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

Martin Desharnais

April 18, 2024

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 Some rules are right unique 41
5.5 Miscellaneous Lemmas 41

6 Invariants **47**
6.1 Initial Literals Generalize Learned, Trail, and Conflict Literals 47
6.2 Trail Literals Are Ground 54
6.3 Trail Literals Are Defined Only Once 56
6.4 Trail Closures Are False In Subtrails 58
6.5 Trail Literals Were Propagated or Decided 60
6.6 Trail Atoms Are Less Than Bound 68
6.7 Trail Resolved Literals Have Unique Polarity 69
6.8 Trail And Conflict Closures Are Ground 71
6.9 Trail And Conflict Closures Are Ground And False 76
6.10 Learned Clauses Are Non-empty 81
6.11 Backtrack Follows Conflict Resolution 82
6.12 Miscellaneous Lemmas 84

7 Soundness **86**
7.1 Sound Trail 86
7.2 Sound State 86
7.3 Initial State Is Sound 87
7.4 SCL Rules Preserve Soundness 87

8 Strategies **94**

9 Monotonicity w.r.t. the Bounding Atom **97**
9.1 Examples 121
9.2 Miscellaneous Lemmas 122
9.3 Well-Defined 123
9.4 Strict Partial Order 124
9.5 Strict Total (w.r.t. Elements in Trail) Order 126
9.6 Well-Founded 126
9.7 Extension on All Literals 131
9.7.1 Well-Founded 133
9.8 Alternative only for terms 133

10 Reasonable Steps **139**
10.1 Invariants . . . 139
10.1.1 No Conflict After Decide . . . 139
10.2 Miscellaneous Lemmas . . . 142

11 Regular Steps **142**
11.1 Invariants . . . 143
11.1.1 Almost No Conflict With Trail . . . 143
11.1.2 Backtrack Follows Regular Conflict Resolution . . . 147
11.2 Miscellaneous Lemmas . . . 159

12 Resolve in Regular Runs **164**

13 Clause Redundancy **170**

14 Trail-Induced Ordering **172**
14.1 Miscellaneous Lemmas . . . 172
14.2 Strict Partial Order . . . 173
14.3 Properties . . . 174

15 Dynamic Non-Redundancy **175**

16 Static Non-Redundancy **190**
16.1 Basic Results . . . 193
16.1.1 Minimal-element characterization of well-foundedness 193

17 Extra Lemmas **201**
17.1 Set Extra . . . 201
17.2 Prod Extra . . . 201
17.3 Wellfounded Extra . . . 202
17.4 FSet Extra . . . 202

18 Termination **203**
18.1 SCL without backtracking terminates . . . 203
18.2 Backtracking can only be done finitely often . . . 211
18.3 Regular SCL terminates . . . 214

theory *Abstract-Renaming-Apart*
imports *Main*
begin

1 Abstract Renaming

locale *renaming-apart* =
fixes
renaming :: *'a set* ⇒ *'a* ⇒ *'a*
assumes

renaming-correct: *finite* $V \Longrightarrow$ *renaming* $V\ x \notin V$ **and**
inj-renaming: *finite* $V \Longrightarrow$ *inj* (*renaming* V)

1.1 Interpretation to Prove That Assumptions Are Consistent

experiment begin

definition *renaming-apart-nats* **where**
renaming-apart-nats V = (*let* m = *Max* V *in* (λx. *Suc* (x + m)))

interpretation *renaming-apart-nats*: *renaming-apart renaming-apart-nats*
proof *unfold-locales*
show $\bigwedge V\ x$. *finite* $V \Longrightarrow$ *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 $\bigwedge V$. *inj* (*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*:
$L \cdot l\ \sigma_L = K \cdot l\ \sigma_K \Longrightarrow$ *atm-of* $L \cdot a\ \sigma_L$ = *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*: *set-mset* ($C \cdot \sigma$) = (λL. $L \cdot l$ σ) ‘ *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*:
$\{x \in \mathit{insert}\ y\ S.\ P\ x\} = (\mathit{if}\ P\ y\ \mathit{then}\ \mathit{insert}\ y\ \mathit{else}\ \mathit{id})\ \{x \in S.\ P\ x\}$
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*: $\bigwedge x\ y\ F.\ \mathit{finite}\ F \Longrightarrow x \neq y \Longrightarrow x \notin F \Longrightarrow y \notin F \Longrightarrow P\ (\mathit{insert}\ y\ F) \Longrightarrow P\ (\mathit{insert}\ x\ (\mathit{insert}\ y\ F))$
shows *P F*
using ‹*finite F*›
proof *induct*
show *P* {} **by** *fact*
next
fix *x F*
assume *F*: *finite F* **and** *P*: *P F*
thus *P* (*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.prems)
    next
      case x-neq-y: False
      show ?thesis
      proof (cases x ∈ F ∨ y ∈ F)
        case True
        then show ?thesis
          by (metis insertCI insertI.IH insertI.prems insert-absorb)
      next
        case False
        show ?thesis
        proof (rule insert-insert)
          show finite F using insertI by simp
        next
          show x ≠ y by (rule x-neq-y)
        next
          show x ∉ F using False by simp
        next
          show y ∉ F using False by simp
        next
          show P (insert y F)
            by (simp add: insertI.prems)
        qed
      qed
    qed
  qed
qed
```

3.3 Product Type Extra

```
lemma insert-Times: insert a A × B = Pair a ‘ B ∪ A × B
  by blast

lemma Times-insert: A × insert b B = (λx. (x, b)) ‘ A ∪ A × B
  by blast

lemma insert-Times-insert′:
  insert a A × insert b B = insert (a, b) ((Pair a ‘ B) ∪ ((λx. (x, b)) ‘ A) ∪ (A
× B))
  (is ?lhs = ?rhs)
  unfolding insert-Times-insert by auto
```

3.4 List Extra

```
lemma lt-lengthD:
  assumes i-lt-xs: i < length xs
  shows ∃ xs1 xi xs2. xs = xs1 @ xi # xs2 ∧ 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*: $i < length\ xs$ **and** *j-lt-xs*: $j < length\ xs$ **and**
i-lt-j: $i < j$
shows $\exists\, xs1\ xi\ xs2\ xj\ xs3.\ xs = xs1\ @\ xi\ \#\ xs2\ @\ xj\ \#\ xs3 \wedge length\ xs1 = i \wedge length\ (xs1\ @\ xi\ \#\ xs2) = j$
proof −
from *i-lt-xs* **obtain** *xs1 xi xs'* **where** $xs = xs1\ @\ xi\ \#\ xs'$ **and** $length\ xs1 = i$
using *lt-lengthD* **by** *blast*
with *j-lt-xs* **obtain** *xs2 xj xs3* **where** $xs = xs1\ @\ xi\ \#\ xs2\ @\ xj\ \#\ xs3$ **and** $length\ (xs1\ @\ xi\ \#\ xs2) = j$
using *lt-lengthD*
by (*smt* (*verit*, *del-insts*) *append.assoc append-Cons append-eq-append-conv i-lt-j list.inject*)
thus *?thesis*
using ‹$length\ xs1 = i$› **by** *blast*
qed

3.5 Sublist Extra

lemma *not-mem-strict-suffix*:
shows $strict\text{-}suffix\ xs\ (y\ \#\ ys) \Longrightarrow y \notin set\ ys \Longrightarrow y \notin set\ xs$
unfolding *strict-suffix-def suffix-def*
by (*metis Cons-eq-append-conv Un-iff set-append*)

lemma *not-mem-strict-suffix'*:
shows $strict\text{-}suffix\ xs\ (y\ \#\ ys) \Longrightarrow f\ y \notin f\ `\ set\ ys \Longrightarrow f\ y \notin f\ `\ set\ xs$
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*:
$multp_{DM}\ R\ M\ N \Longrightarrow \exists\, I\ J\ K.\ N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge (\forall\, k{\in}\#K.\ \exists\, x{\in}\#J.\ R\ k\ x)$
unfolding $multp_{DM}$*-def*
by (*metis subset-mset.le-imp-diff-is-add*)

lemma $multp_{HO}$*-implies-one-step*:
$multp_{HO}\ R\ M\ N \Longrightarrow \exists\, I\ J\ K.\ N = I + J \wedge M = I + K \wedge J \neq \{\#\} \wedge (\forall\, k{\in}\#K.\ \exists\, x{\in}\#J.\ R\ k\ x)$
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* ∈# *M* ⟹ *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* ⊨ *U* ⟹ *D* ∈ *U* ⟹ *N* ⊨ {*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* ⟹ *I* ⊨ *C* ⟷ *I* ⊨ *D*
by (*simp add*: *true-cls-def*)

lemma *true-clss-if-set-mset-eq*: (∀ *D* ∈ 𝒟. ∃ *C* ∈ 𝒞. *set-mset D* = *set-mset C*) ⟹ *I* ⊨*s* 𝒞 ⟹ *I* ⊨*s* 𝒟
using *true-cls-iff-set-mset-eq* **by** (*metis true-clss-def*)

lemma *entails-clss-insert*: *N* ⊨*e insert C U* ⟷ *N* ⊨*e* {*C*} ∧ *N* ⊨*e U*
by *auto*

