

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

Martin Desharnais

September 13, 2023

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)	33
5.2	Rules	34
5.3	Well-Defined	36
5.4	Miscellaneous Lemmas	41
6	Invariants	47
6.1	Initial Literals Generalize Learned, Trail, and Conflict Literals	47
6.2	Trail Literals Come From Clauses	53
6.3	Trail Literals Come From Initial Clauses	56
6.4	Trail Literals Are Ground	57
6.5	Trail Literals Are Defined Only Once	59
6.6	Trail Closures Are False In Subtrails	61
6.7	Trail Literals Were Propagated or Decided	63
6.8	Trail Atoms Are Less Than Bound	71
6.9	Trail Resolved Literals Have Unique Polarity	72
6.10	Trail And Conflict Closures Are Ground	74
6.11	Trail And Conflict Closures Are Ground And False	79
6.12	Learned Clauses Are Non-empty	84
6.13	Backtrack Follows Conflict Resolution	85
6.14	Miscellaneous Lemmas	87
7	Soundness	89
7.1	Sound Trail	89
7.2	Sound State	89
7.3	Initial State Is Sound	90
7.4	SCL Rules Preserve Soundness	90
8	Strategies	97
9	Monotonicity w.r.t. the Bounding Atom	100
9.1	Examples	124
9.2	Miscellaneous Lemmas	125
9.3	Well-Defined	126
9.4	Strict Partial Order	127
9.5	Strict Total (w.r.t. Elements in Trail) Order	129
9.6	Well-Founded	130
9.7	Extension on All Literals	134
9.7.1	Well-Founded	136
9.8	Alternative only for terms	137

10 Reasonable Steps	142
10.1 Invariants	142
10.1.1 No Conflict After Decide	142
10.2 Miscellaneous Lemmas	145
11 Regular Steps	145
11.1 Invariants	146
11.1.1 Almost No Conflict With Trail	146
11.1.2 Backtrack Follows Regular Conflict Resolution	150
11.2 Miscellaneous Lemmas	162
12 Resolve in Regular Runs	167
13 Clause Redundancy	173
14 Trail-Induced Ordering	175
14.1 Miscellaneous Lemmas	175
14.2 Strict Partial Order	176
14.3 Properties	177
15 Dynamic Non-Redundancy	178
16 Static Non-Redundancy	193
16.1 Basic Results	197
16.1.1 Minimal-element characterization of well-foundedness	198
16.2 Bound Restriction and Monotonicity	199
17 Extra Lemmas	208
17.1 Set Extra	208
17.2 Prod Extra	208
17.3 Wellfounded Extra	209
17.4 FSet Extra	209
18 Termination	210
18.1 SCL without backtracking terminates	210
18.2 Backtracking can only be done finitely often	218
18.3 Regular SCL terminates	222
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: $\text{finite } V \implies \text{renaming } V \ x \notin V$ **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*:
 $\{x \in \text{insert } y \text{ } S. P \ x\} = (\text{if } P \ y \text{ then insert } y \text{ else id}) \{x \in S. P \ x\}$
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  $x \in F \vee y \in F$ )
    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  $x \neq y$  by (rule x-neq-y)
  next
  show  $x \notin F$  using False by simp
  next
  show  $y \notin F$  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 Cons-nth-drop-Suc add-diff-cancel-left' add-diff-cancel-right'*
canonically-ordered-monoid-add-class.lessE id-take-nth-drop length-append
length-drop)

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 map f xs f y 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*($x := Var\ x'$)) (*subst-domain* (*Var*($x := Var\ x'$)))
apply (*rule inj-onI*)
apply (*simp add: subst-domain-def*)
by *presburger*
qed

lemma *ex-mgu-if-subst-eq-subst-and-disj-vars*:
fixes $t\ u :: ('f, 'v)\ Term.term$ **and** $\sigma_t\ \sigma_u :: ('f, 'v)\ subst$
assumes $t \cdot \sigma_t = u \cdot \sigma_u$ **and** *vars-term* $t \cap vars-term\ u = \{\}$
shows $\exists \mu :: ('f, 'v)\ subst.$ *Unification.mgu* $t\ u = Some\ \mu$
proof –
from *assms* **obtain** $\sigma :: ('f, 'v)\ subst$ **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: $\forall x \in A.$ *vars-term* ($\sigma_A\ x$) $\cap subst-domain\ \sigma_B = \{\}$
defines $\sigma \equiv restrict-subst-domain\ A\ \sigma_A \circ_s \sigma_B$
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* $A\ \sigma_A\ x = Var\ x$
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** · 67)

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

global-interpretation *substitution-ops subst-apply-term Var subst-compose* .

notation *subst-atm-abbrev* (**infixl** ·a 67)

notation *subst-atm-list* (**infixl** ·al 67)

notation *subst-lit* (**infixl** ·l 67)

notation *subst-cls* (**infixl** · 67)

notation *subst-clss* (**infixl** ·cs 67)

notation *subst-cls-list* (**infixl** ·cl 67)

notation *subst-cls-lists* (**infixl** ··cl 67)

notation *comp-subst-abbrev* (**infixl** \odot 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*) *equals0D equals0I is-ground-atm-def subst-apply-term-empty*)

subst-def subst-simps(1) term.distinct(1) term-subst-eq term-subst-eq-rev)

lemma *is-ground-lit-iff-vars-empty*: *is-ground-lit L* \leftrightarrow *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* \leftrightarrow *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 · a σ)* **and** *v ∈ vars-term A*
shows *is-ground-atm (σ 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 σ)* **and** *v-in-L: v ∈ vars-lit L*
shows *is-ground-atm (σ v)*
proof –
let *?A = atm-of L*
from *v-in-L* **have** *A-p: v ∈ vars-term ?A*
by *auto*
then have *is-ground-atm (?A · a σ)*
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 σ) **and**
v-in-C: v ∈ vars-cls C
shows *is-ground-atm (σ v)*
proof –
from *v-in-C* **obtain** *L* **where** *L-p: L ∈# C v ∈ vars-lit L*
unfolding *vars-cls-def* **by** *auto*
then have *is-ground-lit (subst-lit L σ)*
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* $t \subseteq \text{subst-domain } \gamma$
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* $L \subseteq \text{subst-domain } \gamma$
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* $C \subseteq \text{subst-domain } \sigma$
proof (*rule Set.subsetI*)
fix x **assume** $x \in \text{vars-cls } C$
thus $x \in \text{subst-domain } \sigma$
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 $\forall v \in \text{vars-lit } L. \sigma \ v = \tau \ v$
shows *subst-lit* $L \ \sigma = \text{subst-lit } L \ \tau$
using *assms*
proof (*induction L*)
case (*Pos x*)
then have $\forall v \in \text{vars-term } x. \sigma \ v = \tau \ v \implies \text{subst-atm-abbrev } x \ \sigma = \text{subst-atm-abbrev}$
 $x \ \tau$
using *term-subst-eq* **by** *metis+*
then show *?case*
unfolding *subst-lit-def* **using** *Pos* **by** *auto*
next
case (*Neg x*)
then have $\forall v \in \text{vars-term } x. \sigma \ v = \tau \ v \implies \text{subst-atm-abbrev } x \ \sigma = \text{subst-atm-abbrev}$
 $x \ \tau$
using *term-subst-eq* **by** *metis+*
then show *?case*
unfolding *subst-lit-def* **using** *Neg* **by** *auto*
qed

lemma *same-on-vars-clause*:
assumes $\forall v \in \text{vars-cls } S. \sigma \ v = \tau \ v$
shows *subst-cls* $S \ \sigma = \text{subst-cls } S \ \tau$
by (*smt assms image-eqI image-mset-cong2 mem-simps(9) same-on-vars-lit set-image-mset*
subst-cls-def vars-cls-def)

lemma *same-on-lits-clause*:

assumes $\forall L \in \# C. \text{subst-lit } L \sigma = \text{subst-lit } L \tau$
shows $\text{subst-cls } C \sigma = \text{subst-cls } C \tau$
using *assms unfolding subst-cls-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 $C :: ('f, 'v) \text{ term clause}$ **and** $\sigma :: ('f, 'v) \text{ subst}$
assume $\text{is-ground-cls } (\text{subst-cls } C \sigma)$

hence $\text{ground-atms-}\sigma: \bigwedge v. v \in \text{vars-cls } C \Longrightarrow \text{is-ground-atm } (\sigma v)$
by (*meson is-ground-cls-is-ground-on-var*)

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

have $\text{ground-trm}: \text{is-ground-atm } \text{some-ground-trm}$

unfolding $\text{is-ground-atm-def some-ground-trm-def}$ **by** *auto*

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

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

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

have $\text{all-ground-}\tau: \text{is-ground-atm } (\tau v)$ **for** v

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

case *True*

then show *?thesis*

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

next

case *False*

then show *?thesis*

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

qed

have $\text{is-ground-subst } \tau$

unfolding $\text{is-ground-subst-def}$

proof

fix A

show $\text{is-ground-atm } (A \cdot a \tau)$

proof (*induction* A)

case $(\text{Var } v)$

thus *?case* **using** $\text{all-ground-}\tau$ **by** *auto*

```

next
  case (Fun f As)
  thus ?case using all-ground- $\tau$  by (simp add: is-ground-atm-def)
qed
moreover with  $\tau$ - $\sigma$  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: vars-lit (L · l  $\sigma$ ) = vars-term (atm-of L · a
 $\sigma$ )
  by (cases L) simp-all

lemma vars-subst-lit-eq:
vars-lit (L · 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:
vars-cls (C ·  $\sigma$ ) = ( $\bigcup x \in \text{vars-cls } C. \text{vars-term } (\sigma x)$ )
  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: vars-lit (L · l  $\sigma$ )  $\subseteq$  vars-lit L - subst-domain  $\sigma \cup$ 
range-vars  $\sigma$ 
  using vars-term-subst-apply-term-subset[of atm-of L] by simp

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

lemma vars-cls-plus[simp]: vars-cls (C + D) = vars-cls C  $\cup$  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]: $(\bigcup T \in \{\text{atm-of } ' \text{ set-mset } C\}. \bigcup (\text{vars-term } ' \ T)) = \text{vars-cls } C$
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 (*metis subset-Un-eq subst-cls-union vars-cls-plus-iff vars-subst-cls-eq*)

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

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: disjoint-vars C D \implies E \subseteq # C \implies disjoint-vars E D*
by (*metis disjoint-vars-plus-iff subset-mset.diff-add*)

lemma *disjoint-vars-subst-clsI:*
disjoint-vars C D \implies range-vars $\sigma \cap$ vars-cls D = {} \implies disjoint-vars (C \cdot σ) 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: is-renaming $\rho \longleftrightarrow (\forall x. \text{is-Var } (\rho x)) \wedge \text{inj } \rho$*
(is ?lhs \longleftrightarrow ?rhs)
proof (*rule iffI*)
show *?lhs \implies ?rhs*
unfolding *is-renaming-def*
apply *safe*
apply (*metis subst-apply-eq-Var subst-compose term.disc(1)*)
by (*metis injI subst-compose-def term.sel(1)*)

next
show *?rhs \implies ?lhs*
by (*auto simp: is-renaming-def intro: ex-inverse-of-renaming*)

qed

lemma *subst-cls-idem-if-disj-vars: subst-domain $\sigma \cap$ vars-cls C = {} \implies C \cdot $\sigma =$ C*
by (*metis (mono-tags, lifting) Int-iff empty-iff mem-Collect-eq same-on-vars-clause subst-cls-id-subst subst-domain-def*)

lemma *subst-lit-idem-if-disj-vars: subst-domain $\sigma \cap$ 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: vars-lit L \subseteq V \implies L \cdot l restrict-subst-domain V $\sigma =$ L \cdot l σ*
by (*simp add: restrict-subst-domain-def same-on-vars-lit subsetD*)

lemma *subst-cls-restrict-subst-domain: vars-cls C \subseteq V \implies C \cdot restrict-subst-domain V $\sigma =$ C \cdot σ*
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 η) (U \cdot cs η)*
by (*simp add: subst-cls-def*)

```

lemma valid-grounding-of-renaming:
  assumes is-renaming  $\varrho$ 
  shows  $I \models_s \text{grounding-of-cls } (C \cdot \varrho) \longleftrightarrow I \models_s \text{grounding-of-cls } C$ 
proof –
  have  $\text{grounding-of-cls } (C \cdot \varrho) = \text{grounding-of-cls } C$ 
    by (rule grounding-of-subst-cls-renaming-ident[OF  $\langle \text{is-renaming } \varrho \rangle$ ])
  thus ?thesis
    by simp
qed

lemma is-unifier-iff-mem-unifiers-Times:
  assumes fin-AA: finite  $AA$ 
  shows is-unifier  $v$   $AA \longleftrightarrow v \in \text{unifiers } (AA \times AA)$ 
proof (rule iffI)
  assume unif-v-AA: is-unifier  $v$   $AA$ 
  show  $v \in \text{unifiers } (AA \times AA)$ 
  unfolding unifiers-def mem-Collect-eq
  proof (rule ballI)
    have  $\text{card } (AA \cdot_{\text{set}} v) \leq 1$ 
      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  $\text{card } (AA \cdot_{\text{set}} v) = 1$ 
      using fin-AA  $\langle \text{card } (AA \cdot_{\text{set}} v) \leq 1 \rangle$  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  $p \cdot a \ v = \text{snd } p \cdot a \ v$ 
      by (simp add: p-def)
  qed
next
  assume unif-v-AA:  $v \in \text{unifiers } (AA \times AA)$ 
  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.prems have
         $v\text{-in: } v \in \text{unifiers } ((\text{insert } (a, a) (\text{Pair } a \text{ ' } AA) \cup (\lambda x. (x, a)) \text{ ' } AA) \cup AA \times AA)$ 
      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 $\rho \cdot$ l rename-subst-domain $\rho \sigma = L \cdot$ 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 $\rho \cdot$ rename-subst-domain $\rho \sigma = D \cdot \sigma$*
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 $\rho \sigma) = L \cdot$ 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 : $\forall x. is-Var (\rho x)$*

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

lemma *range-vars-eq-empty-if-is-ground*:
 $\text{is-ground-cls } (C \cdot \gamma) \implies \text{subst-domain } \gamma \subseteq \text{vars-cls } C \implies \text{range-vars } \gamma = \{\}$
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 imageE is-ground-atm-iff-vars-empty is-ground-cls-iff-vars-empty is-ground-cls-is-ground-on-var vars-cls-subset-subst-domain-if-grounding*)

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 $\text{is-imgu } \mu \ \{\{t, u\}\}$
proof –
have $\text{unifiers } (\{t, u\} \times \{t, u\}) = \text{unifiers } \{(t, u)\}$
by (*auto simp: unifiers-def*)
hence $\text{Unifiers.is-imgu } \mu \ (\{t, u\} \times \{t, u\})$
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**

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

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

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

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** $A \in As \ B \in As$

hence $\exists xs \in \text{set } ass. (A, B) \in \text{set } (\text{pairs } xs)$

using *as-in* **by** (*auto simp: As-def intro: pair-in-set-pairs*)

```

      thus  $A \cdot a \ x = B \cdot a \ x$ 
        using  $x\text{-in}$ [unfolded unifiers-def mem-Collect-eq, rule-format, of (A, B),
simplified]
        by simp
      qed
    qed
  next
    fix  $x$  assume  $is\text{-unifs}\text{-}x: is\text{-unifiers } x \ (set \ (map \ set \ ass))$ 
    show  $x \in unifiers \ (set \ (concat \ (map \ pairs \ ass)))$ 
      unfolding unifiers-def mem-Collect-eq
    proof (rule ballI)
      fix  $p$  assume  $p \in set \ (concat \ (map \ pairs \ ass))$ 
      then obtain  $as$  where  $as \in set \ ass$  and  $p\text{-in}: p \in set \ (pairs \ as)$ 
        by auto
      hence  $is\text{-unif}\text{-}x: is\text{-unifier } x \ (set \ as)$ 
        using  $is\text{-unifs}\text{-}x$ [unfolded is-unifiers-def] by simp
      moreover have  $fst \ p \in set \ as$ 
        by (rule p-in[THEN fst-pair-in-set-if-pair-in-pairs])
      moreover have  $snd \ p \in set \ as$ 
        by (rule p-in[THEN snd-pair-in-set-if-pair-in-pairs])
      ultimately show  $fst \ p \cdot a \ x = snd \ p \cdot a \ x$ 
        unfolding is-unifier-alt[of set as, simplified]
        by blast
    qed
  qed
  ultimately show  $is\text{-imgu } \mu \ AAA$ 
    unfolding Unifiers.is-imgu-def is-imgu-def by simp
qed

```

3.8.4 Renaming Extra

context *renaming-apart* **begin**

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

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

lemma *vars-term-subst-term-Var-comp-renaming-disj'*:

assumes $fin-V$: finite $V1$ **and** sub : $V2 \subseteq V1$
shows $vars-term (t \cdot a (Var \circ renaming V1)) \cap V2 = \{\}$
by (*meson disjoint-iff fin-V sub subsetD vars-term-subst-term-Var-comp-renaming-disj*)

lemma *vars-lit-subst-renaming-disj*:

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

lemma *vars-cls-subst-renaming-disj*:

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

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

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

lemma *is-renaming-renaming-wrt*: finite $N \implies is-renaming (renaming-wrt N)$

by (*simp add: inj-Var-comp-renaming is-renaming-iff vars-cls-def*)

lemma *ex-renaming-to-disjoint-vars*:

fixes C :: $(f, 'a)$ *Term.term clause set* **and** N :: $(f, 'a)$ *Term.term clause set*
assumes fin : finite N

shows $\exists \varrho. is-renaming \varrho \wedge vars-cls (C \cdot \varrho) \cap vars-cls N = \{\}$

proof (*intro exI conjI*)

show *SCL-FOL.is-renaming (renaming-wrt N)*

using *fin is-renaming-renaming-wrt* **by** *metis*

next

show $vars-cls (C \cdot renaming-wrt N) \cap vars-cls N = \{\}$

by (*simp add: fin vars-cls-subst-renaming-disj vars-cls-def*)

qed

end

4 SCL State

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

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

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

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

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

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

Note that, in contrast to Bromberger, Schwarz, and Weidenbach, the level

is not part of the state. It would be redundant because it can always be computed from the trail.

abbreviation *initial-state* :: ('f, 'v) state **where**
initial-state \equiv ($\{\}$, $\{\{\}\}$, None)

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

lemma *state-trail-simp*[simp]: *state-trail* (Γ , U , u) = Γ
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* (Γ , 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* (Γ , 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 ($-$, None) = $\{\{\}\}$ |
class-of-trail-lit ($-$, 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 $\{\}$ = $\{\{\}\}$ |
class-of-trail ($Ln \# \Gamma$) = *class-of-trail-lit* $Ln \ |\cup|$ *class-of-trail* Γ

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

lemma *class-of-trail-append*: *class-of-trail* ($\Gamma_0 \ @ \ \Gamma_1$) = *class-of-trail* $\Gamma_0 \ |\cup|$ *class-of-trail* Γ_1
by (*induction* Γ_0) (*simp-all add: funion-assoc*)

fun *class-of-closure* **where**
class-of-closure None = $\{\{\}\}$ |
class-of-closure (Some (C , $-$)) = $\{| C |\}$

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

abbreviation *trail-propagate* ::

$(f, 'v)$ trail \Rightarrow $(f, 'v)$ term literal \Rightarrow $(f, 'v)$ term clause \Rightarrow $(f, 'v)$ subst \Rightarrow
 $(f, 'v)$ trail **where**
trail-propagate Γ L C $\gamma \equiv$ *propagate-lit* L C γ $\#$ Γ

lemma *suffix-trail-propagate[simp]*: *suffix* Γ (*trail-propagate* Γ L C δ)

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

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

class-of-trail (*trail-propagate* Γ L C γ) = *fininsert* (*add-mset* L C) (*class-of-trail* Γ)
unfolding *propagate-lit-def* **by** *simp*

definition *decide-lit* **where**

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

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

trail-decide Γ $L \equiv$ *decide-lit* L $\#$ Γ

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

class-of-trail (*trail-decide* Γ L) = *class-of-trail* Γ
by (*simp add: decide-lit-def*)

definition *is-decision-lit*

:: $(f, 'v)$ term literal \times $(f, 'v)$ closure-with-lit option \Rightarrow bool **where**
is-decision-lit $Ln \longleftrightarrow$ *snd* $Ln = \text{None}$

definition *trail-interp* :: $(f, 'v)$ trail \Rightarrow $(f, 'v)$ term interp **where**

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

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

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

definition *trail-true-lit* :: $(f, 'v)$ trail \Rightarrow $(f, 'v)$ term literal \Rightarrow bool **where**

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

definition *trail-false-lit* :: $(f, 'v)$ trail \Rightarrow $(f, 'v)$ term literal \Rightarrow bool **where**

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

definition *trail-true-cls* :: $(f, 'v)$ trail \Rightarrow $(f, 'v)$ term clause \Rightarrow bool **where**

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

definition *trail-false-cls* :: $(f, 'v)$ trail \Rightarrow $(f, 'v)$ term clause \Rightarrow bool **where**

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

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

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

definition *trail-defined-lit* :: ('f, 'v) trail \Rightarrow ('f, 'v) term literal \Rightarrow bool **where**
trail-defined-lit Γ $L \longleftrightarrow (L \in \text{fst } \text{' set } \Gamma \vee - L \in \text{fst } \text{' set } \Gamma)$

definition *trail-defined-cls* :: ('f, 'v) trail \Rightarrow ('f, 'v) term clause \Rightarrow bool **where**
trail-defined-cls Γ $C \longleftrightarrow (\forall L \in \# C. \text{trail-defined-lit } \Gamma L)$

lemma *trail-defined-lit-iff-true-or-false*:
trail-defined-lit Γ $L \longleftrightarrow \text{trail-true-lit } \Gamma L \vee \text{trail-false-lit } \Gamma L$
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 Γ $C \Longrightarrow \text{trail-true-cls } \Gamma C \vee \text{trail-false-cls } \Gamma C$
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* Γ $\{\#\}$
by (simp add: *trail-false-cls-def*)

lemma *trail-false-cls-add-mset*:
trail-false-cls Γ (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 Γ ($C + D$) $\longleftrightarrow \text{trail-false-cls } \Gamma C \wedge \text{trail-false-cls } \Gamma D$
by (auto simp: *trail-false-cls-def*)

lemma *not-trail-true-Nil[simp]*:
 $\neg \text{trail-true-lit } [] L$
 $\neg \text{trail-true-cls } [] C$
 $N \neq \{\#\} \Longrightarrow \neg \text{trail-true-cls } [] N$
by (simp-all add: *trail-true-lit-def trail-true-cls-def trail-true-cls-def*)

lemma *not-trail-false-Nil[simp]*:
 $\neg \text{trail-false-lit } [] L$
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-false-lit-if-trail-false-suffix*:
suffix $\Gamma' \Gamma \Longrightarrow \text{trail-false-lit } \Gamma' K \Longrightarrow \text{trail-false-lit } \Gamma K$
by (meson image-mono set-mono-suffix subsetD *trail-false-lit-def*)

lemma *trail-false-cls-if-trail-false-suffix*:
suffix $\Gamma' \Gamma \Longrightarrow \text{trail-false-cls } \Gamma' C \Longrightarrow \text{trail-false-cls } \Gamma C$
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*: $(I1 \cup I2) \models_l L \Longrightarrow I1 \models_l L \vee I2 \models_l L$
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 $\text{trail-false-lit } \Gamma M$
unfolding *trail-false-lit-def*
using *M-neq-L*
by (*cases L*; *cases M*) (*simp-all add: trail-interp-def trail-false-lit-def*)

qed

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

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

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

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

shows *trail-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-false-lit* ΓK
proof (*elim disjE*)
show $K \in\# D \cdot \mu \cdot \sigma \implies \text{trail-false-lit } \Gamma K$
using *ingu- μ unif- σ*
by (*metis is-ingu-def subst-cls-comp-subst subst-cls-union tr-false-cls trail-false-cls-def union-iff*)
next
have $L \cdot l \mu \cdot l \sigma = L \cdot l \sigma$
using *ingu- μ unif- σ* **by** (*metis is-ingu-def subst-lit-comp-subst*)
thus $K = L \cdot l \mu \cdot l \sigma \implies \text{trail-false-lit } \Gamma K$
by (*auto intro: tr-false-cls[unfolding trail-false-cls-def, rule-format]*)
qed
qed

lemma *trail-defined-lit-iff-defined-uminus*: *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*: *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*: *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*: $t \notin \text{atm-of 'fst ' set } \Gamma \implies t \notin \text{trail-interp } \Gamma$
by (*metis (no-types, lifting) atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel(2) mem-Collect-eq trail-interp-conv*)

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*:
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*: *trail-consistent* $(\Gamma @ \Gamma') \implies \text{trail-consistent } \Gamma'$
by (*induction* Γ) (*auto elim*: *trail-consistent.cases*)

lemma *trail-consistent-if-suffix*:
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 *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* $L = K \vee L = \neg K$)
case *True*
then show *?thesis*
proof (*elim disjE*)
assume $L = K$
thus *?thesis*
proof (*cases* L ; *cases* K)
fix $t_L t_K$
from $\langle L = K \rangle$ **show** $L = \text{Pos } t_L \implies K = \text{Pos } t_K \implies \text{?thesis}$
by (*simp add*: *trail-interp-def*)
next
fix $t_L t_K$
from $\langle L = K \rangle$ **show** $L = \text{Neg } t_L \implies K = \text{Neg } t_K \implies \text{?thesis}$
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 $L = \neg K$
then show *?thesis*
proof (*cases* L ; *cases* K)
fix $t_L t_K$
from $\langle L = \neg K \rangle$ **show** $L = \text{Pos } t_L \implies K = \text{Neg } t_K \implies \text{?thesis}$
unfolding *trail-interp-Cons'*
using *Cons.hyps(1) Cons.prem*
by (*metis (no-types, lifting) image-insert insertE list.simps(15) literal.distinct(1)*
prod.sel(1) trail-defined-lit-def trail-true-lit-def)

```

next
  fix  $t_L t_K$ 
  from  $\langle L = - K \rangle$  show  $L = \text{Neg } t_L \implies K = \text{Pos } t_K \implies ?thesis$ 
  unfolding trail-interp-Cons'
  using Cons.hyps(1) Cons.prems
  by (metis (no-types, lifting) image-insert insertE list.simps(15) literal.distinct(1)
      prod.sel(1) trail-defined-lit-def trail-true-lit-def)
  qed simp-all
qed
next
case False
with Cons.prems have trail-true-lit  $\Gamma L$ 
  by (simp add: trail-true-lit-def)
with Cons.IH have trail-interp  $\Gamma \Vdash L$ 
  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  $\Gamma$  and trail-true-cls  $\Gamma C$ 
  shows trail-interp  $\Gamma \Vdash C$ 
proof -
  from  $\langle \text{trail-true-cls } \Gamma C \rangle$  obtain  $L$  where  $L \in \# C$  and trail-true-lit  $\Gamma L$ 
  by (auto simp: trail-true-cls-def)
  show ?thesis
  unfolding true-cls-def
  proof (rule beI[OF -  $\langle L \in \# C \rangle$ ])
    show trail-interp  $\Gamma \Vdash L$ 
    by (rule trail-interp-lit-if-trail-true[OF  $\langle \text{trail-consistent } \Gamma \rangle \langle \text{trail-true-lit } \Gamma L \rangle$ ])
  qed
qed

```

```

lemma trail-true-cls-iff-trail-interp-entails:
  assumes trail-consistent  $\Gamma \forall L \in \# C. \text{trail-defined-lit } \Gamma L$ 
  shows trail-true-cls  $\Gamma C \iff \text{trail-interp } \Gamma \Vdash C$ 
proof (rule iffI)
  assume trail-true-cls  $\Gamma C$ 
  thus trail-interp  $\Gamma \Vdash C$ 
  using assms(1) trail-interp-cls-if-trail-true by fast
next
  assume trail-interp  $\Gamma \Vdash C$ 
  then obtain  $L$  where  $L \in \# C$  and trail-interp  $\Gamma \Vdash L$ 
  by (auto simp: true-cls-def)
  show trail-true-cls  $\Gamma C$ 
  proof (cases L)
    case (Pos t)

```

hence $t \in \text{trail-interp } \Gamma$
using $\langle \text{trail-interp } \Gamma \models_l L \rangle$ **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* $\langle L \in \# C \rangle$ $\langle \text{trail-interp } \Gamma \models_l L \rangle$ *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* $\Gamma \forall L \in \# C. \text{trail-defined-lit } \Gamma L$
shows *trail-false-cls* $\Gamma C \longleftrightarrow \neg \text{trail-interp } \Gamma \models C$
proof (*rule iffI*)
show *trail-false-cls* $\Gamma C \implies \neg \text{trail-interp } \Gamma \models C$
by (*metis* (*mono-tags*, *opaque-lifting*) *assms(1)* *trail-false-cls-def* *trail-false-lit-def*
trail-interp-lit-if-trail-true *trail-true-lit-def* *true-cls-def* *true-lit-iff*
true-lit-simps(1) *true-lit-simps(2)* *uminus-Neg* *uminus-Pos*)
next
show $\neg \text{trail-interp } \Gamma \models C \implies \text{trail-false-cls } \Gamma C$
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:
 $(\forall D K \gamma. Kn = \text{propagate-lit } K D \gamma \longrightarrow \text{trail-false-cls } \Gamma (D \cdot \gamma)) \implies$
 $\text{trail-closures-false } \Gamma \implies \text{trail-closures-false } (Kn \# \Gamma)$

lemma *trail-closures-false-ConsD*: $\text{trail-closures-false } (Ln \# \Gamma) \implies \text{trail-closures-false } \Gamma$
by (*auto elim: trail-closures-false.cases*)

lemma *trail-closures-false-appendD*: $\text{trail-closures-false } (\Gamma @ \Gamma') \implies \text{trail-closures-false } \Gamma'$
by (*induction* Γ) (*auto elim: trail-closures-false.cases*)

lemma *is-ground-lit-if-true-in-ground-trail*:
assumes $\forall L \in \text{fst } \text{'set } \Gamma. \text{is-ground-lit } L$
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 $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{is-ground-lit } L$
shows $\text{trail-false-lit } \Gamma L \implies \text{is-ground-lit } L$
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 $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{is-ground-lit } L$
shows $\text{trail-defined-lit } \Gamma L \implies \text{is-ground-lit } L$
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-cls-if-false-in-ground-trail*:
assumes $\forall L \in \text{fst } ' \text{ set } \Gamma. \text{is-ground-lit } L$
shows $\text{trail-false-cls } \Gamma C \implies \text{is-ground-cls } C$
unfolding *trail-false-cls-def is-ground-cls-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** \prec_B 50)
assumes
 $\text{finite-less-B}: \bigwedge \beta. \text{finite } \{x. x \prec_B \beta\}$
begin

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

5.1 Lemmas About (\prec_B)

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

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

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

lemma *finite-lits-less-B*: $\text{finite } \{L. \text{atm-of } L \prec_B \beta\}$
unfolding *lits-less-B-conv*
proof (*rule finite-UN-I*)
show $\text{finite } \{x. x \prec_B \beta\}$
by (*rule finite-less-B*)

next

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

by *simp*

qed

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

using *finite-lits-less-B* **by** (*simp add: lits-less-eq-B-conv*)

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

by *blast*

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

proof –

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

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

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 $\Rightarrow \text{bool}$ **for** $N \beta$ **where**

propagateI: $C \in | N \cup U \implies C = \text{add-mset } L C' \implies \text{is-ground-cls } (C \cdot \gamma) \implies$

$\forall K \in \# C \cdot \gamma. \text{atm-of } K \preceq_B \beta \implies$

$C_0 = \{\#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma\} \implies C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma\} \implies$

trail-false-cls $\Gamma (C_0 \cdot \gamma) \implies \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$

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

propagate $N \beta (\Gamma, U, \text{None}) (\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, \text{None})$

lemma $C \in | N \cup U \implies C = \text{add-mset } L C' \implies \text{is-ground-cls } (C \cdot \gamma) \implies$

$\forall K \in \# C. \text{atm-of } (K \cdot l \gamma) \preceq_B \beta \implies$

$C_0 = \{\#K \in \# C'. K \cdot l \gamma \neq L \cdot l \gamma\} \implies C_1 = \{\#K \in \# C'. K \cdot l \gamma = L \cdot l \gamma\} \implies$

trail-false-cls $\Gamma (C_0 \cdot \gamma) \implies \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \implies$

$is-imgu \mu \{atm-of \text{ ' set-mset (add-mset L C}_1)\} \implies$
 $propagate N \beta (\Gamma, U, None) (trail-propagate \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma, U, None)$
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** $N \beta$ **where**
 $decideI: L \in \bigcup (set-mset \text{ ' fset } N) \implies is-ground-lit (L \cdot l \gamma) \implies$
 $\neg trail-defined-lit \Gamma (L \cdot l \gamma) \implies atm-of L \cdot a \gamma \preceq_B \beta \implies$
 $decide N \beta (\Gamma, U, None) (trail-decide \Gamma (L \cdot l \gamma), U, None)$

lemma $add-mset L C |\in| N \implies is-ground-lit (L \cdot l \gamma) \implies$
 $\neg trail-defined-lit \Gamma (L \cdot l \gamma) \implies atm-of L \cdot a \gamma \preceq_B \beta \implies$
 $decide N \beta (\Gamma, U, None) (trail-decide \Gamma (L \cdot l \gamma), U, None)$
by (auto intro!: decideI)

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

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

lemma $-(fst \mathcal{K}) \notin \# D \cdot \sigma \implies skip N \beta (\mathcal{K} \# \Gamma, U, Some (D, \sigma)) (\Gamma, U, Some$
 $(D, \sigma))$
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** $N \beta$ **where**
 $factorizeI: L \cdot l \gamma = L' \cdot l \gamma \implies is-imgu \mu \{\{atm-of L, atm-of L'\}\} \implies$
 $factorize N \beta (\Gamma, U, Some (add-mset L' (add-mset L D), \gamma)) (\Gamma, U, Some$
 $(add-mset L D \cdot \mu, \gamma))$

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

$\varrho_D) \cdot \mu, \gamma))$

inductive backtrack :: $(f, 'v)$ term clause fset \Rightarrow $(f, 'v)$ term \Rightarrow $(f, 'v)$ state \Rightarrow
 $(f, 'v)$ state \Rightarrow bool **for** $N \beta$ **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
backtrack $N \beta (\Gamma, U, \text{Some } (\text{add-mset } L D, \sigma)) (\Gamma'', \text{finsert } (\text{add-mset } L D) U,$
None)

