,2) decide-lit-def
    unfolding lex-prodp-def
    by simp
  qed
next
assume propagate N β S S'
thus ?less (M β S') (M β S)
proof (cases N β S S' rule: propagate.cases)
  case (propagateI C U L C' γ C0 C1 Γ μ)

  have L · l μ · l γ = L · l γ
  proof -
    have is-unifiers γ {atm-of ‹set-mset (add-mset L C1)›}
      unfolding ‹C1 = {#K ∈# C'. K · l γ = L · l γ#}›
      by (auto simp: is-unifiers-def is-unifier-alt intro: subst-atm-of-eqI)
    hence μ ⊙ γ = γ
      using ‹is-imgu μ {atm-of ‹set-mset (add-mset L C1)›}›[unfolded
is-imgu-def, THEN conjunct2]
      by simp
    thus ?thesis
      by (metis subst-lit-comp-subst)
  qed

  have M-prop-deci β ((L · l γ, Some (C0 · μ, L · l μ, γ)) # Γ) |⊆| M-prop-deci
β Γ
    unfolding M-prop-deci-def fset-of-list-simps fimage-finsert prod.sel
  proof (rule minus-pfsubset-minusI)
    show fst |q fset-of-list Γ |⊆| finsert (L · l γ) (fst |q fset-of-list Γ)
      using ‹q trail-defined-lit Γ (L · l γ)›[unfolded trail-defined-lit-def]
      by (metis (no-types, lifting) finsertCI fset-of-list-elem fset-of-list-map
fsubset-finsertI list.set-map nless-le)
  next
  have insert (L · l γ) (fst ‹set Γ ⊆ {L. atm-of L ≼B β}›)
  proof (intro Set.subsetI Set.CollectI)
    fix K assume K ∈ insert (L · l γ) (fst ‹set Γ ⊆ {L. atm-of L ≼B β}›)
    thus atm-of K ≼B β

```

```

      using ⟨trail-atoms-lt β S⟩
      by (metis image-eqI insert-iff propagateI(1,4,6) state-trail-simp
subst-cls-add-mset
      trail-atoms-lt-def union-single-eq-member)
    qed
  then show finsert (L · l γ) (fst |l fset-of-list Γ) |⊆| Abs-fset {L. atm-of L
≲B β}
    using finite-lits-less-eq-B
    by (simp add: less-eq-fset.rep-eq fset-of-list.rep-eq Abs-fset-inverse)
  qed
  thus ?thesis
    unfolding propagateI(1,2) propagate-lit-def state-proj-simp option.case
    unfolding ⟨L · l μ · l γ = L · l γ⟩
    unfolding lex-prodp-def
    by simp
  qed
next
  assume conflict N β S S'
  thus ?less (M β S') (M β S)
  proof (cases N β S S' rule: conflict.cases)
    case (conflictI D U γ Γ)
    show ?thesis
      unfolding lex-prodp-def conflictI(1,2) by simp
  qed
next
  assume skip N β S S'
  thus ?less (M β S') (M β S)
  proof (cases N β S S' rule: skip.cases)
    case (skipI L D σ n Γ U)
    have (M-skip-fact-reso Γ (D · σ), M-skip-fact-reso ((L, n) # Γ) (D · σ)) ∈
      lenlex {(x, y). x < y}
      by (simp add: lenlex-conv Let-def)
    thus ?thesis
      unfolding lex-prodp-def skipI(1,2) by simp
  qed
next
  assume factorize N β S S'
  thus ?less (M β S') (M β S)
  proof (cases N β S S' rule: factorize.cases)
    case (factorizeI L γ L' μ Γ U D)

    have is-unifier γ {atm-of L, atm-of L'}
      using ⟨L · l γ = L' · l γ⟩[THEN subst-atm-of-eqI]
      by (simp add: is-unifier-alt)
    hence μ ⊙ γ = γ
      using ⟨is-ingu μ {{atm-of L, atm-of L'}}⟩
      by (simp add: is-ingu-def is-unifiers-def)

    have add-mset L D · μ · γ = add-mset L D · γ

```

```

    using  $\langle \mu \odot \gamma = \gamma \rangle$ 
    by (metis subst-cls-comp-subst)
  hence ( $\mathcal{M}$ -skip-fact-reso  $\Gamma$  (add-mset  $L D \cdot \mu \cdot \gamma$ ),
     $\mathcal{M}$ -skip-fact-reso  $\Gamma$  (add-mset  $L' (add-mset L D) \cdot \gamma$ ))  $\in$  (lenlex  $\{(x, y). x$ 
 $< y\}$ )=
    using  $\mathcal{M}$ -skip-fact-reso-add-mset
    by (metis subst-cls-add-mset)
  thus ?thesis
    unfolding lex-prodp-def factorizeI(1,2)
    unfolding add-mset-commute[of  $L' L$ ]
    by simp
qed
next
assume resolve  $N \beta S S'$ 
thus ?less ( $\mathcal{M} \beta S'$ ) ( $\mathcal{M} \beta S$ )
proof (cases  $N \beta S S'$  rule: resolve.cases)
  case (resolveI  $\Gamma \Gamma' K D \gamma_D L \gamma_C \varrho_C \varrho_D C \mu \gamma U$ )
  from  $\langle$ ground-closures  $S$  $\rangle$  have
    ground-conf: is-ground-cls (add-mset  $L C \cdot \gamma_C$ ) and
    ground-prop: is-ground-cls (add-mset  $K D \cdot \gamma_D$ )
    unfolding resolveI(1,2)  $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$ 
    by (simp-all add: ground-closures-def propagate-lit-def)
  hence
     $\forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$ 
     $\forall K \in \# \text{add-mset } K D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$ 
    using resolveI merge-of-renamed-groundings by metis+