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

3.7.2 Clausal Calculus and Abstract Substitution

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

lemma (**in** *substitution*) *is-ground-lit-Neg*[*simp*]: *is-ground-atm atm* ⟹ *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*: $L = - K \Longrightarrow$ *atm-of* $L =$ *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: *vars-term* $t \cap$ *subst-domain* $\delta = \{\}$ **and**
u-inter-σ-empty: *vars-term* $u \cap$ *subst-domain* $\sigma = \{\}$
shows
range-vars $\sigma \cap$ *subst-domain* $\delta = \{\} \Longrightarrow t \cdot \sigma \cdot \delta = u \cdot \sigma \cdot \delta$
range-vars $\delta \cap$ *subst-domain* $\sigma = \{\} \Longrightarrow t \cdot \delta \cdot \sigma = u \cdot \delta \cdot \sigma$
proof −
from *u-inter-σ-empty* **have** $u \cdot \sigma \cdot \delta = u \cdot \delta$
by (*simp add*: *subst-apply-term-ident*)
with ‹$t \cdot \sigma = u \cdot \delta$› **show** *range-vars* $\sigma \cap$ *subst-domain* $\delta = \{\} \Longrightarrow t \cdot \sigma \cdot \delta = u \cdot \sigma \cdot \delta$
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 ‹$t \cdot \sigma = u \cdot \delta$› **show** *range-vars* $\delta \cap$ *subst-domain* $\sigma = \{\} \Longrightarrow t \cdot \delta \cdot \sigma = u \cdot \delta \cdot \sigma$
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**
vars-term $t \cap$ *subst-domain* $\delta = \{\}$ **and**
vars-term $u \cap$ *subst-domain* $\sigma = \{\}$
shows
range-vars $\sigma \cap$ *subst-domain* $\delta = \{\} \Longrightarrow \sigma \circ_s \delta \in$ *unifiers* $\{(t, u)\}$
range-vars $\delta \cap$ *subst-domain* $\sigma = \{\} \Longrightarrow \delta \circ_s \sigma \in$ *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*: $x \notin$ *subst-domain* $\mu \Longrightarrow \mu\ x =$ *Var* x
by (*simp add*: *subst-domain-def*)

lemma *is-renaming* (*Var*($x :=$ *Var* x'))
proof (*unfold is-renaming-def*, *intro conjI allI*)
fix y **show** *is-Var* ((*Var*($x :=$ *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** σ_t σ_u :: ('*f*, '*v*) *subst*
assumes $t \cdot \sigma_t = u \cdot \sigma_u$ **and** *vars-term t* ∩ *vars-term u* = {}
shows ∃ μ :: ('*f*, '*v*) *subst*. *Unification.mgu t u* = *Some* μ
proof −
from *assms* **obtain** σ :: ('*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 σ_A σ_B *A B*
assumes
distinct-domains: *A* ∩ *B* = {} **and**
distinct-range: ∀ *x* ∈ *A*. *vars-term* (σ_A *x*) ∩ *subst-domain* σ_B = {}
defines $\sigma \equiv$ *restrict-subst-domain A* σ_A $\circ_s$ σ_B
shows *x* ∈ *A* ⟹ σ *x* = σ_A *x x* ∈ *B* ⟹ σ *x* = σ_B *x*
proof −
assume *x* ∈ *A*
hence *restrict-subst-domain A* σ_A *x* = σ_A *x*
by (*simp add*: *restrict-subst-domain-def*)
moreover have σ_A *x* · σ_B = σ_A *x*
using *distinct-range*
by (*simp add*: ‹*x* ∈ *A*› *subst-apply-term-ident*)
ultimately show σ *x* = σ_A *x*
by (*simp add*: σ*-def subst-compose-def*)
next
assume *x* ∈ *B*
hence *restrict-subst-domain A* σ_A *x* = *Var x*
using *distinct-domains*
by (*metis Int-iff empty-iff restrict-subst-domain-def*)
then show σ *x* = σ_B *x*
by (*simp add*: σ*-def subst-compose-def*)
qed

lemma *merge-substs-on-disjoint-domains*:
fixes σ_A σ_B *A B*

assumes *distinct-domains*: $A \cap B = \{\}$
defines $\sigma \equiv (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ \sigma_A\ x\ \mathit{else}\ \mathit{if}\ x \in B\ \mathit{then}\ \sigma_B\ x\ \mathit{else}\ \mathit{Var}\ x)$
shows
$x \in A \Longrightarrow \sigma\ x = \sigma_A\ x$
$x \in B \Longrightarrow \sigma\ x = \sigma_B\ x$
$x \notin A \cup B \Longrightarrow \sigma\ x = \mathit{Var}\ x$
proof −
show $x \in A \Longrightarrow \sigma\ x = \sigma_A\ x$
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 $x \notin A \cup B \Longrightarrow \sigma\ x = \mathit{Var}\ x$
by (*simp add*: σ-*def*)
qed

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

lemma *is-grounding-merge-if-mem-then-else*[*simp*]:
fixes $\gamma_A\ \gamma_B\ A\ B$
defines $\gamma \equiv (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ \gamma_A\ x\ \mathit{else}\ \gamma_B\ x)$
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 $\gamma \equiv$ *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: $\forall x{\in}A.\ \mathit{vars\text{-}term}\ (\gamma_A\ x) = \{\}$ **and**
ball-B-ground: $\forall x{\in}B.\ \mathit{vars\text{-}term}\ (\gamma_B\ x) = \{\}$

from *ball-A-ground* **have** $\forall x{\in}A.\ \mathit{vars\text{-}term}\ (\gamma_A\ x) \cap \mathit{subst\text{-}domain}\ \gamma_B = \{\}$
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** $\cdot$ *67*)
no-notation *subst-compose* (**infixl** $\circ_s$ *75*)

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

notation *subst-atm-abbrev* (**infixl** $\cdot a$ *67*)
notation *subst-atm-list* (**infixl** $\cdot al$ *67*)
notation *subst-lit* (**infixl** $\cdot l$ *67*)
notation *subst-cls* (**infixl** $\cdot$ *67*)
notation *subst-clss* (**infixl** $\cdot cs$ *67*)
notation *subst-cls-list* (**infixl** $\cdot cl$ *67*)
notation *subst-cls-lists* (**infixl** $\cdot\cdot cl$ *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* $\Longrightarrow$ *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* $\longleftrightarrow$ *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* ⟷ *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* ⟷ *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* $t \subseteq$ *subst-domain* γ
using *assms*
by (*metis empty-iff is-ground-atm-iff-vars-empty is-ground-atm-is-ground-on-var subsetI*
subst-ident-if-not-in-domain term.set-intros(*3*))

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

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

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

lemma *same-on-vars-clause*:
assumes $\forall\, v \in$ *vars-cls* $S.\ \sigma\ v = \tau\ v$
shows *subst-cls* $S\ \sigma =$ *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 $\forall L \in\# C.$ *subst-lit* $L\ \sigma =$ *subst-lit* $L\ \tau$
  shows *subst-cls* $C\ \sigma =$ *subst-cls* $C\ \tau$
  using *assms* **unfolding** *subst-cls-def*
  by *simp*

global-interpretation *substitution* $(\cdot a)$ *Var* :: - $\Rightarrow$ $('f, 'v)$ *term* $(\odot)$
proof *unfold-locales*
  show $\bigwedge A.$ $A \cdot a$ *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 :: $('f, 'v)$ *term clause* **and** σ :: $('f, 'v)$ *subst*
  assume *is-ground-cls* (*subst-cls* $C\ \sigma$)
  hence *ground-atms-σ*: $\bigwedge v.$ $v \in$ *vars-cls* $C \Longrightarrow$ *is-ground-atm* $(\sigma\ v)$
    by (*meson is-ground-cls-is-ground-on-var*)

  define *some-ground-trm* :: $('f, 'v)$ *term* **where** *some-ground-trm* = (*Fun undefined* [])
  have *ground-trm*: *is-ground-atm some-ground-trm*
    unfolding *is-ground-atm-def some-ground-trm-def* **by** *auto*
  define τ **where** $\tau = (\lambda v.$ *if* $v \in$ *vars-cls* C *then* $\sigma\ v$ *else some-ground-trm*)
  then have *τ-σ*: $\forall\, v \in$ *vars-cls* $C.$ $\sigma\ v = \tau\ v$
    unfolding *τ-def* **by** *auto*

  have *all-ground-τ*: *is-ground-atm* $(\tau\ v)$ **for** v
  proof (*cases* $v \in$ *vars-cls* C)
    case *True*
    then show *?thesis*
      using *ground-atms-σ τ-σ* **by** *auto*
  next
    case *False*
    then show *?thesis*
      unfolding *τ-def* **using** *ground-trm* **by** *auto*
  qed
  have *is-ground-subst* τ
    unfolding *is-ground-subst-def*
  proof
    fix A
    show *is-ground-atm* $(A \cdot a\ \tau)$
    proof (*induction* A)
      case (*Var* v)
      thus *?case* **using** *all-ground-τ* **by** *auto*

```
    next
      case (Fun f As)
      thus ?case using all-ground-τ by (simp add: is-ground-atm-def)
    qed
  qed
  moreover with τ-σ have C · σ = C · τ
    using same-on-vars-clause by auto
  ultimately show ∃τ. is-ground-subst τ ∧ C · τ = C · σ
    by auto
next
  show wfP (strictly-generalizes-atm :: ('f, 'v) term ⇒ - ⇒ -)
    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 σ) = vars-term (atm-of L ·a
σ)
  by (cases L) simp-all

lemma vars-subst-lit-eq:
  vars-lit (L ·l σ) = (⋃x ∈ vars-lit L. vars-term (σ x))
  using vars-term-subst-apply-term by (metis atm-of-subst-lit)

lemma vars-subst-cls-eq:
  vars-cls (C · σ) = (⋃x ∈ vars-cls C. vars-term (σ 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 σ) ⊆ vars-lit L − subst-domain σ ∪
range-vars σ
  using vars-term-subst-apply-term-subset[of atm-of L] by simp

lemma vars-subst-cls-subset: vars-cls (C · σ) ⊆ vars-cls C − subst-domain σ ∪
range-vars σ
  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 · σ) ⊆ vars-cls C ∪ range-vars σ
  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 ∪ vars-cls D
  unfolding vars-cls-def by simp
```

lemma *vars-cls-add-mset*[*simp*]: *vars-cls* (*add-mset L C*) = *vars-lit L* ∪ *vars-cls C*
by (*simp add*: *vars-cls-def*)

lemma *UN-vars-term-atm-of-cls*[*simp*]: (⋃ *T*∈{*atm-of* ‘ *set-mset C*}. ⋃ (*vars-term* ‘ *T*)) = *vars-cls C*
by (*induction C*) *simp-all*