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

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

5.3 Well-Defined

lemma propagate-well-defined:

assumes propagate $N \beta S S'$

shows

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

proof –

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 $\neg \text{decide } N' \beta' S S'$

using S -def S' -def

by (*auto simp add: decide-lit-def propagate-lit-def elim: decide.cases*)

show $\neg \text{conflict } N' \beta' S S'$

using S -def S' -def

by (*auto elim: conflict.cases*)

show $\neg \text{skip } N' \beta' S S'$

```

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, None) and
  S'-def: S' = (trail-decide  $\Gamma$  (L ·l  $\gamma$ ), U, 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 $S\text{-def } S'\text{-def}$
 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**
 $S\text{-def}: S = (\Gamma, U, \text{None})$ **and**
 $S'\text{-def}: S' = (\Gamma, U, \text{Some } (C, \gamma))$
 by (auto elim: conflict.cases)

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

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

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

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

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

show \neg backtrack $N' \beta' S S'$
 using $S\text{-def } S'\text{-def}$
 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 β S S'*

shows

¬ *propagate N β S S'*

¬ *decide N β S S'*

¬ *conflict N β S S'*

¬ *skip N β S S'*

¬ *backtrack N β 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 β S S'*

shows

¬ *propagate N' β' S S'*

¬ *decide N' β' S S'*

¬ *conflict N' β' S S'*

¬ *skip N' β' S S'*

¬ *factorize N' β' S S'*

¬ *resolve N' β' S S'*

proof –

from *assms* **obtain** $\Gamma \Gamma'' U C \gamma$ **where**

S-def: S = (Γ, U, Some (C, γ)) **and**

S'-def: S' = (Γ'', finsert (C) U, None)

by (*auto elim: backtrack.cases*)

show ¬ *propagate N' β' S S'*

using *S-def S'-def*

by (*auto elim: propagate.cases*)

show ¬ *decide N' β' S S'*

using *S-def S'-def*

by (*auto elim: decide.cases*)

show ¬ *conflict N' β' S S'*

using *S-def S'-def*

by (*auto elim: conflict.cases*)

show ¬ *skip N' β' S S'*

using *S-def S'-def*

by (*auto elim: skip.cases*)

show ¬ *factorize N' β' S S'*

using $S\text{-def } S'\text{-def}$
by (*auto elim: factorize.cases*)

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

5.4 Miscellaneous Lemmas

lemma *conflict-set-after-factorization:*

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

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: $\nexists S'. \text{conflict } N \beta S S'$ **and**
could-conf: state-conflict $S = \text{None}$ **and**
C-in: $C \in |N| \cup |U|$ *state-learned* S **and**
gr-C- γ : is-ground-cls $(C \cdot \gamma)$
shows $\neg \text{trail-false-cls } (\text{state-trail } S) (C \cdot \gamma)$
proof (*rule notI*)
assume *tr-false: trail-false-cls* $(\text{state-trail } S) (C \cdot \gamma)$

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

```

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

have conflict N β (Γ, U, None) (Γ, U,
  Some (C, restrict-subst-domain (vars-cls C) γ))
proof (rule conflictI)
  show C |∈| N |∪| U
    using C-in by (simp add: S-def)
next
  show is-ground-cls (C · restrict-subst-domain (vars-cls C) γ)
    using gr-C-γ by (simp add: subst-cls-restrict-subst-domain)
next
  show trail-false-cls Γ (C · restrict-subst-domain (vars-cls C) γ)
    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-empty-not-in-sate-learned:
  scl N β S S' ⇒ {#} |∉| state-learned S ⇒ {#} |∉| state-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-empty-in-initial-clauses-and-no-conflict:
  assumes {#} |∈| N and state-conflict S = None
  shows conflict N β S (state-trail S, state-learned S, Some ({#}, Var))
proof -
  from assms(2) obtain Γ U where S-def: S = (Γ, 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 {#} |∈| N |∪| U
      by simp
    qed simp-all
  qed

lemma conflict-initial-state-if-empty-in-intial-clauses:
  {#} |∈| N ⇒ conflict N β initial-state ([], {||}, Some ({#}, Var))
  using conflict-if-empty-in-initial-clauses-and-no-conflict by auto

lemma conflict-empty-trail:
  assumes conf: conflict N β S S' and empty-trail: state-trail S = []
  shows {#} |∈| N |∪| state-learned S
  using conf
proof (cases N β S S' rule: conflict.cases)
  case (conflictI D U γ Γ)

```

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 $\langle D \mid \in \mid N \mid \cup \mid U \rangle$ **show** *?thesis*
unfolding *conflictI(1,2)* **by** *simp*
qed

lemma *conflict-empty-trail'*:
assumes $\{\#\} \mid \in \mid N \mid \cup \mid U$
shows $\exists S'. \text{conflict } N \beta ([] , U, \text{None}) S'$
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 *mempty-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*: $D \in \text{grounding-of-cls } (fset N \cup fset U)$
shows $\exists S'. \text{conflict } N \beta (\Gamma , U, \text{None}) S'$
proof –
from *D-in* **obtain** $D' \gamma'$ **where**

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

define γ **where**

$\gamma \equiv$ *restrict-subst-domain* (*vars-cls* D') γ'

have *conflict* $N \beta$ ($\Gamma, U, None$) ($\Gamma, U, Some$ (D', γ))

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

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

using gr - D - γ **by** (*simp* *add*: γ -def *subst-cls-restrict-subst-domain*)

next

show *trail-false-cls* Γ ($D' \cdot \gamma$)

using *tr-false- Γ - D* **by** (*simp* *add*: D -def γ -def *subst-cls-restrict-subst-domain*)

qed

thus *?thesis*

by *auto*

qed

lemma *no-conflict-tail-trail*:

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

shows $\nexists S$. *conflict* $N \beta$ ($\Gamma, U, None$) S

proof (*rule* *notI*, *erule* *exE*)

fix S **assume** *conflict* $N \beta$ ($\Gamma, U, None$) S

hence $\exists S$. *conflict* $N \beta$ ($Ln \# \Gamma, U, None$) S

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

case (*conflictI* $D \gamma$)

have *conflict* $N \beta$ ($Ln \# \Gamma, U, None$) ($Ln \# \Gamma, U, Some$ (D, γ))

proof (*rule* *conflict.conflictI*)

show $D \mid \in \mid N \mid \cup \mid U$

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 *trail-false-cls* Γ ($D \cdot \gamma$) \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**
 $\text{dom-}\gamma_C: \text{subst-domain } \gamma_C \subseteq \text{vars-cls } (\text{add-mset } L C)$
shows $\text{subst-domain } (\text{rename-subst-domain } \varrho_C \gamma_C) \subseteq \text{vars-cls } (\text{add-mset } L C \cdot \varrho_C)$
proof –
have $\text{subst-domain } (\text{rename-subst-domain } \varrho_C \gamma_C) \subseteq \text{the-Var } \langle \varrho_C \text{ ' subst-domain } \gamma_C \rangle$
using $\text{subst-domain-rename-subst-domain-subset}[OF \langle \forall x. \text{is-Var } (\varrho_C x) \rangle]$ **by** *simp*
also have $\dots \subseteq \text{the-Var } \langle \varrho_C \text{ ' vars-cls } (\text{add-mset } L C) \rangle$
using $\text{dom-}\gamma_C$ **by** *auto*
also have $\dots = (\bigcup x \in \text{vars-cls } (\text{add-mset } L C). \text{vars-term } (\varrho_C x))$
using $\text{image-the-Var-image-subst-renaming-eq}[OF \langle \forall x. \text{is-Var } (\varrho_C x) \rangle]$ **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 $\text{dom-ren-dom-}\varrho_C\text{-}\gamma_C:$
 $\text{subst-domain } (\text{rename-subst-domain } \varrho_C \gamma_C) \subseteq \text{vars-cls } (\text{add-mset } L C \cdot \varrho_C)$
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**
 $\text{ground-conf}: \text{is-ground-cls } (\text{add-mset } L C \cdot \gamma_C)$ **and**
 $\text{ground-prop}: \text{is-ground-cls } (\text{add-mset } K D \cdot \gamma_D)$ **and**
 $\text{dom-}\gamma_D: \text{subst-domain } \gamma_D \subseteq \text{vars-cls } (\text{add-mset } K D)$ **and**
 $\text{ren-}\varrho_C: \text{is-renaming } \varrho_C$ **and**
 $\text{ren-}\varrho_D: \text{is-renaming } \varrho_D$ **and**
 $\text{disjoint-vars}: \text{vars-cls } (\text{add-mset } L C \cdot \varrho_C) \cap \text{vars-cls } (\text{add-mset } K D \cdot \varrho_D) = \{\}$
defines $\gamma \equiv \text{rename-subst-domain } \varrho_D \gamma_D \odot \text{rename-subst-domain } \varrho_C \gamma_C$
shows
 $\text{subst-renamed-comp-renamed-simp}:$
 $L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C C \cdot \varrho_C \cdot \gamma = C \cdot \gamma_C$
 $K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D D \cdot \varrho_D \cdot \gamma = D \cdot \gamma_D$ **and**
 $\text{imgu-comp-renamed-comp-renamed-simp}:$
 $\text{is-imgu } \mu \{\{\text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D\}\} \implies \mu \odot \gamma = \gamma$
proof –
have $\text{subst-adapt-}\varrho_D\text{-}\gamma_D:$
 $\text{subst-domain } (\text{rename-subst-domain } \varrho_D \gamma_D) \cap \text{vars-cls } (\text{add-mset } L C \cdot \varrho_C) = \{\}$
using $\text{disjoint-vars ren-}\varrho_D \text{ dom-}\gamma_D$
 $\text{subst-domain-rename-subst-domain-subset-vars-cls-subst-cls}$
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 $\gamma\text{-def}$
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 $\text{is-unifiers } \gamma \{ \{ \text{atm-of } L \cdot a \varrho_C, \text{atm-of } K \cdot a \varrho_D \} \}$
by (*simp add: is-unifiers-def is-unifier-alt*)

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

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

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

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

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

unfolding $clss\text{-lits-generalize-clss-lits-def}$ **by** $blast$

lemma $clss\text{-lits-generalize-clss-lits-insert}$:

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

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

unfolding $clss\text{-lits-generalize-clss-lits-def}$ **by** $blast$

lemma $clss\text{-lits-generalize-clss-lits-trans}$:

assumes

$clss\text{-lits-generalize-clss-lits } N1 \ N2$ **and**

$clss\text{-lits-generalize-clss-lits } N2 \ N3$

shows $clss\text{-lits-generalize-clss-lits } N1 \ N3$

proof ($unfold \ clss\text{-lits-generalize-clss-lits-def}$, $rule \ ballI$)

fix $L3$

assume $L3 \in \bigcup (set\text{-mset } 'N3)$

then obtain $L2 \ \sigma2$ **where** $L2 \in \bigcup (set\text{-mset } 'N2)$ **and** $L2 \cdot l \ \sigma2 = L3$

using $assms(2)[unfolded \ clss\text{-lits-generalize-clss-lits-def}]$ **generalizes-lit-def** **by** $meson$

then obtain $L1 \ \sigma1$ **where** $L1 \in \bigcup (set\text{-mset } 'N1)$ **and** $L1 \cdot l \ \sigma1 = L2$

using $assms(1)[unfolded \ clss\text{-lits-generalize-clss-lits-def}]$ **generalizes-lit-def** **by** $meson$

thus $\exists K \in \bigcup (set\text{-mset } 'N1). \text{generalizes-lit } K \ L3$

unfolding $generalizes-lit-def$

proof ($intro \ bexI \ exI \ conjI$)

show $L1 \cdot l \ (\sigma1 \odot \sigma2) = L3$

by ($simp \ add: \langle L1 \cdot l \ \sigma1 = L2 \rangle \langle L2 \cdot l \ \sigma2 = L3 \rangle$)

qed $simp\text{-all}$

qed

lemma $clss\text{-lits-generalize-clss-lits-subst-clss}$:

assumes $clss\text{-lits-generalize-clss-lits } N1 \ N2$

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

unfolding $clss\text{-lits-generalize-clss-lits-def}$

proof ($rule \ ballI$)

fix L **assume** $L \in \bigcup (set\text{-mset } '(\lambda C. C \cdot \sigma) 'N2)$

then obtain $L2$ **where** $L2 \in \bigcup (set\text{-mset } 'N2)$ **and** $L\text{-def}: L = L2 \cdot l \ \sigma$ **by** $auto$

then obtain $L1 \ \sigma1$ **where** $L1\text{-in}: L1 \in \bigcup (set\text{-mset } 'N1)$ **and** $L2\text{-def}: L2 = L1 \cdot l \ \sigma1$

using $assms[unfolded \ clss\text{-lits-generalize-clss-lits-def}]$

unfolding $generalizes-lit-def$ **by** $metis$

show $\exists K \in \bigcup (set\text{-mset } 'N1). \text{generalizes-lit } K \ L$

unfolding $generalizes-lit-def$

proof ($intro \ bexI \ exI$)

show $L1 \in \bigcup (set\text{-mset } 'N1)$

by (rule L_1 -in)
 next
 show $L_1 \cdot l (\sigma_1 \odot \sigma) = L$
 unfolding L -def L_2 -def by simp
 qed
 qed

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

lemma *clss-lits-generalize-clss-lits-subst-clss*:
 assumes $clss-lits-generalize-clss-lits\ N\ \{add-mset\ L1\ (add-mset\ L2\ C)\}$
 shows $clss-lits-generalize-clss-lits\ N\ \{add-mset\ (L1 \cdot l\ \sigma)\ (C \cdot \sigma)\}$
proof (rule *clss-lits-generalize-clss-lits-trans*)
 show $clss-lits-generalize-clss-lits\ N\ \{add-mset\ L1\ (add-mset\ L2\ C) \cdot \sigma\}$
 by (rule *clss-lits-generalize-clss-lits-singleton-subst-clss*[of $N - \sigma$, *OF assms*])
 next
 show $clss-lits-generalize-clss-lits\ \{add-mset\ L1\ (add-mset\ L2\ C) \cdot \sigma\}$
 $\{add-mset\ (L1 \cdot l\ \sigma)\ (C \cdot \sigma)\}$
 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**
 $initial-lits-generalize-learned-trail-conflict\ N\ S \longleftrightarrow clss-lits-generalize-clss-lits\ (fset\ N)$
 $(fset\ (state-learned\ S\ |\cup| clss-of-trail\ (state-trail\ S)\ |\cup|$
 $(case\ state-conflict\ S\ of\ None \Rightarrow \{\|\}\ | Some\ (C, -) \Rightarrow \{|C|\})))$

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

lemma *propagate-preserves-initial-lits-generalize-learned-trail-conflict*:
 $propagate\ N\ \beta\ S\ S' \implies initial-lits-generalize-learned-trail-conflict\ N\ S \implies$
 $initial-lits-generalize-learned-trail-conflict\ N\ S'$

proof (*induction* $S\ S'$ rule: *propagate.induct*)
 case (*propagateI* $C\ U\ L\ C'\ \gamma\ C_0\ C_1\ \Gamma\ \mu$)

from *propagateI.prem*s **have**
 N -generalize: $clss-lits-generalize-clss-lits\ (fset\ N)\ (fset\ (U\ |\cup| clss-of-trail\ \Gamma))$
 unfolding *initial-lits-generalize-learned-trail-conflict-def* by *simp-all*

from *propagateI.hyps* **have**
 C -in: $C \in |N\ |\cup| U$ **and**
 C -def: $C = add-mset\ L\ C'$ **and**
 C_0 -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-mset L C0 · μ) (fset (U |∪| clss-of-trail Γ)))
unfolding clss-lits-generalize-clss-lits-insert
proof (rule conjI)
show ∀ L ∈# add-mset L C0 · μ. ∃ K ∈ ∪ (set-mset ‘ fset N). generalizes-lit K
L
proof (rule ballI)
  fix K assume K ∈# add-mset L C0 · μ
  hence K = L · l μ ∨ (∃ M. M ∈# C0 ∧ K = M · l μ)
    by auto
  then obtain K' where K'-in: K' ∈# C and K-def: K = K' · l μ
    using C0-def C-def by auto

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

show ∃ K' ∈ ∪ (set-mset ‘ fset N). generalizes-lit K' K
proof (rule bexI)
  show generalizes-lit LD K
    using ⟨generalizes-lit LD K'⟩
    by (metis generalizes-lit-def K-def subst-lit-comp-subst)
  next
    show ⟨LD ∈ ∪ (set-mset ‘ fset N)⟩
      using ⟨D |∈| N⟩ ⟨LD ∈# D⟩
      by (meson UN-I)
    qed
  qed
next
  show clss-lits-generalize-clss-lits (fset N) (fset (U |∪| clss-of-trail Γ))
    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 β S S' ⇒ initial-lits-generalize-learned-trail-conflict N S ⇒
  initial-lits-generalize-learned-trail-conflict N S'
proof (induction S S' rule: decide.induct)
  case (decideI L Γ 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 β 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* Γ))
unfolding *conflictI*(1) **by** (*simp add: initial-lits-generalize-learned-trail-conflict-def*)
hence *ball-U- Γ -generalize*:
 $\wedge L. L \in \bigcup (\text{set-mset } 'fset (U \mid \cup \mid \text{clss-of-trail } \Gamma)) \implies$
 $\exists K \in \bigcup (\text{set-mset } 'fset N). \text{generalizes-lit } K L$
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* Γ)))
unfolding *clss-lits-generalize-clss-lits-def*
proof (*rule ballI*)
fix L **assume** $L \in \bigcup (\text{set-mset } 'insert D (fset (U \mid \cup \mid \text{clss-of-trail } \Gamma)))$
hence $L \in \text{set-mset } D \vee L \in \bigcup (\text{set-mset } 'fset (U \mid \cup \mid \text{clss-of-trail } \Gamma))$
by *simp*
thus $\exists K \in \bigcup (\text{set-mset } 'fset N). \text{generalizes-lit } K L$
proof (*elim disjE*)
assume *L-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-in*
by (*meson UN-I generalizes-lit-refl*)
next
assume $D \mid \in \mid U$
hence $\exists K \in \bigcup (\text{set-mset } 'fset N). \text{generalizes-lit } K L$
using *ball-U- Γ -generalize*[*of* L] *L-in*
using *mk-disjoint-finsert* **by** *fastforce*
thus *?thesis*
by *metis*
qed
next
show $L \in \bigcup (\text{set-mset } 'fset (U \mid \cup \mid \text{clss-of-trail } \Gamma)) \implies ?thesis$
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*:
 $skip N \beta S S' \implies \text{initial-lits-generalize-learned-trail-conflict } N S \implies$
 $\text{initial-lits-generalize-learned-trail-conflict } N S'$
proof (*induction* $S S'$ *rule: skip.induct*)
case (*skipI* $L D \sigma Cl \Gamma U$)

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:*
factorize 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: factorize.induct*)
case (*factorizeI L γ L' μ Γ U D*)
moreover have *clss-lits-generalize-clss-lits (fset N) {add-mset (L · l μ) (D · μ)}*
using *factorizeI*
unfolding *initial-lits-generalize-learned-trail-conflict-def*
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*
apply *simp*
using *clss-lits-generalize-clss-lits-insert* **by** *blast*
qed

lemma *resolve-preserves-initial-lits-generalize-learned-trail-conflict:*
resolve 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: 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 · ρ_C + D · ρ_D) · μ}*
proof –
from *resolveI.prem*s **have**
N-lits-sup: clss-lits-generalize-clss-lits (fset N)
(fset (U |∪| clss-of-trail Γ |∪| {add-mset L C}))
unfolding *initial-lits-generalize-learned-trail-conflict-def* **by** *simp*

have *clss-lits-generalize-clss-lits (fset N) {C · ρ_C · 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 · ρ_D · μ}*
proof –
from *N-lits-sup* **have** *clss-lits-generalize-clss-lits (fset N) (fset (clss-of-trail*
Γ))
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 Come From Clauses

```

definition trail-lits-from-clauses where
  trail-lits-from-clauses N S ↔
  ( $\forall L \in \text{fst } ' \text{set } (\text{state-trail } S).$ 
     $\exists L' \in \bigcup (\text{set-mset } ' (\text{fset } N \cup \text{fset } (\text{state-learned } S))). \text{generalizes-lit } L' L$ )

```

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

lemma *propagate-preserves-trail-lits-from-clauses*:
assumes *propagate* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
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 $\exists L' \in \bigcup (fset (set-mset \mid \uparrow (N \mid \cup \mid U)))$. *generalizes-lit* $L' (L \cdot l \mu \cdot l \gamma)$

proof (*rule* *bestI*[*of* - L])

show $L \in \bigcup (fset (set-mset \mid \uparrow (N \mid \cup \mid U)))$

using $\langle C \mid \in \mid N \mid \cup \mid U \rangle \langle C = add-mset L C' \rangle$

by (*meson* *Union-iff fimage-eqI union-single-eq-member*)

next

show *generalizes-lit* $L (L \cdot l \mu \cdot l \gamma)$

unfolding *generalizes-lit-def* **by** (*metis* *subst-lit-comp-subst*)

qed

moreover have $\forall L \in fst \text{ ' set } \Gamma$. $\exists L' \in \bigcup (fset (set-mset \mid \uparrow (N \mid \cup \mid U)))$.
generalizes-lit $L' L$

using *assms(2)* **unfolding** *propagateI* **by** (*simp add: trail-lits-from-clauses-def*)

ultimately show *?thesis*

unfolding *propagateI(2)* **by** (*simp add: trail-lits-from-clauses-def propagate-lit-def*)

qed

lemma *decide-preserves-trail-lits-from-clauses*:
assumes *decide* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
using *assms(1)*

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

case (*decideI* $L \gamma \Gamma U$)

hence $\exists L' \in \bigcup (fset (set-mset \mid \uparrow (N \mid \cup \mid U)))$. *generalizes-lit* $L' (L \cdot l \gamma)$

by (*auto simp: generalizes-lit-def*)

moreover have $\forall L \in fst \text{ ' set } \Gamma$. $\exists L' \in \bigcup (fset (set-mset \mid \uparrow (N \mid \cup \mid U)))$.
generalizes-lit $L' L$

using *assms(2)* **unfolding** *decideI* **by** (*simp add: trail-lits-from-clauses-def*)

ultimately show *?thesis*

unfolding *decideI* **by** (*simp add: trail-lits-from-clauses-def decide-lit-def*)

qed

lemma *conflict-preserves-trail-lits-from-clauses*:
assumes *conflict* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$

shows *trail-lits-from-clauses* $N S'$
using *assms(1)*
proof (*cases* $N \beta S S'$ *rule: conflict.cases*)
case (*conflictI* $D U \gamma \Gamma$)
thus *?thesis*
using *assms(2)* **by** (*simp add: trail-lits-from-clauses-def*)
qed

lemma *skip-preserves-trail-lits-from-clauses:*
assumes *skip* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
using *assms(1)*
proof (*cases* $N \beta S S'$ *rule: skip.cases*)
case (*skipI* $L D \sigma n \Gamma U$)
thus *?thesis*
using *assms(2)* **by** (*simp add: trail-lits-from-clauses-def*)
qed

lemma *factorize-preserves-trail-lits-from-clauses:*
assumes *factorize* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
using *assms(1)*
proof (*cases* $N \beta S S'$ *rule: factorize.cases*)
case (*factorizeI* $L \gamma L' \mu \Gamma U D$)
thus *?thesis*
using *assms(2)* **by** (*simp add: trail-lits-from-clauses-def*)
qed

lemma *resolve-preserves-trail-lits-from-clauses:*
assumes *resolve* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
using *assms(1)*
proof (*cases* $N \beta S S'$ *rule: resolve.cases*)
case (*resolveI* $\Gamma \Gamma' L C \delta \rho U D L' \sigma \mu$)
thus *?thesis*
using *assms(2)* **by** (*simp add: trail-lits-from-clauses-def*)
qed

lemma *backtrack-preserves-trail-lits-from-clauses:*
assumes *backtrack* $N \beta S S'$ **and** *trail-lits-from-clauses* $N S$
shows *trail-lits-from-clauses* $N S'$
using *assms(1)*
proof (*cases* $N \beta S S'$ *rule: backtrack.cases*)
case (*backtrackI* $\Gamma \Gamma' \Gamma'' K L \sigma D U$)
hence *suffix* $\Gamma'' \Gamma$
by (*simp add: suffixI decide-lit-def*)
hence *set* $\Gamma'' \subseteq \text{set } \Gamma$
by (*simp add: set-mono-suffix*)

moreover have $\forall L \in \text{fst } \langle \text{set } \Gamma. \exists L' \in \bigcup (\text{fset } (\text{set-mset } |^q| (N \cup U)))$.
generalizes-lit $L' L$
using *assms(2)* **unfolding** *backtrackI* **by** (*simp add: trail-lits-from-clauses-def*)

ultimately have $\forall L \in \text{fst } \langle \text{set } \Gamma''. \exists L' \in \bigcup (\text{fset } (\text{set-mset } |^q| (N \cup U)))$.
generalizes-lit $L' L$
by *fast*
thus *?thesis*
unfolding *trail-lits-from-clauses-def backtrackI(1,2) state-trail-simp state-learned-simp*
by *auto*
qed

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

6.3 Trail Literals Come From Initial Clauses

definition *trail-lits-from-init-clauses* **where**
trail-lits-from-init-clauses N S \longleftrightarrow
 $(\forall L \in \text{fst } \langle \text{set } (\text{state-trail } S). \exists L' \in \bigcup (\text{set-mset } \langle \text{fset } N \rangle). \text{generalizes-lit } L' L)$

lemma *trail-lits-from-init-clausesI*:
assumes *trail-lits-from-clauses N S* **and** *initial-lits-generalize-learned-trail-conflict N S*
shows *trail-lits-from-init-clauses N S*
unfolding *trail-lits-from-init-clauses-def*
proof (*rule ballI*)
fix L **assume** $L \in \text{fst } \langle \text{set } (\text{state-trail } S)$
with *assms(1)* **obtain** L' **where**
 $L' \in \bigcup (\text{set-mset } \langle \text{fset } N \cup \text{fset } (\text{state-learned } S) \rangle) \wedge \text{generalizes-lit } L' L$
unfolding *trail-lits-from-clauses-def* **by** *metis*
hence $(\exists x \in \text{fset } N. L' \in \# x) \vee (\exists x \in \text{fset } (\text{state-learned } S). L' \in \# x)$ **and**
generalizes-lit $L' L$
by *simp-all*
thus $\exists L' \in \bigcup (\text{set-mset } \langle \text{fset } N \rangle). \text{generalizes-lit } L' L$
proof (*elim disjE bexE*)
fix C **assume** $C \in \text{fset } N$
thus $L' \in \# C \implies ?thesis$
using $\langle \text{generalizes-lit } L' L \rangle$ **by** *auto*
next
fix C **assume** $C \in \text{fset } (\text{state-learned } S)$ **and** $L' \in \# C$
with *assms(2)* **have** $\exists K \in \bigcup (\text{set-mset } \langle \text{fset } N \rangle). \text{generalizes-lit } K L'$

unfolding *initial-lits-generalize-learned-trail-conflict-def*
class-lits-generalize-class-lits-def
by *auto*
thus *?thesis*
using $\langle \text{generalizes-lit } L' L \rangle$ **by** (*metis generalizes-lit-def subst-lit-comp-subst*)
qed
qed

6.4 Trail Literals Are Ground

definition *trail-lits-ground where*

trail-lits-ground $S \longleftrightarrow (\forall L \in \text{fst } ' \text{ set } (\text{state-trail } S). \text{is-ground-lit } L)$

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

shows *trail-lits-ground* 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 $\forall \tau. \text{is-unifiers } \tau \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L C_1) \} \longrightarrow \tau = \mu \odot \tau$

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

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

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

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*)

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

using $\langle \text{trail-lits-ground } S \rangle$ **by** (*simp add: propagateI(1) trail-lits-ground-def*)

ultimately show *?thesis*

by (*simp add: propagateI(2) trail-lits-ground-def propagate-lit-def*)

qed

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

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

shows *trail-lits-ground* S'

```

using assms(1)
proof (cases N β S S' rule: decide.cases)
  case (decideI L γ Γ U)
  hence is-ground-lit (L ·l γ)
    by metis

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

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

lemma conflict-preserves-trail-lits-ground:
  assumes conflict N β S S' and trail-lits-ground S
  shows trail-lits-ground S'
  using assms by (auto simp: trail-lits-ground-def elim!: conflict.cases)

lemma skip-preserves-trail-lits-ground:
  assumes skip N β S S' and trail-lits-ground S
  shows trail-lits-ground S'
  using assms by (auto simp: trail-lits-ground-def elim!: skip.cases)

lemma factorize-preserves-trail-lits-ground:
  assumes factorize N β S S' and trail-lits-ground S
  shows 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 β 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 β 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 β 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

```

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

by (*simp add: is-ingu-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 $N \beta S S'$ 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 $\Gamma \longleftrightarrow (\forall \Gamma' Ln \Gamma''. \Gamma = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' (\text{fst } Ln))$
proof (intro iffI allI impI)
 fix $\Gamma' Ln \Gamma''$
 assume trail-consistent Γ and $\Gamma = \Gamma'' @ Ln \# \Gamma'$
 thus $\neg \text{trail-defined-lit } \Gamma' (\text{fst } Ln)$
proof (induction Γ arbitrary: Γ'' rule: trail-consistent.induct)
 case Nil
 thus ?case
 by simp
 next
 case ind-hyps: (Cons $\Gamma L u$)

```

    thus ?case
      by (cases  $\Gamma''$ ) auto
  qed
next
assume  $\forall \Gamma' Ln \Gamma''. \Gamma = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' (fst Ln)$ 
then show trail-consistent  $\Gamma$ 
proof (induction  $\Gamma$ )
  case Nil
  thus ?case
    by simp
  next
  case (Cons  $Ln \Gamma$ )
  thus ?case
    by (cases  $Ln$ ) (simp add: trail-consistent.Cons)
  qed
qed

```

6.6 Trail Closures Are False In Subtrails

definition *trail-closures-false'* where

trail-closures-false' S \longleftrightarrow *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 β 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' γ C₀ C₁ Γ μ)*

have *is-unifier γ (atm-of ' set-mset (add-mset L C₁))*

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 Γ (C₀ · μ · γ)*

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 β 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 γ Γ 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 β 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 γ Γ*)
with invar show *?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *skip-preserves-trail-closures-false'*:
assumes *step: skip N β 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 σ n Γ 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 β 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 γ L' μ Γ U D*)
with invar show *?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *resolve-preserves-trail-closures-false'*:
assumes *step: resolve N β 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 Γ Γ' K D γ_D L γ_C ϱ_C ϱ_D C μ γ U*)
with invar show *?thesis*
by (*simp add: trail-closures-false'-def*)
qed

lemma *backtrack-preserves-trail-closures-false'*:
assumes *step: backtrack N β S S' and invar: trail-closures-false' S*

```

shows trail-closures-false' S'
using step
proof (cases N β S S' rule: backtrack.cases)
  case (backtrackI Γ Γ' Γ'' K L σ 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 β 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 Γ ↔
  (∀ K D γ Γ' Γ''. Γ = Γ'' @ propagate-lit K D γ # Γ' → trail-false-cls Γ' (D ·
  γ))
proof (intro iffI allI impI)
  fix K D γ Γ' Γ''
  assume trail-closures-false Γ and Γ = Γ'' @ trail-propagate Γ' K D γ
  thus trail-false-cls Γ' (D · γ)
  proof (induction Γ arbitrary: Γ'' Γ' K D γ rule: trail-closures-false.induct)
    case Nil
    thus ?case by simp
  next
    case (Cons u Γ L)
    thus ?case
    by (metis (no-types, opaque-lifting) Cons-eq-append-conv list.inject)
  qed
next
  assume ∀ K D γ Γ' Γ''. Γ = Γ'' @ trail-propagate Γ' K D γ → trail-false-cls Γ'
  (D · γ)
  thus trail-closures-false Γ
  by (induction Γ) (simp-all add: trail-closures-false.Cons)
qed

```

6.7 Trail Literals Were Propagated or Decided

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

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

Propagate:

C |∈| N |∪| U ⇒

C = add-mset L C' ⇒

is-ground-cls (C · γ) ⇒

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

Decide:

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

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.prem*s **obtain** zs **where**

tr-prop-eq: $\text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma = zs @ xs$

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.prem*s **obtain** zs **where**

tr-deci-eq: $\text{trail-decide } \Gamma (L \cdot l \gamma) = zs @ xs$


```

    by (auto simp: suffix-def)
  show ?case
  proof (cases zs)
    case Nil
    with tr-deci-eq have xs = 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 $N \ \beta$ (*state-learned* S) (*state-trail* S)

lemma *trail-propagated-or-decided-learned-finsert*:
assumes *trail-propagated-or-decided* $N \ \beta \ U \ \Gamma$
shows *trail-propagated-or-decided* $N \ \beta$ (*finsert* $C \ U$) Γ
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* $D \ L \ D' \ \gamma \ D_0 \ D_1 \ \Gamma \ \mu$)

from *Propagate.hyps* **have** *D-in*: $D \in N \ \cup \ \text{finsert } C \ U$
by *simp*

have *IH*: *trail-propagated-or-decided* $N \ \beta$ (*finsert* $C \ U$) Γ
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  $C L C' \gamma C_0 C_1 \Gamma \mu$ )
  hence tr-prop-eq- $\Gamma_1$ - $\Gamma_2$ :
     $\Gamma_1 @ \Gamma_2 = \text{trail-propagate } \Gamma (L \cdot l \mu) (C_0 \cdot \mu) \gamma$ 
  by simp
  thus ?case
    unfolding propagate-lit-def append-eq-Cons-conv
  proof (elim disjE conjE exE)
    assume  $\Gamma_1 = []$  and  $\Gamma_2\text{-def}$ :  $\Gamma_2 = (L \cdot l \mu \cdot l \gamma, \text{Some } (C_0 \cdot \mu, L \cdot l \mu, \gamma)) \# \Gamma$ 
    show ?thesis
      unfolding  $\Gamma_2\text{-def}$ 
      by (rule trail-propagated-or-decided.Propagate[unfolded propagate-lit-def];
        rule Propagate.hyps)
  next
    fix  $\Gamma_1'$ 
    assume  $\Gamma_1 = (L \cdot l \mu \cdot l \gamma, \text{Some } (C_0 \cdot \mu, L \cdot l \mu, \gamma)) \# \Gamma_1'$  and  $\Gamma_1' @ \Gamma_2 = \Gamma$ 
    thus ?thesis
      using Propagate.hyps by blast
  qed
next
  case (Decide  $L \gamma \Gamma$ )
  hence  $\Gamma_1 @ \Gamma_2 = \text{trail-decide } \Gamma (L \cdot l \gamma)$ 
  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  $\beta$  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  $\beta$  S S' and 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: propagate.cases*)
case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)

from *propagateI(1)* *assms(2)* **have** *IH: trail-propagated-or-decided* $N \beta U \Gamma$
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* $N \beta S S'$ **and** *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: decide.cases*)
case (*decideI* $L \gamma \Gamma U$)

from *decideI(1)* *assms(2)* **have** *IH: trail-propagated-or-decided* $N \beta U \Gamma$
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* $N \beta$ (*finsert (add-mset L D) U*) Γ''
proof (*rule trail-propagated-or-decided-learned-finsert*)
from *invar* **have** *trail-propagated-or-decided* $N \beta U$ (*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 $\Gamma \longleftrightarrow (\forall (L_\gamma, n) \in \text{set } \Gamma.$
case n *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 $\Gamma \iff (\forall Ln \in \text{set } \Gamma. \forall D K \ \gamma. \text{snd } Ln = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } Ln = K \cdot l \ \gamma)$
 (is *?lhs* \iff *?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* $(L_\gamma, \text{None}) \Rightarrow \text{True} \mid (L_\gamma, \text{Some } (x, L, \gamma)) \Rightarrow L_\gamma = L \cdot l \ \gamma$

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

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

trail-propagated-or-decided $N \ U \ \beta \ \Gamma \implies \text{trail-propagated-wf } \Gamma$

proof (*induction* Γ *rule: trail-propagated-or-decided.induct*)

case *Nil*

then show *?case*

by (*simp add: trail-propagated-wf-def*)

next

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

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' $N \ \beta \ S \implies \text{trail-propagated-wf } (\text{state-trail } S)$

unfolding *trail-propagated-or-decided'-def*

using *trail-propagated-wf-if-trail-propagated-or-decided* .