  have atm-of  $L \cdot a \varrho_C \cdot a \gamma = \text{atm-of } K \cdot a \varrho_D \cdot a \gamma$ 
    using  $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$ 
     $\langle \forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle$  [rule-format, of  $L$ , simplified]
     $\langle \forall K \in \# \text{add-mset } K D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$  [rule-format, of  $K$ ,
simplified]
    by (metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit)
  hence is-unifiers  $\gamma \{\{\text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D\}\}$ 
    by (simp add: is-unifiers-def is-unifier-alt)
  hence  $\mu \odot \gamma = \gamma$ 
    using  $\langle \text{is-ingu } \mu \{\{\text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D\}\} \rangle$ 
    by (auto simp: is-ingu-def)
  hence  $C \cdot \varrho_C \cdot \mu \cdot \gamma = C \cdot \gamma_C$  and  $D \cdot \varrho_D \cdot \mu \cdot \gamma = D \cdot \gamma_D$ 
    using  $\langle \forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle$   $\langle \forall K \in \# \text{add-mset } K$ 
 $D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$ 
    by (metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst
      subst-lit-comp-subst)+
  hence  $(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma = C \cdot \gamma_C + D \cdot \gamma_D$ 
    by (metis subst-cls-comp-subst subst-cls-union)

  have  $L \cdot l \gamma_C \notin \# D \cdot \gamma_D$ 
    using  $\langle \text{trail-resolved-lits-pol } S \rangle$   $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$ 
    unfolding resolveI(1,2)  $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$ 

```

```

    by (simp add: trail-resolved-lits-pol-def propagate-lit-def)

  have ( $\mathcal{M}$ -skip-fact-reso  $\Gamma (C \cdot \gamma_C + D \cdot \gamma_D)$ ,  $\mathcal{M}$ -skip-fact-reso  $\Gamma (add\text{-}mset$ 
 $L C \cdot \gamma_C)$ )  $\in$ 
    lex  $\{(x, y). x < y\}$ 
  unfolding  $\langle \Gamma = trail\text{-}propagate \Gamma' K D \gamma_D \rangle propagate\text{-}lit\text{-}def$ 
  unfolding  $\mathcal{M}$ -skip-fact-reso.simps Let-def prod.sel option.case prod.case
  unfolding lex-conv mem-Collect-eq prod.case
  apply (rule conjI)
  apply simp
  apply (rule exI[of - []])
  apply simp
  using  $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$ 
  apply simp
  unfolding count-eq-zero-iff
  by (rule  $\langle L \cdot l \gamma_C \notin \# D \cdot \gamma_D \rangle$ )
  hence ( $\mathcal{M}$ -skip-fact-reso  $\Gamma (C \cdot \gamma_C + D \cdot \gamma_D)$ ,  $\mathcal{M}$ -skip-fact-reso  $\Gamma (add\text{-}mset$ 
 $L C \cdot \gamma_C)$ )  $\in$ 
    lenlex  $\{(x, y). x < y\}$ 
  unfolding lenlex-conv by simp
  thus ?thesis
  unfolding lex-prodp-def resolveI(1,2)
  unfolding  $\mathcal{M}$ .simps state-proj-simp option.case prod.case prod.sel
  unfolding  $\langle (C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma = C \cdot \gamma_C + D \cdot \gamma_D \rangle$ 
  by simp
qed
qed
next
show wfp-on ( $\mathcal{M} \beta \text{ ' } \{S. \text{invars } S\}$ ) ?less
proof (rule wfp-on-subset)
  show  $\mathcal{M} \beta \text{ ' } \{S. \text{invars } S\} \subseteq UNIV$ 
  by simp
next
show wfp ?less
proof (intro wfp-lex-prodp)
  show wfp ( $\langle \cdot \rangle :: bool \Rightarrow bool \Rightarrow bool$ )
  by (simp add: Wellfounded.wfpUNIVI)
next
show wfp ( $|\cdot|$ )
  by (rule wfp-pfssubset)
next
show wfp ( $\lambda x y. (x, y) \in lenlex \{(x :: - :: wellorder, y). x < y\}$ )
  unfolding Wellfounded.wfp-wf-eq
  using wf-lenlex
  using wf by blast
next
show wfp ( $\langle \cdot \rangle :: nat \Rightarrow nat \Rightarrow bool$ )
  by simp
qed

```

qed
 qed
 qed

corollary *termination-scl-without-back:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**

$\beta :: ('f, 'v) \text{ Term.term}$

defines

$scl\text{-without-backtrack} \equiv propagate\ N\ \beta \sqcup decide\ N\ \beta \sqcup conflict\ N\ \beta \sqcup skip\ N$
 $\beta \sqcup$

$factorize\ N\ \beta \sqcup resolve\ N\ \beta$ **and**

$invars \equiv trail\text{-atoms-lt}\ \beta \sqcap trail\text{-resolved-lits-pol} \sqcap trail\text{-lits-ground} \sqcap$

$initial\text{-lits-generalize-learned-trail-conflict}\ N \sqcap ground\text{-closures}$

shows $wfp\text{-on}\ \{S.\ scl\text{-without-backtrack}^{**}\ initial\text{-state}\ S\}$ $scl\text{-without-backtrack}^{-1-1}$
proof (*rule wfp-on-subset*)

show $wfp\text{-on}\ \{S.\ invars\ S\}$ $scl\text{-without-backtrack}^{-1-1}$

by (*rule termination-scl-without-back-invars(1)[of $\beta\ N$,
 folded invars-def scl-without-backtrack-def]*)

next

have $invars\ initial\text{-state}$

by (*simp add: invars-def*)

moreover have $invars\ S \implies invars\ S'$

if $scl\text{-without-backtrack}\ S\ S'$

for $S\ S'$

proof –

from *that* **have** $scl\ N\ \beta\ S\ S'$

by (*auto simp: scl-without-backtrack-def scl-def*)

thus $invars\ S \implies invars\ S'$

unfolding $invars\text{-def}$

using

$scl\text{-preserves-trail-atoms-lt}$

$scl\text{-preserves-trail-resolved-lits-pol}$

$scl\text{-preserves-trail-lits-ground}$

$scl\text{-preserves-initial-lits-generalize-learned-trail-conflict}$

$scl\text{-preserves-ground-closures}$

by *simp-all*

qed

ultimately have $scl\text{-without-backtrack}^{**}\ initial\text{-state}\ S \implies invars\ S$ **for** S

by (*auto elim: rtranclp-induct*)

thus $\{S.\ scl\text{-without-backtrack}^{**}\ initial\text{-state}\ S\} \subseteq \{S.\ invars\ S\}$

by *auto*

qed

corollary *termination-strategy-without-back:*

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**

$\beta :: ('f, 'v) \text{ Term.term}$