lemma *vars-lit-subst-subset-vars-cls-substI*[*intro*]:
vars-lit L ⊆ *vars-cls C* ⟹ *vars-lit* (*L* ·*l* σ) ⊆ *vars-cls* (*C* · σ)
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*:
vars-cls C ⊆ *vars-cls D* ⟹ *vars-cls* (*C* · σ) ⊆ *vars-cls* (*D* · σ)
by (*simp only*: *vars-subst-cls-eq UN-mono*)

lemma *vars-cls-subst-subset*:
assumes *range-vars-η*: *range-vars* η ⊆ *vars-lit L* ∪ *vars-lit L′*
shows *vars-cls* (*add-mset L D* · η) ⊆ *vars-cls* (*add-mset L′* (*add-mset L D*))
proof −
have *vars-cls* ((*add-mset L D*) · η) ⊆ *vars-cls* (*add-mset L D*) − *subst-domain* η ∪ *range-vars* η
by (*rule vars-subst-cls-subset*[*of add-mset L D* η])
also have ... ⊆ *vars-cls* (*add-mset L D*) − (*vars-lit L* ∪ *vars-lit L′*) ∪ *vars-lit L* ∪ *vars-lit L′*
using *range-vars-η* **by** *blast*
also have ... ⊆ *vars-cls* (*add-mset L D*) ∪ *vars-lit L′* ∪ *vars-lit L*
by *auto*
also have ... ⊆ *vars-cls D* ∪ *vars-lit L′* ∪ *vars-lit L*
by *auto*
also have ... ⊆ *vars-cls* (*add-mset L′* (*add-mset L D*))
by *auto*
finally show *?thesis*
by *assumption*
qed

definition *disjoint-vars* **where**
disjoint-vars C D ⟷ *vars-cls C* ∩ *vars-cls D* = {}

lemma *disjoint-vars-iff-inter-empty*: *disjoint-vars C D* ⟷ *vars-cls C* ∩ *vars-cls D* = {}
by (*rule disjoint-vars-def*)

hide-fact *disjoint-vars-def*

lemma *disjoint-vars-sym*: *disjoint-vars C D* ⟷ *disjoint-vars D C*
unfolding *disjoint-vars-iff-inter-empty* **by** *blast*

lemma *disjoint-vars-plus-iff*: *disjoint-vars* (*C* + *D*) *E* ⟷ *disjoint-vars C E* ∧

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

lemma *disjoint-vars-subst-clsI*:
disjoint-vars C D $\Longrightarrow$ *range-vars* σ $\cap$ *vars-cls D* $= \{\}$ $\Longrightarrow$ *disjoint-vars* $(C \cdot \sigma)$ *D*
unfolding *disjoint-vars-iff-inter-empty*
unfolding *vars-subst-cls-eq*
by (*smt* (*verit*, *best*) *Diff-subset Un-iff disjoint-iff image-cong subsetD vars-subst-cls-eq vars-subst-cls-subset*)

lemma *is-renaming-iff*: *is-renaming* $\varrho \longleftrightarrow (\forall x.$ *is-Var* $(\varrho\ x)) \wedge$ *inj* ϱ
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof (*rule iffI*)
show *?lhs* $\Longrightarrow$ *?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* $\Longrightarrow$ *?lhs*
by (*auto simp*: *is-renaming-def intro*: *ex-inverse-of-renaming*)
qed

lemma *subst-cls-idem-if-disj-vars*: *subst-domain* σ $\cap$ *vars-cls C* $= \{\}$ $\Longrightarrow$ $C \cdot \sigma = C$
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* σ $\cap$ *vars-lit L* $= \{\}$ $\Longrightarrow$ $L \cdot l\ \sigma = L$
by (*rule subst-cls-idem-if-disj-vars*[*of* - $\{\#L\#\}$, *simplified*])

lemma *subst-lit-restrict-subst-domain*: *vars-lit L* $\subseteq$ *V* $\Longrightarrow$ $L \cdot l$ *restrict-subst-domain V* $\sigma = L \cdot l\ \sigma$
by (*simp add*: *restrict-subst-domain-def same-on-vars-lit subsetD*)

lemma *subst-cls-restrict-subst-domain*: *vars-cls C* $\subseteq$ *V* $\Longrightarrow$ $C \cdot$ *restrict-subst-domain V* $\sigma = C \cdot \sigma$
by (*simp add*: *restrict-subst-domain-def same-on-vars-clause subsetD*)

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

lemma *valid-grounding-of-renaming*:
assumes *is-renaming* ϱ
shows $I \models s$ *grounding-of-cls* $(C \cdot \varrho) \longleftrightarrow I \models s$ *grounding-of-cls* C
proof −
have *grounding-of-cls* $(C \cdot \varrho)$ = *grounding-of-cls* C
by (*rule grounding-of-subst-cls-renaming-ident*[*OF* ‹*is-renaming* ϱ›])
thus *?thesis*
by *simp*
qed

lemma *is-unifier-iff-mem-unifiers-Times*:
assumes *fin-AA*: *finite AA*
shows *is-unifier* υ *AA* $\longleftrightarrow \upsilon \in$ *unifiers* $(AA \times AA)$
proof (*rule iffI*)
assume *unif-υ-AA*: *is-unifier* υ *AA*
show $\upsilon \in$ *unifiers* $(AA \times AA)$
unfolding *unifiers-def mem-Collect-eq*
proof (*rule ballI*)
have *card* $(AA \cdot_{set} \upsilon) \leq 1$
by (*rule unif-υ-AA*[*unfolded is-unifier-def subst-atms-def*])

fix p **assume** $p \in AA \times AA$
then obtain a b **where** *p-def*: $p = (a, b)$ **and** $a \in AA$ **and** $b \in AA$
by *auto*
hence *card* $(AA \cdot_{set} \upsilon) = 1$
using *fin-AA* ‹*card* $(AA \cdot_{set} \upsilon) \leq 1$› *antisym-conv2* **by** *fastforce*

hence $a \cdot a \upsilon = b \cdot a \upsilon$
using ‹$a \in AA$› ‹$b \in AA$› *fin-AA is-unifier-subst-atm-eqI unif-υ-AA* **by** *blast*
thus *fst* $p \cdot a \upsilon$ = *snd* $p \cdot a \upsilon$
by (*simp add*: *p-def*)
qed
next
assume *unif-υ-AA*: $\upsilon \in$ *unifiers* $(AA \times AA)$
show *is-unifier* υ *AA*
using *fin-AA unif-υ-AA*
proof (*induction AA arbitrary*: υ *rule*: *finite-induct*)
case *empty*
then show *?case*
by (*simp add*: *is-unifier-def*)
next
case (*insert a AA*)
from *insert.prems* **have**
υ-in: $\upsilon \in$ *unifiers* ((*insert* (a, a) (*Pair a* ‘ *AA*) ∪ $(\lambda x.\ (x, a))$ ‘ *AA*) ∪ $AA \times AA$)
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* υ $\{AA\} \longleftrightarrow$ *Unifiers.is-mgu* υ $(AA \times AA)$
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* υ $\{AA\} \longleftrightarrow$ *Unifiers.is-imgu* υ $(AA \times AA)$
by (*auto simp*: *is-imgu-def Unifiers.is-imgu-def is-unifiers-def*
is-unifier-iff-mem-unifiers-Times[*OF fin*])

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

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

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

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

lemma *subst-domain-rename-subst-domain-subset*′:
assumes *is-var-*ϱ: $\forall x.$ *is-Var* $(\varrho\ x)$

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

lemma *range-vars-eq-empty-if-is-ground*:
is-ground-cls ($C \cdot \gamma$) $\Longrightarrow$ *subst-domain* $\gamma \subseteq$ *vars-cls* $C \Longrightarrow$ *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 = *Some* μ
shows *is-imgu* μ {{t, u}}
proof −
have *unifiers* ({t, u} × {t, u}) = *unifiers* {(t, u)}
by (*auto simp*: *unifiers-def*)
hence *Unifiers.is-imgu* μ ({t, u} × {t, u})
using *mgu-sound*[*OF mgu*]
by (*simp add*: *Unifiers.is-imgu-def*)
thus *?thesis*
by (*simp add*: *is-imgu-singleton-iff-Unifiers-is-imgu-Times*)
qed

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

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

lemma *set-pairs*: *set* (*pairs* xs) = *set* xs × *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*: $a \in set\ as \Longrightarrow b \in set\ as \Longrightarrow (a,\ b) \in set\ (pairs\ as)$
by (*induction as*) *auto*

lemma *fst-pair-in-set-if-pair-in-pairs*: $p \in set\ (pairs\ as) \Longrightarrow fst\ p \in set\ as$
by (*induction as*) *auto*

lemma *snd-pair-in-set-if-pair-in-pairs*: $p \in set\ (pairs\ as) \Longrightarrow snd\ p \in set\ as$
by (*induction as*) *auto*