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

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

by *simp*

lemma *scl-preserves-trail-propagated-lit-wf*:
assumes *step*: $scl\ N\ \beta\ S\ S'$ **and**
inv: $\forall \mathcal{K} \in set\ (state-trail\ S). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
shows $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *step inv*
unfolding *scl-def*
proof (*elim disjE*)
assume *propagate* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv*
by (*smt (verit) fst-conv insert-iff list.simps(15) option.inject propagate.cases propagate-lit-def snd-conv state-trail-simp*)
next
assume *decide* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv*
by (*smt (verit) Pair-inject decide.simps decide-lit-def option.discI set-ConsD sndI state-simp*)
next
assume *conflict* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv conflict.simps by fastforce*
next
assume *skip* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv skip.simps by fastforce*
next
assume *factorize* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv factorize.simps by fastforce*
next
assume *resolve* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv*
by (*smt (verit) resolve.cases state-trail-simp*)
next
assume *backtrack* $N\ \beta\ S\ S'$
thus $\forall \mathcal{K} \in set\ (state-trail\ S'). \forall D\ K\ \gamma. snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K \cdot l\ \gamma$
using *inv*
by (*smt (verit, del-insts) Un-iff fst-conv insertCI list.simps(15) backtrack.cases*)

set-append state-trail-def)
qed

6.8 Trail Atoms Are Less Than Bound

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

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

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

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

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

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* $L \cdot a \mu \cdot a \gamma = \text{atm-of } L \cdot a \gamma$

proof –

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

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

with *propagateI* **have** $\gamma = \mu \odot \gamma$

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 $\langle \text{trail-atoms-lt } \beta \ S \rangle$ **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* $\beta \ S$
 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* $\beta \ S$
 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* $\beta \ S$
 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*

6.9 Trail Resolved Literals Have Unique Polarity

definition *trail-resolved-lits-pol* where
trail-resolved-lits-pol $S \longleftrightarrow$

$(\forall Ln \in \text{set } (\text{state-trail } S). \forall C L \gamma. \text{snd } Ln = \text{Some } (C, L, \gamma) \longrightarrow -(L \cdot l \gamma) \notin \# C \cdot \gamma)$

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 β S S'* **and** *invar: trail-resolved-lits-pol S*
shows *trail-resolved-lits-pol S'*

using *step*

proof (*cases N β S S' rule: propagate.cases*)

case (*propagateI C U L C' γ C₀ C₁ Γ μ*)

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 $\langle \text{is-imgu } \mu \{ \text{atm-of ' set-mset (add-mset L C}_1) \} \rangle$

by (*simp add: is-imgu-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 $-(L \cdot l \mu \cdot l \gamma) \notin \# C_0 \cdot \mu \cdot \gamma$

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* $N \beta S S'$ **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* $N \beta S S'$ **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* $N \beta S S'$ **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* $N \beta S S'$ **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* $N \beta S S'$ **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*

6.10 Trail And Conflict Closures Are Ground

definition *ground-closures* **where**

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

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

lemma *propagate-preserves-ground-closures*:
assumes *step*: *propagate* $N \beta S S'$ **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:  $C = \text{add-mset } L (C_0 + C_1)$ 
    using propagateI(3-) by auto

  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 is-ground-cls ( $\text{add-mset } L C_0 \cdot \mu \cdot \gamma$ )
    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 β S S' and invar: ground-closures S
  shows ground-closures S'
  using assms
  by (cases N β S S' rule: decide.cases) (simp add: ground-closures-def decide-lit-def)

lemma conflict-preserves-ground-closures:
  assumes step: conflict N β S S' and invar: ground-closures S
  shows ground-closures S'
  using step
proof (cases N β S S' rule: conflict.cases)
  case (conflictI D U γ Γ)
  thus ?thesis
    using invar
    unfolding conflictI(1,2)
    by (simp add: ground-closures-def)
qed

lemma skip-preserves-ground-closures:
  assumes step: skip N β S S' and invar: ground-closures S
  shows ground-closures S'
  using assms
  by (cases N β 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 $\langle L \cdot l \gamma = L' \cdot l \gamma \rangle [THEN \text{subst-atm-of-eqI}]$
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* $L D \cdot \mu \cdot \gamma = \text{add-mset } L D \cdot \gamma$
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* $\varrho_C \gamma_C$)
(*vars-cls* ($D \cdot \varrho_D$)) (*rename-subst-domain* $\varrho_D \gamma_D$)

shows
 $\forall L \in \# C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$
 $\forall K \in \# D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$

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: $\forall x \in \text{vars-cls } (C \cdot \rho_C). \gamma x = \text{rename-subst-domain } \rho_C \gamma_C x$

and

ball-D-ρ_D-apply-γ: $\forall x \in \text{vars-cls } (D \cdot \rho_D). \gamma x = \text{rename-subst-domain } \rho_D \gamma_D x$

using *disjoint-vars merge-γ*

unfolding *is-grounding-merge-def*

by *simp-all*

show $\forall L \in \# C. L \cdot l \rho_C \cdot l \gamma = L \cdot l \gamma_C$

proof (*rule ballI*)

fix *L* **assume** *L-in*: $L \in \# C$

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

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** $\rho_C x = \text{Var } x'$

using *ren-ρ_C*

by (*meson is-Var-def is-renaming-iff*)

ultimately have $x' \in \text{vars-lit } (L \cdot l \rho_C)$

using *vars-subst-lit-eq* **by** *fastforce*

hence $\gamma x' = \text{rename-subst-domain } \rho_C \gamma_C x'$

using *ball-C-ρ_C-apply-γ L-in multi-member-split* **by** *force*

thus $(\rho_C \odot \gamma) x = \gamma_C x$

apply (*simp add: subst-compose-def* $\langle \rho_C x = \text{Var } x' \rangle$)

by (*metis (no-types, opaque-lifting) L-in Un-iff* $\langle \rho_C x = \text{Var } x' \rangle \langle x \in \text{vars-lit}$

L

ground-conf image-eqI insert-DiffM is-renaming-iff ren-ρ_C rename-subst-domain-def subsetD the-inv-f-f vars-cls-add-mset vars-cls-subset-subst-domain-if-grounding)

qed

qed

show $\forall K \in \# D. K \cdot l \rho_D \cdot l \gamma = K \cdot l \gamma_D$

proof (*rule ballI*)

fix *K* **assume** *K-in*: $K \in \# D$

show $K \cdot l \rho_D \cdot l \gamma = K \cdot l \gamma_D$

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** $\rho_D x = \text{Var } x'$

using *ren-ρ_D*

by (*meson is-Var-def is-renaming-iff*)

ultimately have $x' \in \text{vars-lit } (K \cdot l \rho_D)$

using *vars-subst-lit-eq* **by** *fastforce*

hence $\gamma x' = \text{rename-subst-domain } \rho_D \gamma_D x'$

using *ball-D-ρ_D-apply-γ K-in multi-member-split* **by** *force*

thus $(\rho_D \odot \gamma) x = \gamma_D x$

apply (*simp add: subst-compose-def* $\langle \rho_D x = \text{Var } x' \rangle$)

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- ϱ_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- ϱ_C : is-renaming ϱ_C **and**

ren- ϱ_D : is-renaming ϱ_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- γ : is-grounding-merge γ

(*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** *inj* ϱ_C **and** $\forall x. \text{is-Var } (\varrho_D x)$ **and** *inj* ϱ_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 β S S'* and *invar: ground-closures S*
 shows *ground-closures S'*
 using *assms*
 by (*cases N β S S' rule: backtrack.cases*)
 (*simp add: ground-closures-def decide-lit-def ball-Un*)

lemma *scl-preserves-ground-closures*:
 assumes *scl N β 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*

6.11 Trail And Conflict Closures Are Ground And False

definition *ground-false-closures* where
 $\text{ground-false-closures } S \longleftrightarrow \text{ground-closures } S \wedge$
 $\text{trail-closures-false } (\text{state-trail } S) \wedge$
 $(\forall C \ \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S) (C \cdot \gamma))$

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 β S S'* and *invar: ground-false-closures S*
 shows *ground-false-closures S'*
 using *step*
proof (*cases N β 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 $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
unfolding *step-hyps*(1,2) **by** *simp*

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *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* $L \gamma \Gamma U$)

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 $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
unfolding *step-hyps*(1,2) **by** *simp*

ultimately show *?thesis*
unfolding *ground-false-closures-def* **by** *metis*
qed

lemma *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 $\forall C \gamma. \text{state-conflict } S' = \text{Some } (C, \gamma) \longrightarrow \text{trail-false-cls } (\text{state-trail } S') (C \cdot \gamma)$
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 β S S'* **and** *invar: ground-false-closures S*
shows *ground-false-closures S'*
using *step*
proof (*cases N β S S' rule: skip.cases*)
case *step-hyps: (skipI L D σ n Γ 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 $\Gamma \Gamma' K D \gamma_D L \gamma_C \varrho_C \varrho_D C \mu \gamma U$)*
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 $atm\text{-of } L \cdot a \varrho_C \cdot a \gamma = atm\text{-of } K \cdot a \varrho_D \cdot a \gamma$
using $\langle K \cdot l \gamma_D = - (L \cdot l \gamma_C) \rangle$
 $\langle \forall L \in \#add\text{-mset } L \ C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle$ [rule-format, of L , simplified]
 $\langle \forall K \in \#add\text{-mset } K \ D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$ [rule-format, of K , simplified]
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
hence $is\text{-unifiers } \gamma \{ \{ atm\text{-of } L \cdot a \varrho_C, atm\text{-of } K \cdot a \varrho_D \} \}$
by (*simp add: is-unifiers-def is-unifier-alt*)
hence $\mu \odot \gamma = \gamma$
using $\langle is\text{-imgu } \mu \{ \{ atm\text{-of } L \cdot a \varrho_C, atm\text{-of } K \cdot a \varrho_D \} \} \rangle$
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 $\langle \forall L \in \#add\text{-mset } L \ C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C \rangle \langle \forall K \in \#add\text{-mset } K \ D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D \rangle$
by (*metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst subst-lit-comp-subst*)
moreover have $trail\text{-false-cls } \Gamma (C \cdot \gamma_C)$
using *invar*
unfolding *step-hyps(1,2)*
by (*simp add: ground-false-closures-def trail-false-cls-add-mset*)
moreover have $trail\text{-false-cls } \Gamma (D \cdot \gamma_D)$
proof (*rule trail-false-cls-if-trail-false-suffix*)
show *suffix* $\Gamma' \Gamma$
unfolding $\langle \Gamma = trail\text{-propagate } \Gamma' \ K \ D \ \gamma_D \rangle$
by (*simp add: suffix-def*)
next
show $trail\text{-false-cls } \Gamma' (D \cdot \gamma_D)$
using *invar*
unfolding *step-hyps(1,2)* $\langle \Gamma = trail\text{-propagate } \Gamma' \ K \ D \ \gamma_D \rangle$
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*

6.12 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*

6.13 Backtrack Follows Conflict Resolution

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

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  $\neq$  None
  by (auto elim: skip.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 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

```

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* $S \neq \text{None}$

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*

6.14 Miscellaneous Lemmas

lemma *before-conflict*:

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

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

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

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 β U Γ rule: trail-propagated-or-decided.cases)
 case Nil
 hence $D \cdot \gamma = \{\#\}$
 using $\langle \text{trail-false-cls } \Gamma (D \cdot \gamma) \rangle$ not-trail-false-Nil(2) by blast
 hence $D = \{\#\}$
 by (simp add: local.conflictI(4))
 moreover from invars(1) have $\{\#\} \notin U$
 by (simp add: conflictI(1) learned-nonempty-def)
 ultimately have $\{\#\} \in N$
 using $\langle D \in N \mid \cup U \rangle$ by simp
 thus ?thesis by simp
 next
 case (Propagate C L C' γ_C C₀ C₁ Γ' μ)
 hence $\exists S0. \text{propagate } N \beta S0 S1$
 unfolding conflictI(1)
 using propagateI by blast
 thus ?thesis by simp
 next
 case (Decide L γ_L Γ')
 hence $\exists S0. \text{decide } N \beta S0 S1$
 unfolding conflictI(1)
 using decideI by blast
 thus ?thesis by simp
 qed
 qed

 lemma before-backtrack:
 assumes backt: backtrack N β S_n S_m and
 invar: conflict-resolution N β S_n
 shows $\exists S0 S1. \text{conflict } N \beta S0 S1 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S1 S_n$
 using backt
 proof (cases N β S_n S_m rule: backtrack.cases)
 case (backtrackI Γ Γ' Γ'' L σ 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-cls (state-trail S) C \implies trail-atoms-lt β 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-cls-def trail-false-lit-def)

7 Soundness

7.1 Sound Trail

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

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

definition *sound-trail* **where**

sound-trail $N \Gamma \longleftrightarrow$

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

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

by (*simp add: sound-trail-def*)

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

by (*meson grounding-of-clss-mono true-clss-mono*)

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

unfolding *sound-trail-def*

by (*meson entails- \mathcal{G} -mono less-eq-fset.rep-eq*)

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

by (*simp add: sound-trail-def*)

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

by (*induction* Γ) (*auto intro: sound-trail-ConsD*)

lemma *sound-trail-propagate*:

assumes

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

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

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

using *assms*

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

lemma *sound-trail-decide*:

sound-trail $N \Gamma \Longrightarrow \text{sound-trail } N (\text{trail-decide } \Gamma L)$

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 $\Rightarrow \text{bool}$ **where**

sound-state $N \beta S \longleftrightarrow$

$(\exists \Gamma U u. S = (\Gamma, U, u) \wedge \text{sound-trail } N \Gamma \wedge \text{fset } N \models_{\mathcal{G}} \text{fset } U \wedge$

$(\text{case } u \text{ of } \text{None} \Rightarrow \text{True} \mid \text{Some } (C, \gamma) \Rightarrow \text{fset } N \models_{\mathcal{G}} \{C\}))$

7.3 Initial State Is Sound

lemma *sound-initial-state*[simp]: *sound-state* $N \beta$ *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* $N \beta S S'$ **and** *sound*: *sound-state* $N \beta S$
shows *sound-state* $N \beta S'$
using *assms(1)*

proof (*cases* $N \beta S S'$ *rule: propagate.cases*)
case (*propagateI* $C U L C' \gamma C_0 C_1 \Gamma \mu$)
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-C₀-γ: *trail-false-cls* $\Gamma (C_0 \cdot \gamma)$ **and**
undef-Γ-L-γ: $\neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma)$ **and**
imgu-μ: *is-imgu* $\mu \{ \text{atm-of } ' \text{ set-mset } (\text{add-mset } L C_1) \}$
by *simp-all*

from sound have
sound-Γ: *sound-trail* $N \Gamma$ **and**
N-entails-U: *fset* $N \Vdash_{\mathcal{G}e} \text{fset } U$
unfolding *sound-state-def S-def* **by auto**

have *vars-C₀*: *vars-cls* $C_0 \subseteq \text{vars-cls } C'$
apply (*simp add: C₀-def*)
by (*metis multiset-partition order-refl sup.coboundedI1 vars-cls-plus-iff*)

have *vars-C₁*: *vars-cls* $C_1 \subseteq \text{vars-cls } C'$
apply (*simp add: C₁-def*)
by (*metis multiset-partition order-refl sup.coboundedI1 vars-cls-plus-iff*)

have *fin-atm-C₁*: *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 $\mu\text{-}\gamma\text{-simp}$: $\mu \odot \gamma = \gamma$
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  $\mu\text{-}\gamma\text{-simp}$  by (metis subst-lit-comp-subst)

have  $C_0 \cdot \mu \cdot \gamma = C_0 \cdot \gamma$ 
using  $\mu\text{-}\gamma\text{-simp}$  by (metis subst-cls-comp-subst)

have sound-trail  $N$  (trail-propagate  $\Gamma$  ( $L \cdot l \ \mu$ ) ( $C_0 \cdot \mu$ )  $\gamma$ )
proof (rule sound-trail-propagate)
  show sound-trail  $N \ \Gamma$ 
  by (rule sound- $\Gamma$ )
next
from C-in N-entails-U have fset  $N \models_{\mathcal{G}e} \{C\}$ 
by (metis (no-types, opaque-lifting) empty-subsetI union-iff grounding-of-cls-mono
  insert-subset true-cls-mono)
hence fset  $N \models_{\mathcal{G}e} \{add\text{-mset } L \ (C_0 + C_1)\}$ 
by (simp add: C-def C0-def C1-def)
hence fset  $N \models_{\mathcal{G}e} \{add\text{-mset } L \ (C_0 + C_1) \cdot \mu\}$ 
by (metis (no-types, opaque-lifting) grounding-of-cls-singleton
  subst-cls-eq-grounding-of-cls-subset-eq true-cls-mono)
hence fset  $N \models_{\mathcal{G}e} \{add\text{-mset } L \ C_0 \cdot \mu\}$ 
proof (rule entails-trans)
have  $\exists C \in \text{grounding-of-cls } \{(C_0 + C_1 + \{\#L\#\}) \cdot \mu\}$ . set-mset  $D = \text{set-mset}$ 
 $C$ 
  if D-in:  $D \in \text{grounding-of-cls } \{(C_0 + \{\#L\#\}) \cdot \mu\}$  for  $I \ D$ 
proof–
  from D-in obtain  $\sigma$  where
    D-def:  $D = add\text{-mset } L \ C_0 \cdot \mu \cdot \sigma$  and gr- $\sigma$ : is-ground-subst  $\sigma$ 
  by (auto simp: grounding-of-cls-def)
  show ?thesis
  proof (rule bexI)
    from imgu- $\mu$  have is-unifier  $\mu$  (atm-of ‘set-mset (add-mset  $L \ C_1$ ))
    by (simp add: is-imgu-def is-unifiers-def)
    hence  $\forall A \in \text{atm-of } \text{‘ set-mset } (add\text{-mset } L \ C_1)$ .  $\forall B \in \text{atm-of } \text{‘ set-mset}$ 
    (add-mset  $L \ C_1$ ).
       $A \cdot a \ \mu = B \cdot a \ \mu$ 
      using is-unifier-alt
      by (metis (mono-tags, opaque-lifting) finite-imageI finite-set-mset)
    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 C1-def
      by (metis (mono-tags, lifting) atm-of-subst-lit image-eqI literal.expand
mem-Collect-eq
      set-mset-filter subst-lit-is-neg)
    hence set-mset  $((add\text{-mset } L \ C_1) \cdot \mu) = \{L \cdot l \ \mu\}$ 
      by auto
    hence set-mset  $((C_0 + C_1 + \{\#L\#\}) \cdot \mu) = \text{set-mset } ((C_0 + \{\#L\#\}) \cdot$ 
```

μ)
by *auto*
thus *set-mset* $D = \text{set-mset } ((C_0 + C_1 + \{\#L\}) \cdot \mu \cdot \sigma)$
unfolding $D\text{-def set-mset-subst-cls-conv[of - } \sigma]$
by *simp*
next
show $(C_0 + C_1 + \{\#L\}) \cdot \mu \cdot \sigma \in \text{grounding-of-cls } \{(C_0 + C_1 + \{\#L\}) \cdot \mu\}$
using $gr\text{-}\sigma$ **by** (*auto simp: grounding-of-cls-def*)
qed
qed
then show $\{add\text{-mset } L (C_0 + C_1) \cdot \mu\} \models_{\mathcal{G}e} \{add\text{-mset } L C_0 \cdot \mu\}$
by (*auto elim: true-cls-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 grounding-of-cls-singleton subst-cls-add-mset*)
qed
thus *?thesis*
unfolding $S'\text{-def sound-state-def}$
using $N\text{-entails-}U$ **by** *simp*
qed

lemma *decide-preserves-sound-state:*
assumes $step: \text{decide } N \beta S S'$ **and** $sound: \text{sound-state } N \beta S$
shows $\text{sound-state } N \beta S'$
using $assms(1)$
proof ($cases N \beta S S'$ *rule: decide.cases*)
case ($decideI L \gamma \Gamma U$)
from $decideI(1) sound$ **have**
 $sound\text{-}\Gamma: \text{sound-trail } N \Gamma$ **and**
 $N\text{-entails-}U: fset N \models_{\mathcal{G}e} fset U$
unfolding $sound\text{-state-def}$ **by** *auto*

moreover have $sound\text{-trail } N (\text{trail-decide } \Gamma (L \cdot l \gamma))$
by ($simp \text{ add: local.decideI}(4) \text{ local.decideI}(5) sound\text{-}\Gamma sound\text{-trail-decide}$)

ultimately show *?thesis*
unfolding $decideI sound\text{-state-def}$ **by** *simp*
qed

lemma *conflict-preserves-sound-state:*
assumes $step: \text{conflict } N \beta S S'$ **and** $sound: \text{sound-state } N \beta S$
shows $\text{sound-state } N \beta S'$
using $assms(1)$
proof ($cases N \beta S S'$ *rule: conflict.cases*)
case ($conflictI D U \gamma \Gamma$)
hence $D\text{-in: } D \in | N \cup U$ **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
    unfolding grounding-of-clss-singleton
    by (metis (mono-tags, opaque-lifting) N-entails-U UN-I funion-iff ground-
ing-of-clss-def
      true-clss-def)
  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 β S S' and sound: sound-state N β S
  shows sound-state N β S'
  using step
proof (cases N β S S' rule: skip.cases)
  case (skipI L D σ Cl Γ 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 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: factorize.cases)
  case (factorizeI L γ L' μ Γ U D)

from factorizeI(1) sound have
  sound-Γ: sound-trail N Γ 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 μ {{atm-of L, atm-of L'}}
  by simp

```

from *factorizeI* **have** $L\text{-eq-}L'\text{-}\gamma: L \cdot l \ \gamma = L' \cdot l \ \gamma$ **by** *simp*

from $L\text{-eq-}L'\text{-}\gamma$ **have** $\text{unif-}\gamma: \text{is-unifier } \gamma \{ \text{atm-of } L, \text{atm-of } L' \}$
by (*auto simp: is-unifier-alt intro: subst-atm-of-eqI*)
hence $\text{unifs-}\gamma: \text{is-unifiers } \gamma \{ \{ \text{atm-of } L, \text{atm-of } L' \} \}$
by (*simp add: is-unifiers-def*)

from *imgu- μ* **have** $\text{is-unifier } \mu \{ \text{atm-of } L, \text{atm-of } L' \}$
by (*auto simp add: is-unifiers-def dest: is-imgu-is-mgu[THEN is-mgu-is-unifiers]*)
hence $L\text{-eq-}L'\text{-}\mu: L \cdot l \ \mu = L' \cdot l \ \mu$
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 $\{ D + \{ \#L, L'\# \} \} \models_{\mathcal{G}e} \{ (D + \{ \#L, L'\# \}) \cdot \mu \}$
using *subst-cls-eq-grounding-of-cls-subset-eq true-cls-mono*
by (*metis (mono-tags, lifting) grounding-of-cls-singleton*)

moreover have $\{ (D + \{ \#L, L'\# \}) \cdot \mu \} \models_{\mathcal{G}e} \{ (D + \{ \#L\# \}) \cdot \mu \}$
apply (*intro allI impI*)
by (*erule true-cls-if-set-mset-eq[rotated]*) (*auto simp add: L-eq-L'- μ grounding-of-cls-def*)

ultimately show $\{ D + \{ \#L, L'\# \} \} \models_{\mathcal{G}e} \{ (D + \{ \#L\# \}) \cdot \mu \}$
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
 $\Gamma\text{-def: } \Gamma = \text{trail-propagate } \Gamma' \ K \ D \ \gamma_D$ **and**
 $\text{imgu-}\mu: \text{is-imgu } \mu \{ \{ \text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D \} \}$
by *simp-all*

from *sound* **have**

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

N-entails-U: *fset* $N \Vdash_{\mathcal{G}e}$ *fset* U **and**

N-entails-conf: *fset* $N \Vdash_{\mathcal{G}e}$ $\{add\text{-}mset\ L\ C\}$

unfolding *resolveI*(1,2) *sound-state-def* **by** *simp-all*

from *sound-Γ* **have**

N-entails-prop: *fset* $N \Vdash_{\mathcal{G}e}$ $\{add\text{-}mset\ K\ D\}$

unfolding *sound-trail-def* $\Gamma\text{-def}$

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

moreover **have** *fset* $N \Vdash_{\mathcal{G}e}$ $\{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$

proof –

have *: *fset* $N \Vdash_{\mathcal{G}e}$ $\{add\text{-}mset\ L\ C \cdot \varrho_C \cdot \mu\}$

using *N-entails-conf*

by (*metis* (*no-types*, *opaque-lifting*) *grounding-of-clss-singleton* *grounding-of-subst-clss-subset* *true-clss-mono*)

have **: *fset* $N \Vdash_{\mathcal{G}e}$ $\{add\text{-}mset\ K\ D \cdot \varrho_D \cdot \mu\}$

using *N-entails-prop*

by (*metis* (*no-types*, *opaque-lifting*) *grounding-of-clss-singleton* *grounding-of-subst-clss-subset* *true-clss-mono*)

show *fset* $N \Vdash_{\mathcal{G}e}$ $\{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$

unfolding *true-clss-def*

proof (*intro allI impI ballI*)

fix $I\ E$

assume *I-entails*: $\forall E \in \text{grounding-of-clss } (fset\ N). I \Vdash E$ **and**

E-in: $E \in \text{grounding-of-clss } \{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$

then obtain γ_E **where**

E-def: $E = (C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma_E$ **and** *gr-γ_E*: *is-ground-subst* γ_E

unfolding *grounding-of-clss-def* *grounding-of-clss-def* **by** *auto*

have $I \Vdash_l L \cdot l \varrho_C \cdot l \mu \cdot l \gamma_E \vee (\exists x \in \# C \cdot \varrho_C \cdot \mu \cdot \gamma_E. I \Vdash_l x)$

proof –

have *add-mset* $L\ C \cdot \varrho_C \cdot \mu \cdot \gamma_E \in \text{grounding-of-clss } \{add\text{-}mset\ L\ C \cdot \varrho_C \cdot \mu\}$

using *gr-γ_E* **unfolding** *grounding-of-clss-def* *grounding-of-clss-def* **by** *auto*

hence $\exists K \in \# add\text{-}mset\ L\ C \cdot \varrho_C \cdot \mu \cdot \gamma_E. I \Vdash_l K$

using *[*rule-format*, *unfolded true-clss-def*, *OF I-entails*]

by (*metis true-clss-def*)

thus *?thesis*

by *simp*

qed

moreover **have** $I \Vdash_l K \cdot l \varrho_D \cdot l \mu \cdot l \gamma_E \vee (\exists x \in \# D \cdot \varrho_D \cdot \mu \cdot \gamma_E. I \Vdash_l x)$

proof –

have *add-mset* $K\ D \cdot \varrho_D \cdot \mu \cdot \gamma_E \in \text{grounding-of-clss } \{add\text{-}mset\ K\ D \cdot \varrho_D \cdot \mu\}$

using *gr- γ_E* **unfolding** *grounding-of-clss-def grounding-of-clc-def* **by** *auto*
hence $\exists K \in \# \text{ add-mset } K \ D \cdot \varrho_D \cdot \mu \cdot \gamma_E. I \models K$
using ***[rule-format, unfolded true-clss-def, OF I-entails]*
by *(metis true-clc-def)*
thus *?thesis*
by *simp*
qed

ultimately show $I \models E$
proof *(elim disjE)*
assume $I \models L \cdot l \ \varrho_C \cdot l \ \mu \cdot l \ \gamma_E$ **and** $I \models K \cdot l \ \varrho_D \cdot l \ \mu \cdot l \ \gamma_E$
moreover have $\text{atm-of } L \cdot a \ \varrho_C \cdot a \ \mu = \text{atm-of } K \cdot a \ \varrho_D \cdot a \ \mu$
using *imgu- μ [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 $N :: ('f, 'v)\ Term.term\ clause\ fset$
shows $scl\ N\ \beta\ S\ S' \implies sound-state\ N\ \beta\ S \implies sound-state\ N\ \beta\ S'$
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* $N \beta S S' \iff reasonable-scl N \beta S S' \wedge$
 $((\exists S''. conflict N \beta S S'') \longrightarrow conflict N \beta S S')$
(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 $(\exists S''. conflict N \beta S S'') \longrightarrow conflict N \beta S S'$
using *conf* **by** *simp*
qed
next
assume $\neg (\exists S''. conflict N \beta S S'')$ **and** *reasonable-scl* $N \beta S S'$
thus *?rhs*
by *simp*
qed
next
assume *?rhs*
thus *?lhs*
proof (*cases* $\exists S''. conflict N \beta S S''$)
case *True*
with $\langle ?rhs \rangle$ **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 $\langle ?rhs \rangle$ **by** *simp*
qed
qed

definition *ex-conflict* **where**

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

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

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

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

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

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

shortest-backtrack-strategy *regular-scl* $N \beta S S' \Longrightarrow \text{regular-scl } N \beta S S'$

by (*simp add: shortest-backtrack-strategy-def*)

lemma *strategy-restrictions*:

shows

shortest-backtrack-strategy *regular-scl* $N \beta S S' \Longrightarrow \text{regular-scl } N \beta S S'$ **and**

regular-scl $N \beta S S' \Longrightarrow \text{reasonable-scl } N \beta S S'$ **and**

reasonable-scl $N \beta S S' \Longrightarrow \text{scl } N \beta S S'$

using *regular-scl-if-shortest-backtrack-strategy* *reasonable-if-regular* *scl-if-reasonable*

by *metis+*

primrec *shortest-backtrack* **where**

shortest-backtrack $C [] = [] \mid$

shortest-backtrack $C (Ln \# \Gamma) =$

(*if ex-conflict* $C (Ln \# \Gamma)$ *then*

shortest-backtrack $C \Gamma$

else

$Ln \# \Gamma$)

lemma *suffix-shortest-backtrack*: *suffix* (*shortest-backtrack* $C \Gamma$) Γ

by (*induction* Γ) (*simp-all add: suffix-Cons*)

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

$C = \{\#\}$

by (*induction* Γ) (*auto simp add: ex-conflict-def*)

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

$C \neq \{\#\} \Longrightarrow \text{is-shortest-backtrack } C \Gamma$ (*shortest-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  $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$  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  $\beta$   $S_0$   $S_1$ 
  with assms(1) have propagate N  $\beta'$   $S_0$   $S_1$ 
    using propagateI propagate.cases
    by (smt (verit) is-ground-cls-imp-is-ground-lit is-ground-lit-def)
  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)
  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  $\bigwedge A.$  is-ground-atm  $A \implies A \preceq_B \beta \implies A \preceq_B \beta'$  and reasonable-scl N
   $\beta$   $S_0$   $S_1$ 
  shows reasonable-scl N  $\beta'$   $S_0$   $S_1$ 
  unfolding reasonable-scl-def
proof (intro conjI impI)
  show scl N  $\beta'$   $S_0$   $S_1$ 
    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  $\exists S_1'.$  conflict N  $\beta$   $S_0$   $S_1'$  and reasonable-scl N  $\beta$   $S_0$   $S_1$ 
  have  $\exists S_1'.$  conflict N  $\beta'$   $S_0$   $S_1'$ 
    using  $\langle \exists S_1'.$  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
 $\bigwedge A. \text{is-ground-atm } A \implies A \preceq_B \beta \implies A \preceq_B \beta'$ **and**
shortest-backtrack-strategy regular-scl N β S_0 S_1
shows *shortest-backtrack-strategy regular-scl N β' S_0 S_1*
unfolding *shortest-backtrack-strategy-def*
proof (*intro conjI impI*)
from *assms(2)* **have** *regular-scl N β S_0 S_1*
 by (*simp add: shortest-backtrack-strategy-def*)
with *assms(1)* **show** *regular-scl N β' S_0 S_1*
 using *regular-scl-monotone-wrt-bound*
 by *metis*
next
assume *backtrack N β' S_0 S_1*
with *assms(2)* **have** *backtrack N β 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 S_0))) (state-trail S_0) (state-trail S_1))
 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 S_0 $S_1 \implies scl N \beta' S_0 S_1$ and
reasonable-scl N β S_0 $S_1 \implies reasonable-scl N \beta' S_0 S_1$ and
regular-scl N β S_0 $S_1 \implies regular-scl N \beta' S_0 S_1$ and
shortest-backtrack-strategy regular-scl N β S_0 $S_1 \implies$
shortest-backtrack-strategy regular-scl N β' S_0 S_1
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
 $\beta \prec_B \beta'$
shows

$scl\ N\ \beta\ S_0\ S_1 \implies scl\ N\ \beta'\ S_0\ S_1$ **and**
 $reasonable\ scl\ N\ \beta\ S_0\ S_1 \implies reasonable\ scl\ N\ \beta'\ S_0\ S_1$ **and**
 $regular\ scl\ N\ \beta\ S_0\ S_1 \implies regular\ scl\ N\ \beta'\ S_0\ S_1$ **and**
 $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta\ S_0\ S_1 \implies$
 $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta'\ S_0\ S_1$

proof –

from $\langle transp\ on\ \{A.\ is\ ground\ atm\ A\}\ \langle \prec_B \rangle$ **have** $transp\ on\ \{A.\ is\ ground\ atm\ A\}\ (\preceq_B)$

by (*metis transp-on-reflcp*)

moreover from $\langle \beta\ \prec_B\ \beta' \rangle$ **have** $\beta\ \preceq_B\ \beta'$

by *blast*

ultimately have $\bigwedge A.\ is\ ground\ atm\ A \implies A\ \preceq_B\ \beta \implies A\ \preceq_B\ \beta'$

using $\langle is\ ground\ atm\ \beta \rangle\ \langle is\ ground\ atm\ \beta' \rangle$

by (*meson CollectI transp-onD*)

thus

$scl\ N\ \beta\ S_0\ S_1 \implies scl\ N\ \beta'\ S_0\ S_1$ **and**
 $reasonable\ scl\ N\ \beta\ S_0\ S_1 \implies reasonable\ scl\ N\ \beta'\ S_0\ S_1$ **and**
 $regular\ scl\ N\ \beta\ S_0\ S_1 \implies regular\ scl\ N\ \beta'\ S_0\ S_1$ **and**
 $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta\ S_0\ S_1 \implies$
 $shortest\ backtrack\ strategy\ regular\ scl\ N\ \beta'\ S_0\ S_1$

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*: $sound\ state\ N\ \beta\ S$ **and** *conflict-S*: $state\ conflict\ S = Some\ (\{\#\}, \gamma)$

shows $\neg\ satisfiable\ (grounding\ of\ class\ (fset\ N))$

proof –

from *sound-S* *conflict-S* **have** $fset\ N\ \Vdash_{Ge}\ \{\#\}$

unfolding *sound-state-def* *state-conflict-def* **by** *auto*

thus *?thesis* **by** *simp*

qed

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

assumes

trail-lt-beta: $trail\ atoms\ lt\ \beta\ S_2$ **and**

no-conf: $\nexists S_1.\ conflict\ N\ \beta\ S_0\ S_1$ **and** *decide*: $decide\ N\ \beta\ S_0\ S_2$ **and** *conflict*: $conflict\ N\ \beta\ S_2\ S_3$

shows $\exists S_4.\ propagate\ N\ \beta\ S_0\ S_4$

proof –

from *deci* **obtain** $L \ \gamma \ \Gamma \ U$ **where**

S_0 -def: $S_0 = (\Gamma, U, \text{None})$ **and**

S_2 -def: $S_2 = (\text{trail-decide } \Gamma (L \cdot l \ \gamma), U, \text{None})$ **and**

$L \in \bigcup (\text{set-mset } \text{'fset } N)$ **and**

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

\neg *trail-defined-lit* $\Gamma (L \cdot l \ \gamma)$ **and**

atm-of $L \cdot a \ \gamma \preceq_B \beta$

by (*elim decide.cases*) *blast*

from *conf* S_2 -def **obtain** $D \ \gamma_D$ **where**

S_3 -def: $S_3 = (\text{trail-decide } \Gamma (L \cdot l \ \gamma), U, \text{Some } (D, \gamma_D))$ **and**

D -in: $D \mid \in \mid N \mid \cup \mid U$ **and**

ground-D- σ : *is-ground-cls* $(D \cdot \gamma_D)$ **and**

tr- Γ -L-false-D: *trail-false-cls* $(\text{trail-decide } \Gamma (L \cdot l \ \gamma)) (D \cdot \gamma_D)$

by (*auto elim: conflict.cases*)

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

using *ground-D- σ vars-cls-subset-subst-domain-if-grounding* **by** *blast*

moreover **have** \neg *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** $(L \cdot l \ \gamma) = L' \cdot l$

γ_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 $trail\text{-}lt\text{-}\beta'$: $\forall A \in atm\text{-}of \text{ 'fst ' set (trail\text{-}decide } \Gamma (L \cdot l \gamma)). A \preceq_B \beta$
using $trail\text{-}lt\text{-}\beta$ **by** (*simp add: trail\text{-}atoms\text{-}lt\text{-}def S₂\text{-}def*)

assume $K\text{-}in$: $K \in\# D' \cdot \gamma_D$
hence $atm\text{-}of K \in atm\text{-}of \text{ 'fst ' set (trail\text{-}decide } \Gamma (L \cdot l \gamma))$
using $tr\text{-}\Gamma\text{-}L\text{-}false\text{-}D$ [*unfolded D\text{-}def*]
by (*metis D\text{-}def \langle K \in\# D \cdot \gamma_D \rangle atm\text{-}of\text{-}in\text{-}atm\text{-}of\text{-}set\text{-}iff\text{-}in\text{-}set\text{-}or\text{-}u\text{-}minus\text{-}in\text{-}set trail\text{-}false\text{-}cls\text{-}def trail\text{-}false\text{-}lit\text{-}def*)
moreover from $trail\text{-}lt\text{-}\beta$ **have** $\forall A \in atm\text{-}of \text{ 'fst ' set (trail\text{-}decide } \Gamma (L \cdot l \gamma)). A \preceq_B \beta$
by (*simp add: trail\text{-}atoms\text{-}lt\text{-}def S₂\text{-}def*)
ultimately show *?thesis*
by *blast*
qed
qed

have $tr\text{-}false\text{-}C_1$: $trail\text{-}false\text{-}cls \Gamma (C_0 \cdot \gamma_D)$
proof (*rule subtrail\text{-}falseI*)
show $trail\text{-}false\text{-}cls ((L \cdot l \gamma, None) \# \Gamma) (C_0 \cdot \gamma_D)$
unfolding $C_0\text{-}def$ $trail\text{-}false\text{-}cls\text{-}def$
apply (*rule ballI*)
apply (*rule tr\text{-}\Gamma\text{-}L\text{-}false\text{-}D*[*unfolded D\text{-}def trail\text{-}false\text{-}cls\text{-}def decide\text{-}lit\text{-}def, rule\text{-}format*])
by *auto*
next
show $-(L \cdot l \gamma) \notin\# C_0 \cdot \gamma_D$
unfolding $C_0\text{-}def$
using $\langle -(L \cdot l \gamma) = L' \cdot l \gamma_D \rangle$ **by** *fastforce*
qed

have $not\text{-}def\text{-}L'\text{-}\rho\text{-}\sigma_\rho$: $\neg trail\text{-}defined\text{-}lit \Gamma (L' \cdot l \gamma_D)$
using $\langle \neg trail\text{-}defined\text{-}lit \Gamma (L \cdot l \gamma) \rangle$
by (*metis \langle -(L \cdot l \gamma) = L' \cdot l \gamma_D \rangle trail\text{-}defined\text{-}lit\text{-}iff\text{-}defined\text{-}u\text{-}minus*)

obtain xs **where** $mset\ xs = add\text{-}mset\ L'\ C_1$
using $ex\text{-}mset$ **by** *auto*
hence $set\text{-}xs\text{-}conv$:
 $set\ xs = set\text{-}mset (add\text{-}mset\ L'\ C_1)$
by (*metis set\text{-}mset\text{-}mset*)

have $unifiers (set (pairs (map\ atm\text{-}of\ xs))) \neq \{\}$
proof (*rule not\text{-}empty\text{-}if\text{-}mem*)
have $set (pairs (map\ atm\text{-}of\ xs)) =$
 $atm\text{-}of \text{ ' set\text{-}mset (add\text{-}mset\ L'\ C_1) \times atm\text{-}of \text{ ' set\text{-}mset (add\text{-}mset\ L'\ C_1)$
unfolding $set\text{-}pairs\ list.set\text{-}map\ set\text{-}xs\text{-}conv$ **by** *simp*
also have $\dots =$
 $atm\text{-}of \text{ ' insert\ L' (set\text{-}mset\ C_1) \times atm\text{-}of \text{ ' insert\ L' (set\text{-}mset\ C_1)$
unfolding $set\text{-}mset\text{-}add\text{-}mset\text{-}insert$ **by** *simp*

also have ... =
atm-of 'insert $L' \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\} \times$
atm-of 'insert $L' \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\}$
unfolding C_1 -def set-mset-filter by simp
finally have set-pairs-xs-simp: set (pairs (map atm-of xs)) =
atm-of 'insert $L' \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\} \times$
atm-of 'insert $L' \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\}$
by assumption

show $\gamma_D \in \text{unifiers}(\text{set}(\text{pairs}(\text{map atm-of xs})))$

unfolding unifiers-def mem-Collect-eq

proof (rule ballI)

fix p **assume** p -in: $p \in \text{set}(\text{pairs}(\text{map atm-of xs}))$

then obtain $K1 K2$ **where** p -def: $p = (\text{atm-of } K1, \text{atm-of } K2)$ **and**

$K1 = L' \vee K1 \in \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\}$ **and**

$K2 = L' \vee K2 \in \{K. K \in \# D' \wedge K \cdot l \gamma_D = L' \cdot l \gamma_D\}$

by (auto simp: set-pairs-xs-simp)

hence $K1 \cdot l \gamma_D = L' \cdot l \gamma_D \wedge K2 \cdot l \gamma_D = L' \cdot l \gamma_D$

by auto

thus $\text{fst } p \cdot a \gamma_D = \text{snd } p \cdot a \gamma_D$

unfolding p -def prod.sel

by (metis atm-of-subst-lit)

qed

qed

then obtain ys **where**

$\text{unify-pairs: unify}(\text{pairs}(\text{map atm-of xs})) \sqsupseteq \text{Some } ys$

using ex-unify-if-unifiers-not-empty[OF - refl] **by blast**

define μ **where**

$\mu = \text{subst-of } ys$

have $\text{imgu-}\mu$: is-imgu $\mu \{ \text{atm-of 'set-mset}(\text{add-mset } L' C_1) \}$

proof (intro is-imgu-if-mgu-sets[unfolded mgu-sets-def] exI conjI)

show set (map set [(map atm-of xs)]) = {atm-of 'set-mset (add-mset $L' C_1$)}

by (simp add: set-xs-conv)

next

show map-option subst-of (unify (concat (map pairs [map atm-of xs]))) \sqsupseteq =

Some μ

by (simp add: unify-pairs μ -def)

qed

show ?thesis

using propagateI[OF D -in D -def, of γ_D , unfolded subst-cls-comp-subst subst-lit-comp-subst,

OF ground- D - σ ball-atms-lt- β C_0 -def C_1 -def tr-false- C_1 not-def- L' - ρ - σ_ρ

$\text{imgu-}\mu$]

unfolding S_0 -def **by blast**

qed

theorem correct-termination:

fixes *gnd-N* **and** *gnd-N-lt-β*
assumes
sound-S: *sound-state* $N \beta S$ **and**
invars: *trail-atoms-lt* βS *trail-propagated-wf* (*state-trail* S) *trail-lits-consistent*
 S
ground-false-closures S **and**
no-new-conflict: $\nexists S'. \text{conflict } N \beta S S'$ **and**
no-new-propagate: $\nexists S'. \text{propagate } N \beta S S'$ **and**
no-new-decide: $\nexists S'. \text{decide } N \beta S S' \wedge (\nexists S''. \text{conflict } N \beta S' S'')$ **and**
no-new-skip: $\nexists S'. \text{skip } N \beta S S'$ **and**
no-new-resolve: $\nexists S'. \text{resolve } N \beta S S'$ **and**
no-new-backtrack: $\nexists S'. \text{backtrack } N \beta S S' \wedge$
is-shortest-backtrack (*fst* (*the* (*state-conflict* S))) (*state-trail* S) (*state-trail* S')
defines
gnd-N \equiv *grounding-of-clss* (*fset* N) **and**
gnd-N-lt-β \equiv $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
shows \neg *satisfiable* *gnd-N* \wedge ($\exists \gamma. \text{state-conflict } S = \text{Some} (\{\#\}, \gamma)$) \vee
satisfiable *gnd-N-lt-β* \wedge *trail-true-clss* (*state-trail* S) *gnd-N-lt-β*
proof –
obtain $\Gamma U u$ **where** *S-def*: $S = (\Gamma, U, u)$
using *prod-cases3* **by** *blast*

from *sound-S* **have** *sound-Γ*: *sound-trail* $N \Gamma$
by (*simp-all add*: *S-def sound-state-def*)

from \langle *ground-false-closures* S \rangle **have** *ground-closures* S
by (*simp add*: *ground-false-closures-def*)

have *trail-consistent*: *trail-consistent* (*state-trail* S)
using \langle *trail-lits-consistent* S \rangle **by** (*simp add*: *trail-lits-consistent-def*)

show *?thesis*
proof (*cases* u)
case *u-def*: *None*
hence *state-conflict* $S = \text{None}$
by (*simp add*: *S-def*)

have *tr-true*: *trail-true-clss* Γ *gnd-N-lt-β*
unfolding *trail-true-clss-def* *gnd-N-lt-β-def* *gnd-N-def*
proof (*rule ballI*, *unfold mem-Collect-eq*, *erule conjE*)
fix C **assume** *C-in*: $C \in \text{grounding-of-clss} (\text{fset } N)$ **and** *C-lt-β*: $\forall L \in \# C.$
atm-of $L \preceq_B \beta$

from *C-in* **have** *is-ground-cls* C
by (*rule grounding-ground*)

from *C-in* **obtain** $C' \gamma$ **where** *C'-in*: $C' \mid \in \mid N$ **and** *C-def*: $C = C' \cdot \gamma$
unfolding *grounding-of-clss-def* *grounding-of-cls-def*
by (*smt* (*verit*, *ccfv-threshold*) *UN-iff mem-Collect-eq*)

from *no-new-decide* **have** Γ -defined- C : *trail-defined-cls* Γ C
proof (*rule contrapos- η*)
 assume \neg *trail-defined-cls* Γ C
 then obtain L **where** L -in: $L \in \# C$ **and** \neg *trail-defined-lit* Γ L
 using *trail-defined-cls-def* **by** *blast*
 then obtain L' **where** L' -in: $L' \in \# C'$ **and** $L = L' \cdot l \ \gamma$
 using C -def *Melem-subst-cls* **by** *blast*

have *deci*: *decide* $N \ \beta$ (Γ , U , *None*) (*trail-decide* Γ ($L' \cdot l \ \gamma$), U , *None*)
proof (*rule decideI*)
 show $L' \in \bigcup$ (*set-mset* ‘*fset* N)
 using C' -in L' -in
 by (*meson UN-I*)
next
 show *is-ground-lit* ($L' \cdot l \ \gamma$)
 using L -in $\langle L = L' \cdot l \ \gamma \rangle$ *is-ground-cls* C *is-ground-cls-def* **by** *blast*
next
 show \neg *trail-defined-lit* Γ ($L' \cdot l \ \gamma$)
 using $\langle L = L' \cdot l \ \gamma \rangle$ $\langle \neg$ *trail-defined-lit* Γ $L \rangle$ **by** *blast*
next
 show *atm-of* $L' \cdot a \ \gamma \preceq_B \ \beta$
 using $\langle L = L' \cdot l \ \gamma \rangle$ C -lt- β L -in **by** *fastforce*
qed

moreover have $\nexists S''$. *conflict* $N \ \beta$ (*trail-decide* Γ ($L' \cdot l \ \gamma$), U , *None*) S''
proof (*rule notI, elim exE*)
 fix S''
 assume *conf*: *conflict* $N \ \beta$ (*trail-decide* Γ ($L' \cdot l \ \gamma$), U , *None*) S''
 moreover have *trail-atoms-lt* β (*trail-decide* Γ ($L' \cdot l \ \gamma$), 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* Γ C
 using Γ -defined- C [*THEN trail-true-or-false-cls-if-defined*]
proof (*elim disjE*)
 show *trail-true-cls* Γ $C \implies$ *trail-true-cls* Γ C
 by *assumption*
next