```

defines
  scl-without-backtrack  $\equiv$  propagate  $N \beta \sqcup$  decide  $N \beta \sqcup$  conflict  $N \beta \sqcup$  skip  $N \beta \sqcup$ 
  factorize  $N \beta \sqcup$  resolve  $N \beta$ 
assumes strategy-stronger:  $\bigwedge S S'. \text{strategy } S S' \implies \text{scl-without-backtrack } S S'$ 
shows wfp-on  $\{S. \text{strategy}^{**} \text{initial-state } S\} \text{strategy}^{-1-1}$ 
proof (rule wfp-on-mono-strong)
show wfp-on  $\{S. \text{strategy}^{**} \text{initial-state } S\} \text{scl-without-backtrack}^{-1-1}$ 
proof (rule wfp-on-subset)
show wfp-on  $\{S. \text{scl-without-backtrack}^{**} \text{initial-state } S\} \text{scl-without-backtrack}^{-1-1}$ 
unfolding scl-without-backtrack-def
using termination-scl-without-back by metis
next
show  $\{S. \text{strategy}^{**} \text{initial-state } S\} \subseteq \{S. \text{scl-without-backtrack}^{**} \text{initial-state } S\}$ 
using strategy-stronger
by (metis (no-types, opaque-lifting) Collect-mono mono-rtranclp)
qed
next
show  $\bigwedge S' S. \text{strategy}^{-1-1} S' S \implies \text{scl-without-backtrack}^{-1-1} S' S$ 
using strategy-stronger by simp
qed simp

```

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

$fclss-no-dup \beta = fPow (\text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\})$

lemma image-fset-fset-fPow-eq: $fset ' fset (fPow A) = Pow (fset A)$

proof (rule Set.equalityI)

show $fset ' fset (fPow A) \subseteq Pow (fset A)$

by (meson PowI fPowD image-subset-iff less-eq-fset.rep-eq)

next

show $Pow (fset A) \subseteq fset ' fset (fPow A)$

proof (rule Set.subsetI)

fix x **assume** $x \in Pow (fset A)$

moreover **hence** finite x

by (metis PowD finite-fset rev-finite-subset)

ultimately **show** $x \in fset ' fset (fPow A)$

unfolding image-iff

by (metis PowD fPowI fset-cases less-eq-fset.rep-eq mem-Collect-eq)

qed

qed

lemma

assumes $\forall L \in \# C. \text{count } C L = 1$

shows $\exists C'. C = \text{mset-set } C'$

using assms

by (metis count-eq-zero-iff count-mset-set(1) count-mset-set(3) finite-set-mset multiset-eqI)

lemma *fmember-fclss-no-dup-if*:

assumes $\forall L \mid \in \mid C. \text{atm-of } L \preceq_B \beta$

shows $C \mid \in \mid \text{fclss-no-dup } \beta$

proof –

show *?thesis*

unfolding *fclss-no-dup-def fPow-iff*

proof (*rule fsubsetI*)

fix K **assume** $K \mid \in \mid C$

with *assms* **show** $K \mid \in \mid \text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\}$

using *Abs-fset-inverse[simplified, OF finite-lits-less-eq-B]*

by *simp*

qed

qed

definition *M-back* :: $- \Rightarrow (f, v) \text{ state} \Rightarrow (f, v) \text{ Term.term literal fset fset}$
where

M-back $\beta S = \text{Abs-fset } (\text{fset } (\text{fclss-no-dup } \beta) -$

Abs-fset 'set-mset 'grounding-of-clss (fset (state-learned S))))

lemma *M-back-after-regular-backtrack*:

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* $N \beta Sn Sn'$

defines $U \equiv \text{state-learned } Sn$

shows

$\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

set-mset $(C \cdot \gamma) \notin \text{set-mset 'grounding-of-clss (fset } N \cup \text{fset } U)$ **and**

M-back $\beta Sn' \mid \subset \mid \text{M-back } \beta Sn$

proof –

from *regular-run* **have** (*scl* $N \beta$)** *initial-state* $S0$

by (*induction* $S0$ *rule*: *rtranclp-induct*)

(*auto intro*: *scl-if-regular rtranclp.rtrancl-into-rtrancl*)

with *conflict* **have** (*scl* $N \beta$)** *initial-state* $S1$

by (*meson regular-scl-if-conflict rtranclp.rtrancl-into-rtrancl scl-if-regular*)

with *resolution* **have** *scl-run*: (*scl* $N \beta$)** *initial-state* Sn

by (*metis (no-types, lifting) Nitpick.rtranclp-unfold mono-rtranclp*

regular-run-if-skip-factorize-resolve-run rtranclp-tranclp-tranclp scl-if-regular)

from *scl-run* **have** *ground-false-closures* Sn

by (*induction* Sn *rule*: *rtranclp-induct*)

(*auto intro*: *scl-preserves-ground-false-closures*)

hence *ground-closures* Sn

using *ground-false-closures-def* **by** *blast*