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

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

lemma *is-imgu-if-mgu-sets*:
assumes *mgu-AAA*: *mgu-sets* μ *AAA*
shows *is-imgu* μ *AAA*
proof −
from *mgu-AAA* **obtain** *ass xs* **where**
AAA-def: $AAA = set\ (map\ set\ ass)$ **and**
unify: $unify\ (concat\ (map\ pairs\ ass))\ [] = Some\ xs$ **and**
$subst\text{-}of\ xs = \mu$
unfolding *mgu-sets-def* **by** *auto*
hence *Unifiers.is-imgu* μ $(set\ (concat\ (map\ pairs\ ass)))$
using *unify-sound*[*OF unify*] **by** *simp*
moreover have $unifiers\ (set\ (concat\ (map\ pairs\ ass))) = \{\upsilon.\ is\text{-}unifiers\ \upsilon\ AAA\}$
unfolding *AAA-def*
proof (*rule Set.equalityI*; *rule Set.subsetI*; *unfold mem-Collect-eq*)
fix *x* **assume** *x-in*: $x \in unifiers\ (set\ (concat\ (map\ pairs\ ass)))$
show $is\text{-}unifiers\ x\ (set\ (map\ set\ ass))$
unfolding *is-unifiers-def*
proof (*rule ballI*)
fix *As* **assume** $As \in set\ (map\ set\ ass)$
hence *finite As* **by** *auto*

from ‹$As \in set\ (map\ set\ ass)$› **obtain** *as* **where**
as-in: $as \in set\ ass$ **and** *As-def*: $As = set\ as$
by *auto*

show $is\text{-}unifier\ x\ As$
unfolding *is-unifier-alt*[*OF* ‹*finite As*›]
proof (*intro ballI*)
fix *A B* **assume** $A \in As\ B \in As$
hence $\exists xs \in set\ ass.\ (A,\ B) \in set\ (pairs\ xs)$
using *as-in* **by** (*auto simp*: *As-def intro*: *pair-in-set-pairs*)

thus $A \cdot a\ x = B \cdot a\ x$
using *x-in*[*unfolded unifiers-def mem-Collect-eq*, *rule-format*, *of* (A, B), *simplified*]
by *simp*
qed
qed
next
fix x **assume** *is-unifs-x*: *is-unifiers* x (*set* (*map set ass*))
show $x \in$ *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-in*: $p \in$ *set* (*pairs as*)
by *auto*
hence *is-unif-x*: *is-unifier* x (*set as*)
using *is-unifs-x*[*unfolded is-unifiers-def*] **by** *simp*
moreover have *fst* $p \in$ *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-imgu* μ *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* $V \Longrightarrow$ *inj* (*Var* $\circ$ *renaming* V)
using *inj-compose inj-renaming* **by** (*metis inj-def term.inject*(*1*))

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

lemma *vars-term-subst-term-Var-comp-renaming-disj*:
assumes *fin-V*: *finite* V
shows *vars-term* $(t \cdot a\ (Var \circ renaming\ V)) \cap V = \{\}$
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* ·*a* (*Var* ∘ *renaming V1*)) ∩ *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* ·*l* (*Var* ∘ *renaming V*)) ∩ *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* · (*Var* ∘ *renaming V*)) ∩ *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* ⇒ - ⇒ ('*f*, -) *Term.term*
where
renaming-wrt N ≡ *Var* ∘ *renaming* (*vars-clss N*)

lemma *is-renaming-renaming-wrt*: *finite N* ⟹ *is-renaming* (*renaming-wrt N*)
by (*simp add*: *inj-Var-comp-renaming is-renaming-iff vars-clss-def*)

lemma *ex-renaming-to-disjoint-vars*:
fixes *C* :: ('*f*, '*a*) *Term.term clause* **and** *N* :: ('*f*, '*a*) *Term.term clause set*
assumes *fin*: *finite N*
shows ∃ ϱ. *is-renaming* ϱ ∧ *vars-cls* (*C* · ϱ) ∩ *vars-clss 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* · *renaming-wrt N*) ∩ *vars-clss N* = {}
by (*simp add*: *fin vars-cls-subst-renaming-disj vars-clss-def*)
qed

end

4 SCL State

type-synonym ('*f*, '*v*) *closure* = ('*f*, '*v*) *term clause* × ('*f*, '*v*) *subst*
type-synonym ('*f*, '*v*) *closure-with-lit* =
('*f*, '*v*) *term clause* × ('*f*, '*v*) *term literal* × ('*f*, '*v*) *subst*
type-synonym ('*f*, '*v*) *trail* = (('*f*, '*v*) *term literal* × ('*f*, '*v*) *closure-with-lit option*) *list*
type-synonym ('*f*, '*v*) *state* =
('*f*, '*v*) *trail* × ('*f*, '*v*) *term clause fset* × ('*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 ≡ ([], {||}, *None*)

definition *state-trail* :: ('*f*, '*v*) *state* ⇒ ('*f*, '*v*) *trail* **where**
state-trail S = *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* ⇒ ('*f*, '*v*) *term clause fset* **where**
state-learned S = *fst* (*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* ⇒ ('*f*, '*v*) *closure option* **where**
state-conflict S = *snd* (*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 *clss-of-trail-lit* **where**
clss-of-trail-lit (-, *None*) = {||} |
clss-of-trail-lit (-, *Some* (*C*, *L*, -)) = {|*add-mset L C*|}

primrec *clss-of-trail* :: ('*f*, '*v*) *trail* ⇒ ('*f*, '*v*) *term clause fset* **where**
clss-of-trail [] = {||} |
clss-of-trail (*Ln* # Γ) = *clss-of-trail-lit Ln* |∪| *clss-of-trail* Γ

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

lemma *clss-of-trail-append*: *clss-of-trail* (Γ$_0$ @ Γ$_1$) = *clss-of-trail* Γ$_0$ |∪| *clss-of-trail* Γ$_1$
by (*induction* Γ$_0$) (*simp-all add*: *funion-assoc*)

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

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

abbreviation *trail-propagate* ::
(*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term literal* ⇒ (*'f*, *'v*) *term clause* ⇒ (*'f*, *'v*) *subst* ⇒
(*'f*, *'v*) *trail* **where**
trail-propagate Γ *L C* γ ≡ *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 *clss-of-trail-trail-propagate*[*simp*]:
clss-of-trail (*trail-propagate* Γ *L C* γ) = *finsert* (*add-mset L C*) (*clss-of-trail* Γ)
unfolding *propagate-lit-def* **by** *simp*

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

abbreviation *trail-decide* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term literal* ⇒ (*'f*, *'v*) *trail*
where
trail-decide Γ *L* ≡ *decide-lit L* # Γ

lemma *clss-of-trail-trail-decide*[*simp*]:
clss-of-trail (*trail-decide* Γ *L*) = *clss-of-trail* Γ
by (*simp add*: *decide-lit-def*)

definition *is-decision-lit*
:: (*'f*, *'v*) *term literal* × (*'f*, *'v*) *closure-with-lit option* ⇒ *bool* **where**
is-decision-lit Ln ⟷ *snd Ln* = *None*

definition *trail-interp* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term interp* **where**
trail-interp Γ = ⋃((λ*L*. *case L of Pos A* ⇒ {*A*} | *Neg A* ⇒ {}) ‘ *fst* ‘ *set* Γ)

lemma *trail-interp* Γ = (⋃ *Ln* ∈ *set* Γ. *case fst Ln of Pos t* ⇒ {*t*} | *Neg t* ⇒ {})
unfolding *trail-interp-def* **by** *simp*

definition *trail-true-lit* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term literal* ⇒ *bool* **where**
trail-true-lit Γ *L* ⟷ *L* ∈ *fst* ‘ *set* Γ

definition *trail-false-lit* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term literal* ⇒ *bool* **where**
trail-false-lit Γ *L* ⟷ − *L* ∈ *fst* ‘ *set* Γ

definition *trail-true-cls* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term clause* ⇒ *bool* **where**
trail-true-cls Γ *C* ⟷ (∃ *L* ∈# *C*. *trail-true-lit* Γ *L*)

definition *trail-false-cls* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term clause* ⇒ *bool* **where**
trail-false-cls Γ *C* ⟷ (∀ *L* ∈# *C*. *trail-false-lit* Γ *L*)

definition *trail-true-clss* :: (*'f*, *'v*) *trail* ⇒ (*'f*, *'v*) *term clause set* ⇒ *bool* **where**
trail-true-clss Γ *N* ⟷ (∀ *C* ∈ *N*. *trail-true-cls* Γ *C*)

definition *trail-defined-lit* :: ('*f*, '*v*) *trail* ⇒ ('*f*, '*v*) *term literal* ⇒ *bool* **where**
trail-defined-lit Γ *L* ⟷ (*L* ∈ *fst* ‘ *set* Γ ∨ − *L* ∈ *fst* ‘ *set* Γ)

definition *trail-defined-cls* :: ('*f*, '*v*) *trail* ⇒ ('*f*, '*v*) *term clause* ⇒ *bool* **where**
trail-defined-cls Γ *C* ⟷ (∀ *L* ∈# *C*. *trail-defined-lit* Γ *L*)

lemma *trail-defined-lit-iff-true-or-false*:
trail-defined-lit Γ *L* ⟷ *trail-true-lit* Γ *L* ∨ *trail-false-lit* Γ *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* ⟹ *trail-true-cls* Γ *C* ∨ *trail-false-cls* Γ *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*) ⟷ *trail-false-lit* Γ *L* ∧ *trail-false-cls* Γ *C*
by (*auto simp*: *trail-false-cls-def*)

lemma *trail-false-cls-plus*:
trail-false-cls Γ (*C* + *D*) ⟷ *trail-false-cls* Γ *C* ∧ *trail-false-cls* Γ *D*
by (*auto simp*: *trail-false-cls-def*)

lemma *not-trail-true-Nil*[*simp*]:
¬ *trail-true-lit* [] *L*
¬ *trail-true-cls* [] *C*
N ≠ {} ⟹ ¬ *trail-true-clss* [] *N*
by (*simp-all add*: *trail-true-lit-def trail-true-cls-def trail-true-clss-def*)

lemma *not-trail-false-Nil*[*simp*]:
¬ *trail-false-lit* [] *L*
trail-false-cls [] *C* ⟷ *C* = {#}
by (*simp-all add*: *trail-false-lit-def trail-false-cls-def*)

lemma *not-trail-defined-lit-Nil*[*simp*]: ¬ *trail-defined-lit* [] *L*
by (*simp add*: *trail-defined-lit-iff-true-or-false*)

lemma *trail-false-lit-if-trail-false-suffix*:
suffix Γ′ Γ ⟹ *trail-false-lit* Γ′ *K* ⟹ *trail-false-lit* Γ *K*
by (*meson image-mono set-mono-suffix subsetD trail-false-lit-def*)

lemma *trail-false-cls-if-trail-false-suffix*:
suffix Γ′ Γ ⟹ *trail-false-cls* Γ′ *C* ⟹ *trail-false-cls* Γ *C*
using *trail-false-cls-def trail-false-lit-if-trail-false-suffix* **by** *metis*