```

assume trail-false-cls  $\Gamma$   $C$ 

define  $\varrho :: 'v \Rightarrow ('f, 'v)$  term where
   $\varrho = \text{renaming-wrt } (fset (N \mid \cup \mid U \mid \cup \mid \text{cls-of-trail } \Gamma))$ 

define  $\gamma_\varrho$  where
   $\gamma_\varrho = \text{rename-subst-domain } \varrho (\text{restrict-subst-domain } (\text{vars-cls } C') \gamma)$ 

have conflict  $N$   $\beta$   $(\Gamma, U, \text{None}) (\Gamma, U, \text{Some } (C', \text{restrict-subst-domain } (\text{vars-cls } C') \gamma))$ 
proof (rule conflictI)
  show  $C' \mid \in \mid N \mid \cup \mid U$ 
    using  $C'\text{-in}$  by simp
next
  show is-ground-cls  $(C' \cdot \text{restrict-subst-domain } (\text{vars-cls } C') \gamma)$ 
    using  $\langle \text{is-ground-cls } C \rangle [\text{unfolded } C\text{-def}]$ 
    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  $\langle \text{trail-false-cls } \Gamma C \rangle [\text{unfolded } C\text{-def}]$ 
    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  $\Gamma$   $C$  ..
qed
qed

moreover have satisfiable gnd-N-lt- $\beta$ 
  unfolding true-cls-def gnd-N-lt- $\beta$ -def
proof (intro exI ballI, unfold mem-Collect-eq, elim conjE)
  fix  $C$ 
  have trail-consistent  $\Gamma$ 
    using  $S\text{-def trail-consistent}$  by auto
  show  $C \in \text{gnd-N} \implies \forall L \in \# C. \text{atm-of } L \preceq_B \beta \implies \text{trail-interp } \Gamma \models C$ 
    using  $\text{tr-true} [\text{unfolded } \text{gnd-N-lt-}\beta\text{-def}]$ 
    using  $\text{trail-interp-cls-if-trail-true} [OF \langle \text{trail-consistent } \Gamma \rangle]$ 
    by (simp add: trail-true-cls-def)
qed

ultimately show ?thesis
  by (simp add: S-def)
next
case (Some Cl)
then obtain  $C$   $\gamma_C$  where  $u\text{-def}: u = \text{Some } (C, \gamma_C)$  by force

from  $\langle \text{ground-false-closures } S \rangle$  have  $\Gamma\text{-false-}C\text{-}\gamma_C: \text{trail-false-cls } \Gamma (C \cdot \gamma_C)$ 
  by (simp add: S-def u-def ground-false-closures-def)

```

```

show ?thesis
proof (cases C = {#})
  case True
    hence  $\neg$  satisfiable gnd-N  $\wedge$  ( $\exists \gamma$ . state-conflict S = 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  $\Gamma$ )
      case Nil
        with  $\Gamma$ -false-C- $\gamma_C$  show False
        using C-not-empty by simp
      next
        case (Cons Ln  $\Gamma'$ )
        then obtain  $K_\Gamma$  n where  $\Gamma$ -def:  $\Gamma = (K_\Gamma, n) \# \Gamma'$ 
          by fastforce

        show False
        proof (cases  $- K_\Gamma \in\# C \cdot \gamma_C$ )
          case True — Literal cannot be skipped
            then obtain C' L where C-def:  $C = \text{add-mset } L \ C'$  and  $K\text{-}\gamma$ :  $L \cdot l \ \gamma_C$ 
            =  $- K_\Gamma$ 
              by (metis Melem-subst-cls multi-member-split)
            hence L-eq-uminus-K- $\gamma$ :  $K_\Gamma = - (L \cdot l \ \gamma_C)$ 
              by simp

          show False
          proof (cases n)
            case None — Conflict clause can be backtracked
              hence  $\Gamma$ -def:  $\Gamma = \text{trail-decide } \Gamma' (- (L \cdot l \ \gamma_C))$ 
                by (simp add:  $\Gamma$ -def L-eq-uminus-K- $\gamma$  decide-lit-def)

              from suffix-shortest-backtrack[of add-mset L C'  $\Gamma'$ ] obtain  $\Gamma''$  where
                 $\Gamma'$ -def:  $\Gamma' = \Gamma'' @ \text{shortest-backtrack } (\text{add-mset } L \ C') \ \Gamma'$ 
                using suffixE by blast

              define S' :: ('f, 'v) state where
                S' = (shortest-backtrack (add-mset L C')  $\Gamma'$ , finsert (add-mset L C')
U, None)

              have backtrack N  $\beta$  S S'
                unfolding S-def[unfolded u-def C-def] S'-def
                proof (rule backtrackI[OF - refl])
                  show  $\Gamma = \text{trail-decide } (\Gamma'' @ \text{shortest-backtrack } (\text{add-mset } L \ C') \ \Gamma') (- (L \cdot l \ \gamma_C))$ 
                    using  $\Gamma$ -def  $\Gamma'$ -def by simp
                next

```

$\cdot \gamma$)
show $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } L \ C' \cdot \gamma) \wedge$
 $\text{trail-false-cls } (\text{shortest-backtrack } (\text{add-mset } L \ C') \ \Gamma') \ (\text{add-mset } L \ C'$
 $\cdot \gamma)$
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 $S\text{-def}$ [*unfolded* $u\text{-def}$ $C\text{-def}$] $S'\text{-def}$
apply *simp*
using *is-shortest-backtrack-shortest-backtrack*[of *add-mset* $L \ C'$, *simplified*]
 γ_C) \rangle **by** (*metis* $C\text{-def}$ $\Gamma\text{-def}$ $\Gamma\text{-false-}C\text{-}\gamma_C$ $\langle S = (\Gamma, U, \text{Some } (\text{add-mset } L \ C',$
 $\langle \text{ground-closures } S \rangle \text{ex-conflict-def ground-closures-def is-shortest-backtrack-def}$
 $\text{state-proj-simp}(3) \text{suffix-Cons}$)
ultimately show *False*
using *no-new-backtrack* **by** *metis*
next
case *Some* — Literal can be resolved
then obtain $D \ K \ \gamma_D$ **where** $n\text{-def}: n = \text{Some } (D, K, \gamma_D)$
by (*metis prod-cases3*)
with $\langle \text{trail-propagated-wf } (\text{state-trail } S) \rangle$ **have** $L\text{-def}: K_\Gamma = K \cdot l \ \gamma_D$
by (*simp add: \Gamma-def S-def trail-propagated-wf-def*)
hence $1: \Gamma = \text{trail-propagate } \Gamma' \ K \ D \ \gamma_D$
using $\Gamma\text{-def}$ $n\text{-def}$
by (*simp add: propagate-lit-def*)
from $\langle \text{ground-closures } S \rangle$ **have**
 $\text{ground-conf}: \text{is-ground-cls } (\text{add-mset } L \ C' \cdot \gamma_C)$ **and**
 $\text{ground-prop}: \text{is-ground-cls } (\text{add-mset } K \ D \cdot \gamma_D)$
unfolding $S\text{-def}$ $\text{ground-closures-def}$
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 $\text{ren-}\varrho: \text{is-renaming } \varrho$
unfolding $\varrho\text{-def}$
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 $\text{disjoint-vars}: \bigwedge C. \text{vars-cls } (C \cdot \varrho) \cap \text{vars-cls } (\text{add-mset } K \ D \cdot \text{Var})$
 $= \{$
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 ? $\gamma_D'$  = restrict-subst-domain (vars-cls (add-mset K D))  $\gamma_D$ 

have K · l ? $\gamma_D'$  = K · l  $\gamma_D$  and D · ? $\gamma_D'$  = D ·  $\gamma_D$ 
by (simp-all add: subst-lit-restrict-subst-domain subst-cls-restrict-subst-domain)
hence K · l ? $\gamma_D'$  = - (L · l  $\gamma_C$ ) and ground-prop': is-ground-cls (add-mset
K D · ? $\gamma_D'$ )
using 2 ground-prop by simp-all

have dom- $\gamma_D'$ : subst-domain ? $\gamma_D'$   $\subseteq$  vars-cls (add-mset K D)
by simp

let ? $\gamma$  =  $\lambda x$ .
if  $x \in$  vars-cls (add-mset L C' ·  $\varrho$ ) then
  rename-subst-domain  $\varrho$   $\gamma_C$  x
else
   $\gamma_D$  x
have L · l  $\varrho$  · l ? $\gamma$  = L · l  $\gamma_C$  and K · l ? $\gamma$  = K · 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-of L · a  $\varrho$  · a ? $\gamma$  = atm-of K · a ? $\gamma$ 
by (smt (verit, best) 2 atm-of-uminus subst-lit-id-subst atm-of-subst-lit)
then obtain  $\mu$  where Unification.mgu (atm-of L · a  $\varrho$ ) (atm-of K) =
Some  $\mu$ 
using ex-mgu-if-subst-apply-term-eq-subst-apply-term
by blast
hence imgu- $\mu$ : is-imgu  $\mu$  {{atm-of L · a  $\varrho$ , atm-of K · a 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- $\varrho$  is-renaming-id-subst disjoint-vars imgu- $\mu$ 
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  $K_\Gamma$  C  $\gamma_C$  N  $\beta$  n  $\Gamma'$  U])
with no-new-skip show False
by (metis S-def  $\Gamma$ -def u-def)
qed
qed
thus ?thesis ..
qed
qed

```


qed

corollary *correct-termination-strategy*:

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

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

$gnd-N \equiv grounding-of-cls\ (fset\ N)$ **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 –

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* $\beta\ S$

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  $\nexists S'. \text{conflict } N \beta S S'$ 
    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  $\nexists S'. \text{propagate } N \beta S S'$ 
    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  $\nexists S'. \text{decide } N \beta S S' \wedge (\nexists S''. \text{conflict } N \beta S' S'')$ 
    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  $\nexists S'. \text{skip } N \beta S S'$ 
    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: $\nexists S'. \text{scl } N \beta S S'$

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 - $N \beta$, folded *gnd-N-def*, folded *gnd-N-lt- β -def*])
show (*scl* $N \beta$)** *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'
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 $N \beta$)** initial-state S and*
no-step: $\nexists S'. \text{reasonable-scl } N \beta S S'$
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 $\neg \text{satisfiable gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
satisfiable gnd-N-lt- β \wedge trail-true-cls (state-trail S) gnd-N-lt- β
proof (*rule correct-termination-strategy*[of - $N \beta$, folded *gnd-N-def*, folded *gnd-N-lt- β -def*])
show (*reasonable-scl* $N \beta$)** *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-β*
assumes
*run: (regular-scl N β)** initial-state S and*
no-step: $\nexists S'. \text{regular-scl } N \beta S S'$
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 $\neg \text{satisfiable } \text{gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable } \text{gnd-N-lt-}\beta \wedge \text{trail-true-cls} (\text{state-trail } S) \text{gnd-N-lt-}\beta$
proof (*rule correct-termination-strategy*[*of - N β, folded gnd-N-def, folded gnd-N-lt-β-def*])
show (*regular-scl N β)** initial-state S*
by (*rule run*)
next
show $\nexists S'. \text{regular-scl } N \beta S S'$
by (*rule no-step*)
next
show $\bigwedge S S'. \text{shortest-backtrack-strategy } \text{regular-scl } N \beta S S' \implies \text{regular-scl } N \beta S S'$
by (*simp add: reasonable-if-regular regular-scl-if-shortest-backtrack-strategy*)
next
show $\bigwedge N \beta S S'. \text{regular-scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
using *scl-preserves-sound-state*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N \beta S S'. \text{regular-scl } N \beta S S' \implies \text{trail-atoms-lt } \beta S \implies \text{trail-atoms-lt } \beta S'$

$\beta S'$
using *scl-preserves-trail-atoms-lt*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N \beta S S'. \text{regular-scl } N \beta S S' \implies \text{trail-propagated-or-decided}' N \beta S$
 \implies
 $\text{trail-propagated-or-decided}' N \beta S'$
using *scl-preserves-trail-propagated-or-decided*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N \beta S S'. \text{regular-scl } N \beta S S' \implies \text{trail-lits-consistent } S \implies \text{trail-lits-consistent } S'$
using *scl-preserves-trail-lits-consistent*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N \beta S S'. \text{regular-scl } N \beta S S' \implies \text{ground-false-closures } S \implies \text{ground-false-closures } S'$
using *scl-preserves-ground-false-closures*[*OF scl-if-regular*] **by** *simp*
qed

corollary *correct-termination-shortest-backtrack-strategy-regular-scl*:
fixes *gnd-N* **and** *gnd-N-lt- β*
assumes
run: (*shortest-backtrack-strategy regular-scl N β*)** *initial-state S* **and**
no-step: $\nexists S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S'$
defines
gnd-N \equiv *grounding-of-class (fset N)* **and**
gnd-N-lt- β \equiv $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
shows $\neg \text{satisfiable gnd-N} \wedge (\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)) \vee$
 $\text{satisfiable gnd-N-lt-}\beta \wedge \text{trail-true-class (state-trail } S) \text{ gnd-N-lt-}\beta$
proof (*rule correct-termination-strategy*[*of - N β , folded gnd-N-def, folded gnd-N-lt- β -def*])
show (*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 $\bigwedge S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{shortest-backtrack-strategy regular-scl } N \beta S S'$
by *simp*
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$
using *scl-preserves-sound-state*[*OF scl-if-regular*]
by (*auto simp: shortest-backtrack-strategy-def*)
next
show $\bigwedge N \beta S S'. \text{shortest-backtrack-strategy regular-scl } N \beta S S' \implies \text{trail-atoms-lt } \beta S \implies \text{trail-atoms-lt } \beta S'$
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 ⇒
  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 ∧ N β S S'. shortest-backtrack-strategy regular-scl N β S S' ⇒ trail-lits-consistent
S ⇒ trail-lits-consistent S'
  using scl-preserves-trail-lits-consistent[OF scl-if-regular]
  by (auto simp: shortest-backtrack-strategy-def)
next
show ∧ N β S S'. shortest-backtrack-strategy regular-scl N β S S' ⇒ ground-false-closures
S ⇒ 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 ∧ (∄ S'. scl N β S S') ∨
(reasonable-scl N β)** initial-state S ∧ (∄ S'. reasonable-scl N β S S') ∨
(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 $r :: 'a \text{ rel}$ **and** $s :: 'b \text{ rel}$ **and** $f :: 'a \Rightarrow 'b$
assumes *wf s* **and** *convertible*: $\bigwedge x y. (x, y) \in r \Longrightarrow (f x, f y) \in s$
shows *wf r*
proof (*rule wfI-min*[*of r*])
fix $x :: 'a$ **and** $Q :: 'a \text{ set}$
assume $x \in Q$
then obtain y **where** $y \in Q$ **and** $\bigwedge z. (f z, f y) \in s \Longrightarrow z \notin Q$
by (*auto elim: wfE-min*[*OF wf-inv-image*[*of s f*, *OF <wf s*], *unfolded in-inv-image*])
thus $\exists z \in Q. \forall y. (y, z) \in r \longrightarrow y \notin Q$
by (*auto intro: convertible*)
qed

lemma *wfP-if-convertible-to-wfP*: $wfP S \Longrightarrow (\bigwedge x y. R x y \Longrightarrow S (f x) (f y)) \Longrightarrow wfP R$
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 $f :: - \Rightarrow nat$
shows $(\bigwedge x y. R x y \Longrightarrow f x < f y) \Longrightarrow wfP R$
by (*rule wfP-if-convertible-to-wfP*[*of (< :: nat \Rightarrow nat \Rightarrow 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**

$trail-less' Ls = (\lambda L K.$
 $(\exists i. i < length Ls \wedge L = Ls ! i \wedge K = - (Ls ! i)) \vee$
 $(\exists i. Suc i < length Ls \wedge L = - (Ls ! Suc i) \wedge K = Ls ! i))^{++}$

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

proof (*rule transpI*)

show $\bigwedge x y z. trail-less' Ls x y \implies trail-less' Ls y z \implies trail-less' Ls x z$

by (*metis (no-types, lifting) trail-less'-def tranclp-trans*)

qed

lemma *trail-less'-Suc*:

assumes $Suc i < length Ls$

shows $trail-less' Ls (Ls ! Suc i) (Ls ! i)$

proof –

have $trail-less' Ls (Ls ! Suc i) (- (Ls ! Suc i))$

unfolding *trail-less'-def*

using *assms by blast*

moreover have $trail-less' Ls (- (Ls ! Suc i)) (Ls ! i)$

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 $trail-less' Ls (- (Ls ! Suc i)) (- (Ls ! i))$

proof –

have $trail-less' Ls (- (Ls ! Suc i)) (Ls ! i)$

unfolding *trail-less'-def*

using *assms Suc-lessD by blast*

moreover have $trail-less' Ls (Ls ! i) (- (Ls ! i))$

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:
   $j < i \implies i < \text{length } Ls \implies \text{trail-less}' Ls (- (Ls ! i)) (- (Ls ! j))$ 
proof (induction i arbitrary: j)
  case 0
  then show ?case
    by simp
next
  case (Suc i)
  then show ?case
    using trail-less'-comp-Suc-comp
    by (metis Suc-lessD less-Suc-eq transpD transp-trail-less')
qed

lemma trail-less'-id-comp:
  assumes  $j < i$  and  $i < \text{length } Ls$ 
  shows  $\text{trail-less}' Ls (Ls ! i) (- (Ls ! j))$ 
proof -
  have  $\text{trail-less}' Ls (Ls ! j) (- (Ls ! j))$ 
    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 :: uminus) list
  assumes
    uminus-not-id:  $\bigwedge x :: 'a. - x \neq x$  and
    uminus-uminus-id:  $\bigwedge x :: 'a. - (- x) = x$  and
    pairwise-distinct:
       $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$ 
  j)
  shows trail-less Ls = trail-less' Ls
proof (intro ext iffI)
  fix L K
  show trail-less Ls L K  $\implies$  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  $i < \text{length } Ls$  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  $\text{Suc } i < \text{length } Ls$  and  $L = - (Ls ! \text{Suc } i)$  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 ⟨y = Ls ! i⟩ ⟨z = - Ls ! i⟩)
    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 ⟨i < length Ls⟩ ⟨y = Ls ! i⟩ ⟨z = - Ls ! i⟩ 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 ⟨i < length Ls⟩ ⟨y = Ls ! i⟩ ⟨z = - Ls ! i⟩ 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 ⟹ trail-less Ls L z
    by (metis ⟨Suc i < length Ls⟩ ⟨y = - Ls ! Suc i⟩ ⟨z = Ls ! i⟩ 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)  $\langle Suc\ i < length\ Ls \rangle \langle y = -\ Ls\ !\ Suc\ i \rangle \langle z = Ls\ !\ i \rangle$ 
      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  $\langle Suc\ i < length\ Ls \rangle \langle y = -\ Ls\ !\ Suc\ i \rangle$ 
       $\langle z = Ls\ !\ i \rangle$ 
      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  $\langle Suc\ i < length\ Ls \rangle \langle y = -\ Ls\ !\ Suc\ i \rangle \langle z = Ls\ !\ i \rangle$ 
      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 $(0 :: nat) \leq 0$ **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 $(1 :: \text{nat}) \leq 1$ **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 $(1 :: \text{nat}) < 2$ **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 $(2 :: \text{nat}) \leq 2$ **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 $(0 :: \text{nat}) < 1$ **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 $(1 :: \text{nat}) < 2$ **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 $(0 :: \text{nat}) < 1$ **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 $(1 :: \text{nat}) < 2$ **by** *presburger*
qed *simp-all*

end

9.2 Miscellaneous Lemmas

lemma *not-trail-less-Nil*: $\neg \text{trail-less } [] L K$
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 $L \in \text{set } Ls \cup \text{uminus } 'a \text{ set } Ls\ K \in \text{set } Ls \cup \text{uminus } 'a \text{ set } Ls$
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 $\neg \text{trail-less } Ls\ L\ K - \text{trail-less } Ls\ K\ L$
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 $L \in \text{set } Ls \cup \text{uminus } 'a \text{ set } Ls \iff L \in \text{set } Ls \vee - L \in \text{set } Ls$
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 *trail-less-id-id* *Ls L K*
 \neg *trail-less-comp-id* *Ls L K*
 \neg *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 *trail-less-id-id* *Ls L K*
 \neg *trail-less-id-comp* *Ls L K*
 \neg *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