```

from scl-run have trail-atoms-lt  $\beta$  Sn
  by (induction Sn rule: rtranclp-induct)
    (auto intro: scl-preserves-trail-atoms-lt)

obtain C  $\gamma$  where
  conf: state-conflict Sn = Some (C,  $\gamma$ ) and
  set-conf-not-in-set-groundings:
    set-mset (C ·  $\gamma$ )  $\notin$  set-mset ‘ grounding-of-clss (fset N  $\cup$  fset (state-learned
S1))
  using dynamic-non-redundancy-regular-scl[OF assms(1,2,3,4)]
  using standard-lit-less-preserves-term-less
  by metis

have 1: state-learned Sn' = finsert C (state-learned Sn)
  using backtrack conf by (auto elim: backtrack.cases)

have 2: state-learned Sn = state-learned S1
  using resolution
proof (induction Sn rule: tranclp-induct)
  case (base y)
  thus ?case
    by (auto elim: skip.cases factorize.cases resolve.cases)
next
  case (step y z)
  from step.hyps(2) have state-learned z = state-learned y
    by (auto elim: skip.cases factorize.cases resolve.cases)
  with step.IH show ?case
    by simp
qed
with conf set-conf-not-in-set-groundings show  $\exists$  C  $\gamma$ . state-conflict Sn = Some
(C,  $\gamma$ )  $\wedge$ 
  set-mset (C ·  $\gamma$ )  $\notin$  set-mset ‘ grounding-of-clss (fset N  $\cup$  fset U)
  by (simp add: U-def)

have Diff-strict-subsetI:  $x \in A \implies x \in B \implies A - B \subset A$  for  $x \in A \in B$ 
  by auto

have fset (fclss-no-dup  $\beta$ ) - Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned
Sn')) =
  fset (fclss-no-dup  $\beta$ ) - Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned
Sn)) -
  Abs-fset ‘ set-mset ‘ grounding-of-clss C
  unfolding 1 finsert.rep-eq grounding-of-clss-insert image-Un
  by auto