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

lemma *trail-interp-Cons'*: *trail-interp* $(Ln \# \Gamma) = (\textit{case fst Ln of Pos } A \Rightarrow \{A\} \mid \textit{Neg } A \Rightarrow \{\}) \cup \textit{trail-interp}\ \Gamma$
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*: *trail-false-cls* $((L,\ Cl) \#\ \Gamma)\ C$ **and** *L-not-in*: $-L \notin\# C$
shows *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*: *trail-false-lit* $((L,\ Cl) \#\ \Gamma)\ M$
unfolding *trail-false-cls-def* **by** *simp*
thus *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*:
set-mset $C =$ *set-mset* $D \Longrightarrow$ *trail-false-cls* $\Gamma\ C \longleftrightarrow$ *trail-false-cls* $\Gamma\ D$
by (*simp add*: *trail-false-cls-def*)

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

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

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

shows *trail-false-cls* Γ ((*D* + {#*L*#}) · μ · σ)
unfolding *trail-false-cls-def*
proof (*rule ballI*)
fix *K*
assume *K* ∈# (*D* + {#*L*#}) · μ · σ
hence *K* ∈# *D* · μ · σ ∨ *K* = *L* ·*l* μ ·*l* σ **by** *force*
thus *trail-false-lit* Γ *K*
proof (*elim disjE*)
show *K* ∈# *D* · μ · σ ⟹ *trail-false-lit* Γ *K*
using *imgu-μ unif-σ*
by (*metis is-imgu-def subst-cls-comp-subst subst-cls-union tr-false-cls trail-false-cls-def union-iff*)
next
have *L* ·*l* μ ·*l* σ = *L* ·*l* σ
using *imgu-μ unif-σ* **by** (*metis is-imgu-def subst-lit-comp-subst*)
thus *K* = *L* ·*l* μ ·*l* σ ⟹ *trail-false-lit* Γ *K*
by (*auto intro*: *tr-false-cls*[*unfolded trail-false-cls-def*, *rule-format*])
qed
qed

lemma *trail-defined-lit-iff-defined-uminus*: *trail-defined-lit* Γ *L* ⟷ *trail-defined-lit* Γ (−*L*)
by (*auto simp add*: *trail-defined-lit-def*)

lemma *trail-defined-lit-iff*: *trail-defined-lit* Γ *L* ⟷ *atm-of L* ∈ *atm-of* ‘ *fst* ‘ *set* Γ
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* Γ = *atm-of* ‘ {*L* ∈ *fst* ‘ *set* Γ. *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* ∉ *atm-of* ‘ *fst* ‘ *set* Γ ⟹ *t* ∉ *trail-interp* Γ
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: ¬ *trail-defined-lit* Γ *L* ⟹ *trail-consistent* Γ ⟹ *trail-consistent* ((*L*, *u*) # Γ)

```
lemma distinct-atm-of-trail-if-trail-consistent:
  trail-consistent Γ ⟹ distinct (map (atm-of ∘ fst) Γ)
  by (induction Γ rule: trail-consistent.induct)
    (simp-all add: image-comp trail-defined-lit-iff)

lemma trail-consistent-appendD: trail-consistent (Γ @ Γ′) ⟹ trail-consistent Γ′
  by (induction Γ) (auto elim: trail-consistent.cases)

lemma trail-consistent-if-suffix:
  trail-consistent Γ ⟹ suffix Γ′ Γ ⟹ trail-consistent Γ′
  by (auto simp: suffix-def intro: trail-consistent-appendD)

lemma trail-interp-lit-if-trail-true:
  shows trail-consistent Γ ⟹ trail-true-lit Γ L ⟹ trail-interp Γ ⊨l L
proof (induction Γ rule: trail-consistent.induct)
  case Nil
  thus ?case
    by (simp add: trail-true-lit-def)
next
  case (Cons Γ K u)
  show ?case
  proof (cases L = K ∨ L = - K)
    case True
    then show ?thesis
    proof (elim disjE)
      assume L = K
      thus ?thesis
      proof (cases L; cases K)
        fix t_L t_K
        from ‹L = K› show L = Pos t_L ⟹ K = Pos t_K ⟹ ?thesis
          by (simp add: trail-interp-def)
      next
        fix t_L t_K
        from ‹L = K› show L = Neg t_L ⟹ K = Neg t_K ⟹ ?thesis
          using Cons.hyps(1)
          by (simp add: trail-defined-lit-iff trail-interp-Cons′
              not-in-trail-interp-if-not-in-trail)
      qed simp-all
    next
      assume L = - K
      then show ?thesis
      proof (cases L; cases K)
        fix t_L t_K
        from ‹L = - K› show L = Pos t_L ⟹ K = Neg t_K ⟹ ?thesis
          unfolding trail-interp-Cons′
          using Cons.hyps(1) Cons.prems
            by (metis (no-types, lifting) image-insert insertE list.simps(15) lit-
eral.distinct(1)
            prod.sel(1) trail-defined-lit-def trail-true-lit-def)
```

```
      next
        fix t_L t_K
        from ‹L = - K› show L = Neg t_L ⟹ K = Pos t_K ⟹ ?thesis
          unfolding trail-interp-Cons'
          using Cons.hyps(1) Cons.prems
              by (metis (no-types, lifting) image-insert insertE list.simps(15) lit-
eral.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 Γ L
      by (simp add: trail-true-lit-def)
    with Cons.IH have trail-interp Γ ⊨l L
      by simp
    with False show ?thesis
      by (cases L; cases K) (simp-all add: trail-interp-def del: true-lit-iff)
  qed
qed

lemma trail-interp-cls-if-trail-true:
  assumes trail-consistent Γ and trail-true-cls Γ C
  shows trail-interp Γ ⊨ C
proof −
  from ‹trail-true-cls Γ C› obtain L where L ∈# C and trail-true-lit Γ L
    by (auto simp: trail-true-cls-def)
  show ?thesis
    unfolding true-cls-def
  proof (rule bexI[OF - ‹L ∈# C›])
    show trail-interp Γ ⊨l L
     by (rule trail-interp-lit-if-trail-true[OF ‹trail-consistent Γ› ‹trail-true-lit Γ L›])
  qed
qed

lemma trail-true-cls-iff-trail-interp-entails:
  assumes trail-consistent Γ ∀ L ∈# C. trail-defined-lit Γ L
  shows trail-true-cls Γ C ⟷ trail-interp Γ ⊨ C
proof (rule iffI)
  assume trail-true-cls Γ C
  thus trail-interp Γ ⊨ C
    using assms(1) trail-interp-cls-if-trail-true by fast
next
  assume trail-interp Γ ⊨ C
  then obtain L where L ∈# C and trail-interp Γ ⊨l L
    by (auto simp: true-cls-def)
  show trail-true-cls Γ C
  proof (cases L)
    case (Pos t)
```

hence $t \in$ *trail-interp* Γ
using ‹*trail-interp* $\Gamma \models l\ L$› **by** *simp*
then show *?thesis*
unfolding *trail-true-cls-def*
using ‹$L \in\#\ C$› *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* ‹$L \in\#\ C$› ‹*trail-interp* $\Gamma \models l\ L$› *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* Γ $\forall L \in\#\ C$. *trail-defined-lit* Γ L
shows *trail-false-cls* Γ $C \longleftrightarrow \neg$ *trail-interp* $\Gamma \models C$
proof (*rule iffI*)
show *trail-false-cls* Γ $C \Longrightarrow \neg$ *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 $\neg$ *trail-interp* $\Gamma \models C \Longrightarrow$ *trail-false-cls* Γ 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 = $ *propagate-lit* $K\ D\ \gamma \longrightarrow$ *trail-false-cls* $\Gamma\ (D \cdot \gamma)) \Longrightarrow$
trail-closures-false $\Gamma \Longrightarrow$ *trail-closures-false* $(Kn\ \#\ \Gamma)$

lemma *trail-closures-false-ConsD*: *trail-closures-false* $(Ln\ \#\ \Gamma) \Longrightarrow$ *trail-closures-false* Γ
by (*auto elim*: *trail-closures-false.cases*)

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

lemma *is-ground-lit-if-true-in-ground-trail*:
assumes $\forall L \in$ *fst* ‘ *set* Γ. *is-ground-lit* L
shows *trail-true-lit* Γ $L \Longrightarrow$ *is-ground-lit* L
using *assms* **by** (*metis trail-true-lit-def*)

lemma *is-ground-lit-if-false-in-ground-trail*:
assumes $\forall\, L \in \mathit{fst}\ `\ \mathit{set}\ \Gamma.\ \mathit{is\text{-}ground\text{-}lit}\ L$
shows $\mathit{trail\text{-}false\text{-}lit}\ \Gamma\ L \Longrightarrow \mathit{is\text{-}ground\text{-}lit}\ L$
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 $\forall\, L \in \mathit{fst}\ `\ \mathit{set}\ \Gamma.\ \mathit{is\text{-}ground\text{-}lit}\ L$
shows $\mathit{trail\text{-}defined\text{-}lit}\ \Gamma\ L \Longrightarrow \mathit{is\text{-}ground\text{-}lit}\ L$
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 $\forall\, L \in \mathit{fst}\ `\ \mathit{set}\ \Gamma.\ \mathit{is\text{-}ground\text{-}lit}\ L$
shows $\mathit{trail\text{-}false\text{-}cls}\ \Gamma\ C \Longrightarrow \mathit{is\text{-}ground\text{-}cls}\ C$
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* :: $'v\ \mathit{set} \Rightarrow 'v \Rightarrow 'v$ +
fixes *less-B* :: $('f, 'v)\ \mathit{term} \Rightarrow ('f, 'v)\ \mathit{term} \Rightarrow \mathit{bool}$ (**infix** $\prec_B$ *50*)
assumes
finite-less-B: $\bigwedge\beta.\ \mathit{finite}\ \{x.\ x \prec_B \beta\}$
begin