\neg *trail-less-id-id* *Ls L K*
 \neg *trail-less-id-comp* *Ls L K*
 \neg *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:
       $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls !$ 
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 asymp-trail-less:

```


fixes $Ls :: ('a :: \text{uminus}) \text{ list}$
assumes
 $\text{uminus-not-id}: \bigwedge x :: 'a. - x \neq x$ **and**
 $\text{uminus-uminus-id}: \bigwedge x :: 'a. - (- x) = x$ **and**
 $\text{pairwise-distinct}:$
 $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
shows $\text{asymptotic} (\text{trail-less } Ls)$
using $\text{irreflp-trail-less}[OF \text{ assms}] \text{ transp-trail-less}[OF \text{ assms}]$
using $\text{asymptotic-on-iff-irreflp-on-iff-transp-on}$
by auto

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

lemma $\text{totalp-on-trail-less}:$
 $\text{totalp-on} (\text{set } Ls \cup \text{uminus } ' \text{ set } Ls) (\text{trail-less } Ls)$
proof ($\text{rule } \text{totalp-onI}, \text{unfold } \text{Un-iff}, \text{elim } \text{disjE}$)
fix $L K$
assume $L \in \text{set } Ls$ **and** $K \in \text{set } Ls$ **and** $L \neq K$
then obtain $i j$ **where** $i < \text{length } Ls$ $Ls ! i = L$ $j < \text{length } Ls$ $Ls ! j = K$
unfolding in-set-conv-nth **by** auto
thus $\text{trail-less } Ls L K \vee \text{trail-less } Ls K L$
using $\langle L \neq K \rangle \text{less-linear}[of i j]$
by ($\text{meson } \text{trail-less-def } \text{trail-less-id-id-def}$)
next
fix $L K$
assume $L \in \text{set } Ls$ **and** $K \in \text{uminus } ' \text{ set } Ls$ **and** $L \neq K$
then obtain $i j$ **where** $i < \text{length } Ls$ $Ls ! i = L$ $j < \text{length } Ls$ $-(Ls ! j) = K$
unfolding $\text{in-set-conv-nth } \text{image-set } \text{length-map}$ **by** auto
thus $\text{trail-less } Ls L K \vee \text{trail-less } Ls K L$
using $\text{less-linear}[of i j]$
by ($\text{metis } \text{le-eq-less-or-eq } \text{trail-less-comp-id-def } \text{trail-less-def } \text{trail-less-id-comp-def}$)
next
fix $L K$
assume $L \in \text{uminus } ' \text{ set } Ls$ **and** $K \in \text{set } Ls$ **and** $L \neq K$
then obtain $i j$ **where** $i < \text{length } Ls$ $-(Ls ! i) = L$ $j < \text{length } Ls$ $Ls ! j = K$
unfolding $\text{in-set-conv-nth } \text{image-set } \text{length-map}$ **by** auto
thus $\text{trail-less } Ls L K \vee \text{trail-less } Ls K L$
using $\text{less-linear}[of i j]$
by ($\text{metis } \text{le-eq-less-or-eq } \text{trail-less-comp-id-def } \text{trail-less-def } \text{trail-less-id-comp-def}$)
next
fix $L K$
assume $L \in \text{uminus } ' \text{ set } Ls$ **and** $K \in \text{uminus } ' \text{ set } Ls$ **and** $L \neq K$
then obtain $i j$ **where** $i < \text{length } Ls$ $-(Ls ! i) = L$ $j < \text{length } Ls$ $-(Ls ! j) = K$
unfolding $\text{in-set-conv-nth } \text{image-set } \text{length-map}$ **by** auto
thus $\text{trail-less } Ls L K \vee \text{trail-less } Ls K L$
using $\langle L \neq K \rangle \text{less-linear}[of i j]$
by ($\text{metis } \text{trail-less-comp-comp-def } \text{trail-less-def}$)

qed

9.6 Well-Founded

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

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

assumes

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

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

pairwise-distinct:

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

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

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

proof (rule *notI*, *unfold trail-less-def*, *elim disjE*)

show $\text{trail-less-id-id } (L \# Ls) (- L) L \Longrightarrow \text{False}$

unfolding *trail-less-id-id-def*

using *pairwise-distinct uminus-not-id by metis*

next

show $\text{trail-less-comp-id } (L \# Ls) (- L) L \Longrightarrow \text{False}$

unfolding *trail-less-comp-id-def*

using *pairwise-distinct uminus-uminus-id*

by (*metis less-not-refl*)

next

show $\text{trail-less-id-comp } (L \# Ls) (- L) L \Longrightarrow \text{False}$

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 $\text{trail-less-comp-comp } (L \# Ls) (- L) L \Longrightarrow \text{False}$

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 $L :: 'a :: \text{uminus}$

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

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

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

less-Cons: $\text{trail-less } (L \# Ls) H K$

```

shows 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.prems 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  $Q\text{-neq-empty}$ : False
        have  $Q\text{-minus-subset}$ :  $Q - \{L, -L\} \subseteq \text{set } Ls \cup \text{uminus } \text{'set } Ls$  using
          Cons-ind.prem(1) 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  $Q\text{-minus-subset}$  pairwise-distinct-Ls]
        proof (elim disjE bexE)
          assume  $Q - \{L, -L\} = \{\}$ 
          with  $Q\text{-neq-empty}$  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  $Q\text{-neq-empty}$   $\langle Q \subseteq \{L, -L\} \rangle$  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  $K\text{-in-}Q\text{-minus}$ :  $K \in Q - \{L, -L\}$  and  $\forall y. \text{trail-less } Ls \ y \ K \longrightarrow y$ 
               $\notin Q - \{L, -L\}$ 
            from  $K\text{-in-}Q\text{-minus}$  have  $L \neq K - L \neq K$  by auto
            from  $K\text{-in-}Q\text{-minus}$  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  $Ls :: ('a :: \text{uminus}) \text{ list}$ 
  assumes
    uminus-not-id:  $\bigwedge x :: 'a. - x \neq x$  and
    uminus-uminus-id:  $\bigwedge x :: 'a. - (- x) = x$  and
    pairwise-distinct:
       $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$ 
  shows wfP (trail-less  $Ls$ )
  unfolding wfP-eq-minimal
  proof (intro allI impI)
    fix  $M :: 'a \text{ set}$  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
  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 $L \in \text{set } Ls \vee - L \in \text{set } Ls$ then
 if $K \in \text{set } Ls \vee - K \in \text{set } Ls$ then
 trail-less *Ls* *L* *K*
 else
 True
 else
 if $K \in \text{set } Ls \vee - K \in \text{set } Ls$ then
 False
 else
 lt *L* *K*)

lemma

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

assumes

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

shows $K \in \text{set } Ls \vee - K \in \text{set } Ls \implies \text{trail-less-ex } lt \ Ls \ L \ K \longleftrightarrow \text{trail-less } Ls \ L \ K$

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 minus-minus-id*, *THEN iffD1*]]
by *auto*

lemma

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

assumes

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

shows $L \in \text{set } Ls \cup \text{uminus } ' \text{ set } Ls \implies K \notin \text{set } Ls \cup \text{uminus } ' \text{ set } Ls \implies$
trail-less-ex lt Ls L K

using *defined-conv minus-minus-id*

by (*auto simp add: trail-less-ex-def*)

lemma *irreflp-trail-ex-less*:

fixes $Ls :: ('a :: \text{uminus}) \text{ list}$ **and** $lt :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

assumes

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

minus-minus-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 minus-not-id minus-minus-id pairwise-distinct*] *irreflp-lt*

by (*simp add: irreflpD irreflpI*)

lemma *transp-trail-less-ex*:

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

assumes

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

minus-minus-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 minus-not-id minus-minus-id pairwise-distinct*] *transp-lt*

by (*smt (verit, ccfv-SIG) transp-def*)

lemma *asympt-trail-less-ex*:

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

assumes

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

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

pairwise-distinct:

$\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
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 :: uminus) list*
assumes
uminus-not-id: $\bigwedge x :: 'a. - x \neq x$ and
uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ and
pairwise-distinct:
 $\forall i < \text{length } Ls. \forall j < \text{length } Ls. i \neq j \longrightarrow Ls ! i \neq Ls ! j \wedge Ls ! i \neq - (Ls ! j)$
j) and
wfP-lt: wfP lt
shows *wfP (trail-less-ex lt Ls)*
unfolding *wfP-eq-minimal*
proof (*intro allI impI*)
fix *Q :: 'a 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 (set Ls \cup uminus ' set Ls) = {}*)
case *True*
then show *?thesis*
using *wfP-lt[unfolded 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,*
unfolded wfP-eq-minimal, of Q \cap (set Ls \cup uminus ' 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**

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

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-asympt-on[OF asympt-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 $y \in set\ xs$

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

syntax (input)
-fBall    :: pttrn ⇒ 'a fset ⇒ bool ⇒ bool    ((∃! (-/|:-)/ -) [0, 0, 10] 10)
-fBex     :: pttrn ⇒ 'a fset ⇒ bool ⇒ bool    ((∃? (-/|:-)/ -) [0, 0, 10] 10)

syntax
-fBall    :: pttrn ⇒ 'a fset ⇒ bool ⇒ bool    ((∃∀ (-/|∈|-)/ -) [0, 0, 10] 10)
-fBex     :: pttrn ⇒ 'a fset ⇒ bool ⇒ bool    ((∃∃ (-/|∈|-)/ -) [0, 0, 10] 10)

translations
∀ x|∈|A. P ⇔ CONST fBall A (λx. P)
∃ x|∈|A. P ⇔ CONST fBex A (λx. P)

print-translation ‹
[Syntax-Trans.preserve-binder-abs2-tr' const-syntax ‹fBall› syntax-const ‹-fBall›,
Syntax-Trans.preserve-binder-abs2-tr' const-syntax ‹fBex› syntax-const ‹-fBex›]
› — to avoid eta-contraction of body

global-interpretation comp-finsert-commute: comp-fun-commute finsert
proof (unfold-locales)
  show ∧ y x. finsert y ∘ finsert x = finsert x ∘ finsert y
    by auto
  qed

definition fset-mset :: 'a multiset ⇒ 'a fset
  where fset-mset = fold-mset finsert {||}

lemma fset-mset-mempty[simp]: fset-mset {#} = {||}
  by (simp add: fset-mset-def)

```

lemma *fset-mset-add-mset*[simp]: $fset\text{-}mset\ (add\text{-}mset\ x\ M) = finsert\ x\ (fset\text{-}mset\ M)$

by (*simp add: fset-mset-def*)

lemma *fset-fset-mset*[simp]: $fset\ (fset\text{-}mset\ M) = set\text{-}mset\ M$

by (*induction M rule: multiset-induct*) *simp-all*

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

by (*induction M rule: multiset-induct*) *simp-all*

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

by *simp*

lemma *fBex-fset-mset-iff*[simp]: $(\exists x\ |\in|\ fset\text{-}mset\ M.\ P\ x) \longleftrightarrow (\exists x\ \in\# \ M.\ P\ x)$

by *simp*

lemma *fmember-ffUnion-iff*: $a\ |\in|\ ffUnion\ (f\ |\cdot|\ A) \longleftrightarrow (\exists x\ |\in|\ A.\ a\ |\in|\ f\ x)$

unfolding *ffUnion.rep-eq* **by** *simp*

lemma *fBex-ffUnion-iff*: $(\exists z\ |\in|\ ffUnion\ (f\ |\cdot|\ A).\ P\ z) \longleftrightarrow (\exists x\ |\in|\ A.\ \exists z\ |\in|\ f\ x.\ P\ z)$

unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

lemma *fBall-ffUnion-iff*: $(\forall z\ |\in|\ ffUnion\ (f\ |\cdot|\ A).\ P\ z) \longleftrightarrow (\forall x\ |\in|\ A.\ \forall z\ |\in|\ f\ x.\ P\ z)$

unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

abbreviation *grounding-lits-of-clss* **where**

grounding-lits-of-clss $N \equiv \{L \cdot l\ \gamma \mid L\ \gamma.\ L \in \bigcup (set\text{-}mset\ 'N) \wedge is\text{-}ground\text{-}lit\ (L \cdot l\ \gamma)\}$

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**
 $L' \in \bigcup (\text{set-mset } ' \text{fset } (\text{state-learned } S))$ **and**
is-ground-lit $(L' \cdot l \ \gamma)$
by *auto*
then obtain $L_N \ \sigma_N$ **where** $L_N \in \bigcup (\text{set-mset } ' \text{fset } N)$ **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*:
 $\text{grounding-lits-of-clss } N = \{L \mid L \ C. \ \text{add-mset } L \ C \in \text{grounding-of-clss } N\}$
(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 $C \cdot \gamma \cdot \gamma_C \in \text{grounding-of-clss } N$
using $\langle C \in N \rangle \langle \text{is-ground-cls } (C \cdot \gamma \cdot \gamma_C) \rangle$
unfolding *grounding-of-clss-def*
by (*metis (no-types, opaque-lifting) UN-iff grounding-of-clss-ground*
grounding-of-subst-clss-subset insert-subset subsetD)

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-cls-def
  by (smt (verit, best) UN-iff grounding-of-cls-def mem-Collect-eq)
then obtain  $L' \ C'$  where  $CC = \text{add-mset } L' \ C'$  and  $L = L' \cdot l \ \gamma$  and  $C = C'$ 
   $\cdot \ \gamma$ 
  by (metis (no-types, lifting) msed-map-invR subst-cls-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 is-ground-lit  $(L' \cdot l \ \gamma)$ 
    using  $\langle \text{add-mset } L \ C \in \text{grounding-of-cls } N \rangle \langle L = L' \cdot l \ \gamma \rangle$ 
    by (metis grounding-ground is-ground-cls-add-mset)
qed
qed

corollary
  assumes initial-lits-generalize-learned-trail-conflict  $N \ S$ 
  defines  $U \equiv \text{state-learned } S$ 
  shows  $\{L \mid L \ C. \ \text{add-mset } L \ C \in \text{grounding-of-cls } (\text{fset } U)\} \subseteq$ 
     $\{L \mid L \ C. \ \text{add-mset } L \ C \in \text{grounding-of-cls } (\text{fset } N)\}$ 
  using assms grounding-lits-of-learned-subset-grounding-lits-of-initial
  unfolding grounding-lits-of-cls-conv
  by simp

end

end

theory Multiset-Order-Extra
  imports HOL-Library.Multiset-Order
begin

lemma strict-subset-implies-multpHO:  $A \subset\# B \implies \text{multp}_{HO} \ r \ A \ B$ 
  unfolding multpHO-def
  by (simp add: leD mset-subset-eq-count)

end

theory Non-Redundancy
  imports
    SCL-FOL
    Trail-Induced-Ordering
    Initial-Literals-Generalize-Learned-Literals
    Multiset-Order-Extra

```

begin

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

10 Reasonable Steps

lemma *reasonable-scl-sound-state*:

reasonable-scl $N \beta S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$

using *scl-preserves-sound-state* *reasonable-scl-def* **by** *blast*

lemma *reasonable-run-sound-state*:

(reasonable-scl $N \beta)^{**}$ $S S' \implies \text{sound-state } N \beta S \implies \text{sound-state } N \beta S'$

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* $N \beta U \square \mid$

Cons: (*is-decision-lit* $Ln \longrightarrow (\nexists S'. \text{conflict } N \beta (Ln \# \Gamma, U, \text{None}) S')$) \implies

no-conflict-after-decide $N \beta U \Gamma \implies \text{no-conflict-after-decide } N \beta U (Ln \# \Gamma)$

definition *no-conflict-after-decide'* **where**

no-conflict-after-decide' $N \beta S = \text{no-conflict-after-decide } N \beta (\text{state-learned } S)$
(*state-trail* S)

lemma *no-conflict-after-decide'-initial-state*[*simp*]: *no-conflict-after-decide'* $N \beta$ *initial-state*

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** $\nexists S''. \text{conflict } N \beta S' S''$ **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 β S S' and no-conflict-after-decide' N β S
shows no-conflict-after-decide' N β 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 β S S' and no-conflict-after-decide' N β S
shows no-conflict-after-decide' N β S'
using assms
by (auto simp: no-conflict-after-decide'-def elim: factorize.cases)

lemma resolve-preserves-no-conflict-after-decide':
assumes resolve N β S S' and no-conflict-after-decide' N β S
shows no-conflict-after-decide' N β 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) 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 β (Γ, finsert C U, None :: ('f, 'v) closure option) S'
then show  $\exists S'. \text{conflict } N \beta (\Gamma, U, \text{None}) S'$ 
proof (cases N β (Γ, finsert C U, None :: ('f, 'v) closure option) S' rule:
conflict.cases)
case (conflictI D γ)
then show ?thesis
using assms conflict.intros by blast
qed
qed

lemma backtrack-preserves-no-conflict-after-decide':
assumes step: backtrack N β S S' and invar: no-conflict-after-decide' N β S
shows no-conflict-after-decide' N β S'
using step
proof (cases N β S S' rule: backtrack.cases)
case (backtrackI Γ Γ' Γ'' K L σ D U)
have no-conflict-after-decide N β U (Γ' @ Γ'')
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* Γ') (*auto elim: no-conflict-after-decide.cases*)
hence *no-conflict-after-decide* $N \beta$ (*finsert* (*add-mset* $L D$) U) Γ''
using *backtrackI*(5)
proof (*induction* Γ'')
case *Nil*
show *?case*
by (*auto intro: no-conflict-after-decide.Nil*)
next
case (*Cons* $Ln \Gamma''$)
hence $\nexists \gamma. \text{is-ground-cls } (add\text{-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } (Ln \# \Gamma'')$ (*add-mset* $L D \cdot \gamma$)
by *metis*
hence $\nexists \gamma. \text{is-ground-cls } (add\text{-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } \Gamma''$ (*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 $1: \text{no-conflict-after-decide } N \beta$ (*finsert* (*add-mset* $L D$) U) Γ''
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 $\nexists S'. \text{conflict } N \beta$ ($Ln \# \Gamma'', \text{finsert } (add\text{-mset } L D) U, \text{None}$) S'
using *learning-clause-without-conflict-preserves-nex-conflict*
using $\langle \nexists \gamma. \text{is-ground-cls } (add\text{-mset } L D \cdot \gamma) \wedge \text{trail-false-cls } (Ln \# \Gamma'')$
(*add-mset* $L D \cdot \gamma$)
by *blast*
next
show *no-conflict-after-decide* $N \beta$ (*finsert* (*add-mset* $L D$) U) Γ''
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*(β) **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*: *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*:

(*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 \[] \mid$

Cons: ($\nexists S'. \text{conflict } N \beta (Ln \# \Gamma, U, \text{None}) S'$) \implies

$\text{no-conflict-with-trail } N \beta U \Gamma \implies \text{no-conflict-with-trail } N \beta U (Ln \# \Gamma)$

lemma *nex-conflict-if-no-conflict-with-trail*:

assumes *no-conflict-with-trail* $N \beta U \Gamma$

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

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*:

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*:

no-conflict-with-trail $N \beta U \Gamma \implies D \mid \in \mid N \mid \cup \mid U \implies D \neq \{\#\} \implies \text{is-ground-cls}$
 $(D \cdot \gamma) \implies$

$\neg \text{trail-false-cls } \Gamma (D \cdot \gamma)$

proof (*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 Ln \# \Gamma \Rightarrow$ *if is-decision-lit* Ln *then* $Ln \# \Gamma$ *else*

Γ)

lemma *nex-conflict-if-no-conflict-with-trail''*:
assumes *no-conf*: *state-conflict* $S = \text{None}$ **and** $\{\#\} \notin N$ **and** *learned-nonempty* S
no-conflict-with-trail $N \beta$ (*state-learned* S) (*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** $\{\#\} \notin U$
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 $\langle \{\#\} \notin N \rangle \langle \{\#\} \notin U \rangle$
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* $N \beta$ (*state-learned* S) (*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 Gamma*)
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 $N \beta U \Gamma \implies$ 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 $N \beta$ 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 ΓU 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 $\nexists S''. \text{conflict } N \beta S' S''$

```

    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) # Γ else
 Γ)
 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 $S = (\Gamma, U, Some Cl1)$ and $S' = (\Gamma, U,$
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  $\beta$ )** initial-state S1 and
  conf: conflict N  $\beta$  S1 S2 and
  {#} | $\notin$ | N
shows  $\exists S0$ . (regular-scl N  $\beta$ )** initial-state S0  $\wedge$  regular-scl N  $\beta$  S0 S1  $\wedge$ 
(propagate N  $\beta$  S0 S1)
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 reason-
    able-if-regular]])

  from step.hyps(1) have trail-propagated-or-decided' N  $\beta$  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  $\beta$  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  $\beta$  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  $\beta$  S1
  by (simp add: regular-scl-preserves-almost-no-conflict-with-trail)

show ?case
proof (intro exI conjI)
  show (regular-scl N  $\beta$ )** initial-state S0
  using step.hyps by simp
next
  show regular-scl N  $\beta$  S0 S1
  using step.hyps by simp
next
from step.premis obtain  $\Gamma$  U C  $\gamma$  where
  S1-def: S1 = ( $\Gamma$ , U, None) and
  S2-def: S2 = ( $\Gamma$ , U, Some (C,  $\gamma$ )) and
  C-in: C | $\in$ | N | $\cup$ | 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 β 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* $S = None$ **and** *state-conflict* $S' = None$
by (*auto elim: decide.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  $\langle \text{state-conflict } S = \text{None} \rangle$  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  $\langle \text{state-conflict } S' = \text{Some } (C, \gamma) \rangle$ 
    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  $\langle \text{state-conflict } S = \text{None} \rangle$  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  $\exists S0 S1 S2 S3. (\text{regular-scl } N \beta)** \text{initial-state } S0 \wedge$ 
      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 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-def:  $S = (\Gamma, U, Cl)$ 
    by (metis state-simp)

  show ?thesis
  proof (cases  $\Gamma$ )
    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  $\Gamma'$ )
  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: (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

from reg-run have (regular-scl  $N \beta$ )** initial-state  $S1$ 
  using  $\langle \text{regular-scl } N \beta S0 S1 \rangle$  by simp

```

hence $\langle \text{regular-scl } N \ \beta \rangle^{**} \text{ initial-state } S2$
using $\langle \text{regular-scl } N \ \beta \ S1 \ S2 \rangle$ **by** *simp*
hence $\langle \text{regular-scl } N \ \beta \rangle^{**} \text{ initial-state } S3$
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 $(\text{elim } \text{factorize.cases})$ *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** $\text{almost-no-conflict-with-trail } N$
 $\beta \ S3$
using *regular-scl-preserves-almost-no-conflict-with-trail*
by $(\text{induction } S3 \ \text{rule: } r\text{tranclp-induct})$ *simp-all*

show $\exists S0 \ S1 \ S2 \ S3. (\text{regular-scl } N \ \beta)^{**} \ \text{initial-state } S0 \ \wedge$
 $\text{propagate } N \ \beta \ S0 \ S1 \ \wedge \ \text{regular-scl } N \ \beta \ S0 \ S1 \ \wedge$
 $\text{conflict } N \ \beta \ S1 \ S2 \ \wedge \ \text{regular-scl } N \ \beta \ S1 \ S2 \ \wedge$
 $(\text{factorize } N \ \beta)^{**} \ S2 \ S3 \ \wedge \ (\text{regular-scl } N \ \beta)^{**} \ S2 \ S3 \ \wedge$
 $(S3 = S' \vee (\exists S4. \text{resolve } N \ \beta \ S3 \ S4 \ \wedge \ (\text{skip } N \ \beta \sqcup \text{factorize } N \ \beta \sqcup \text{resolve } N \ \beta)^{**} \ S4 \ S'))$
using *reg-run propa confl facto*
proof $(\text{intro impI exI conjI})$
show $S3 = S' \vee (\exists S4. \text{resolve } N \ \beta \ S3 \ S4 \ \wedge \ (\text{skip } N \ \beta \sqcup \text{factorize } N \ \beta \sqcup \text{resolve } N \ \beta)^{**} \ S4 \ S')$
using *maybe-reso*

```

proof (elim disjE exE conjE)
  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 {#} |∄| U
  using ⟨{#} |∄| state-learned S⟩ ⟨S3 = S⟩ ⟨state-learned S3 = state-learned
S1⟩
  unfolding conflictI(1,2)
  by simp
  thus D ≠ {#}

```

```

    using ⟨{#} |≠| N⟩ ⟨D |∈| N |∪| U⟩
    by auto
  next
    show is-ground-cls (D · γD)
    by (rule ⟨is-ground-cls (D · γD)⟩)
  qed

  ultimately have - (L · l γ) ∈# D · γD
  by (metis subtrail-falseI propagate-lit-def)

  moreover have state-conflict S2 = Some (D, γD)
  unfolding conflictI(1,2) by simp

  ultimately show ?thesis
  by metis
  qed
  then obtain D γD where state-conflict S2 = Some (D, γD) and - (L · l
  γ) ∈# D · γD
  by metis

  with ⟨(factorize N β)** S2 S3⟩
  have ∃ D' γD'. state-conflict S3 = Some (D', γD') ∧ - (L · l γ) ∈# D' ·
  γD'
  proof (induction S3 arbitrary: rule: rtranclp-induct)
    case base
    thus ?case by simp
  next
    case (step y z)
    then obtain D' γD' where state-conflict y = Some (D', γD') and - (L
    · l γ) ∈# D' · γD'
    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
  γ⟩
  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 β S S'* **and** *reg-step: regular-scl N β S S'* **and**
invar: regular-conflict-resolution N β S

shows *regular-conflict-resolution* $N \beta S'$
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 $\{\#\} \mid \notin \mid N$
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: $S3 = S \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{\text{**}} S4 S)$
unfolding *regular-conflict-resolution-def* $\langle \text{state-conflict } S = \text{Some } (C, \gamma) \rangle$
unfolding *option.cases*
by *metis*

show $\exists S0 S1 S2 S3. (\text{regular-scl } N \beta)^{\text{**}} \text{initial-state } S0 \wedge$
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* $N \beta$)^{**} $S2 S3 \wedge (\text{regular-scl } N \beta)^{\text{**}} S2 S3 \wedge$
 $(S3 = S' \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{\text{**}} S4 S'))$
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 $\langle (\text{factorize } N \beta)^{\text{**}} S2 S3 \rangle$ *step*
by (*simp add*: $\langle S3 = S \rangle$)
next
show (*regular-scl* $N \beta$)^{**} $S2 S'$
using $\langle (\text{regular-scl } N \beta)^{\text{**}} S2 S3 \rangle$ *reg-step*
by (*simp add*: $\langle S3 = S \rangle$)
next
show $S' = S' \vee (\exists S4. \text{resolve } N \beta S' S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{\text{**}} S4 S')$
by *simp*
qed
next
fix $S4$ **assume** *hyps*: *resolve* $N \beta S3 S4$ $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{\text{**}} S4 S$

```

show ?thesis
  using reg-run propa confl facto
proof (intro exI conjI)
  show  $S3 = S' \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup$ 
   $\text{resolve } N \beta)^{**} S4 S')$ 
    using 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
     $\text{state-conflict } S = \text{Some } (C, \gamma)$  and  $\text{state-conflict } S' = \text{Some } (C', \gamma')$ 
    by (auto elim!: resolve.cases)

```

```

show ?thesis
  unfolding regular-conflict-resolution-def ‹ $\text{state-conflict } S' = \text{Some } (C', \gamma')$ ›
  unfolding option.cases
proof (intro impI)
  from step have  $\text{state-conflict } S \neq \text{None}$ 
    by (auto elim: resolve.cases)

```

```

assume {#} | $\notin$ |  $N$ 
with invar obtain  $S0 S1 S2 S3$  where
  reg-run:  $(\text{regular-scl } N \beta)^{**} \text{initial-state } S0$  and
  propagate  $N \beta S0 S1$   $\text{regular-scl } N \beta S0 S1$  and
  conflict  $N \beta S1 S2$   $\text{regular-scl } N \beta S1 S2$  and
   $(\text{factorize } N \beta)^{**} S2 S3$   $(\text{regular-scl } N \beta)^{**} S2 S3$  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 ‹ $\text{state-conflict } S = \text{Some } (C, \gamma)$ ›
  unfolding option.cases
  by metis

```

```

then show  $\exists S0 S1 S2 S3. (\text{regular-scl } N \beta)^{**} \text{initial-state } S0 \wedge$ 
   $\text{propagate } N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$ 
   $\text{conflict } N \beta S1 S2 \wedge \text{regular-scl } N \beta S1 S2 \wedge$ 
   $(\text{factorize } N \beta)^{**} S2 S3 \wedge (\text{regular-scl } N \beta)^{**} S2 S3 \wedge$ 
   $(S3 = S' \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{**} S4 S'))$ 
proof (intro exI conjI)
  show  $S3 = S' \vee (\exists S4. \text{resolve } N \beta S3 S4 \wedge (\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup$ 
   $\text{resolve } N \beta)^{**} S4 S')$ 

```



```

    using maybe-reso 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 L_n Γ' **where** $\Gamma = L_n \# \Gamma'$

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 L_n then L_n # 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 $(\text{regular-scl } N \ \beta)^{**}$ *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** $(\text{regular-scl } N \ \beta)^{**}$ $S' \ S$
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** $S'\text{-def}$: $S' = ([], \{\|\}, \text{Some } (\{\#\}, \gamma))$
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 $S'\text{-def}$
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 $N \ \beta \ S0 \ S1 \ \wedge$ *regular-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 *backt* **obtain** $L \ C \ \gamma$ **where** *conflict-S*: *state-conflict* $S = \text{Some } (\text{add-mset } L \ C, \gamma)$
by (*auto elim: backtrack.cases*)

have $\{\#\} \notin N$

```

proof (rule mempty-not-in-initial-clauses-if-non-empty-regular-conflict)
  show state-conflict  $S = \text{Some} (\text{add-mset } L \ C, \ \gamma)$ 
    by (rule ⟨state-conflict  $S = \text{Some} (\text{add-mset } L \ C, \ \gamma)$ ⟩)
next
  show add-mset  $L \ C \neq \{\#\}$ 
    by simp
next
  show almost-no-conflict-with-trail  $N \ \beta \ S$ 
    by (rule ⟨almost-no-conflict-with-trail  $N \ \beta \ S$ ⟩)
next
  show sound-state  $N \ \beta \ S$ 
    by (rule ⟨sound-state  $N \ \beta \ S$ ⟩)
next
  show ground-false-closures  $S$ 
    by (rule ⟨ground-false-closures  $S$ ⟩)
qed

then obtain  $S0 \ S1 \ S2 \ S3$  where
  reg-run: (regular-scl  $N \ \beta$ )** initial-state  $S0$  and
  propa: propagate  $N \ \beta \ S0 \ S1$  regular-scl  $N \ \beta \ S0 \ S1$  and
  confl: conflict  $N \ \beta \ S1 \ S2$  and
  fact: (factorize  $N \ \beta$ )**  $S2 \ S3$  and
  maybe-resolution:  $S3 = S \vee$ 
    ( $\exists S4. \text{resolve } N \ \beta \ S3 \ S4 \wedge (\text{skip } N \ \beta \sqcup \text{factorize } N \ \beta \sqcup \text{resolve } N \ \beta)$ **  $S4 \ S$ )
using ⟨regular-conflict-resolution  $N \ \beta \ S$ ⟩ ⟨state-conflict  $S = \text{Some} (\text{add-mset } L$ 
 $C, \ \gamma)$ ⟩
  unfolding regular-conflict-resolution-def conflict-S option.case
  by metis

have  $S3 \neq S$ 
proof (rule notI)
  from ⟨(factorize  $N \ \beta$ )**  $S2 \ S3$ ⟩ have state-trail  $S3 = \text{state-trail } S2$ 
    by (induction  $S3$  rule: rtranclp-induct) (auto elim: factorize.cases)
  also from ⟨conflict  $N \ \beta \ S1 \ S2$ ⟩ have ... = state-trail  $S1$ 
    by (auto elim: conflict.cases)
  finally have state-trail  $S3 = \text{state-trail } S1$ 
    by assumption
  from ⟨propagate  $N \ \beta \ S0 \ S1$ ⟩ obtain  $L \ C \ \gamma$  where
    state-trail  $S1 = \text{trail-propagate} (\text{state-trail } S0) \ L \ C \ \gamma$ 
    by (auto elim: propagate.cases)

from ⟨(factorize  $N \ \beta$ )**  $S2 \ S3$ ⟩ have state-learned  $S3 = \text{state-learned } S2$ 
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 $\langle \text{conflict } N \beta S1 S2 \rangle$ **have** $\dots = \text{state-learned } S1$
by $(\text{auto elim: conflict.cases})$
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 $(\text{auto elim: propagate.cases})$

assume $S3 = S$
hence $\text{no-conf-with-trail: no-conflict-with-trail } N \beta (\text{state-learned } S0) (\text{state-trail } S0)$
using $\langle \text{almost-no-conflict-with-trail } N \beta S \rangle \langle \{\#\} \mid \notin \mid N \rangle$
 $\langle \text{state-trail } S1 = \text{trail-propagate } (\text{state-trail } S0) L C \gamma \rangle \langle \text{state-trail } S3 = \text{state-trail } S1 \rangle$
 $\langle \text{state-learned } S3 = \text{state-learned } S1 \rangle \langle \text{state-learned } S1 = \text{state-learned } S0 \rangle$
by $(\text{simp add: almost-no-conflict-with-trail-def propagate-lit-def is-decision-lit-def})$
hence $\{\#\} \mid \notin \mid \text{state-learned } S0$
using $\text{nex-conflict-if-no-conflict-with-trail}'[\text{folded mempty-in-iff-ex-conflict}]$ **by**
 simp

have $\exists D \gamma_D. \text{state-conflict } S2 = \text{Some } (D, \gamma_D) \wedge - (L \cdot l \gamma) \in \# D \cdot \gamma_D$
using $\langle \text{conflict } N \beta S1 S2 \rangle$
proof $(\text{cases } N \beta S1 S2 \text{ rule: conflict.cases})$
case $(\text{conflictI } D U \gamma_D \Gamma)$
hence $\text{trail-false-cls } (\text{trail-propagate } (\text{state-trail } S0) L C \gamma) (D \cdot \gamma_D)$
using $\langle \text{state-trail } S1 = \text{trail-propagate } (\text{state-trail } S0) L C \gamma \rangle$
by simp

moreover from $\text{no-conf-with-trail}$ **have** $\neg \text{trail-false-cls } (\text{state-trail } S0) (D \cdot \gamma_D)$
proof $(\text{rule not-trail-false-cls-if-no-conflict-with-trail})$
show $D \mid \in \mid N \mid \cup \mid \text{state-learned } S0$
using $\langle \text{state-learned } S1 = \text{state-learned } S0 \rangle \text{local.conflictI}(1) \text{local.conflictI}(3)$
by fastforce

next
have $\{\#\} \mid \notin \mid U$
using $\langle \{\#\} \mid \notin \mid \text{state-learned } S0 \rangle \langle S3 = S \rangle \langle \text{state-learned } S3 = \text{state-learned } S1 \rangle$
 $\langle \text{state-learned } S1 = \text{state-learned } S0 \rangle$
unfolding $\text{conflictI}(1,2)$
by simp
thus $D \neq \{\#\}$
using $\langle \{\#\} \mid \notin \mid N \rangle \langle D \mid \in \mid N \mid \cup \mid U \rangle$
by auto

next
show $\text{is-ground-cls } (D \cdot \gamma_D)$
by $(\text{rule } \langle \text{is-ground-cls } (D \cdot \gamma_D) \rangle)$
qed

ultimately have $\neg (L \cdot l \ \gamma) \in\# D \cdot \gamma_D$
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** $\neg (L \cdot l \ \gamma) \in\# D \cdot \gamma_D$
by *metis*

with $\langle \text{factorize } N \ \beta \rangle^{**} S2 \ S3 \rangle$
have $\exists D' \ \gamma_{D'}. \text{state-conflict } S3 = \text{Some } (D', \gamma_{D'}) \wedge \neg (L \cdot l \ \gamma) \in\# D' \cdot \gamma_{D'}$
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** $\neg (L \cdot l \ \gamma) \in\# D' \cdot \gamma_{D'}$
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 $(\text{regular-scl } N \ \beta)^{**} \text{initial-state } S0$
by *(rule* $\langle (\text{regular-scl } N \ \beta)^{**} \text{initial-state } S0 \rangle$ *)*

next
show *propagate* $N \ \beta \ S0 \ S1$
by *(rule* $\langle \text{propagate } N \ \beta \ S0 \ S1 \rangle$ *)*

next
show *regular-scl* $N \ \beta \ S0 \ S1$
by *(rule propa(2))*

next
show *conflict* $N \ \beta \ S1 \ S2$
by *(rule* $\langle \text{conflict } N \ \beta \ S1 \ S2 \rangle$ *)*

next