also have ...  $\subset$ 
  fset (fclss-no-dup  $\beta$ ) - Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned
Sn))
proof (rule Diff-strict-subsetI)

```

from $\langle \text{ground-closures } Sn \rangle$ **have** $C \cdot \gamma \in \text{grounding-of-clc } C$
unfolding *ground-closures-def conf*
using *grounding-of-clc-ground grounding-of-subst-clc-subset* **by** *blast*
thus $\text{Abs-fset } (\text{set-mset } (C \cdot \gamma)) \in \text{Abs-fset } \text{' set-mset ' grounding-of-clc } C$
by *blast*
next
have *Abs-fset-in-image-Abs-fset-iff*: $\text{Abs-fset } A \in \text{Abs-fset } \text{' AA } \longleftrightarrow A \in AA$
if $\text{finite } A \wedge (\forall B \in AA. \text{finite } B)$
for $A \text{ } AA$
apply (*rule iffI*)
using *that*
apply (*metis Abs-fset-inverse imageE mem-Collect-eq*)
using *that*
by *blast*

have $\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \text{' grounding-of-clc } (\text{fset } (\text{state-learned } S1))$
using *set-conf-not-in-set-groundings*
by (*auto simp: grounding-of-clc-union*)
then have $\text{Abs-fset } (\text{set-mset } (C \cdot \gamma)) \notin$
 $\text{Abs-fset } \text{' set-mset ' grounding-of-clc } (\text{fset } (\text{state-learned } Sn))$
unfolding 2
using *Abs-fset-in-image-Abs-fset-iff*
by (*metis finite-set-mset image-iff*)

moreover have $\text{Abs-fset } (\text{set-mset } (C \cdot \gamma)) \in \text{fset } (\text{fclss-no-dup } \beta)$
proof (*intro fmember-fclss-no-dup-if ballI*)
fix L **assume** $L \in \text{Abs-fset } (\text{set-mset } (C \cdot \gamma))$
hence $L \in \# C \cdot \gamma$
by (*metis fset-fset-mset fset-inverse*)
moreover have $\text{trail-false-clc } (\text{state-trail } Sn) (C \cdot \gamma)$
using $\langle \text{ground-false-closures } Sn \rangle$ *conf* **by** (*auto simp: ground-false-closures-def*)
ultimately show $\text{atm-of } L \preceq_B \beta$
using *ball-less-B-if-trail-false-and-trail-atoms-lt[OF - $\langle \text{trail-atoms-lt } \beta Sn \rangle$]*
by *metis*
qed

ultimately show $\text{Abs-fset } (\text{set-mset } (C \cdot \gamma)) \in \text{fset } (\text{fclss-no-dup } \beta) -$
 $\text{Abs-fset } \text{' set-mset ' grounding-of-clc } (\text{fset } (\text{state-learned } Sn))$
by *simp*
qed

finally show $\mathcal{M}\text{-back } \beta Sn' \mid \subset \mid \mathcal{M}\text{-back } \beta Sn$
unfolding *M-back-def*
unfolding *fminus-conv*
by (*simp add: Abs-fset-inverse[simplified]*)
qed

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 (*rule wfp-on-mono-strong*)

fix $S S'$ **assume** $(\text{regular-scl } N \beta)^{-1-1} S S'$

thus $(\text{backtrack } N \beta \sqcup (\text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta$
 $\sqcup \text{factorize } N \beta \sqcup$
 $\text{resolve } N \beta))^{-1-1} S S'$

by (*auto simp: regular-scl-def reasonable-scl-def scl-def*)

next

show $\text{wfp-on } \{S. \text{invars } S\} (\text{backtrack } N \beta \sqcup (\text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup$
 $\text{conflict } N \beta \sqcup$

$\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta))^{-1-1}$

unfolding *converse-join[of backtrack N beta]*

proof (*rule wfp-on-sup-if-convertible-to-wfp, unfold mem-Collect-eq*)

show $\text{wfp-on } \{S. \text{invars } S\} (\text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip}$
 $N \beta \sqcup$

$\text{factorize } N \beta \sqcup \text{resolve } N \beta)^{-1-1}$

using *termination-scl-without-back-invars(1)[of beta N]*

by (*auto simp: invars-def inf-assoc elim: wfp-on-subset*)

next

show $\text{wfp-on } (\mathcal{M}\text{-back } \beta \text{ ' } \{S. \text{invars } S\}) (|\subset|)$

proof (*rule wfp-on-subset*)

show $\text{wfp } (|\subset|)$

by (*rule wfp-pfsubset*)

qed *simp*

next

fix $S' S$

assume $\text{invars } S'$ **and** $\text{invars } S$ **and** $(\text{backtrack } N \beta)^{-1-1} S' S$

moreover from $\langle \text{invars } S \rangle$ **have** $\text{sound-state } N \beta S$

by (*simp add: invars-def*)

moreover from $\langle \text{invars } S \rangle$ **have** $\text{almost-no-conflict-with-trail } N \beta S$

by (*simp add: invars-def*)

moreover from $\langle \text{invars } S \rangle$ **have** $\text{regular-conflict-resolution } N \beta S$

by (*simp add: invars-def*)

moreover from $\langle \text{invars } S \rangle$ **have** $\text{ground-false-closures } S$

by (*simp add: invars-def*)

ultimately obtain $S0\ S1\ S2\ S3\ S4$ **where**
reg-run: (*regular-scl* $N\ \beta$)** *initial-state* $S0$ **and**
propa: *propagate* $N\ \beta\ S0\ S1$ *regular-scl* $N\ \beta\ S0\ S1$ **and**
confl: *conflict* $N\ \beta\ S1\ S2$ **and**
facto: (*factorize* $N\ \beta$)** $S2\ S3$ **and**
resol: *resolve* $N\ \beta\ S3\ S4$ **and**
reg-res: (*skip* $N\ \beta\ \sqcup$ *factorize* $N\ \beta\ \sqcup$ *resolve* $N\ \beta$)** $S4\ S$
using *before-regular-backtrack* **by** *blast*

show $\mathcal{M}\text{-back}\ \beta\ S' \mid\sqsubset\ \mathcal{M}\text{-back}\ \beta\ S$