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

5.1 Lemmas About $(\prec_B)$

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

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

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

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

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

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

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

lemma *finite-lit-clss-nodup-less-B*: *finite* $\{C.\ \forall L \in\# C.\ atm\text{-}of\ L \prec_B \beta \wedge count\ C\ L = 1\}$
proof −
have *1*: $(\forall L \in\# C.\ P\ L \wedge count\ C\ L = 1) \longleftrightarrow (\exists C'.\ C = mset\text{-}set\ C' \wedge finite\ C' \wedge (\forall L \in C'.\ P\ L))$
for *C P*
by (*smt* (*verit*) *count-eq-zero-iff count-mset-set' finite-set-mset finite-set-mset-mset-set multiset-eqI*)

have *2*: *finite* $\{C'.\ \forall L \in C'.\ atm\text{-}of\ L \prec_B \beta\}$
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* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒ ('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
propagateI: $C\ |\in|\ N\ |\cup|\ U \Longrightarrow C = add\text{-}mset\ L\ C' \Longrightarrow is\text{-}ground\text{-}cls\ (C \cdot \gamma) \Longrightarrow$
$\forall K \in\# C \cdot \gamma.\ atm\text{-}of\ K \preceq_B \beta \Longrightarrow$
$C_0 = \{\#K \in\# C'.\ K \cdot l\ \gamma \neq L \cdot l\ \gamma\#\} \Longrightarrow C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\} \Longrightarrow$
trail-false-cls Γ $(C_0 \cdot \gamma) \Longrightarrow \neg$ *trail-defined-lit* Γ $(L \cdot l\ \gamma) \Longrightarrow$
is-imgu μ $\{atm\text{-}of\ `\ set\text{-}mset\ (add\text{-}mset\ L\ C_1)\} \Longrightarrow$
propagate N β (Γ, *U*, *None*) (*trail-propagate* Γ $(L \cdot l\ \mu)$ $(C_0 \cdot \mu)\ \gamma$, *U*, *None*)

lemma $C\ |\in|\ N\ |\cup|\ U \Longrightarrow C = add\text{-}mset\ L\ C' \Longrightarrow is\text{-}ground\text{-}cls\ (C \cdot \gamma) \Longrightarrow$
$\forall K \in\# C.\ atm\text{-}of\ (K \cdot l\ \gamma) \preceq_B \beta \Longrightarrow$
$C_0 = \{\#K \in\# C'.\ K \cdot l\ \gamma \neq L \cdot l\ \gamma\#\} \Longrightarrow C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\} \Longrightarrow$
trail-false-cls Γ $(C_0 \cdot \gamma) \Longrightarrow \neg$ *trail-defined-lit* Γ $(L \cdot l\ \gamma) \Longrightarrow$

is-imgu μ {*atm-of* ‘ *set-mset* (*add-mset L* C_1)} $\Longrightarrow$
propagate N β (Γ, *U*, *None*) (*trail-propagate* Γ (*L* $\cdot l$ μ) (C_0 $\cdot$ μ) γ, *U*, *None*)
apply (*rule propagateI*[*of C N U L C*′ γ β - - Γ μ]; *assumption?*)
by (*metis Melem-subst-cls*)

inductive *decide* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒
('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
decideI: *is-ground-lit* (*L* $\cdot l$ γ) $\Longrightarrow$
¬ *trail-defined-lit* Γ (*L* $\cdot l$ γ) $\Longrightarrow$ *atm-of L* $\cdot a$ γ $\preceq_B$ β $\Longrightarrow$
decide N β (Γ, *U*, *None*) (*trail-decide* Γ (*L* $\cdot l$ γ), *U*, *None*)

inductive *conflict* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒
('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
conflictI: *D* |∈| *N* |∪| *U* $\Longrightarrow$ *is-ground-cls* (*D* $\cdot$ γ) $\Longrightarrow$ *trail-false-cls* Γ (*D* $\cdot$ γ)
$\Longrightarrow$
conflict N β (Γ, *U*, *None*) (Γ, *U*, *Some* (*D*, γ))

inductive *skip* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒
('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
skipI: −*L* ∉# *D* $\cdot$ σ $\Longrightarrow$
skip N β ((*L*, *n*) # Γ, *U*, *Some* (*D*, σ)) (Γ, *U*, *Some* (*D*, σ))

lemma −(*fst* $\mathcal{K}$) ∉# *D* $\cdot$ σ $\Longrightarrow$ *skip N* β ($\mathcal{K}$ # Γ, *U*, *Some* (*D*, σ)) (Γ, *U*, *Some*
(*D*, σ))
by (*metis prod.exhaust-sel skipI*)

inductive *factorize* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒
('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
factorizeI: *L* $\cdot l$ γ = *L*′ $\cdot l$ γ $\Longrightarrow$ *is-imgu* μ {{*atm-of L*, *atm-of L*′}} $\Longrightarrow$
factorize N β (Γ, *U*, *Some* (*add-mset L*′ (*add-mset L D*), γ)) (Γ, *U*, *Some*
(*add-mset L D* $\cdot$ μ, γ))

inductive *resolve* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒
('*f*, '*v*) *state* ⇒ *bool* **for** *N* β **where**
resolveI: Γ = *trail-propagate* Γ′ *K D* γ_D $\Longrightarrow$ *K* $\cdot l$ γ_D = −(*L* $\cdot l$ γ_C) $\Longrightarrow$
is-renaming ϱ_C $\Longrightarrow$ *is-renaming* ϱ_D $\Longrightarrow$
vars-cls (*add-mset L C* $\cdot$ ϱ_C) ∩ *vars-cls* (*add-mset K D* $\cdot$ ϱ_D) = {} $\Longrightarrow$
is-imgu μ {{*atm-of L* $\cdot a$ ϱ_C, *atm-of K* $\cdot a$ ϱ_D}} $\Longrightarrow$
is-grounding-merge γ
(*vars-cls* (*add-mset L C* $\cdot$ ϱ_C)) (*rename-subst-domain* ϱ_C γ_C)
(*vars-cls* (*add-mset K D* $\cdot$ ϱ_D)) (*rename-subst-domain* ϱ_D γ_D) $\Longrightarrow$
resolve N β (Γ, *U*, *Some* (*add-mset L C*, γ_C)) (Γ, *U*, *Some* ((*C* $\cdot$ ϱ_C + *D* $\cdot$
ϱ_D) $\cdot$ μ, γ))

inductive *backtrack* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒

(′f, ′v) *state* ⇒ *bool* **for** *N* *β* **where**
backtrackI: Γ = *trail-decide* (Γ′ @ Γ′′) *K* ⟹ *K* = − (*L* ·*l* *σ*) ⟹
∄*γ*. *is-ground-cls* (*add-mset L D* · *γ*) ∧ *trail-false-cls* Γ′′ (*add-mset L D* · *γ*)
⟹
backtrack N β (Γ, *U*, *Some* (*add-mset L D*, *σ*)) (Γ′′, *finsert* (*add-mset L D*) *U*, *None*)

definition *scl* :: (′f, ′v) *term clause fset* ⇒ (′f, ′v) *term* ⇒ (′f, ′v) *state* ⇒
(′f, ′v) *state* ⇒ *bool* **where**
scl N β S S′ ⟷ *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′* ∨ *backtrack N β S S′*

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

5.3 Well-Defined

lemma *propagate-well-defined*:
assumes *propagate N β S S′*
shows
¬ *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′*
¬ *backtrack N′ β′ S S′*
proof −
from *assms* **obtain** *L C γ* Γ *U* **where**
S-def: *S* = (Γ, *U*, *None*) **and**
S′-def: *S′* = (*trail-propagate* Γ *L C γ*, *U*, *None*)
by (*auto elim*: *propagate.cases*)

show ¬ *decide N′ β′ S S′*
using *S-def S′-def*
by (*auto simp add*: *decide-lit-def propagate-lit-def 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-def S′-def*

```
    by (auto elim: factorize.cases)

  show ¬ resolve N' β' S S'
    using S-def S'-def
    by (auto elim: resolve.cases)

  show ¬ backtrack N' β' S S'
    using S-def S'-def
    by (auto elim: backtrack.cases)
qed

lemma decide-well-defined:
  assumes decide N β S S'
  shows
    ¬ propagate N' β' S S'
    ¬ conflict N' β' S S'
    ¬ skip N' β' S S'
    ¬ factorize N' β' S S'
    ¬ resolve N' β' S S'
    ¬ backtrack N' β' S S'
proof −
  from assms obtain L γ Γ U where
    S-def: S = (Γ, U, None) and
    S'-def: S' = (trail-decide Γ (L ·l γ), U, None)
    by (auto elim: decide.cases)

  show ¬ propagate N' β' S S'
    using S-def S'-def
    by (auto simp add: decide-lit-def propagate-lit-def elim: propagate.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-def S'-def
    by (auto elim: factorize.cases)

  show ¬ resolve N' β' S S'
    using S-def S'-def
    by (auto elim: resolve.cases)

  show ¬ backtrack N' β' S S'
    using S-def S'-def
    by (auto elim: backtrack.cases)
```

qed

lemma *conflict-well-defined*:
  assumes *conflict N β S S′*
  shows
    ¬ *propagate N′ β′ S S′*
    ¬ *decide N′ β′ S S′*
    ¬ *skip N′ β′ S S′*
    ¬ *factorize N′ β′ S S′*
    ¬ *resolve N′ β′ S S′*
    ¬ *backtrack N′ β′ S S′*
proof −
  from *assms* **obtain** *C γ Γ U* **where**
    S-def: *S* = (Γ, *U*, *None*) **and**
    S′-def: *S′* = (Γ, *U*, *Some* (*C*, *γ*))
    by (*auto elim*: *conflict.cases*)

  show ¬ *propagate N′ β′ S S′*
    using *S-def S′-def*
    by (*auto simp add*: *decide-lit-def propagate-lit-def elim*: *propagate.cases*)

  show ¬ *decide N′ β′ S S′*
    using *S-def S′-def*
    by (*auto elim*: *decide.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-def S′-def*
    by (*auto elim*: *factorize.cases*)

  show ¬ *resolve N′ β′ S S′*
    using *S-def S′-def*
    by (*auto elim*: *resolve.cases*)