```

show (factorize  $N \beta$ )**  $S_2 S_3$ 
  by (rule  $\langle$ factorize  $N \beta$ )**  $S_2 S_3$  $\rangle$ )
next
show resolve  $N \beta S_3 S_4$ 
  by (rule  $\langle$ resolve  $N \beta S_3 S_4$  $\rangle$ )
next
show (skip  $N \beta \sqcup$  factorize  $N \beta \sqcup$  resolve  $N \beta$ )**  $S_4 S$ 
  by (rule  $\langle$ skip  $N \beta \sqcup$  factorize  $N \beta \sqcup$  resolve  $N \beta$ )**  $S_4 S$  $\rangle$ )
qed
qed

```

12 Resolve in Regular Runs

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

assumes

no-conf: $\nexists S_1$. *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 $\exists S_3$. *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 *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 $\forall x$. *is-Var* (ϱx) **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: ϱ -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- γ_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- μ' : is-imgu μ' $\{\{\text{atm-of } L' \cdot a \ \varrho, \text{atm-of } (L \cdot l \ \mu)\}\}$*
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* μ' $\{\{\text{atm-of } L' \cdot a \ \varrho, \text{atm-of } (L \cdot l \ \mu) \cdot a \ \text{Var}\}\}$
using *imgu- μ'* **by** *simp*

next
show *is-grounding-merge* $? \gamma \text{reso}$
 $(\text{vars-cls } (\text{add-mset } L' \ D' \cdot \varrho)) (\text{rename-subst-domain } \varrho \ \gamma_D)$
 $(\text{vars-cls } (\text{add-mset } (L \cdot l \ \mu) (C_0 \cdot \mu) \cdot \text{Var})) (\text{rename-subst-domain } \text{Var} \ \gamma)$
using *is-grounding-merge-if-mem-then-else*
by (*metis rename-subst-domain-Var-lhs*)

qed *simp-all*
thus *?thesis*
unfolding $S_2\text{-def } D\text{-def}$ **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 $\exists S_4. \text{resolve } N \beta S_3 S_4$

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* $\mu_L \{\{\text{atm-of } LL, \text{atm-of } LL'\}\}$

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* (*add-mset* $L C \cdot \gamma_C$)

unfolding *resolveI*(1,2)

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

have *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** $LL \cdot l \mu_L = LL' \cdot l \mu_L$

using *imgu- μ_L* [*unfolded is-imgu-def*, *THEN conjunct1*, *unfolded is-unifiers-def*, *simplified*]

using $\langle LL \cdot l \gamma_C = LL' \cdot l \gamma_C \rangle$

by (*metis* (*no-types*, *opaque-lifting*) *atm-of-subst-lit finite.emptyI finite.insertI insertCI*)

is-unifier-alt literal.expand subst-lit-is-neg)

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 $\langle \text{add-mset } L C = \text{add-mset } LL' (\text{add-mset } LL CC) \rangle$

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 $\varrho\varrho$ **where**
 $\varrho\varrho = renaming\text{-}wrt\ \{add\text{-}mset\ K\ D\}$

have $ren\text{-}\varrho\varrho$: *is-renaming* $\varrho\varrho$
 by (*metis* $\varrho\varrho\text{-}def$ *finite.emptyI finite.insertI is-renaming-renaming-wrt*)

have *disjoint-vars*: $\bigwedge C. vars\text{-}cls\ (C \cdot \varrho\varrho) \cap vars\text{-}cls\ (add\text{-}mset\ K\ D) = \{\}$
 by (*simp add:* $\varrho\varrho\text{-}def$ *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 $\exists \mu. Unification.mgu\ (atm\text{-}of\ L \cdot a\ \mu_L \cdot a\ \varrho\varrho)\ (atm\text{-}of\ K) = Some\ \mu$
proof (*rule ex-mgu-if-subst-eq-subst-and-disj-vars*)
show $vars\text{-}term\ (atm\text{-}of\ L \cdot a\ \mu_L \cdot a\ \varrho\varrho) \cap vars\text{-}lit\ K = \{\}$
using *disjoint-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-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 $atm\text{-}of\ L \cdot a\ \mu_L \cdot a\ \varrho\varrho \cdot a\ rename\text{-}subst\text{-}domain\ \varrho\varrho\ \gamma_C = atm\text{-}of\ L \cdot a\ \mu_L$
 $\cdot a\ \gamma_C$
using *ren- $\varrho\varrho$*
by (*simp add: is-renaming-iff renaming-cancels-rename-subst-domain*)
thus $atm\text{-}of\ L \cdot a\ \mu_L \cdot a\ \varrho\varrho \cdot a\ rename\text{-}subst\text{-}domain\ \varrho\varrho\ \gamma_C = atm\text{-}of\ K \cdot a\ \gamma_D$
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** *imgu- $\mu\mu$* : *is-imgu* $\mu\mu\ \{\{atm\text{-}of\ L \cdot a\ \mu_L \cdot a\ \varrho\varrho, atm\text{-}of\ K\}\}$
using *is-imgu-if-mgu-eq-Some* **by** *auto*

show *?thesis*
unfolding $S_3\text{-}def\ \langle add\text{-}mset\ LL\ CC \cdot \mu_L = add\text{-}mset\ L\ CCC \cdot \mu_L \rangle$
using *resolve.resolveI*[*OF* $\langle \Gamma = trail\text{-}propagate\ \Gamma'\ K\ D\ \gamma_D \rangle \langle K \cdot l\ \gamma_D = -\ (L \cdot l\ \mu_L \cdot l\ \gamma_C) \rangle$ *ren- $\varrho\varrho$*
is-renaming-id-subst, unfolded subst-atm-id-subst subst-cls-id-subst atm-of-subst-lit, OF disjoint-vars imgu- $\mu\mu$ is-grounding-merge-if-mem-then-else, of N β U

```

CCC ·  $\mu_L$ ]
  by auto
qed

```

The following lemma corresponds to Lemma 7 in the paper.

```

lemma no-backtrack-after-conflict-if:
  assumes conf: conflict  $N \beta S1 S2$  and trail-S2: state-trail  $S1 = \text{trail-propagate}$ 
 $\Gamma L C \gamma$ 
  shows  $\nexists S4. \text{backtrack } N \beta S2 S4$ 
proof –
  from trail-S2 conf have state-trail  $S2 = \text{trail-propagate } \Gamma L C \gamma$ 
  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  $N \beta S S' \implies \text{suffix } (\text{state-trail } S') (\text{state-trail } S)$ 
  by (auto simp: suffix-def elim: skip.cases)

```

```

lemma factorize-state-trail: factorize  $N \beta S S' \implies \text{state-trail } S' = \text{state-trail } S$ 
  by (auto elim: factorize.cases)

```

```

lemma resolve-state-trail: resolve  $N \beta S S' \implies \text{state-trail } S' = \text{state-trail } S$ 
  by (auto elim: resolve.cases)

```

```

lemma mempty-not-in-initial-clauses-if-run-leads-to-trail:
  assumes
    reg-run: (regular-scl  $N \beta$ )** initial-state  $S1$  and
    trail-lit: state-trail  $S1 = Lc \# \Gamma$ 
  shows  $\{\#\} \notin N$ 
proof (rule notI)
  assume  $\{\#\} \in N$ 
  then obtain  $\gamma$  where conflict  $N \beta$  initial-state ( $\{\}, \{\|\}, \text{Some } (\{\#\}, \gamma)$ )
  using conflict-initial-state-if-mempty-in-initial-clauses by auto
  moreover hence  $\nexists S'. \text{local.scl } N \beta (\{\}, \{\|\}, \text{Some } (\{\#\}, \gamma)) S'$  for  $\gamma$ 
  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: (skip  $N \beta \sqcup$  factorize  $N \beta \sqcup$  resolve  $N \beta$ )**  $S2 Sn$  and
backtrack:  $\exists Sn'. \text{backtrack } N \beta Sn Sn'$  and
mempty-not-in-init-clss:  $\{\#\} \notin N$  and
invars: learned-nonempty  $S1$  trail-propagated-or-decided'  $N \beta S1$  no-conflict-after-decide'
 $N \beta S1$ 
shows  $\neg \text{is-decision-lit } Lc \wedge \text{strict-suffix } (\text{state-trail } Sn) (\text{state-trail } S1)$ 
proof –
  obtain  $S0$  where propa: propagate  $N \beta S0 S1$ 
  using mempty-not-in-init-clss before-reasonable-conflict[OF 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  $\neg \text{is-decision-lit } Lc$ 
  by (auto simp: propagate-lit-def is-decision-lit-def elim!: propagate.cases)

show ?thesis
proof (rule conjI)
  show  $\neg \text{is-decision-lit } Lc$ 
  by (rule  $\langle \neg \text{is-decision-lit } Lc \rangle$ )
next
show strict-suffix (state-trail  $Sn$ ) (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 (state-trail  $Sn$ ) (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  $\langle \neg \text{is-decision-lit } Lc \rangle$  show state-trail  $Sn \neq \text{state-trail } S1$ 
  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-redundant $lt\ N\ C \longleftrightarrow \{D \in N. lt\ D\ C\} \models_e \{C\}$

definition *redundant* **where**

redundant $lt\ N\ C \longleftrightarrow$
 $(\forall C' \in \text{grounding-of-clss } C. \text{ground-redundant } lt\ (\text{grounding-of-clss } N)\ C')$

lemma *redundant* $lt\ N\ C \longleftrightarrow (\forall C' \in \text{grounding-of-clss } C. \{D' \in \text{grounding-of-clss } N. lt\ D'\ C'\} \models_e \{C'\})$

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

lemma *ground-redundant-iff*:

ground-redundant $lt\ N\ C \longleftrightarrow (\exists M \subseteq N. M \models_e \{C\} \wedge (\forall D \in M. lt\ D\ C))$

proof (*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}_{Fs} *Red-F_G* *Red-F*

defines

$\mathcal{G}_F \equiv \text{grounding-of-clss}$ **and**

$\mathcal{G}_{Fs} \equiv \text{grounding-of-clss}$ **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 $Red-F\text{-def } Red-F_{\mathcal{G}}\text{-def } \mathcal{G}_{F_s}\text{-def } \mathcal{G}_F\text{-def } \text{redundant-def}$ **by** *auto*

lemma *ground-redundant-if-strict-subset*:
assumes $D \in N$ **and** $D \subset\# C$
shows $\text{ground-redundant } (multp_{HO} \ R) \ N \ C$
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 $\text{redundant } (multp_{HO} \ R) \ N \ C$
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 $\text{ground-redundant } (multp_{HO} \ R) (\text{grounding-of-clss } N) \ C'$
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-clss-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 $\text{redundant } (multp_{HO} \ R) \ N \ C$
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 $\text{ground-redundant } (multp_{HO} \ R) (\text{grounding-of-clss } N) \ C'$
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*:
 $ground\text{-}redundant\ R\ N\ C \implies (\bigwedge x. x \in N \implies R\ x\ C \implies S\ x\ C) \implies ground\text{-}redundant\ S\ N\ C$
unfolding *ground-redundant-def*
by (*simp add: true-clss-def*)

lemma *redundant-mono-strong*:
 $redundant\ R\ N\ C \implies$
 $(\bigwedge x\ y. x \in grounding\text{-}of\text{-}clss\ N \implies y \in grounding\text{-}of\text{-}cls\ C \implies R\ x\ y \implies S\ x\ y) \implies$
 $redundant\ S\ N\ C$
by (*auto simp: redundant-def*)
*intro: ground-redundant-mono-strong[*of* R grounding-of-clss N - S]*

lemma *redundant-multp-if-redundant-strict-subset*:
 $redundant\ (\subset\#)\ N\ C \implies redundant\ (multp_{HO}\ R)\ N\ C$
by (*auto intro: strict-subset-implies-multp_{HO} elim!: redundant-mono-strong*)

lemma *redundant-multp-if-redundant-subset*:
 $redundant\ (\subset\#)\ N\ C \implies redundant\ (multp\ (trail\text{-}less\text{-}ex\ lt\ Ls))\ N\ C$
by (*auto intro: subset-implies-multp elim!: redundant-mono-strong*)

lemma *not-be_x-subset-mset-if-not-ground-redundant*:
assumes *is-ground-cls C and is-ground-cls N*
shows $\neg ground\text{-}redundant\ (\subset\#)\ N\ C \implies \neg (\exists D \in N. D \subset\# C)$
using *assms* **unfolding** *ground-redundant-def*
apply (*simp add: true-cls-def true-clss-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 Γ
defines $Ls \equiv (map\ fst\ \Gamma)$
shows *trail-consistent* $\Gamma \implies$
 $\forall i < length\ Ls. \forall j < length\ Ls. i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq - (Ls\ !\ j)$

```

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 :: 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-lt j-lt  $\langle i \neq j \rangle$  L-distinct Cons.IH[rule-format]
  using less-Suc-eq-0-disj by force
  qed
qed
qed

```

14.2 Strict Partial Order

lemma *irreflp-trail-less-if-trail-consistant*:
 $\text{trail-consistent } \Gamma \implies \text{irreflp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$
using *irreflp-trail-less*[*OF*
Clausal-Logic.uminus-not-id'
Clausal-Logic.uminus-of-uminus-id
pairwise-distinct-if-trail-consistent]
by *assumption*

lemma *transp-trail-less-if-trail-consistant*:
 $\text{trail-consistent } \Gamma \implies \text{transp } (\text{trail-less } (\text{map } \text{fst } \Gamma))$
using *transp-trail-less*[*OF*

Clausal-Logic.uminus-not-id'
Clausal-Logic.uminus-of-uminus-id
pairwise-distinct-if-trail-consistent]
by *assumption*

lemma *asympt-trail-less-if-trail-consistant:*
trail-consistent $\Gamma \implies$ *asympt* (*trail-less* (*map fst* Γ))
using *asympt-trail-less*[*OF*
Clausal-Logic.uminus-not-id'
Clausal-Logic.uminus-of-uminus-id
pairwise-distinct-if-trail-consistent]
by *assumption*

14.3 Properties

lemma *trail-defined-lit-if-trail-term-less:*
assumes *trail-term-less* (*map* (*atm-of o fst*) Γ) (*atm-of* L) (*atm-of* K)
shows *trail-defined-lit* Γ L *trail-defined-lit* Γ K
proof –
from *assms* **have** *atm-of* $L \in$ *set* (*map* (*atm-of o fst*) Γ) **and** *atm-of* $K \in$ *set*
(*map* (*atm-of o fst*) Γ)
by (*auto simp: trail-term-less-def*)
hence *atm-of* $L \in$ *atm-of ' fst ' set* Γ **and** *atm-of* $K \in$ *atm-of ' fst ' set* Γ
by *auto*
hence $L \in$ *fst ' set* $\Gamma \vee - L \in$ *fst ' set* Γ **and** $K \in$ *fst ' set* $\Gamma \vee - K \in$ *fst ' set*
 Γ
by (*simp-all add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*)
thus *trail-defined-lit* Γ L **and** *trail-defined-lit* Γ K
by (*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: *multp_{HO}* (*lit-less* (*trail-term-less* (*map* (*atm-of o fst*) Γ))) C D **and**
tr-def-D: *trail-defined-cls* Γ D **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 *trail-defined-cls* Γ C
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  $\Gamma$   $L$ 
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  $L' \in \# J$  and lit-less (trail-term-less (map (atm-of
o fst)  $\Gamma$ ))  $L L'$ 
        using * by blast
      hence (trail-term-less (map (atm-of o fst)  $\Gamma$ )) $==$  (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 $\Gamma \implies \neg (\text{trail-true-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma L)$
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 $\Gamma \implies \neg (\text{trail-true-cls } \Gamma C \wedge \text{trail-false-cls } \Gamma C)$
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 $R (Pos\ t1) (Pos\ t2) = R\ t1\ t2 \mid$
standard-lit-less $R (Pos\ t1) (Neg\ t2) = R^{==}\ t1\ t2 \mid$
standard-lit-less $R (Neg\ t1) (Pos\ t2) = R\ t1\ t2 \mid$
standard-lit-less $R (Neg\ t1) (Neg\ t2) = R\ t1\ t2$

lemma *standard-lit-less-preserves-term-less*:
shows *standard-lit-less* $R\ L1\ L2 \implies R^{==}\ (\text{atm-of } L1)\ (\text{atm-of } L2)$
by (*cases L1; cases L2*) *simp-all*

theorem *learned-clauses-in-regular-runs-invars*:

fixes Γ *lit-less*

assumes

sound-S0: *sound-state* $N\ \beta\ S0$ **and**

invars: *learned-nonempty* $S0$ *trail-propagated-or-decided'* $N\ \beta\ S0$

no-conflict-after-decide' $N\ \beta\ S0$ *almost-no-conflict-with-trail* $N\ \beta\ S0$

trail-lits-consistent $S0$ *trail-closures-false'* $S0$ *ground-false-closures* $S0$ **and**

conflict: *conflict* $N\ \beta\ S0\ S1$ **and**

resolution: (*skip* $N\ \beta \sqcup$ *factorize* $N\ \beta \sqcup$ *resolve* $N\ \beta$)⁺⁺ $S1\ Sn$ **and**

backtrack: *backtrack* $N\ \beta\ Sn\ Sn'$ **and**

lit-less-preserves-term-order: $\bigwedge R\ L1\ L2. \text{lit-less } R\ L1\ L2 \implies R^{==}\ (\text{atm-of } L1)$

(*atm-of* $L2$)

defines

$\Gamma \equiv \text{state-trail } S1$ **and**

$U \equiv \text{state-learned } S1$ **and**

trail-ord $\equiv \text{multp}_{HO}\ (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o\ \text{fst})\ \Gamma)))$

shows $(\exists C\ \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \text{'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin (\text{fset } N \cup \text{fset } U) \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U)\ C)$

proof –

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*: *state-conflict* $S1 = \text{Some } (C1,$

```

γ1)
  by (smt (verit, best) conflict.simps state-conflict-simp)
  with backtrack obtain Cn γn where conflict-Sn: state-conflict Sn = Some (Cn,
γn) and Cn ≠ {#}
  by (auto elim: backtrack.cases)
  with resolution conflict-S1 have C1 ≠ {#}
  proof (induction Sn arbitrary: C1 γ1 Cn γn 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 γy where
    conf-y: state-conflict y = Some (Cy, γy) and Cy ≠ {#}
  by (auto elim: skip.cases factorize.cases resolve.cases)
  show ?case
    using step.IH[OF - conf-y ⟨Cy ≠ {#}⟩] step.prem.s
    by simp
qed

from conflict have sound-S1: sound-state N β S1 and ground-false-closures S1
  using sound-S0 ⟨ground-false-closures S0⟩
  by (simp-all add: conflict-preserves-sound-state conflict-preserves-ground-false-closures)
with resolution have sound-Sn: sound-state N β 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  $\models_{Ge}$  {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) ∧

```

```

  (∃ Cn γn. state-conflict Sn = Some (Cn, γn) ∧ trail-false-cls (state-trail S1)
(Cn · γn))
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.premis 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  $\langle \text{suffix } (state\text{-trail } Sm') (state\text{-trail } S1) \rangle$ 
    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
     $\gamma m\text{-def: } \gamma m = \gamma$ 
      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  $\langle \text{suffix } (state\text{-trail } Sm') (state\text{-trail } S1) \rangle$ 
          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  $\gamma m\text{-def: } \gamma m =$ 
     $\gamma_C$ 
      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 $\langle \text{ground-false-closures } Sm \rangle$ **have**

ground-conf: *is-ground-cls* $(\text{add-mset } L \ C \cdot \gamma_C)$ **and**
ground-prop: *is-ground-cls* $(\text{add-mset } K \ D \cdot \gamma_D)$
unfolding *resolveI*(1,2) $\langle \Gamma = \text{trail-propagate } \Gamma' \ K \ D \ \gamma_D \rangle$
by (*simp-all add: ground-false-closures-def 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 *is-ingu* $\mu \{ \{ \text{atm-of } L \cdot a \ \varrho_C, \text{atm-of } K \cdot a \ \varrho_D \} \}$
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*)

moreover have *trail-false-cls* (*state-trail* $S1$) $(D \cdot \gamma_D)$
proof (*rule trail-false-cls-if-trail-false-suffix*)
show *suffix* $\Gamma' (state-trail \ S1)$
using *resolveI* $\langle \text{suffix } (state-trail \ Sm') (state-trail \ S1) \rangle$
by (*metis (no-types, opaque-lifting) state-trail-simp suffix-order.trans suffix-trail-propagate*)

next

from $\langle \text{trail-closures-false}' \ Sm \rangle$ **have** *trail-closures-false* Γ
unfolding *resolveI*(1,2)
by (*simp add: trail-closures-false'-def*)
thus *trail-false-cls* $\Gamma' (D \cdot \gamma_D)$
using *resolveI*(3-)
by (*auto simp: propagate-lit-def elim: trail-closures-false.cases*)

qed

ultimately have *trail-false-cls* (*state-trail* $S1$) $((C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma)$
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-γn: trail-false-cls (state-trail S1) (Cn · γn)
  by auto

from ‹ground-false-closures Sn› conflict-Sn have Cn-γn-in: Cn · γn ∈ 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-preserves-trail-lits-consistent[OF conflict ‹trail-lits-consistent S0›]
  by (simp add: trail-lits-consistent-def)

have ∀ L ∈ #Cn · γn. trail-defined-lit (state-trail S1) L
  using tr-false-S1-Cn-γn trail-defined-lit-iff-true-or-false trail-false-cls-def by
blast
  hence trail-interp (state-trail S1) ⊨ Cn · γn ↔ trail-true-cls (state-trail S1)
(Cn · γn)
  using tr-consistent-S1 trail-true-cls-iff-trail-interp-entails by auto
  hence not-trail-S1-entails-Cn-γn: ¬ trail-interp (state-trail S1) ⊨s {Cn · γn}
  using tr-false-S1-Cn-γn not-trail-true-and-false-cls[OF tr-consistent-S1] by
auto

have trail-defined-cls (state-trail S1) (Cn · γn)
  using ‹∀ L ∈ #Cn · γn. trail-defined-lit (state-trail S1) L› trail-defined-cls-def
by blast

have {#} |≠| N
  by (rule mempty-not-in-initial-clauses-if-non-empty-regular-conflict[OF con-
flict-S1 ‹C1 ≠ {#}›
  ‹almost-no-conflict-with-trail N β S1› sound-S1 ‹ground-false-closures S1›])
then obtain S where propagate N β S S0
  using before-reasonable-conflict[OF conflict ‹learned-nonempty S0›
  ‹trail-propagated-or-decided' N β S0› ‹no-conflict-after-decide' N β S0›]
  by auto

```



```

have state-learned  $S = \text{state-learned } S0$ 
  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  $S = \text{state-learned } S1$ 
  by assumption

have state-conflict  $S = \text{None}$ 
  using  $\langle \text{propagate } N \beta S S0 \rangle$  by (auto simp add: propagate.simps)

from conflict have state-trail  $S1 = \text{state-trail } S0$ 
  by (smt (verit) conflict.cases state-trail-simp)

obtain  $L C \gamma$  where trail-S0-eq:  $\text{state-trail } S0 = \text{trail-propagate } (\text{state-trail } S)$ 
 $L C \gamma$ 
  using  $\langle \text{propagate } N \beta S S0 \rangle$  unfolding propagate.simps by auto

with backtrack have strict-suffix ( $\text{state-trail } Sn$ ) ( $\text{state-trail } S0$ )
  using conflict-with-literal-gets-resolved[OF - conflict resolution][THEN tran-
clp-into-rtranclp] -
   $\langle \{\#\} \mid \notin N \rangle \langle \text{learned-nonempty } S0 \rangle \langle \text{trail-propagated-or-decided}' N \beta S0 \rangle$ 
   $\langle \text{no-conflict-after-decide}' N \beta S0 \rangle$ 
  by (metis (no-types, lifting) propagate-lit-def)
hence suffix ( $\text{state-trail } Sn$ ) ( $\text{state-trail } S$ )
  unfolding trail-S0-eq propagate-lit-def
  by (metis suffix-Cons suffix-order.le-less suffix-order.less-irrefl)

moreover have  $\neg \text{trail-defined-lit } (\text{state-trail } S) (L \cdot l \gamma)$ 
proof -
  have trail-consistent ( $\text{state-trail } S0$ )
    using  $\langle \text{state-trail } S1 = \text{state-trail } S0 \rangle \langle \text{trail-consistent } (\text{state-trail } S1) \rangle$  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  $\neg \text{trail-defined-lit } (\text{state-trail } Sn) (L \cdot l \gamma)$ 
  by (metis trail-defined-lit-def trail-false-lit-def trail-false-lit-if-trail-false-suffix
uminus-of-uminus-id)

moreover have trail-false-cls ( $\text{state-trail } Sn$ ) ( $Cn \cdot \gamma n$ )
  using  $\langle \text{ground-false-closures } Sn \rangle$  conflict-Sn by (auto simp add: ground-false-closures-def)

ultimately have  $L \cdot l \gamma \notin \# Cn \cdot \gamma n \wedge \neg (L \cdot l \gamma) \notin \# Cn \cdot \gamma n$ 
  unfolding trail-false-cls-def trail-false-lit-def trail-defined-lit-def
  by (metis uminus-of-uminus-id)

have no-conf-at-S:  $\nexists S'. \text{conflict } N \beta S S'$ 

```

```

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:  $\exists S'. \text{conflict } N \beta S S'$ 
  if  $D$ -in:  $D \in \text{grounding-of-clss } (fset N \cup fset U)$  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-clss (fset  $N \cup fset U$ )) ( $Cn \cdot \gamma n$ )
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  $\neg$  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:  $D \in \text{gnds-le-}Cn$ - $\gamma n$  and  $\neg$  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- $Cn$ - $\gamma n$ : trail-ord  $D (Cn \cdot \gamma n)$ 
  unfolding gnds-le- $Cn$ - $\gamma n$ -def by simp-all

```

from $D\text{-le-}Cn\text{-}\gamma n$ **have** $trail\text{-}defined\text{-}cls$ ($state\text{-}trail$ $S1$) D
using $\langle trail\text{-}defined\text{-}cls$ ($state\text{-}trail$ $S1$) ($Cn \cdot \gamma n$) \rangle
using $trail\text{-}defined\text{-}cls\text{-}if\text{-}lt\text{-}defined$
using $\Gamma\text{-}def$ $lit\text{-}less\text{-}preserves\text{-}term\text{-}order$ $tr\text{-}consistent\text{-}S1$ $trail\text{-}ord\text{-}def$
by $metis$
hence $trail\text{-}false\text{-}cls$ ($state\text{-}trail$ $S1$) D
using $\langle \neg$ $trail\text{-}interp$ ($state\text{-}trail$ $S1$) $\models D$ \rangle
using $\langle trail\text{-}consistent$ ($state\text{-}trail$ $S1$) \rangle $trail\text{-}interp\text{-}cls\text{-}if\text{-}trail\text{-}true$
 $trail\text{-}true\text{-}or\text{-}false\text{-}cls\text{-}if\text{-}defined$ **by** $blast$

have $L \cdot l \gamma \notin\# D \wedge \neg (L \cdot l \gamma) \notin\# D$
proof $-$
from $D\text{-le-}Cn\text{-}\gamma n$ **have** $D\text{-lt-}Cn\text{-}\gamma n'$:
 $multp_{HO}$ ($lit\text{-}less$ ($trail\text{-}term\text{-}less$ (map ($atm\text{-}of \circ fst$) ($state\text{-}trail$ $S1$)))) D
 $(Cn \cdot \gamma n)$
unfolding $trail\text{-}ord\text{-}def$ $\Gamma\text{-}def$.

have $\forall K \in\# Cn \cdot \gamma n. \neg K \in fst \text{ ' set } (state\text{-}trail S1)$
by ($rule$ $\langle trail\text{-}false\text{-}cls$ ($state\text{-}trail$ $S1$) ($Cn \cdot \gamma n$) \rangle [$unfolded$ $trail\text{-}false\text{-}cls\text{-}def$
 $trail\text{-}false\text{-}lit\text{-}def$])
hence $\forall K \in\# Cn \cdot \gamma n. \neg K \in insert$ ($L \cdot l \gamma$) ($fst \text{ ' set } (state\text{-}trail S)$)
unfolding $\langle state\text{-}trail S1 = state\text{-}trail S0$ \rangle
 $\langle state\text{-}trail S0 = trail\text{-}propagate$ ($state\text{-}trail$ S) $L C \gamma$ \rangle
 $propagate\text{-}lit\text{-}def$ $list.set$ $image\text{-}insert$ $prod.sel$
by $simp$
hence $*$: $\forall K \in\# Cn \cdot \gamma n. \neg K \in fst \text{ ' set } (state\text{-}trail S)$
by ($metis$ $\langle L \cdot l \gamma \notin\# Cn \cdot \gamma n \wedge \neg (L \cdot l \gamma) \notin\# Cn \cdot \gamma n \rangle$ $insert\text{-}iff$
 $uminus\text{-}lit\text{-}swap$)

have $**$: $\forall K \in\# Cn \cdot \gamma n. trail\text{-}term\text{-}less$ (map ($atm\text{-}of \circ fst$) ($state\text{-}trail S1$))
 $(atm\text{-}of K)$ ($atm\text{-}of (L \cdot l \gamma)$)
unfolding $\langle state\text{-}trail S1 = state\text{-}trail S0$ \rangle
 $\langle state\text{-}trail S0 = trail\text{-}propagate$ ($state\text{-}trail$ S) $L C \gamma$ \rangle
 $propagate\text{-}lit\text{-}def$ $comp\text{-}def$ $prod.sel$ $list.map$
proof ($rule$ $ballI$)
fix K **assume** $K \in\# Cn \cdot \gamma n$
with $*$ **have** $\neg K \in fst \text{ ' set } (state\text{-}trail S)$
by $metis$
hence $atm\text{-}of K \in set$ (map ($\lambda x. atm\text{-}of (fst x)$) ($state\text{-}trail S$))
by ($metis$ ($no\text{-}types$, $lifting$) $atm\text{-}of\text{-}in\text{-}atm\text{-}of\text{-}set\text{-}iff\text{-}in\text{-}set\text{-}or\text{-}uminus\text{-}in\text{-}set$
 $comp\text{-}eq\text{-}dest\text{-}lhs$ $image\text{-}comp$ $image\text{-}cong$ $list.set\text{-}map$)
thus $trail\text{-}term\text{-}less$ ($atm\text{-}of (L \cdot l \gamma)$) $\# map$ ($\lambda x. atm\text{-}of (fst x)$) ($state\text{-}trail$
 S)
 $(atm\text{-}of K)$ ($atm\text{-}of (L \cdot l \gamma)$)
using $trail\text{-}term\text{-}less\text{-}Cons\text{-}if\text{-}mem$ **by** $metis$
qed