proof (*rule* *$\mathcal{M}\text{-back-after-regular-backtrack}$*)
show (*regular-scl* $N\ \beta$)** *initial-state* $S1$
using *reg-run propa*(2) **by** *simp*
next
show *conflict* $N\ \beta\ S1\ S2$
by (*rule* *confl*)
next
have (*skip* $N\ \beta\ \sqcup$ *factorize* $N\ \beta\ \sqcup$ *resolve* $N\ \beta$)** $S2\ S3$
using *facto*
by (*rule* *mono-rtranclp*[*rule-format*, *rotated*]) *simp*
also have (*skip* $N\ \beta\ \sqcup$ *factorize* $N\ \beta\ \sqcup$ *resolve* $N\ \beta$)** $S3\ S4$
using *resol* **by** *auto*
finally show (*skip* $N\ \beta\ \sqcup$ *factorize* $N\ \beta\ \sqcup$ *resolve* $N\ \beta$)** $S2\ S$
using *reg-res* **by** *simp*
next
from $\langle(\text{backtrack } N\ \beta)^{-1-1}\ S'\ S\rangle$ **show** *backtrack* $N\ \beta\ S\ S'$
by *simp*
qed
next
fix $S'\ S$
assume *invars* S' **and** *invars* S **and**
 $(\text{propagate } N\ \beta\ \sqcup\ \text{decide } N\ \beta\ \sqcup\ \text{conflict } N\ \beta\ \sqcup\ \text{skip } N\ \beta\ \sqcup\ \text{factorize } N\ \beta\ \sqcup\ \text{resolve } N\ \beta)^{-1-1}\ S'\ S$
hence *state-learned* $S' = \text{state-learned } S$
by (*auto elim: propagate.cases decide.cases conflict.cases skip.cases factorize.cases*
resolve.cases)
hence $\mathcal{M}\text{-back}\ \beta\ S' = \mathcal{M}\text{-back}\ \beta\ S$
by (*simp add: $\mathcal{M}\text{-back-def}$*)
thus $\mathcal{M}\text{-back}\ \beta\ S' \mid\sqsubset\ \mathcal{M}\text{-back}\ \beta\ S \vee \mathcal{M}\text{-back}\ \beta\ S' = \mathcal{M}\text{-back}\ \beta\ S ..$
qed
qed *simp*

corollary *termination-regular-scl*:
fixes
 $N :: ('f, 'v)\ \text{Term.term clause fset}$ **and**
 $\beta :: ('f, 'v)\ \text{Term.term}$

defines
 $invars \equiv trail\text{-}atoms\text{-}lt\ \beta \sqcap trail\text{-}resolved\text{-}lits\text{-}pol \sqcap trail\text{-}lits\text{-}ground \sqcap$
 $initial\text{-}lits\text{-}generalize\text{-}learned\text{-}trail\text{-}conflict\ N \sqcap ground\text{-}closures \sqcap ground\text{-}false\text{-}closures$
 \sqcap
 $sound\text{-}state\ N\ \beta \sqcap almost\text{-}no\text{-}conflict\text{-}with\text{-}trail\ N\ \beta \sqcap regular\text{-}conflict\text{-}resolution$
 $N\ \beta$
shows $wfp\text{-}on\ \{S. (regular\text{-}scl\ N\ \beta)^{**}\ initial\text{-}state\ S\} (regular\text{-}scl\ N\ \beta)^{-1-1}$
proof (rule $wfp\text{-}on\text{-}subset$)
show $wfp\text{-}on\ \{S. invars\ S\} (regular\text{-}scl\ N\ \beta)^{-1-1}$
by (rule $termination\text{-}regular\text{-}scl\text{-}invars(1)$)[of $\beta\ N$, folded $invars\text{-}def$]
next
note $rea\text{-}to\text{-}scl = scl\text{-}if\text{-}reasonable$
note $reg\text{-}to\text{-}rea = reasonable\text{-}if\text{-}regular$
note $reg\text{-}to\text{-}scl = reg\text{-}to\text{-}rea[THEN\ rea\text{-}to\text{-}scl]$

have $invars\ initial\text{-}state$
by ($simp\ add: invars\text{-}def$)

moreover have $\bigwedge S\ S'. regular\text{-}scl\ N\ \beta\ S\ S' \implies invars\ S \implies invars\ S'$
unfolding $invars\text{-}def$
using
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}trail\text{-}atoms\text{-}lt]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}trail\text{-}resolved\text{-}lits\text{-}pol]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}trail\text{-}lits\text{-}ground]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}initial\text{-}lits\text{-}generalize\text{-}learned\text{-}trail\text{-}conflict]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}ground\text{-}closures]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}ground\text{-}false\text{-}closures]$
 $reg\text{-}to\text{-}scl[THEN\ scl\text{-}preserves\text{-}sound\text{-}state]$
 $regular\text{-}scl\text{-}preserves\text{-}almost\text{-}no\text{-}conflict\text{-}with\text{-}trail$
 $regular\text{-}scl\text{-}preserves\text{-}regular\text{-}conflict\text{-}resolution$
by $simp$
ultimately have $(regular\text{-}scl\ N\ \beta)^{**}\ initial\text{-}state\ S \implies invars\ S$ **for** S
by ($auto\ elim: rtranclp\text{-}induct$)
thus $\{S. (regular\text{-}scl\ N\ \beta)^{**}\ initial\text{-}state\ S\} \subseteq \{S. invars\ S\}$
by $auto$
qed

corollary $termination\text{-}projectable\text{-}strategy$:
fixes
 $N :: ('f, 'v)\ Term.term\ clause\ fset$ **and**
 $\beta :: ('f, 'v)\ Term.term$ **and**
 $strategy$ **and** $strategy\text{-}init$ **and** $proj$
assumes $strategy\text{-}restricts\text{-}regular\text{-}scl$:
 $\bigwedge S\ S'. strategy^{**}\ strategy\text{-}init\ S \implies strategy\ S\ S' \implies regular\text{-}scl\ N\ \beta\ (proj\ S)$
 $(proj\ S')$ **and**
 $initial\text{-}state: proj\ strategy\text{-}init = initial\text{-}state$
shows $wfp\text{-}on\ \{S. strategy^{**}\ strategy\text{-}init\ S\} strategy^{-1-1}$
proof (rule $wfp\text{-}on\text{-}mono\text{-}stronger$)
show $wfp\text{-}on\ \{proj\ S \mid S. strategy^{**}\ strategy\text{-}init\ S\} (regular\text{-}scl\ N\ \beta)^{-1-1}$