  show ¬ *backtrack N′ β′ S S′*
    using *S-def S′-def*
    by (*auto elim*: *backtrack.cases*)
qed

lemma *skip-well-defined*:
  assumes *skip N β S S′*
  shows
    ¬ *propagate N′ β′ S S′*
    ¬ *decide N′ β′ S S′*
    ¬ *conflict N′ β′ S S′*
    ¬ *factorize N′ β′ S S′*

```
    ¬ resolve N' β' S S'
    ¬ backtrack N' β' S S'
proof −
  from assms obtain Ln Γ U opt where
    S-def: S = (Ln # Γ, U, opt) and
    S'-def: S' = (Γ, U, opt)
    by (auto elim: skip.cases)

  show ¬ propagate N' β' S S'
    using S-def S'-def
    by (auto simp add: decide-lit-def propagate-lit-def 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 ¬ factorize N' β' S S'
    using S-def S'-def
    by (auto elim: factorize.cases)

  show ¬ resolve N' β' S S'
    using S-def S'-def
    by (auto elim: resolve.cases)

  show ¬ backtrack N' β' S S'
    using S-def S'-def
    by (auto elim: backtrack.cases)
qed

lemma factorize-well-defined:
  assumes factorize 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 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 Γ Γ′′ U C γ 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-def S′-def
    by (auto elim: factorize.cases)

  show ¬ resolve N′ β′ S S′
    using S-def S′-def
```

by (*auto elim*: *resolve.cases*)
qed

5.4 Some rules are right unique

lemma *right-unique-skip*: *right-unique* (*skip N β*)
proof (*rule right-uniqueI*)
fix *S S′ S″*
assume *step1*: *skip N β S S′* **and** *step2*: *skip N β S S″*
show *S′ = S″*
using *step1*
proof (*cases N β S S′ rule*: *skip.cases*)
case *hyps1*: (*skipI L D σ n Γ U*)
show *?thesis*
using *step2*[*unfolded hyps1*]
proof (*cases N β* ((*L*, *n*) # Γ, *U*, *Some* (*D*, *σ*)) *S″ rule*: *skip.cases*)
case *skipI*
with *hyps1* **show** *?thesis*
by *simp*
qed
qed
qed

5.5 Miscellaneous Lemmas

lemma *conflict-set-after-factorization*:
assumes *fact*: *factorize N β S S′* **and** *conflict-S*: *state-conflict S = Some* (*C*, *γ*)
shows ∃ *C′ γ′*. *state-conflict S′ = Some* (*C′*, *γ′*) ∧ *set-mset* (*C* · *γ*) = *set-mset* (*C′* · *γ′*)
using *fact*
proof (*cases N β S S′ rule*: *factorize.cases*)
case (*factorizeI L γ L′ μ Γ U D*)

from ‹*L* ·*l γ* = *L′* ·*l γ*› **have** *is-unifier γ* {*atm-of L*, *atm-of L′*}
by (*auto intro*!: *is-unifier-alt*[*THEN iffD2*] *intro*: *subst-atm-of-eqI*)
hence *μ* ⊙ *γ* = *γ*
using ‹*is-imgu μ* {{*atm-of L*, *atm-of L′*}}›
by (*simp add*: *is-imgu-def is-unifiers-def*)

have *L* ·*l μ* ·*l γ* = *L* ·*l γ*
using ‹*μ* ⊙ *γ* = *γ*›
by (*metis subst-lit-comp-subst*)

moreover have *D* · *μ* · *γ* = *D* · *γ*
using ‹*μ* ⊙ *γ* = *γ*›
by (*metis subst-cls-comp-subst*)

ultimately show *?thesis*
using *conflict-S*[*symmetric*]
unfolding *factorizeI*(*1*,*2*)

by (*simp add*: ‹$L \cdot l\ \gamma = L' \cdot l\ \gamma$›)
qed

lemma *not-trail-false-ground-cls-if-no-conflict*:
assumes
no-conf: $\nexists S'$. *conflict* $N\ \beta\ S\ S'$ **and**
could-conf: *state-conflict* S = *None* **and**
C-in: C |∈| N |∪| *state-learned* S **and**
gr-C-γ: *is-ground-cls* $(C \cdot \gamma)$
shows ¬ *trail-false-cls* (*state-trail* S) $(C \cdot \gamma)$
proof (*rule notI*)
assume *tr-false*: *trail-false-cls* (*state-trail* S) $(C \cdot \gamma)$

from *could-conf* **obtain** Γ U **where** *S-def*: $S = (\Gamma,\ U,\ None)$
by (*metis prod-cases3 state-conflict-simp*)

have *conflict* $N\ \beta$ (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-mempty-not-in-sate-learned*:
scl $N\ \beta\ S\ S' \Longrightarrow$ {#} |∉| *state-learned* $S \Longrightarrow$ {#} |∉| *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-mempty-in-initial-clauses-and-no-conflict*:
assumes {#} |∈| N **and** *state-conflict* S = *None*
shows *conflict* $N\ \beta\ S$ (*state-trail* S, *state-learned* S, *Some* ({#}, *Var*))
proof −
from *assms*(*2*) **obtain** Γ U **where** *S-def*: $S = (\Gamma,\ U,\ None)$
by (*metis snd-conv state-conflict-def surj-pair*)

show *?thesis*
unfolding *S-def state-trail-simp state-learned-simp*
proof (*rule conflictI*[*of* {#} N - *Var* - -, *unfolded subst-cls-empty*])
from *assms*(*1*) **show** {#} |∈| N |∪| U

```
    by simp
  qed simp-all
qed

lemma conflict-initial-state-if-mempty-in-intial-clauses:
  {#} |∈| N ⟹ conflict N β initial-state ([], {||}, Some ({#}, Var))
  using conflict-if-mempty-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 Γ = []
    unfolding conflictI(1,2) by simp
  with ‹trail-false-cls Γ (D · γ)› have D = {#}
    using not-trail-false-Nil(2) subst-cls-empty-iff by blast
  with ‹D |∈| N |∪| U› show ?thesis
    unfolding conflictI(1,2) by simp
qed

lemma conflict-empty-trail':
  assumes {#} |∈| N |∪| U
  shows ∃ S'. conflict N β ([], U, None) S'
 by (metis assms is-ground-cls-empty not-trail-false-ground-cls-if-no-conflict state-conflict-simp
     state-learned-simp subst-cls-empty trail-false-cls-mempty)

lemma mempty-in-iff-ex-conflict: {#} |∈| N |∪| U ⟷ (∃ S'. conflict N β ([], U,
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 β initial-state S
  shows ∃ γ. S = ([], {||}, Some ({#}, γ))
  using assms(1)
proof (cases rule: conflict.cases)
  case (conflictI D γ)

  from ‹trail-false-cls [] (D · γ)› have D · γ = {#}
    using not-trail-false-Nil(2) by blast
  hence D = {#}
    by simp
  thus ?thesis
    using ‹S = ([], {||}, Some (D, γ))› by simp
qed

lemma no-more-step-if-conflict-mempty:
  assumes state-trail S = [] state-conflict S = Some ({#}, γ)
```

shows $\nexists S'.\ scl\ N\ \beta\ S\ S'$
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* Γ *D* **and** *D-in*: $D \in$ *grounding-of-clss* (*fset* $N \cup$ *fset* U)
shows $\exists S'.\ conflict\ N\ \beta\ (\Gamma,\ U,\ None)\ S'$
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-clss-def grounding-of-cls-def*
by (*smt* (*verit*, *ccfv-threshold*) *D-in UN-iff grounding-ground mem-Collect-eq union-fset*)

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

have *conflict* $N\ \beta\ (\Gamma,\ U,\ None)\ (\Gamma,\ U,\ Some\ (D',\ \gamma))$
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* $N\ \beta\ (Ln\ \#\ \Gamma,\ U,\ None)\ (Ln\ \#\ \Gamma,\ U,\ Some\ (D,\ \gamma))$
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* # Γ) (*D* · γ)
using ‹*trail-false-cls* Γ (*D* · γ)›
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 ∀ *x*. *is-Var* (ϱ*C* *x*) **and**
*dom-*γ*C*: *subst-domain* γ*C* ⊆ *vars-cls* (*add-mset L C*)
shows *subst-domain* (*rename-subst-domain* ϱ*C* γ*C*) ⊆ *vars-cls* (*add-mset L C* · ϱ*C*)
proof −
have *subst-domain* (*rename-subst-domain* ϱ*C* γ*C*) ⊆ *the-Var* ‘ ϱ*C* ‘ *subst-domain* γ*C*
using *subst-domain-rename-subst-domain-subset*[*OF* ‹∀ *x*. *is-Var* (ϱ*C* *x*)›] **by** *simp*
also have ... ⊆ *the-Var* ‘ ϱ*C* ‘ *vars-cls* (*add-mset L C*)
using *dom-*γ*C* **by** *auto*
also have ... = (⋃ *x* ∈ *vars-cls* (*add-mset L C*). *vars-term* (ϱ*C* *x*))
using *image-the-Var-image-subst-renaming-eq*[*OF* ‹∀ *x*. *is-Var* (ϱ*C* *x*)›] **by** *simp*
also have ... = *vars-cls* (*add-mset L C* · ϱ*C*)
using *vars-subst-cls-eq* **by** *metis*
finally show *dom-ren-dom-*ϱ*C-*γ*C*:
subst-domain (*rename-subst-domain* ϱ*C* γ*C*) ⊆ *vars-cls* (*add-mset L C* · ϱ*C*)
by *assumption*
qed