have $\neg (L \cdot l \gamma \in\# D \vee \neg (L \cdot l \gamma) \in\# D)$

```

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-γn1]
  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 · l γ) ∈ # K
          then obtain j where
            j-in: j ∈ # J and
            uminus-L-γ-lt-j: lit-less (trail-term-less (map (atm-of ∘ fst) (state-trail
S1))) (¬ (L · l γ)) j
            using ⟨∀ k ∈ #K. ∃ x ∈ #J. lit-less (trail-term-less (map (atm-of ∘ fst)
(state-trail S1))) k x⟩
            by blast

          from j-in have
            trail-term-less (map (atm-of ∘ fst) (state-trail S1)) (atm-of j) (atm-of (L
· l γ))
            using **
            by (auto simp: ⟨Cn · γn = I + J⟩)

          moreover from uminus-L-γ-lt-j have trail-term-less (map (atm-of ∘ fst)
(state-trail S1)) (atm-of (L · l γ)) (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 trail-false-cls (state-trail S1) D \rangle*

unfolding *\langle state-trail S1 = state-trail S0 \rangle*

\langle state-trail S0 = trail-propagate (state-trail S) L C γ \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- γ n-in* **by** *auto*

moreover have *Cn \cdot γ n \notin grounding-of-cls (fset N \cup fset U)*

proof –

have *is-ground-cls (Cn \cdot γ n)*

using *Cn- γ 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 *($\{D \in$ grounding-of-cls (fset N \cup fset U). trail-ord D (Cn \cdot γ n) $\}$ \models_e $\{Cn \cdot \gamma n\}$)*

using *not-gr-red-Cn- γ n*

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

thus *?thesis*

using *\langle suffix (state-trail Sn) (state-trail S) \rangle conf-at-S-if no-conf-at-S trail-Sn-false-Cn- γ n trail-false-cls-if-trail-false-suffix* **by** *blast*

qed

moreover have *set-mset (Cn \cdot γ n) \notin set-mset ‘grounding-of-cls (fset N \cup fset U)*

proof *(rule notI)*

assume *set-mset (Cn \cdot γ n) \in set-mset ‘grounding-of-cls (fset N \cup fset U)*

then obtain *D where*

D-in: D \in grounding-of-cls (fset N \cup fset U) and

set-mset-eq-D-Cn- γ n: set-mset D = set-mset (Cn \cdot γ n)

by (auto simp add: image-iff)

have $\neg \text{trail-interp } (\text{state-trail } S1) \Vdash D$
unfolding true-cls-iff-set-mset-eq[OF set-mset-eq-D-Cn- γn]
using not-trail-S1-entails-Cn- γn
by simp

have trail-defined-cls (state-trail S1) D
using $\langle \forall L \in \# Cn \cdot \gamma n. \text{trail-defined-lit } (\text{state-trail } S1) L \rangle$
unfolding set-mset-eq-D-Cn- γn [symmetric]
by (simp add: trail-defined-cls-def)
hence trail-false-cls (state-trail S1) D
using $\langle \neg \text{trail-interp } (\text{state-trail } S1) \Vdash D \rangle$
using tr-consistent-S1 trail-interp-cls-if-trail-true trail-true-or-false-cls-if-defined
by blast

have $L \cdot l \gamma \notin \# D \wedge - (L \cdot l \gamma) \notin \# D$
using $\langle L \cdot l \gamma \notin \# Cn \cdot \gamma n \wedge - (L \cdot l \gamma) \notin \# Cn \cdot \gamma n \rangle$
unfolding set-mset-eq-D-Cn- γn [symmetric]
by assumption
hence trail-false-cls (state-trail S) D
using D-in $\langle \text{trail-false-cls } (\text{state-trail } S1) D \rangle$
unfolding $\langle \text{state-trail } S1 = \text{state-trail } S0 \rangle$
 $\langle \text{state-trail } S0 = \text{trail-propagate } (\text{state-trail } S) L C \gamma \rangle$
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

moreover have $\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = Cn)$
by (metis (no-types, lifting) Cn- γn -in Set.set-insert UnCI calculation(2)
grounding-of-cls-insert grounding-of-subst-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: (regular-scl N β)** initial-state S0 and

conflict: conflict N β S0 S1 and

resolution: (skip N $\beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta$)⁺⁺ S1 Sn and

backtrack: backtrack N β Sn Sn' and

lit-less-preserves-term-order: $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{\text{==}} (\text{atm-of } L1)$
(atm-of L2)

defines

$\Gamma \equiv$ *state-trail* $S1$ **and**

$U \equiv$ *state-learned* $S1$ **and**

trail-ord \equiv *multp*_{HO} (*lit-less* (*trail-term-less* (*map* (*atm-of o fst*) Γ)))

shows (*regular-scl* $N \beta$)^{**} *initial-state* Sn' \wedge

($\exists C \gamma.$ *state-conflict* $Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin$ *grounding-of-cls* (*fset* $N \cup$ *fset* U) \wedge

set-mset ($C \cdot \gamma$) \notin *set-mset* ' *grounding-of-cls* (*fset* $N \cup$ *fset* U) \wedge

$C \notin$ *fset* $N \cup$ *fset* U \wedge

$\neg (\exists D \in$ *fset* $N \cup$ *fset* $U. \exists \sigma. D \cdot \sigma = C) \wedge$

\neg *redundant trail-ord* (*fset* $N \cup$ *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-preserves-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-preserves-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-preserves-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** $(\text{regular-scl } N \beta)^{**}$ *initial-state* $S1$
by $(\text{meson regular-scl-def rtranclp.simps})$
also from *resolution* **have** $\text{reg-run-}S1\text{-}Sn: (\text{regular-scl } N \beta)^{**} \dots Sn$
using *regular-run-if-skip-factorize-resolve-run tranclp-into-rtranclp* **by** *metis*
also have $(\text{regular-scl } N \beta)^{**} \dots Sn'$
proof $(\text{rule r-into-rtranclp})$
from *backtrack* **have** $\text{scl } N \beta Sn Sn'$
by $(\text{simp add: scl-def})$
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 $(\text{smt (verit) conflict.simps option.simps(3) backtrack.cases state-conflict-simp})$
qed
finally have $(\text{regular-scl } N \beta)^{**}$ *initial-state* Sn' **by** *assumption*
thus *?thesis*
using *learned-clauses-in-regular-runs-invars*[*OF sound-S0* $\langle \text{learned-nonempty } S0 \rangle$
 $\langle \text{trail-propagated-or-decided}' N \beta S0 \rangle$
 $\langle \text{no-conflict-after-decide}' N \beta S0 \rangle$ $\langle \text{almost-no-conflict-with-trail } N \beta S0 \rangle$
 $\langle \text{trail-lits-consistent } S0 \rangle$ $\langle \text{trail-closures-false}' S0 \rangle$ $\langle \text{ground-false-closures } S0 \rangle$
conflict resolution backtrack]
using *lit-less-preserves-term-order*
using *U-def* $\Gamma\text{-def trail-ord-def}$ **by** *presburger*
qed

theorem *dynamic-non-redundancy-strategy:*

fixes Γ

assumes

run: $(\text{strategy } N \beta)^{**}$ *initial-state* $S0$ **and**

conflict: *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: *backtrack* $N \beta Sn Sn'$ **and**

strategy-imp-regular-scl: $\bigwedge S S'. \text{strategy } N \beta S S' \implies \text{regular-scl } N \beta S S'$ **and**

lit-less-preserves-term-order: $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{==}$ (*atm-of* $L1$)

(*atm-of* $L2$)

defines

$\Gamma \equiv \text{state-trail } S1$ **and**

$U \equiv \text{state-learned } S1$ **and**

trail-ord $\equiv \text{multp}_{HO} (\text{lit-less } (\text{trail-term-less } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma)))$

shows $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$

$C \cdot \gamma \notin \text{grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } \text{'grounding-of-cls } (\text{fset } N \cup \text{fset } U) \wedge$

$C \notin \text{fset } N \cup \text{fset } U \wedge$

$\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$

$\neg \text{redundant trail-ord } (\text{fset } N \cup \text{fset } U) C$)

proof –

from *backtrack* **have** *backtrack'*: *backtrack* $N \beta Sn Sn'$

by (*simp add: shortest-backtrack-strategy-def*)

have $(\exists C \gamma. \text{state-conflict } Sn = \text{Some } (C, \gamma) \wedge$
 $C \cdot \gamma \notin \text{grounding-of-clss } (\text{fset } N \cup \text{fset } U) \wedge$
 $\text{set-mset } (C \cdot \gamma) \notin \text{set-mset } ' \text{grounding-of-clss } (\text{fset } N \cup \text{fset } U) \wedge$
 $C \notin \text{fset } N \cup \text{fset } U \wedge$
 $\neg (\exists D \in \text{fset } N \cup \text{fset } U. \exists \sigma. D \cdot \sigma = C) \wedge$
 $\neg \text{redundant } (\text{multp}_{HO} (\text{lit-less}$
 $\quad (\text{trail-term-less } (\text{map } (\text{atm-of } \circ \text{fst}) (\text{state-trail } S1))))))$
 $(\text{fset } N \cup \text{fset } U) C)$

unfolding *U-def*

proof (*rule dynamic-non-redundancy-regular-scl[THEN conjunct2]*)

from *run show* (*regular-scl* $N \beta$)** *initial-state* $S0$

by (*induction* $S0$ *rule: rtranclp-induct*) (*auto dest: strategy-imp-regular-scl*)

next

from *assms show* *conflict* $N \beta S0 S1$

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* $\bigwedge R L1 L2. \text{lit-less } R L1 L2 \implies R^{==} (\text{atm-of } L1) (\text{atm-of } L2)$

by *simp*

qed

thus *?thesis*

by (*auto simp add: trail-ord-def* Γ -*def*)

qed

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 $\exists S0 S1 S2 S3 S4. (\text{regular-scl } N \beta)$ ** *initial-state* $S0 \wedge$

propagate $N \beta S0 S1 \wedge \text{regular-scl } N \beta S0 S1 \wedge$

conflict $N \beta S1 S2 \wedge (\text{factorize } N \beta)$ ** $S2 S3 \wedge \text{resolve } N \beta S3 S4 \wedge$

$(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)$ ** $S4 S$

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-scl $N \beta$)** initial-state $S0$ **and**
propa: propagate $N \beta S0 S1$ regular-scl $N \beta S0 S1$ **and**
conft: conflict $N \beta S1 S2$ **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-scl $N \beta$)** initial-state S **and**
step: backtrack $N \beta S S'$

defines

$U \equiv$ state-learned S

shows $\exists C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \wedge \neg (\exists D \in \text{fset } N \cup \text{fset } U. \text{subsumes } D C)$

proof –

from before-regular-backtrack'[OF run step] **obtain** $S0 S1 S2 S3 S4$ **where**
run-S0: (regular-scl $N \beta$)** initial-state $S0$ **and**
propa: propagate $N \beta S0 S1$ regular-scl $N \beta S0 S1$ **and**
conft: conflict $N \beta S1 S2$ **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 $(\text{skip } N \beta \sqcup \text{factorize } N \beta \sqcup \text{resolve } N \beta)^{++} S3 S4$
using *resol* **by** *auto*
finally show *?thesis*
using *reg-res* **by** *simp*
qed

ultimately obtain $C \gamma lt$ **where**
reg-run: $(\text{regular-scl } N \beta)^{**}$ *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: $\neg \text{redundant } (\text{multp}_{HO} (\text{standard-lit-less}$
 $\quad (\text{trail-term-less } (\text{map } (\text{atm-of } \circ \text{fst}) (\text{state-trail } S2))))))$
 $(\text{fset } N \cup \text{fset } (\text{state-learned } S2)) C$
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 *metis*
qed

corollary *static-non-subsumption-strategy*:
assumes
run: $(\text{strategy } N \beta)^{**}$ *initial-state* S **and**
step: *backtrack* $N \beta S S'$ **and**
strategy-imp-regular-scl: $\bigwedge S S'. \text{strategy } N \beta S S' \implies \text{regular-scl } N \beta S S'$

```

defines
   $U \equiv \text{state-learned } S$ 
  shows  $\exists C \gamma. \text{state-conflict } S = \text{Some } (C, \gamma) \wedge \neg (\exists D \in \text{fset } N \cup \text{fset } U. \text{subsumes } D C)$ 
  unfolding  $U\text{-def}$ 
proof (rule static-non-subsumption-regular-scl)
  from run show (regular-scl  $N \beta$ )** initial-state  $S$ 
  by (induction  $S$  rule: rtranclp-induct)
  (auto intro: rtranclp.rtrancl-into-rtrancl strategy-imp-regular-scl)
next
  from step show backtrack  $N \beta S S'$ 
  by simp
qed

end

end
theory Wellfounded-Extra
imports
  Main
  Ordered-Resolution-Prover.Lazy-List-Chain
begin

definition wf-on ::  $'a \text{ set} \Rightarrow 'a \text{ rel} \Rightarrow \text{bool}$ 
  where  $\text{wf-on } A r \iff (\forall P. (\forall x \in A. (\forall y \in A. (y, x) \in r \longrightarrow P y) \longrightarrow P x) \longrightarrow (\forall x \in A. P x))$ 

abbreviation wf ::  $'a \text{ rel} \Rightarrow \text{bool}$  where
   $\text{wf} \equiv \text{wf-on } \text{UNIV}$ 

definition wfp-on ::  $'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$  where
   $\text{wfp-on } A R \iff (\forall P. (\forall x \in A. (\forall y \in A. R y x \longrightarrow P y) \longrightarrow P x) \longrightarrow (\forall x \in A. P x))$ 

abbreviation wfp ::  $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{bool}$  where
   $\text{wfp} \equiv \text{wfp-on } \text{UNIV}$ 

lemma wf-def[no-atp]:  $\text{wf } r \iff (\forall P. (\forall x. (\forall y. (y, x) \in r \longrightarrow P y) \longrightarrow P x) \longrightarrow (\forall x. P x))$ 
  by (simp add: wf-on-def)

lemma wf-onI:
   $(\bigwedge P x. (\bigwedge y. y \in A \implies (\bigwedge z. z \in A \implies (z, y) \in r \implies P z) \implies P y) \implies x \in A \implies P x) \implies \text{wf-on } A r$ 
  unfolding wf-on-def by metis

lemma wfI:  $(\bigwedge P x. (\bigwedge y. (\bigwedge z. (z, y) \in r \implies P z) \implies P y) \implies P x) \implies \text{wf } r$ 
  by (auto simp: wf-on-def)

```

lemma *wf-on-induct*[*consumes 1, case-names less in-dom*]:

assumes

wf-on A r **and**

$\bigwedge x. x \in A \implies (\bigwedge y. y \in A \implies (y, x) \in r \implies P y) \implies P x$ **and**
 $x \in A$

shows $P x$

using *assms unfolding wf-on-def by metis*

lemma *wf-on UNIV r* \longleftrightarrow *Wellfounded.wf r*

by (*simp add: wf-on-def Wellfounded.wf-def*)

lemma *wfp-iff-wfP: wfp R* \longleftrightarrow *Wellfounded.wfP R*

by (*metis (no-types, opaque-lifting) UNIV-I wfpUNIVI wfP-induct-rule wfp-on-def*)

lemma *wfp-on-wf-on-iff*[*pred-set-conv*]: *wfp-on A* $(\lambda x y. (x, y) \in r) \longleftrightarrow$ *wf-on A*
r

by (*simp add: wfp-on-def wf-on-def*)

16.1 Basic Results

Point-free characterization of well-foundedness

lemma *wf-onE-pf*:

assumes *wf: wf-on A r* **and** $B \subseteq A$ **and** $B \subseteq r$ “ B

shows $B = \{\}$

proof –

from *wf* **have** $x \notin B$ **if** $x \in A$ **for** x

proof (*induction x rule: wf-on-induct*)

case (*less x*)

then show *?case*

by (*metis ImageE assms(2) assms(3) in-mono*)

next

case *in-dom*

show *?case*

using *that* .

qed

with $\langle B \subseteq A \rangle$ **show** *?thesis*

by *blast*

qed

lemma *wfE-pf*: *wf R* $\implies A \subseteq R$ “ $A \implies A = \{\}$

by (*auto elim!: wf-onE-pf*)

lemma *wf-onI-pf*:

assumes $\bigwedge B. B \subseteq A \implies B \subseteq R$ “ $B \implies B = \{\}$

shows *wf-on A R*

unfolding *wf-on-def*

proof (*intro allI impI ballI*)

fix $P :: 'a \Rightarrow \text{bool}$ **and** $x :: 'a$

let $?B = \{x \in A. \neg P x\}$
assume $\forall x \in A. (\forall y \in A. (y, x) \in R \longrightarrow P y) \longrightarrow P x$
hence $?B \subseteq R$ “ $?B$ **by** *blast*
with *assms(1)[of ?B]* **have** $\{x \in A. \neg P x\} = \{\}$
 by *simp*
moreover assume $x \in A$
ultimately show $P x$
 by *simp*
qed

lemma *wfI-pf*: $(\bigwedge A. A \subseteq R \text{ “ } A \Longrightarrow A = \{\}) \Longrightarrow \text{wf } R$
 by (*auto intro!*: *wf-onI-pf*)

16.1.1 Minimal-element characterization of well-foundedness

lemma *minimal-if-wf-on*:
assumes *wf*: $\text{wf-on } A \ R$ **and** $B \subseteq A$ **and** $B \neq \{\}$
shows $\exists z \in B. \forall y. (y, z) \in R \longrightarrow y \notin B$
using *wf-onE-pf[OF wf ‹B ⊆ A›]*
by (*metis Image-iff assms(3) subsetI*)

lemma *wfE-min*:
assumes *wf*: $\text{wf } R$ **and** $Q: x \in Q$
obtains z **where** $z \in Q \wedge \forall y. (y, z) \in R \Longrightarrow y \notin Q$
using Q *wfE-pf[OF wf, of Q]* **by** *blast*

lemma *wfE-min'*:
 $\text{wf } R \Longrightarrow Q \neq \{\} \Longrightarrow (\bigwedge z. z \in Q \Longrightarrow (\bigwedge y. (y, z) \in R \Longrightarrow y \notin Q) \Longrightarrow \text{thesis})$
 $\Longrightarrow \text{thesis}$
using *wfE-min[of R - Q]* **by** *blast*

lemma *wf-on-if-minimal*:
assumes $\bigwedge B. B \subseteq A \Longrightarrow B \neq \{\} \Longrightarrow \exists z \in B. \forall y. (y, z) \in R \longrightarrow y \notin B$
shows $\text{wf-on } A \ R$
proof (*rule wf-onI-pf*)
 fix B
 show $B \subseteq A \Longrightarrow B \subseteq R$ “ $B \Longrightarrow B = \{\}$
 using *assms* **by** (*metis ImageE subset-eq*)
qed

lemma *wfI-min*:
assumes $a: \bigwedge x \in Q. x \in Q \Longrightarrow \exists z \in Q. \forall y. (y, z) \in R \longrightarrow y \notin Q$
shows $\text{wf } R$
proof (*rule wfI-pf*)
 fix A
 assume $b: A \subseteq R$ “ A
 have *False* **if** $x \in A$ **for** x
 using a [*OF that*] b **by** *blast*
 then show $A = \{\}$ **by** *blast*

qed

lemma *wf-on-iff-minimal*: $wf\text{-on } A \ r \longleftrightarrow (\forall B \subseteq A. B \neq \{\} \longrightarrow (\exists z \in B. \forall y. (y, z) \in r \longrightarrow y \notin B))$
using *minimal-if-wf-on wf-on-if-minimal* by *metis*

lemma *wf-iff-minimal*: $wf \ r \longleftrightarrow (\forall B. B \neq \{\} \longrightarrow (\exists z \in B. \forall y. (y, z) \in r \longrightarrow y \notin B))$
using *wf-on-iff-minimal[of UNIV, simplified]* by *metis*

lemma *wf-eq-minimal[no-atp]*: $wf \ r \longleftrightarrow (\forall Q \ x. x \in Q \longrightarrow (\exists z \in Q. \forall y. (y, z) \in r \longrightarrow y \notin Q))$
unfolding *wf-iff-minimal* by *auto*

lemma *wfp-on-iff-minimal*: $wfp\text{-on } A \ R \longleftrightarrow (\forall B \subseteq A. B \neq \{\} \longrightarrow (\exists z \in B. \forall y. R \ y \ z \longrightarrow y \notin B))$
using *wf-on-iff-minimal[of A, to-pred]* by *simp*

lemma *wfp-iff-minimal*: $wfp \ R \longleftrightarrow (\forall B. B \neq \{\} \longrightarrow (\exists z \in B. \forall y. R \ y \ z \longrightarrow y \notin B))$
by (*simp add: wfp-on-iff-minimal*)

lemmas *wfP-eq-minimal[no-atp] = wf-eq-minimal[to-pred]*

lemma *ex-trans-min-element-if-wf-on*:
assumes *wf*: $wf\text{-on } A \ r$ and *x-in*: $x \in A$
shows $\exists y \in A. (y, x) \in r^* \wedge \neg(\exists z \in A. (z, y) \in r)$
using *wf*
proof (*induction x rule: wf-on-induct*)
case (*less x*)
thus ?*case*
by (*metis rtrancl.rtrancl-into-rtrancl rtrancl.rtrancl-refl*)
next
case *in-dom*
thus ?*case*
using *x-in* by *metis*
qed

lemma *ex-trans-min-element-if-wfp-on*: $wfp\text{-on } A \ R \Longrightarrow x \in A \Longrightarrow \exists y \in A. R^{**} \ y \ x \wedge \neg(\exists z \in A. R \ z \ y)$
by (*rule ex-trans-min-element-if-wf-on[to-pred]*)

16.2 Bound Restriction and Monotonicity

lemma *wf-on-subset*: $wf\text{-on } A \ r \Longrightarrow B \subseteq A \Longrightarrow wf\text{-on } B \ r$
unfolding *wf-on-iff-minimal*
by (*meson order-trans*)

lemma *wfp-on-subset*: $wfp\text{-on } A R \implies B \subseteq A \implies wfp\text{-on } B R$
by (*rule wfp-on-subset[of A - B, to-pred]*)

lemma *wf-on-mono-strong*:
 $wfp\text{-on } A r \implies (\bigwedge x y. x \in A \implies y \in A \implies (x, y) \in q \implies (x, y) \in r) \implies wfp\text{-on } A q$
unfolding *wf-on-iff-minimal*
by (*meson in-mono*)

lemma *wfp-on-mono-strong*:
 $wfp\text{-on } A R \implies (\bigwedge x y. x \in A \implies y \in A \implies Q x y \implies R x y) \implies wfp\text{-on } A Q$
by (*rule wfp-on-mono-strong[to-pred]*)

Well-foundedness of subsets

lemma *wf-subset*: $wf r \implies p \subseteq r \implies wf p$
by (*simp add: wf-eq-minimal*) *fast*

lemmas *wfP-subset = wf-subset* [*to-pred*]

Well-foundedness of the empty relation

lemma *wf-empty [iff]*: $wf \{\}$
by (*simp add: wf-def*)

lemma *wfP-empty [iff]*: $wfp (\lambda x y. False)$

proof –

have *wfp bot*
by (*simp add: wfp-on-def*)
then show *?thesis*
by (*simp add: bot-fun-def*)

qed

lemma *wf-Int1*: $wf r \implies wf (r \cap r')$
by (*erule wf-subset*) (*rule Int-lower1*)

lemma *wf-Int2*: $wf r \implies wf (r' \cap r)$
by (*erule wf-subset*) (*rule Int-lower2*)

definition *inv-imagep-on* :: $'a \text{ set} \Rightarrow ('b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ **where**

$inv\text{-imagep-on } A R f = (\lambda x y. x \in A \wedge y \in A \wedge R (f x) (f y))$

lemma *wfp-on-inv-imagep*:
assumes $wf: wfp\text{-on } (f ' A) R$
shows $wfp\text{-on } A (inv\text{-imagep } R f)$
unfolding *wfp-on-iff-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 $wf[unfolded wfp-on-iff-minimal, rule-format, of f ' B]$ **by** *blast*

thus $\exists z \in B. \forall y. \text{inv-imagep } R \text{ f } y \ z \longrightarrow y \notin B$
unfolding *inv-imagep-def*
by (*metis image-iff*)
qed

lemma *wfp-on-if-convertible-to-wfp*:

assumes
wf: *wfp-on* (*f* ' *A*) *Q* **and**
convertible: $(\bigwedge x \ y. x \in A \implies y \in A \implies R \ x \ y \implies Q \ (f \ x) \ (f \ y))$
shows *wfp-on* *A* *R*
unfolding *wfp-on-iff-minimal*
proof (*intro allI impI*)
fix *B* **assume** $B \subseteq A$ **and** $B \neq \{\}$
moreover from *wf* **have** *wfp-on* *A* (*inv-imagep* *Q* *f*)
by (*rule wfp-on-inv-imagep*)
ultimately obtain *y* **where** $y \in B$ **and** $\bigwedge z. Q \ (f \ z) \ (f \ y) \implies z \notin B$
unfolding *wfp-on-iff-minimal in-inv-imagep* **by** *metis*
thus $\exists z \in B. \forall y. R \ y \ z \longrightarrow y \notin B$
using $\langle B \subseteq A \rangle$ *convertible* **by** *blast*
qed

definition *lex-prodp* **where**

lex-prodp $R_A \ R_B \ x \ y \longleftrightarrow R_A \ (fst \ x) \ (fst \ y) \vee fst \ x = fst \ y \wedge R_B \ (snd \ x) \ (snd \ y)$

lemma *lex-prodp-lex-prod-iff*[*pred-set-conv*]:

lex-prodp $R_A \ R_B \ x \ y \longleftrightarrow (x, y) \in \text{lex-prod } \{(x, y). R_A \ x \ y\} \{(x, y). R_B \ x \ y\}$
unfolding *lex-prodp-def lex-prod-def* **by** *auto*

lemma *lex-prod-lex-prodp-iff*:

lex-prod $\{(x, y). R_A \ x \ y\} \{(x, y). R_B \ x \ y\} = \{(x, y). \text{lex-prodp } R_A \ R_B \ x \ y\}$
unfolding *lex-prodp-def lex-prod-def* **by** *auto*

lemma *wf-on-lex-prod*:

assumes *wfA*: *wf-on* *A* r_A **and** *wfB*: *wf-on* *B* r_B
shows *wf-on* $(A \times B)$ $(r_A \langle *lex* \rangle r_B)$
unfolding *wf-on-iff-minimal*
proof (*intro allI impI*)
fix *AB* **assume** $AB \subseteq A \times B$ **and** $AB \neq \{\}$
hence *fst* ' $AB \subseteq A$ **and** *snd* ' $AB \subseteq B$
by *auto*

from $\langle \text{fst} \ ' \ AB \subseteq A \rangle \langle AB \neq \{\} \rangle$ **obtain** *a* **where**

a-in: $a \in \text{fst} \ ' \ AB$ **and**
a-minimal: $(\forall y. (y, a) \in r_A \longrightarrow y \notin \text{fst} \ ' \ AB)$
using *wfA*[*unfolded wf-on-iff-minimal, rule-format, of fst ' AB*]
by *auto*

from $\langle \text{snd} \ ' \ AB \subseteq B \rangle \langle AB \neq \{\} \rangle$ *a-in* **obtain** *b* **where**

b-in: $b \in \text{snd} \ ' \ \{p \in AB. \text{fst } p = a\}$ **and**

b-minimal: $(\forall y. (y, b) \in r_B \longrightarrow y \notin \text{snd } \{p \in AB. \text{fst } p = a\})$
using *wfB[unfolded wf-on-iff-minimal, rule-format, of snd ' {p ∈ AB. fst p = a}]*
by *blast*

show $\exists z \in AB. \forall y. (y, z) \in r_A \langle *lex* \rangle r_B \longrightarrow y \notin AB$
proof (*rule bezI*)
show $(a, b) \in AB$
using *b-in by fastforce*
next
show $\forall y. (y, (a, b)) \in r_A \langle *lex* \rangle r_B \longrightarrow y \notin AB$
proof (*intro allI impI*)
fix *p* **assume** $(p, (a, b)) \in r_A \langle *lex* \rangle r_B$
hence $(\text{fst } p, a) \in r_A \vee \text{fst } p = a \wedge (\text{snd } p, b) \in r_B$
unfolding *lex-prod-def* **by** *auto*
then show $p \notin AB$
proof (*elim disjE conjE*)
assume $(\text{fst } p, a) \in r_A$
hence $\text{fst } p \notin \text{fst } \{p \in AB. \text{fst } p = a\}$
using *a-minimal by simp*
thus *?thesis*
by *auto*
next
assume $\text{fst } p = a$ **and** $(\text{snd } p, b) \in r_B$
hence $\text{snd } p \notin \text{snd } \{p \in AB. \text{fst } p = a\}$
using *b-minimal by simp*
thus $p \notin AB$
using $\langle \text{fst } p = a \rangle$ **by** *auto*
qed
qed
qed
qed

lemma *wfp-on-lex-prodp*: $wfp\text{-on } A \ R_A \Longrightarrow wfp\text{-on } B \ R_B \Longrightarrow wfp\text{-on } (A \times B)$
 $(lex\text{-prodp } R_A \ R_B)$
by (*rule wf-on-lex-prod[to-pred, unfolded lex-prod-lex-prodp-iff, to-pred]*)

corollary *wfp-lex-prodp*: $wfp \ R_A \Longrightarrow wfp \ R_B \Longrightarrow wfp \ (lex\text{-prodp } R_A \ R_B)$
by (*rule wfp-on-lex-prodp[of UNIV - UNIV, unfolded UNIV-Times-UNIV]*)

lemma *wfp-on-sup-if-convertible-to-wfp*:
includes *lattice-syntax*
assumes
wf-S: $wfp\text{-on } A \ S$ **and**
wf-Q: $wfp\text{-on } (f \ ' \ A) \ Q$ **and**
convertible-R: $\bigwedge x \ y. x \in A \Longrightarrow y \in A \Longrightarrow R \ x \ y \Longrightarrow Q \ (f \ x) \ (f \ y)$ **and**
convertible-S: $\bigwedge x \ y. x \in A \Longrightarrow y \in A \Longrightarrow S \ x \ y \Longrightarrow Q \ (f \ x) \ (f \ y) \vee f \ x = f \ y$
shows $wfp\text{-on } A \ (R \sqcup S)$
proof (*rule wfp-on-if-convertible-to-wfp*)

```

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  $x \in A \implies y \in A \implies (R \sqcup S) x y \implies \text{lex-prodp } Q S (f x, x) (f y, y)$ 
  using convertible- $R$  convertible- $S$ 
  by (auto simp add: lex-prodp-def)
qed

lemma wf-on-iff-wf: wf-on  $A r \longleftrightarrow \text{wf } \{(x, y) \in r. x \in A \wedge y \in A\}$ 
proof (rule iffI)
  assume wf: wf-on  $A r$ 
  show wf  $\{(x, y) \in r. x \in A \wedge y \in A\}$ 
  unfolding wf-def
  proof (intro allI impI ballI)
    fix  $P x$ 
    assume IH:  $\forall x. (\forall y. (y, x) \in \{(x, y). (x, y) \in r \wedge x \in A \wedge y \in A\} \longrightarrow P y)$ 
     $\longrightarrow P x$ 
    show  $P x$ 
    proof (cases  $x \in A$ )
      case True
      from wf show ?thesis
      proof (induction  $x$  rule: wf-on-induct)
        case (less  $x$ )
        then show ?case
        by (auto intro: IH[rule-format])
      next
      case in-dom
      show ?case
      using True .
    qed
  next
  case False
  then show ?thesis
  by (auto intro: IH[rule-format])
  qed
qed
next
assume wf: wf  $\{(x, y). (x, y) \in r \wedge x \in A \wedge y \in A\}$ 
show wf-on  $A r$ 
  unfolding wf-on-def
  proof (intro allI impI ballI)
    fix  $P x$ 

```

```

assume IH:  $\forall x \in A. (\forall y \in A. (y, x) \in r \longrightarrow P y) \longrightarrow P x$  and  $x \in A$ 
show  $P x$ 
  using wf  $\langle x \in A \rangle$ 
proof (induction x rule: wf-on-induct)
  case (less y)
  hence  $\bigwedge z. (z, y) \in r \implies z \in A \implies P z$ 
    by simp
  thus ?case
    using IH[rule-format, OF  $\langle y \in A \rangle$ ] by simp
next
  case in-dom
  show ?case
    by simp
  qed
qed
qed

lemma wfp-on-iff-wfp:  $wfp\text{-on } A R \longleftrightarrow wfp (\lambda x y. R x y \wedge x \in A \wedge y \in A)$ 
by (smt (verit, del-insts) UNIV-I subsetI wfp-on-def wfp-on-mono-strong wfp-on-subset)

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  $xs :: 'a\ list$ 
assumes Lazy-List-Chain.chain  $(\lambda x y. R x y) (LCons\ init\ xs)$ 
shows Lazy-List-Chain.chain  $(\lambda x y. R x y \wedge R^{**}\ init\ x) (LCons\ init\ xs)$ 
proof (intro lnth-rel-chain allI impI conjI)
  show  $\neg lnull (LCons\ init\ xs)$ 
    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:  $R^{**}$   $x_0$   $x$ 
  shows  $\exists xs.$  lfinite xs  $\wedge$  chain ( $\lambda y z.$   $R$   $y$   $z \wedge R^{**}$   $x_0$   $y$ ) (LCons  $x_0$  xs)  $\wedge$  llast
  (LCons  $x_0$  xs) =  $x$ 
  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 ( $\lambda y z.$   $R$   $y$   $z \wedge R^{**}$   $x_0$   $y$ ) (LCons  $x_0$  xs) and llast (LCons
   $x_0$  xs) =  $y$ 
  by auto
  let ?xs = lappend xs (LCons  $z$  LNil)
  show ?case
  proof (intro exI conjI)
    show lfinite ?xs
    using  $\langle$ lfinite xs $\rangle$  by simp
  next
    show chain ( $\lambda y z.$   $R$   $y$   $z \wedge R^{**}$   $x_0$   $y$ ) (LCons  $x_0$  ?xs)
    using  $\langle$ chain ( $\lambda y z.$   $R$   $y$   $z \wedge R^{**}$   $x_0$   $y$ ) (LCons  $x_0$  xs) $\rangle$   $\langle$ llast (LCons  $x_0$  xs)
  =  $y$  $\rangle$ 
    chain.chain-singleton chain-lappend step.hyps(1) step.hyps(2)
    by fastforce
  next
    show llast (LCons  $x_0$  ?xs) =  $z$ 
    by (simp add:  $\langle$ lfinite xs $\rangle$  llast-LCons)
  qed
qed

lemma chain-conj-rtranclpD:
  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** x0 x
  by auto
thus ?thesis
proof (cases R x0 x rule: rtrancl.cases)
  case rtrancl-refl
  then show ?thesis
    using chain local.chain-cons(1) by force
next
case (rtrancl-into-rtrancl xn)
then obtain ys where
  lfin-ys: lfinite ys and
  chain-ys: chain (λy z. R y z ∧ R** x0 y) (LCons x0 ys) and
  llast-ys: llast (LCons x0 ys) = xn
  by (auto dest: rtrancl-implies-ex-lfinite-chain)
show ?thesis
proof (intro exI conjI)
  show lfinite (LCons x0 ys)
    using lfin-ys
    by simp
next
have R (llast (LCons x0 ys)) (lhd xs)
  using rtrancl-into-rtrancl(2) chain-cons(1) llast-ys
  by simp
moreover have R** x0 (llast (LCons x0 ys))
  using rtrancl-into-rtrancl(1,2)
  using lappend-code(2)[of x0 ys xs]
  lhd-LCons[of x0 (lappend ys xs)] local.chain-cons(1)
  using llast-ys
  by fastforce
ultimately show chain (λy z. R y z ∧ R** x0 y) (lappend (LCons x0 ys) xs)
  using chain-lappend[OF chain-ys chain]
  by metis
next
show lhd (lappend (LCons x0 ys) xs) = x0
  by simp
qed
qed
qed

```

lemma *wfp-on-rtrancl-conversep-iff-no-infinite-down-chain-llist:*

```

  fixes R x0
  shows wfp-on {x. R** x0 x} R-1-1 ↔ (∄ xs. ¬ lfinite xs ∧ Lazy-List-Chain.chain
R (LCons x0 xs))
proof (rule iffI)
  assume wfp-on {x. R** x0 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 $Wellfounded.wfP (\lambda z y. R y z \wedge R^{**} x_0 y)$
by (*simp add: wfp-iff-wfP*)
hence $\nexists xs. \neg lfinite xs \wedge Lazy-List-Chain.chain (\lambda y z. R y z \wedge R^{**} x_0 y) xs$
unfolding *wfP-iff-no-infinite-down-chain-llist*
by (*metis (no-types, lifting) Lazy-List-Chain.chain-mono conversepI*)
hence $\nexists xs. \neg lfinite xs \wedge Lazy-List-Chain.chain (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons$
 $x_0 xs)$
by (*meson lfinite-LCons*)
thus $\nexists xs. \neg lfinite xs \wedge Lazy-List-Chain.chain R (LCons x_0 xs)$
using *chain-conj-rtranclpI*
by *fastforce*
next
assume $\nexists xs. \neg lfinite xs \wedge Lazy-List-Chain.chain R (LCons x_0 xs)$
hence *no-inf-chain: $\nexists xs. \neg lfinite xs \wedge chain (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons$*
 $x_0 xs)$
by (*metis (mono-tags, lifting) Lazy-List-Chain.chain-mono*)
have $\nexists xs. \neg lfinite xs \wedge chain (\lambda y z. R y z \wedge R^{**} x_0 y) xs$
proof (*rule notI, elim exE conjE*)
fix xs **assume** $\neg lfinite xs$ **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 $\exists xs. \neg lfinite xs \wedge chain (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons x_0 xs)$
proof (*intro exI conjI*)
show $\neg lfinite (ltl (lappend ys xs))$
using $\langle \neg lfinite xs \rangle lfinite-lappend lfinite-ltl$
by *blast*
next
show $chain (\lambda y z. R y z \wedge R^{**} x_0 y) (LCons x_0 (ltl (lappend ys xs)))$
using $\langle chain (\lambda y z. R y z \wedge R^{**} x_0 y) (lappend ys xs) \rangle \langle lhd (lappend ys xs) \rangle$
 $= x_0 \rangle$
 $chain-not-lnull lhd-LCons-ltl$
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 $wfp (\lambda z y. R y z \wedge y \in \{x. R^{**} x_0 x\})$
unfolding *wfp-iff-wfP by argo*
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** x0 x} R-1-1
  unfolding wfp-on-iff-wfp[of {x. R** x0 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 ⊂ B and B ⊆ A
  shows (A − B ⊂ A − C)
proof (rule Set.psubsetI)
  show A − B ⊆ A − C
    using assms(1) by blast
next
  show A − B ≠ A − C
    using assms by blast
qed

```

```

lemma wfP-pfsubset: wfP (|C|)
proof (rule wfP-if-convertible-to-nat)
  show  $\bigwedge x y. x \text{ |C| } y \implies \text{fcard } x < \text{fcard } y$ 
    by (rule pfsubset-fcard-mono)
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 \times B) (lex-prodp RA RB)*

proof (*rule totalp-onI*)

fix *x y* **assume** *x \in A \times B and y \in A \times B and x \neq y*

then show *lex-prodp RA RB x y \vee 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 Wellfounded Extra