```

proof (rule wfp-on-subset)
  show wfp-on {S. (regular-scl N β)** initial-state S} (regular-scl N β)-1-1
    using termination-regular-scl by metis
next
  show {proj S | S. strategy** strategy-init S} ⊆ {S. (regular-scl N β)** ini-
tial-state S}
  proof (intro Collect-mono impI, elim exE conjE)
    fix s S assume s = proj S and strategy** strategy-init S
    show (regular-scl N β)** initial-state s
      unfolding ⟨s = proj S⟩
      using ⟨strategy** strategy-init S⟩
    proof (induction S rule: rtranclp-induct)
      case base
      thus ?case
      unfolding initial-state by simp
    next
      case (step y z)
      thus ?case
      using strategy-restricts-regular-scl
      by (meson rtranclp.simps)
    qed
  qed
qed
next
  show proj ‘ {S. strategy** strategy-init S} ⊆ {proj S | S. strategy** strategy-init
S}
    by blast
next
  show ∧S' S. S ∈ {S. strategy** strategy-init S} ⇒ strategy-1-1 S' S ⇒
(regular-scl N β)-1-1 (proj S') (proj S)
    using strategy-restricts-regular-scl by simp
qed

corollary termination-strategy:
  fixes
    N :: ('f, 'v) Term.term clause fset and
    β :: ('f, 'v) Term.term
  assumes strategy-restricts-regular-scl: ∧S S'. strategy S S' ⇒ regular-scl N β
S S'
  shows wfp-on {S. strategy** initial-state S} strategy-1-1
  using termination-projectable-strategy[of strategy initial-state N β λx. x]
  using assms by metis

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 $N \beta$ *gnd-N*

defines

$gnd-N \equiv$ *grounding-of-class* (*fset* N) **and**

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

assumes

unsat: \neg *satisfiable* *gnd-N-lt-beta* **and**

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

no-more-step: $\nexists S'. regular-scl N \beta S S'$

shows $\exists \gamma. state-conflict S = Some (\{\#\}, \gamma)$

using *unsat correct-termination-regular-scl-run*[*OF run no-more-step*]

by (*simp add: gnd-N-lt-beta-def gnd-N-def*)

theorem (in *scl-fol-calculus*)

fixes $N \beta$ *gnd-N*

defines

$gnd-N \equiv$ *grounding-of-class* (*fset* N) **and**

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

assumes *unsat*: \neg *satisfiable* *gnd-N-lt-beta*

shows $\exists S. (regular-scl N \beta)** initial-state S \wedge$

$(\nexists S'. regular-scl N \beta S S') \wedge$

$(\exists \gamma. state-conflict S = Some (\{\#\}, \gamma))$

proof –

obtain S **where**

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

no-more-step: $(\nexists S'. regular-scl N \beta S S')$

using *termination-regular-scl*[*THEN* *bex-rtranclp-min-element-if-wfp-on*, *of*

initial-state]

by (*metis* (*no-types*, *opaque-lifting*) *conversep-iff mem-Collect-eq rtranclp.rtrancl-into-rtrancl*
rtranclp.rtrancl-refl)

moreover have $\exists \gamma. state-conflict S = Some (\{\#\}, \gamma)$

using *unsat correct-termination-regular-scl-run*[*OF run no-more-step*]

by (*simp add: gnd-N-lt-beta-def gnd-N-def*)

ultimately show *?thesis*

by *metis*

qed

lemma (in *scl-fol-calculus*) *no-infinite-down-chain*:

$\nexists Ss. \neg lfinite Ss \wedge Lazy-List-Chain.chain (\lambda S S'. regular-scl N \beta S S') (LCons$
initial-state $Ss)$

using *termination-regular-scl wfp-on-rtranclp-conversep-iff-no-infinite-down-chain-llist*
by *metis*

theorem (in *scl-fol-calculus*) *completeness-wrt-bound*:
fixes $N \beta$ *gnd-N*
defines
 $gnd-N \equiv \text{grounding-of-cls} (\text{fset } N)$ **and**
 $gnd-N\text{-lt-}\beta \equiv \{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
assumes *unsat*: \neg *satisfiable gnd-N-lt-}\beta
shows
 $\nexists Ss. \neg \text{lfinit} Ss \wedge \text{Lazy-List-Chain.chain } (\lambda S S'. \text{regular-scl } N \beta S S')$
 $(\text{LCons initial-state } Ss)$ **and**
 $\forall S. (\text{regular-scl } N \beta)^{**} \text{initial-state } S \longrightarrow (\nexists S'. \text{regular-scl } N \beta S S') \longrightarrow$
 $(\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma))$
using *assms regular-scl-run-derives-contradiction-if-unsat no-infinite-down-chain*
by *simp-all**

locale *compact-scl* =
scl-fol-calculus renaming-vars ($<$) :: $(f :: \text{weighted}, 'v)$ *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
 $gnd-N \equiv \text{grounding-of-cls} (\text{fset } N)$
assumes *unsat*: \neg *satisfiable gnd-N*
shows $\exists \beta. \neg$ *satisfiable* $\{C \in gnd-N. \forall L \in \# C. \text{atm-of } L \leq \beta\}$
proof –
from *unsat* **obtain** $gnd-N'$ **where**
 $gnd-N' \subseteq gnd-N$ **and** *finite gnd-N'* **and** *unsat'*: \neg *satisfiable gnd-N'*
using *clausal-logic-compact[of gnd-N]* **by** *metis*