17.4 FSet Extra

lemma *finset-Abs-fset*: *finite A \implies finset a (Abs-fset A) = Abs-fset (insert a A)*

by (*simp add: eq-onp-same-args finset.abs-eq*)

lemma *minus-pfssubset-minusI*:

assumes *C $|<$ B and B $|<=$ A*

shows *(A $|<$ B $|<$ A $|<$ C)*

proof (*rule FSet.pfssubsetI*)

show *A $|<$ B $|<=$ A $|<$ C*

using *assms(1)* **by** *blast*

next

show *A $|<$ B \neq A $|<$ C*

using *assms* **by** *blast*

qed

lemma *Abs-fset-minus*: $finite\ A \implies finite\ B \implies Abs-fset\ (A - B) = Abs-fset\ A \mid - \mid Abs-fset\ B$
by (*metis Abs-fset-inverse fset-inverse mem-Collect-eq minus-fset*)

lemma *fminus-conv*: $A \mid \subset \mid B \iff fset\ A \subset fset\ B \wedge finite\ (fset\ A) \wedge finite\ (fset\ B)$
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 $\beta\ \Gamma = Abs-fset\ \{L.\ atm-of\ L \preceq_B\ \beta\} \mid - \mid (fst \mid \uparrow \mid fset-of-list\ \Gamma)$

primrec *M-skip-fact-reso* **where**

M-skip-fact-reso $\square\ C = \square \mid$
M-skip-fact-reso $(Ln\ \#\ \Gamma)\ C =$
 (*let* $n = count\ C\ (-\ (fst\ Ln))$ *in*
 (*case* $snd\ Ln$ *of* $None \Rightarrow 0 \mid Some\ - \Rightarrow n$) $\#\$
M-skip-fact-reso $\Gamma\ (C + (case\ snd\ Ln\ of\ None \Rightarrow \{\#\} \mid Some\ (D, -, \gamma) \Rightarrow repeat-mset\ n\ (D \cdot \gamma)))$)

fun *M-skip-fact-reso'* **where**

M-skip-fact-reso' $C\ \square = \square \mid$
M-skip-fact-reso' $C\ ((-, None)\ \#\ \Gamma) = 0\ \#\ M-skip-fact-reso'\ C\ \Gamma \mid$
M-skip-fact-reso' $C\ ((K, Some\ (D, -, \gamma))\ \#\ \Gamma) =$
 (*let* $n = count\ C\ (-\ K)$ *in* $n\ \#\ M-skip-fact-reso'\ (C + repeat-mset\ n\ (D \cdot \gamma))$)
 Γ)

lemma *M-skip-fact-reso* $\Gamma\ C = M-skip-fact-reso'\ C\ \Gamma$

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 *M-skip-fact-reso'* $C\ (decide-lit\ K\ \#\ \Gamma) = 0\ \#\ M-skip-fact-reso'\ C\ \Gamma$
by (*simp add: decide-lit-def*)

lemma \mathcal{M} -skip-fact-reso' C (propagate-lit $K D \gamma \# \Gamma$) =
 (let $n = \text{count } C \text{ } (- (K \cdot l \gamma))$ in $n \# \mathcal{M}$ -skip-fact-reso' ($C + \text{repeat-mset } n (D \cdot \gamma)$) Γ)
by (simp add: propagate-lit-def)

fun $\mathcal{M} :: - \Rightarrow - \Rightarrow ('f, 'v) \text{ state} \Rightarrow$
 $\text{bool} \times ('f, 'v) \text{ Term.term literal fset} \times \text{nat list} \times \text{nat}$ **where**
 $\mathcal{M} N \beta (\Gamma, U, \text{None}) = (\text{True}, \mathcal{M}\text{-prop-deci } \beta \Gamma, [], 0) |$
 $\mathcal{M} N \beta (\Gamma, U, \text{Some } (C, \gamma)) = (\text{False}, \{\|\}, \mathcal{M}\text{-skip-fact-reso } \Gamma (C \cdot \gamma), \text{size } C)$

lemma $\text{length-}\mathcal{M}\text{-skip-fact-reso}$ [simp]: $\text{length } (\mathcal{M}\text{-skip-fact-reso } \Gamma C) = \text{length } \Gamma$
by (induction Γ arbitrary: C) (simp-all add: Let-def)

lemma \mathcal{M} -skip-fact-reso-add-mset:
 $(\mathcal{M}\text{-skip-fact-reso } \Gamma C, \mathcal{M}\text{-skip-fact-reso } \Gamma (\text{add-mset } L C)) \in (\text{List.lenlex } \{(x, y). x < y\})^=$
proof (induction Γ arbitrary: C)
case Nil
show ?case **by** simp
next
case ($\text{Cons } Ln \Gamma$)
show ?case
proof (cases $\text{snd } Ln$)
case None
then show ?thesis
using Cons.IH [of C]
by (simp add: Cons-lenlex-iff)
next
case ($\text{Some } cl$)
show ?thesis
proof (cases $L = - \text{fst } Ln$)
case True
then show ?thesis
by (simp add: $\text{Let-def Some Cons-lenlex-iff}$)
next
case False
then show ?thesis
using Cons.IH
by (auto simp add: $\text{Let-def Some Cons-lenlex-iff}$)
qed
qed
qed

lemma $\text{termination-scl-without-back-invars}$:
fixes $N \beta$
defines
 $\text{scl-without-backtrack} \equiv \text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta \sqcup$
 $\text{factorize } N \beta \sqcup \text{resolve } N \beta$ **and**

invars \equiv *trail-atoms-lt* β \sqcap *trail-resolved-lits-pol* \sqcap *trail-lits-ground* \sqcap
trail-lits-from-clauses N \sqcap *initial-lits-generalize-learned-trail-conflict* N \sqcap
ground-closures

shows *wfp-on* $\{S. \text{invars } S\}$ *scl-without-backtrack*⁻¹⁻¹

proof –

let *?less* =

lex-prodp ($(<) :: \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}$)
lex-prodp ($|\subset|$)
lex-prodp ($\lambda x y. (x, y) \in \text{List.lenlex } \{(x :: - :: \text{wellorder}, y). x < y\}$)
 $((<) :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}))$

show *wfp-on* $\{S. \text{invars } S\}$ *scl-without-backtrack*⁻¹⁻¹

proof (*rule wfp-on-if-convertible-to-wfp*)

fix $S' S :: ('f, 'v) \text{ state}$

assume $S' \in \{S. \text{invars } S\}$ **and** $S \in \{S. \text{invars } S\}$ **and** *step*: *scl-without-backtrack*⁻¹⁻¹

$S' S$

hence

trail-atoms-lt β S **and**
trail-resolved-lits-pol S **and**
trail-lits-ground S **and**
trail-lits-from-clauses $N S$ **and**
initial-lits-generalize-learned-trail-conflict $N S$ **and**
ground-closures S
trail-lits-from-clauses $N S'$ **and**
initial-lits-generalize-learned-trail-conflict $N S'$
by (*simp-all add: invars-def*)

have *trail-lits-from-init-clauses* $N S$
using $\langle \text{trail-lits-from-clauses } N S \rangle \langle \text{initial-lits-generalize-learned-trail-conflict } N S \rangle$
by (*simp add: trail-lits-from-init-clausesI*)

have *trail-lits-from-init-clauses* $N S'$
using $\langle \text{trail-lits-from-clauses } N S' \rangle \langle \text{initial-lits-generalize-learned-trail-conflict } N S' \rangle$
by (*simp add: trail-lits-from-init-clausesI*)

from *step* **show** *?less* $(\mathcal{M} N \beta S') (\mathcal{M} N \beta S)$

unfolding *conversep-iff scl-without-backtrack-def sup-apply sup-bool-def*

proof (*elim disjE*)

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

thus *?less* $(\mathcal{M} N \beta S') (\mathcal{M} N \beta S)$

proof (*cases N beta S S' rule: decide.cases*)

case (*decideI* $L \gamma \Gamma U$)

have *M-prop-deci* $\beta ((L \cdot l \gamma, \text{None}) \# \Gamma) |\subset| \mathcal{M}\text{-prop-deci } \beta \Gamma$
unfolding *M-prop-deci-def fset-of-list-simps fimage-finsert prod.sel*

proof (*rule minus-pfsubset-minusI*)

show *fst* $|\supset| \text{fset-of-list } \Gamma |\subset| \text{finsert } (L \cdot l \gamma) (\text{fst } |\supset| \text{fset-of-list } \Gamma)$
using $\langle \neg \text{trail-defined-lit } \Gamma (L \cdot l \gamma) \rangle \langle \text{unfolded trail-defined-lit-def} \rangle$

```

    by (metis (no-types, lifting) fininsertCI fset-of-list-elem fset-of-list-map
        fsubset-fininsertI list.set-map nless-le)
  next
  have  $L \cdot l \ \gamma \in \{L. \text{atm-of } L \preceq_B \beta\}$ 
    using  $\langle \text{atm-of } L \cdot a \ \gamma \preceq_B \beta \rangle$ 
    by simp
  moreover have  $\text{fst } \langle \text{set } \Gamma \subseteq \{L. \text{atm-of } L \preceq_B \beta\}$ 
    using  $\langle \text{trail-atoms-lt } \beta \ S \rangle$ 
    by (auto simp: trail-atoms-lt-def decideI(1))
  ultimately have  $\text{insert } (L \cdot l \ \gamma) (\text{fst } \langle \text{set } \Gamma \subseteq \{L. \text{atm-of } L \preceq_B \beta\}$ 
    by simp
  then show  $\text{fininsert } (L \cdot l \ \gamma) (\text{fst } |\cdot| \text{ fset-of-list } \Gamma) \mid\subseteq\mid \text{Abs-fset } \{L. \text{atm-of } L$ 
 $\preceq_B \beta\}$ 
    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  $N \ \beta \ S \ S'$ 
thus ?less  $(\mathcal{M} \ N \ \beta \ S') (\mathcal{M} \ N \ \beta \ S)$ 
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  $\gamma \ \{\text{atm-of } \langle \text{set-mset } (\text{add-mset } L \ C_1) \rangle\}$ 
      unfolding  $\langle C_1 = \{\#K \in \# \ C'. \ K \cdot l \ \gamma = L \cdot l \ \gamma \# \}$ 
      by (auto simp: is-unifiers-def is-unifier-alt intro: subst-atm-of-eqI)
    hence  $\mu \odot \gamma = \gamma$ 
      using  $\langle \text{is-imgu } \mu \ \{\text{atm-of } \langle \text{set-mset } (\text{add-mset } L \ C_1) \rangle\} \rangle$  [unfolded
is-imgu-def, THEN conjunct2]
      by simp
    thus ?thesis
      by (metis subst-lit-comp-subst)
  qed

  have  $\mathcal{M}\text{-prop-deci } \beta \ ((L \cdot l \ \gamma, \text{Some } (C_0 \cdot \mu, L \cdot l \ \mu, \gamma)) \# \Gamma) \mid\subseteq\mid \mathcal{M}\text{-prop-deci}$ 
 $\beta \ \Gamma$ 
    unfolding  $\mathcal{M}\text{-prop-deci-def fset-of-list-simps fimage-fininsert prod.sel}$ 
  proof (rule minus-pfsubset-minusI)
    show  $\text{fst } |\cdot| \text{ fset-of-list } \Gamma \mid\subseteq\mid \text{fininsert } (L \cdot l \ \gamma) (\text{fst } |\cdot| \text{ fset-of-list } \Gamma)$ 
      using  $\langle \neg \text{trail-defined-lit } \Gamma \ (L \cdot l \ \gamma) \rangle$  [unfolded trail-defined-lit-def]
      by (metis (no-types, lifting) fininsertCI fset-of-list-elem fset-of-list-map
          fsubset-fininsertI 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)
  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 |1 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 N β S') (M N 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 N β S') (M N 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 N β S') (M N 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  $\mu$  {{atm-of L, atm-of L'}}⟩
by (simp add: is-ingu-def is-unifiers-def)

have add-mset L D ·  $\mu$  ·  $\gamma$  = add-mset L D ·  $\gamma$ 
using ⟨ $\mu \odot \gamma = \gamma$ ⟩
by (metis subst-cls-comp-subst)
hence (M-skip-fact-reso  $\Gamma$  (add-mset L D ·  $\mu$  ·  $\gamma$ ),
M-skip-fact-reso  $\Gamma$  (add-mset L' (add-mset L D) ·  $\gamma$ )) ∈ (lenlex {(x, y). x
< y})=
using 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 (M N  $\beta$  S') (M N  $\beta$  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 ⟨ground-closures S⟩ have
  ground-conf: is-ground-cls (add-mset L C ·  $\gamma_C$ ) and
  ground-prop: is-ground-cls (add-mset K D ·  $\gamma_D$ )
unfolding resolveI(1,2) ⟨ $\Gamma = \text{trail-propagate } \Gamma' K D \gamma_D$ ⟩
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 · a  $\varrho_C$  · a  $\gamma$  = atm-of K · a  $\varrho_D$  · a  $\gamma$ 
using ⟨K · l  $\gamma_D = - (L \cdot l \gamma_C)$ ⟩
  ⟨ $\forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$ ⟩[rule-format, of L, simplified]
  ⟨ $\forall K \in \# \text{add-mset } K D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$ ⟩[rule-format, of K,
simplified]
by (metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit)
hence is-unifiers  $\gamma$  {{atm-of L · a  $\varrho_C$ , atm-of K · a  $\varrho_D$ }}
by (simp add: is-unifiers-def is-unifier-alt)
hence  $\mu \odot \gamma = \gamma$ 
using ⟨is-ingu  $\mu$  {{atm-of L · a  $\varrho_C$ , atm-of K · a  $\varrho_D$ }}⟩
by (auto simp: is-ingu-def)
hence C ·  $\varrho_C$  ·  $\mu$  ·  $\gamma$  = C ·  $\gamma_C$  and D ·  $\varrho_D$  ·  $\mu$  ·  $\gamma$  = D ·  $\gamma_D$ 
using ⟨ $\forall L \in \# \text{add-mset } L C. L \cdot l \varrho_C \cdot l \gamma = L \cdot l \gamma_C$ ⟩ ⟨ $\forall K \in \# \text{add-mset } K
D. K \cdot l \varrho_D \cdot l \gamma = K \cdot l \gamma_D$ ⟩
by (metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst
subst-lit-comp-subst)+
hence (C ·  $\varrho_C$  + D ·  $\varrho_D$ ) ·  $\mu$  ·  $\gamma$  = C ·  $\gamma_C$  + D ·  $\gamma_D$ 
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  $\text{resolveI}(1,2) \langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle$ 
  by (simp add: trail-resolved-lits-pol-def propagate-lit-def)

have ( $\mathcal{M}\text{-skip-fact-reso } \Gamma (C \cdot \gamma_C + D \cdot \gamma_D)$ ,  $\mathcal{M}\text{-skip-fact-reso } \Gamma (\text{add-mset } L C \cdot \gamma_C)$ )  $\in$ 
   $\text{lex } \{(x, y). x < y\}$ 
  unfolding  $\langle \Gamma = \text{trail-propagate } \Gamma' K D \gamma_D \rangle \text{ propagate-lit-def}$ 
  unfolding  $\mathcal{M}\text{-skip-fact-reso.simps Let-def prod.sel option.case prod.case}$ 
  unfolding  $\text{lex-conv mem-Collect-eq prod.case}$ 
  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 ( $\mathcal{M}\text{-skip-fact-reso } \Gamma (C \cdot \gamma_C + D \cdot \gamma_D)$ ,  $\mathcal{M}\text{-skip-fact-reso } \Gamma (\text{add-mset } L C \cdot \gamma_C)$ )  $\in$ 
   $\text{lenlex } \{(x, y). x < y\}$ 
  unfolding lenlex-conv by simp
thus ?thesis
  unfolding lex-prodp-def resolveI(1,2)
  unfolding  $\mathcal{M}\text{-simps state-proj-simp option.case prod.case prod.sel}$ 
  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 ( $\mathcal{M} N \beta ' \{S. \text{invars } S\}$ ) ?less
proof (rule wfp-on-subset)
  show  $\mathcal{M} N \beta ' \{S. \text{invars } S\} \subseteq UNIV$ 
  by simp
next
show wfp ?less
proof (intro wfp-lex-prodp)
  show wfp ( $(<) :: \text{bool} \Rightarrow \text{bool} \Rightarrow \text{bool}$ )
  unfolding wfp-iff-wfP
  by (simp add: Wellfounded.wfPUNIVI)
next
show wfp ( $|\cdot|$ )
  unfolding wfp-iff-wfP
  by (rule wfP-pfssubset)
next
show wfp ( $\lambda x y. (x, y) \in \text{lenlex } \{(x :: - :: \text{wellorder}, y). x < y\}$ )
  unfolding wfp-iff-wfP Wellfounded.wfP-wf-eq

```



```

    using wf-lenlex
    using wf by blast
  next
    show wfp ((<) :: nat ⇒ nat ⇒ bool)
      unfolding wfp-iff-wfP by simp
    qed
  qed
qed
qed

corollary termination-scl-without-back:
  fixes
    N :: ('f, 'v) Term.term clause fset and
    β :: ('f, 'v) Term.term
  defines
    scl-without-backtrack ≡ propagate N β ⊔ decide N β ⊔ conflict N β ⊔ skip N
    β ⊔
    factorize N β ⊔ resolve N β and
    invars ≡ trail-atoms-lt β ⊔ trail-resolved-lits-pol ⊔ trail-lits-ground ⊔
    trail-lits-from-clauses N ⊔ initial-lits-generalize-learned-trail-conflict N ⊔
    ground-closures
  shows wfp-on {S. scl-without-backtrack** initial-state S} scl-without-backtrack-1-1
proof (rule wfp-on-subset)
  show wfp-on {S. invars S} scl-without-backtrack-1-1
    by (rule termination-scl-without-back-invars(1)[of β N,
      folded invars-def scl-without-backtrack-def])
  next
    have invars initial-state
      by (simp add: invars-def)

    moreover have invars S ⇒ invars S'
      if scl-without-backtrack S S'
      for S S'
    proof –
      from that have scl N β S S'
        by (auto simp: scl-without-backtrack-def scl-def)
      thus invars S ⇒ invars S'
        unfolding invars-def
        using
          scl-preserved-trail-atoms-lt
          scl-preserved-trail-resolved-lits-pol
          scl-preserved-trail-lits-ground
          scl-preserved-trail-lits-from-clauses
          scl-preserved-initial-lits-generalize-learned-trail-conflict
          scl-preserved-ground-closures
        by simp-all
    qed
  ultimately have scl-without-backtrack** initial-state S ⇒ invars S for S
    by (auto elim: rtranclp-induct)

```

thus $\{S. \text{scl-without-backtrack}^{**} \text{ initial-state } S\} \subseteq \{S. \text{invars } S\}$
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
 $\text{scl-without-backtrack} \equiv \text{propagate } N \beta \sqcup \text{decide } N \beta \sqcup \text{conflict } N \beta \sqcup \text{skip } N \beta \sqcup$
 $\text{factorize } N \beta \sqcup \text{resolve } N \beta$
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

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

$$\text{fclss-no-dup } \beta = \text{fPow } (\text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\})$$

lemma *image-fset-fset-fPow-eq*: $\text{fset ' fset } (\text{fPow } A) = \text{Pow } (\text{fset } A)$

proof (*rule Set.equalityI*)

show $\text{fset ' fset } (\text{fPow } A) \subseteq \text{Pow } (\text{fset } A)$

by (*meson PowI fPowD image-subset-iff less-eq-fset.rep-eq*)

next

show $\text{Pow } (\text{fset } A) \subseteq \text{fset ' fset } (\text{fPow } A)$

proof (*rule Set.subsetI*)

fix x **assume** $x \in \text{Pow } (\text{fset } A)$

moreover hence *finite* x

by (*metis PowD finite-fset rev-finite-subset*)

ultimately show $x \in \text{fset ' fset } (\text{fPow } A)$

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 $\forall L \in C. \text{atm-of } L \preceq_B \beta$
 shows $C \in \text{fclss-no-dup } \beta$

proof –

show *?thesis*

unfolding *fclss-no-dup-def fPow-iff*

proof (*rule fsubsetI*)

fix *K* assume $K \in C$

with *assms* show $K \in \text{Abs-fset } \{L. \text{atm-of } L \preceq_B \beta\}$

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* $N \ \beta$)** $S1 \ S_n$ **and**

backtrack: *backtrack* $N \ \beta \ S_n \ S_n'$

defines $U \equiv \text{state-learned } S_n$

shows

$\exists C \ \gamma. \text{state-conflict } S_n = \text{Some } (C, \gamma) \wedge$

$\text{set-mset } (C \cdot \gamma) \notin \text{set-mset ' grounding-of-clss (fset } N \cup \text{fset } U)$ **and**

$\text{M-back } \beta \ S_n' \sqsubset \text{M-back } \beta \ S_n$

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* S_n

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  $x A B$ 
  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-cls* *C*
unfolding 1 *finsert.rep-eq grounding-of-clss-insert image-Un*
by *auto*

also have ... \subset

fset (fclss-no-dup β) - Abs-fset ‘ *set-mset* ‘ *grounding-of-clss (fset (state-learned S_n))*

proof (*rule Diff-strict-subsetI*)

from $\langle \text{ground-closures } S_n \rangle$ **have** $C \cdot \gamma \in \text{grounding-of-cls } C$

unfolding *ground-closures-def conf*

using *grounding-of-cls-ground grounding-of-subst-cls-subset* **by** *blast*

thus *Abs-fset (set-mset (C · γ))* \in *Abs-fset* ‘ *set-mset* ‘ *grounding-of-cls C*

by *blast*

next

have *Abs-fset-in-image-Abs-fset-iff*: *Abs-fset A* \in *Abs-fset* ‘ *AA* \longleftrightarrow *A* \in *AA*

if *finite A* \wedge ($\forall B \in AA.$ *finite B*)

for *A AA*

apply (*rule iffI*)

using *that*

apply (*metis Abs-fset-inverse imageE mem-Collect-eq*)

using *that*

by *blast*

have *set-mset (C · γ)* \notin *set-mset* ‘ *grounding-of-clss (fset (state-learned S_1))*

using *set-conf-not-in-set-groundings*

by (*auto simp: grounding-of-clss-union*)

then have *Abs-fset (set-mset (C · γ))* \notin

Abs-fset ‘ *set-mset* ‘ *grounding-of-clss (fset (state-learned S_n))*

unfolding 2

using *Abs-fset-in-image-Abs-fset-iff*

by (*metis finite-set-mset image-iff*)

moreover have *Abs-fset (set-mset (C · γ))* \in *fset (fclss-no-dup β)*

proof (*intro fmember-fclss-no-dup-if ballI*)

fix *L* **assume** $L \in$ *Abs-fset (set-mset (C · γ))*

hence $L \in \# C \cdot \gamma$

by (*metis fset-fset-mset fset-inverse*)

moreover have *trail-false-cls (state-trail S_n) (C · γ)*

using $\langle \text{ground-false-closures } S_n \rangle$ *conf* **by** (*auto simp: ground-false-closures-def*)

ultimately show *atm-of L* \preceq_B β

using *ball-less-B-if-trail-false-and-trail-atoms-lt[OF - $\langle \text{trail-atoms-lt } \beta S_n \rangle$]*

by *metis*

qed

ultimately show *Abs-fset (set-mset (C · γ))* \in *fset (fclss-no-dup β)* –

Abs-fset ‘ *set-mset* ‘ *grounding-of-clss (fset (state-learned S_n))*

by *simp*

qed

finally show $\mathcal{M}\text{-back } \beta \text{ Sn}' \mid\!\!| \mathcal{M}\text{-back } \beta \text{ Sn}$
unfolding $\mathcal{M}\text{-back-def}$
unfolding $f\text{minus-conv}$
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

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$
 $\text{trail-lits-from-clauses } N \sqcap \text{initial-lits-generalize-learned-trail-conflict } N \sqcap$
 $\text{ground-closures} \sqcap \text{ground-false-closures} \sqcap \text{sound-state } N \beta \sqcap$
 $\text{almost-no-conflict-with-trail } N \beta \sqcap$
 $\text{regular-conflict-resolution } N \beta$

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 $\text{wfp-on } (\mathcal{M}\text{-back } \beta \text{ ' } \{S. \text{invars } S\}) (\mid\!\!|)$

proof (*rule wfp-on-subset*)

show $\text{wfp } (\mid\!\!|)$

unfolding *wfp-iff-wfP*

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** *almost-no-conflict-with-trail* $N \beta S$
by (*simp add: invars-def*)

moreover from $\langle \text{invars } S \rangle$ **have** *regular-conflict-resolution* $N \beta S$
by (*simp add: invars-def*)

moreover from $\langle \text{invars } S \rangle$ **have** *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*
(propagate $N \beta \sqcup$ decide $N \beta \sqcup$ conflict $N \beta \sqcup$ skip $N \beta \sqcup$ factorize $N \beta \sqcup$
resolve $N \beta$)⁻¹⁻¹ $S' S$
hence *state-learned $S' =$ state-learned S*
by (*auto elim: propagate.cases decide.cases conflict.cases skip.cases factor-*
ize.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 $\mathcal{M}\text{-back } \beta S' \mid \subset \mid \mathcal{M}\text{-back } \beta S \vee \mathcal{M}\text{-back } \beta S' = \mathcal{M}\text{-back } \beta S \dots$
qed
qed

corollary *termination-regular-scl*:

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**

$\beta :: ('f, 'v) \text{ Term.term}$

defines

$\text{invars} \equiv \text{trail-atoms-lt } \beta \sqcap \text{trail-resolved-lits-pol} \sqcap \text{trail-lits-ground} \sqcap$
 $\text{trail-lits-from-clauses } N \sqcap \text{initial-lits-generalize-learned-trail-conflict } N \sqcap$
 $\text{ground-closures} \sqcap \text{ground-false-closures} \sqcap \text{sound-state } N \beta \sqcap$
 $\text{almost-no-conflict-with-trail } N \beta \sqcap$
 $\text{regular-conflict-resolution } N \beta$

shows $\text{wfp-on } \{S. (\text{regular-scl } N \beta)^{**} \text{ initial-state } S\} (\text{regular-scl } N \beta)^{-1-1}$

proof (*rule wfp-on-subset*)

show $\text{wfp-on } \{S. \text{invars } S\} (\text{regular-scl } N \beta)^{-1-1}$

by (*rule termination-regular-scl-invars(1)*)[*of* βN , *folded invars-def*]

next

note $\text{rea-to-scl} = \text{scl-if-reasonable}$

note $\text{reg-to-rea} = \text{reasonable-if-regular}$

note $\text{reg-to-scl} = \text{reg-to-rea}[\text{THEN } \text{rea-to-scl}]$

have $\text{invars initial-state}$

by (*simp add: invars-def*)

moreover have $\bigwedge S S'. \text{regular-scl } N \beta S S' \implies \text{invars } S \implies \text{invars } S'$

unfolding *invars-def*

using

$\text{reg-to-scl}[\text{THEN } \text{scl-preserves-trail-atoms-lt}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-trail-resolved-lits-pol}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-trail-lits-ground}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-trail-lits-from-clauses}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-initial-lits-generalize-learned-trail-conflict}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-ground-closures}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-ground-false-closures}]$
 $\text{reg-to-scl}[\text{THEN } \text{scl-preserves-sound-state}]$
 $\text{regular-scl-preserves-almost-no-conflict-with-trail}$
 $\text{regular-scl-preserves-regular-conflict-resolution}$

by *simp*

ultimately have $(\text{regular-scl } N \beta)^{**} \text{ initial-state } S \implies \text{invars } S$ **for** S

by (*auto elim: rtranclp-induct*)

thus $\{S. (\text{regular-scl } N \beta)^{**} \text{ initial-state } S\} \subseteq \{S. \text{invars } S\}$

by *auto*

qed

corollary *termination-strategy*:

fixes

$N :: ('f, 'v) \text{ Term.term clause fset}$ **and**


```

     $\beta :: ('f, 'v) \text{Term.term}$ 
assumes strategy-restricts-regular-scl:  $\bigwedge S S'. \text{strategy } S S' \implies \text{regular-scl } N \beta$ 
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  $\{S. \text{strategy}^{**} \text{initial-state } S\} (\text{regular-scl } N \beta)^{-1-1}$ 
proof (rule wfp-on-subset)
    show wfp-on  $\{S. (\text{regular-scl } N \beta)^{**} \text{initial-state } S\} (\text{regular-scl } N \beta)^{-1-1}$ 
    using termination-regular-scl by metis
next
    show  $\{S. \text{strategy}^{**} \text{initial-state } S\} \subseteq \{S. (\text{regular-scl } N \beta)^{**} \text{initial-state } S\}$ 
    using strategy-restricts-regular-scl
    by (metis (no-types, opaque-lifting) Collect-mono mono-rtranclp)
qed
next
show  $\bigwedge S' S. \text{strategy}^{-1-1} S' S \implies (\text{regular-scl } N \beta)^{-1-1} S' S$ 
using strategy-restricts-regular-scl by simp
qed
end
end
theory Completeness
imports
    Correct-Termination
    Termination
    Functional-Ordered-Resolution-Prover.IsaFoR-Term
begin

lemma (in scl-fol-calculus) regular-scl-run-derives-contradiction-if-unsat:
fixes  $N \beta \text{gnd-N}$ 
defines
     $\text{gnd-N} \equiv \text{grounding-of-clss } (\text{fset } N)$  and
     $\text{gnd-N-lt-}\beta \equiv \{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$ 
assumes
    unsat:  $\neg \text{satisfiable gnd-N-lt-}\beta$  and
    run:  $(\text{regular-scl } N \beta)^{**} \text{initial-state } S$  and
    no-more-step:  $\nexists S'. \text{regular-scl } N \beta S S'$ 
shows  $\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)$ 
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  $N \beta \text{gnd-N}$ 
defines
     $\text{gnd-N} \equiv \text{grounding-of-clss } (\text{fset } N)$  and
     $\text{gnd-N-lt-}\beta \equiv \{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$ 
assumes unsat:  $\neg \text{satisfiable gnd-N-lt-}\beta$ 
shows  $\exists S. (\text{regular-scl } N \beta)^{**} \text{initial-state } S \wedge$ 

```

$(\nexists S'. \text{regular-scl } N \beta S S') \wedge$
 $(\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma))$
proof –
obtain S **where**
run: $(\text{regular-scl } N \beta)^{**}$ *initial-state* S **and**
no-more-step: $(\nexists S'. \text{regular-scl } N \beta S S')$
using *termination-regular-scl*[*THEN ex-trans-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 $\exists \gamma. \text{state-conflict } S = \text{Some } (\{\#\}, \gamma)$
using *unsat correct-termination-regular-scl-run*[*OF run no-more-step*]
by (*simp add: gnd-N-lt- β -def gnd-N-def*)

ultimately show *?thesis*
by *metis*
qed

lemma (**in** *scl-fol-calculus*) *no-infinite-down-chain*:
 $\nexists Ss. \neg \text{lfinite } Ss \wedge \text{Lazy-List-Chain.chain } (\lambda S S'. \text{regular-scl } N \beta S S') (LCons \text{initial-state } Ss)$
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 $N \beta \text{gnd-N}$
defines
 $\text{gnd-N} \equiv \text{grounding-of-class } (fset N)$ **and**
 $\text{gnd-N-lt-}\beta \equiv \{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \preceq_B \beta\}$
assumes *unsat*: $\neg \text{satisfiable gnd-N-lt-}\beta$
shows
 $\nexists Ss. \neg \text{lfinite } Ss \wedge \text{Lazy-List-Chain.chain } (\lambda S S'. \text{regular-scl } N \beta S S') (LCons \text{initial-state } Ss)$ **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  $\equiv$  grounding-of-cls (fset N)
assumes unsat:  $\neg$  satisfiable gnd-N
shows  $\exists \beta. \neg$  satisfiable  $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \leq \beta\}$ 
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-in:  $C \in \text{gnd-N}'$  and C-min:  $\forall x \in \text{gnd-N}'. x \leq C$ 
    using finite-has-maximal[OF  $\langle \text{finite gnd-N}' \rangle$   $\langle \text{gnd-N}' \neq \{\} \rangle$ ] by force

show ?thesis
proof (cases C)
    case empty
    let ?S = ( $\{\}$ ,  $\{\|\}$ , Some ( $\{\#\}$ , Var))

    show ?thesis
    proof (rule exI)
        have  $\{\#\} \mid \in \mid N$ 
            using C-in  $\langle \text{gnd-N}' \subseteq \text{gnd-N} \rangle$ 
            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  $\{\#\} \in \text{gnd-N}$ 
            using C-in  $\langle \text{gnd-N}' \subseteq \text{gnd-N} \rangle$  local.empty by blast
        hence  $\{\#\} \in \{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L < \text{undefined}\}$ 
            by force
        thus  $\neg$  satisfiable  $\{C \in \text{gnd-N}. \forall L \in \# C. \text{atm-of } L \leq \text{undefined}\}$ 
            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  $L \in \# C$  and L-min:  $\forall x \in \# C. x \leq L$ 
    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  $\forall D \in \text{gnd-N}'. \forall K \in \# D. \text{atm-of } K \leq \text{atm-of } L$ 
    by (meson dual-order.trans ex-gt-imp-less-multiset leq-imp-less-eq-atm-of
    verit-comp-simplify1 (3))
hence  $\text{gnd-N}' \subseteq \{D \in \text{gnd-N}. \forall K \in \# D. (\text{atm-of } K) \leq (\text{atm-of } L)\}$ 
    using  $\langle \text{gnd-N}' \subseteq \text{gnd-N} \rangle$  subset-Collect-iff by auto

```

hence \neg *satisfiable* $\{D \in \text{gnd-}N. \forall K \in \# D. (\text{atm-of } K) \leq (\text{atm-of } L)\}$
using $\langle \neg$ *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
 $\text{scl } N \beta S S' \implies \text{trail-lits-ground } S \implies \text{trail-lits-ground } S' \text{ and}$
inv-trail-atoms-lt:
trail-atoms-lt β initial-state
 $\text{scl } N \beta S S' \implies \text{trail-atoms-lt } \beta S \implies \text{trail-atoms-lt } \beta S' \text{ and}$
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*)
 $\text{scl } N \beta S S' \implies$
 $(\forall \Gamma' Ln \Gamma''. \text{state-trail } S = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' (\text{fst } Ln))$
 \implies
 $(\forall \Gamma' Ln \Gamma''. \text{state-trail } S' = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg \text{trail-defined-lit } \Gamma' (\text{fst } Ln)) \text{ and}$
inv-ground-closures:
ground-closures initial-state
 $\text{scl } N \beta S S' \implies \text{ground-closures } S \implies \text{ground-closures } S' \text{ and}$
inv-ground-false-closures:
ground-false-closures initial-state
 $\text{scl } N \beta S S' \implies \text{ground-false-closures } S \implies \text{ground-false-closures } S' \text{ and}$
inv-trail-propagated-lits-wf:
 $\forall \mathcal{K} \in \text{set } (\text{state-trail initial-state}). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma$
 $\text{scl } N \beta S S' \implies$
 $(\forall \mathcal{K} \in \text{set } (\text{state-trail } S). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma) \implies$
 $(\forall \mathcal{K} \in \text{set } (\text{state-trail } S'). \forall D K \gamma. \text{snd } \mathcal{K} = \text{Some } (D, K, \gamma) \longrightarrow \text{fst } \mathcal{K} = K \cdot l \gamma) \text{ and}$
inv-trail-resolved-lits-pol:
trail-resolved-lits-pol initial-state

```

    scl N β S S' ⇒ trail-resolved-lits-pol S ⇒ trail-resolved-lits-pol S' and
  inv-initial-lits-generalize-learned-trail-conflict:
    initial-lits-generalize-learned-trail-conflict N initial-state
    scl N β S S' ⇒ initial-lits-generalize-learned-trail-conflict N S ⇒ ini-
tial-lits-generalize-learned-trail-conflict N S' and
  inv-trail-lits-from-clauses:
    trail-lits-from-clauses N initial-state
    scl N β S S' ⇒ trail-lits-from-clauses N S ⇒ trail-lits-from-clauses N S'
and
  inv-sound-state:
    sound-state N β initial-state
    scl N β S S' ⇒ sound-state N β S ⇒ sound-state N β 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]
  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
  scl-preserves-initial-lits-generalize-learned-trail-conflict
using trail-lits-from-clauses-initial-state scl-preserves-trail-lits-from-clauses
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.