have $gnd-N' \neq \{\}$
using $\langle \neg$ *satisfiable gnd-N'* \rangle **by** *force*

obtain C **where** $C\text{-in}$: $C \in gnd-N'$ **and** $C\text{-min}$: $\forall x \in gnd-N'. x \leq C$
using *finite-has-maximal[OF* \langle *finite gnd-N'* \rangle \langle $gnd-N' \neq \{\}$ \rangle \rangle **by** *force*

show *?thesis*
proof (*cases C*)
case *empty*
let $?S = (\[], \{\|\}, \text{Some } (\{\#\}, \text{Var}))$

show *?thesis*
proof (*rule exI*)
have $\{\#\} \mid \in \mid N$
using $C\text{-in}$ \langle $gnd-N' \subseteq gnd-N$ \rangle
unfolding *empty gnd-N-def*
by (*smt* (*verit*, *del-insts*) *SCL-FOL.grounding-of-cls-def*)

```

      SCL-FOL.subst-cls-empty-iff UN-E mem-Collect-eq subset-iff
      substitution-ops.grounding-of-cls-def)
  hence {#} ∈ gnd-N
  using C-in ⟨gnd-N' ⊆ gnd-N⟩ local.empty by blast
  hence {#} ∈ {C ∈ gnd-N. ∀ L ∈ #C. atm-of L < undefined}
  by force
  thus ¬ satisfiable {C ∈ gnd-N. ∀ L ∈ #C. atm-of L ≤ undefined}
  using unsat'
  by (smt (verit, best) C-min le-multiset-empty-right local.empty mem-Collect-eq
      nless-le
      subset-entailed subset-iff)
  qed
next
case (add x C')
then obtain L where L ∈ # C and L-min: ∀ x ∈ #C. x ≤ L
  using Multiset.bex-greatest-element[of C]
  by (metis empty-not-add-mset finite-set-mset infinite-growing linorder-le-less-linear
      set-mset-eq-empty-iff)

  from L-min C-min have ∀ D ∈ gnd-N'. ∀ K ∈ #D. atm-of K ≤ atm-of L
    by (meson dual-order.trans ex-gt-imp-less-multiset leq-imp-less-eq-atm-of
        verit-comp-simplify1 (3))
  hence gnd-N' ⊆ {D ∈ gnd-N. ∀ K ∈ # D. (atm-of K) ≤ (atm-of L)}
  using ⟨gnd-N' ⊆ gnd-N⟩ subset-Collect-iff by auto
  hence ¬ satisfiable {D ∈ gnd-N. ∀ K ∈ # D. (atm-of K) ≤ (atm-of L)}
  using ⟨¬ satisfiable gnd-N'⟩
  by (meson satisfiable-antimono)
  thus ?thesis
  by auto
  qed
qed

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 β S S' ⇒ trail-lits-ground S ⇒ trail-lits-ground S' and

inv-trail-atoms-lt:

trail-atoms-lt β initial-state

scl N β S S' ⇒ trail-atoms-lt β S ⇒ trail-atoms-lt β S' and

inv-undefined-trail-lits:
 $\forall \Gamma' Ln \Gamma''. \text{state-trail initial-state} = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma'$
(fst Ln)
 $\text{scl } N \beta S S' \Longrightarrow$
 $(\forall \Gamma' Ln \Gamma''. \text{state-trail } S = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' \text{ (fst Ln)})$
 \Longrightarrow
 $(\forall \Gamma' Ln \Gamma''. \text{state-trail } S' = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' \text{ (fst Ln)})$ **and**
inv-ground-closures:
ground-closures initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{ground-closures } S \Longrightarrow \text{ground-closures } S'$ **and**
inv-ground-false-closures:
ground-false-closures initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{ground-false-closures } S \Longrightarrow \text{ground-false-closures } S'$ **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
 $\text{scl } N \beta S S' \Longrightarrow \text{initial-lits-generalize-learned-trail-conflict } N S \Longrightarrow \text{initial-lits-generalize-learned-trail-conflict } N S'$ **and**
inv-sound-state:
sound-state N beta initial-state
 $\text{scl } N \beta S S' \Longrightarrow \text{sound-state } N \beta S \Longrightarrow \text{sound-state } N \beta S'$
using *trail-lits-ground-initial-state scl-preserves-trail-lits-ground*
using *trail-atoms-lt-initial-state scl-preserves-trail-atoms-lt*
using *trail-lits-consistent-initial-state[unfolded trail-lits-consistent-def trail-consistent-iff]*
 $\text{scl-preserves-trail-lits-consistent[unfolded trail-lits-consistent-def trail-consistent-iff]}$
using *ground-closures-initial-state scl-preserves-ground-closures*
using *ground-false-closures-initial-state scl-preserves-ground-false-closures*
using *trail-propagated-lit-wf-initial-state scl-preserves-trail-propagated-lit-wf*
using *trail-resolved-lits-pol-initial-state scl-preserves-trail-resolved-lits-pol*
using *initial-lits-generalize-learned-trail-conflict-initial-state*
 $\text{scl-preserves-initial-lits-generalize-learned-trail-conflict}$
using *sound-initial-state scl-preserves-sound-state*
by *metis+*

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.