lemma *renamed-comp-renamed-simp*:
fixes γ*C* γ*D*
assumes
K ·*l* γ*D* = − (*L* ·*l* γ*C*) **and**
ground-conf: *is-ground-cls* (*add-mset L C* · γ*C*) **and**
ground-prop: *is-ground-cls* (*add-mset K D* · γ*D*) **and**
*dom-*γ*D*: *subst-domain* γ*D* ⊆ *vars-cls* (*add-mset K D*) **and**
*ren-*ϱ*C*: *is-renaming* ϱ*C* **and**
*ren-*ϱ*D*: *is-renaming* ϱ*D* **and**
disjoint-vars: *vars-cls* (*add-mset L C* · ϱ*C*) ∩ *vars-cls* (*add-mset K D* · ϱ*D*) = {}
defines γ ≡ *rename-subst-domain* ϱ*D* γ*D* ⊙ *rename-subst-domain* ϱ*C* γ*C*
shows
subst-renamed-comp-renamed-simp:
L ·*l* ϱ*C* ·*l* γ = *L* ·*l* γ*C* *C* · ϱ*C* · γ = *C* · γ*C*

```
    K ·l ϱ_D ·l γ = K ·l γ_D  D · ϱ_D · γ = D · γ_D and
   imgu-comp-renamed-comp-renamed-simp:
    is-imgu μ {{atm-of L ·a ϱ_C, atm-of K ·a ϱ_D}} ⟹ μ ⊙ γ = γ
proof −
  have subst-adapt-ϱ_D-γ_D:
   subst-domain (rename-subst-domain ϱ_D γ_D) ∩ vars-cls (add-mset L C · ϱ_C) =
{}
    using disjoint-vars ren-ϱ_D dom-γ_D
     subst-domain-rename-subst-domain-subset-vars-cls-subst-cls
    by (metis Int-commute Orderings.order-eq-iff ground-prop is-renaming-iff
       subst-renaming-subst-adapted vars-cls-subset-subst-domain-if-grounding)

  show C · ϱ_C · γ = C · γ_C
  proof −
    have C · ϱ_C · rename-subst-domain ϱ_C γ_C = C · γ_C
    proof (rule subst-renaming-subst-adapted[OF ren-ϱ_C])
      show vars-cls C ⊆ subst-domain γ_C
        using vars-cls-subset-subst-domain-if-grounding[OF ground-conf] by simp
    qed
    moreover have C · ϱ_C · rename-subst-domain ϱ_D γ_D = C · ϱ_C
    proof (rule subst-cls-idem-if-disj-vars)
      show subst-domain (rename-subst-domain ϱ_D γ_D) ∩ vars-cls (C · ϱ_C) = {}
        using subst-adapt-ϱ_D-γ_D by auto
    qed
    ultimately show ?thesis
      unfolding γ-def by simp
  qed

  show D · ϱ_D · γ = D · γ_D
  proof −
    have D · ϱ_D · rename-subst-domain ϱ_D γ_D = D · γ_D
    proof (rule subst-renaming-subst-adapted[OF ren-ϱ_D])
      show vars-cls D ⊆ subst-domain γ_D
        using vars-cls-subset-subst-domain-if-grounding[OF ground-prop] by simp
    qed
    moreover have D · γ_D · rename-subst-domain ϱ_C γ_C = D · γ_D
      using ground-prop by simp
    ultimately show ?thesis
      unfolding γ-def by simp
  qed

  show L ·l ϱ_C ·l γ = L ·l γ_C
  proof −
    have L ·l ϱ_C ·l rename-subst-domain ϱ_C γ_C = L ·l γ_C
    proof (rule subst-lit-renaming-subst-adapted[OF ren-ϱ_C])
      show vars-lit L ⊆ subst-domain γ_C
        using ground-conf
        by (simp add: vars-lit-subset-subst-domain-if-grounding)
    qed
```

moreover have $L \cdot l\ \varrho_C \cdot l$ *rename-subst-domain* $\varrho_D\ \gamma_D = L \cdot l\ \varrho_C$
proof (*rule subst-lit-idem-if-disj-vars*)
show *subst-domain* (*rename-subst-domain* $\varrho_D\ \gamma_D$) $\cap$ *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 $K \cdot l\ \varrho_D \cdot l$ *rename-subst-domain* $\varrho_D\ \gamma_D = K \cdot l\ \gamma_D$
proof (*rule subst-lit-renaming-subst-adapted*[*OF ren-*ϱ_D])
show *vars-lit* $K \subseteq$ *subst-domain* γ_D
using *ground-prop*
by (*simp add*: *vars-lit-subset-subst-domain-if-grounding*)
qed
ultimately show *?thesis*
unfolding γ*-def*
by (*simp add*: *literal.expand*)
qed
ultimately have *atm-of* $L \cdot a\ \varrho_C \cdot a\ \gamma$ = *atm-of* $K \cdot a\ \varrho_D \cdot a\ \gamma$
using ‹$K \cdot l\ \gamma_D = -\ (L \cdot l\ \gamma_C)$›
by (*metis atm-of-subst-lit atm-of-uminus*)
hence *is-unifiers* γ {{*atm-of* $L \cdot a\ \varrho_C$, *atm-of* $K \cdot a\ \varrho_D$}}
by (*simp add*: *is-unifiers-def is-unifier-alt*)

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

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

6 Invariants

6.1 Initial Literals Generalize Learned, Trail, and Conflict Literals

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

lemma *clss-lits-generalize-clss-lits-if-superset*[*simp*]:
assumes *N2* ⊆ *N1*
shows *clss-lits-generalize-clss-lits N1 N2*
proof (*unfold clss-lits-generalize-clss-lits-def*, *rule ballI*)

fix *L*
assume *L-in*: $L \in \bigcup$ (*set-mset* ‘ *N2*)
show $\exists K \in \bigcup$ (*set-mset* ‘ *N1*). *generalizes-lit K L*
unfolding *generalizes-lit-def*
proof (*intro bexI exI conjI*)
show $L \in \bigcup$ (*set-mset* ‘ *N1*)
using *L-in* ‹*N2* ⊆ *N1*› **by** *blast*
next
show *L* ·*l Var* = *L*
by *simp*
qed
qed

lemma *clss-lits-generalize-clss-lits-subset*:
clss-lits-generalize-clss-lits N U1 ⟹ *U2* ⊆ *U1* ⟹ *clss-lits-generalize-clss-lits N U2*
unfolding *clss-lits-generalize-clss-lits-def* **by** *blast*

lemma *clss-lits-generalize-clss-lits-insert*:
clss-lits-generalize-clss-lits N (*insert C U*) ⟷
(∀ *L* ∈# *C*. ∃ *K* ∈ ⋃(*set-mset* ‘ *N*). *generalizes-lit K L*) ∧ *clss-lits-generalize-clss-lits N U*
unfolding *clss-lits-generalize-clss-lits-def* **by** *blast*

lemma *clss-lits-generalize-clss-lits-trans*:
assumes
clss-lits-generalize-clss-lits N1 N2 **and**
clss-lits-generalize-clss-lits N2 N3
shows *clss-lits-generalize-clss-lits N1 N3*
proof (*unfold clss-lits-generalize-clss-lits-def*, *rule ballI*)
fix *L3*
assume *L3* ∈ ⋃ (*set-mset* ‘ *N3*)
then obtain *L2 σ2* **where** *L2* ∈ ⋃ (*set-mset* ‘ *N2*) **and** *L2* ·*l σ2* = *L3*
using *assms*(*2*)[*unfolded clss-lits-generalize-clss-lits-def*] *generalizes-lit-def* **by** *meson*
then obtain *L1 σ1* **where** *L1* ∈ ⋃ (*set-mset* ‘ *N1*) **and** *L1* ·*l σ1* = *L2*
using *assms*(*1*)[*unfolded clss-lits-generalize-clss-lits-def*] *generalizes-lit-def* **by** *meson*

thus ∃ *K* ∈ ⋃ (*set-mset* ‘ *N1*). *generalizes-lit K L3*
unfolding *generalizes-lit-def*
proof (*intro bexI exI conjI*)
show *L1* ·*l* (*σ1* ⊙ *σ2*) = *L3*
by (*simp add*: ‹*L1* ·*l σ1* = *L2*› ‹*L2* ·*l σ2* = *L3*›)
qed *simp-all*
qed

lemma *clss-lits-generalize-clss-lits-subst-clss*:
assumes *clss-lits-generalize-clss-lits N1 N2*

shows *clss-lits-generalize-clss-lits N1* ((λ*C*. *C* · σ) ‘ *N2*)
unfolding *clss-lits-generalize-clss-lits-def*
proof (*rule ballI*)
fix *L* **assume** *L* ∈ ⋃ (*set-mset* ‘ (λ*C*. *C* · σ) ‘ *N2*)
then obtain L_2 **where** L_2 ∈ ⋃ (*set-mset* ‘ *N2*) **and** *L-def*: *L* = L_2 ·*l* σ **by** *auto*
then obtain L_1 σ_1 **where** L_1*-in*: L_1 ∈ ⋃ (*set-mset* ‘ *N1*) **and** L_2*-def*: L_2 = L_1 ·*l* σ_1
using *assms*[*unfolded clss-lits-generalize-clss-lits-def*]
unfolding *generalizes-lit-def* **by** *metis*

show ∃ *K* ∈ ⋃ (*set-mset* ‘ *N1*). *generalizes-lit K L*
unfolding *generalizes-lit-def*
proof (*intro bexI exI*)
show L_1 ∈ ⋃ (*set-mset* ‘ *N1*)
by (*rule* L_1*-in*)
next
show L_1 ·*l* (σ_1 ⊙ σ) = *L*
unfolding *L-def* L_2*-def* **by** *simp*
qed
qed

lemma *clss-lits-generalize-clss-lits-singleton-subst-cls*:
clss-lits-generalize-clss-lits N {*C*} ⟹ *clss-lits-generalize-clss-lits N* {*C* · σ}
by (*rule clss-lits-generalize-clss-lits-subst-clss*[*of N* {*C*} σ, *simplified*])

lemma *clss-lits-generalize-clss-lits-subst-cls*:
assumes *clss-lits-generalize-clss-lits N* {*add-mset L1* (*add-mset L2 C*)}
shows *clss-lits-generalize-clss-lits N* {*add-mset* (*L1* ·*l* σ) (*C* · σ)}
proof (*rule clss-lits-generalize-clss-lits-trans*)
show *clss-lits-generalize-clss-lits N* {*add-mset L1* (*add-mset L2 C*) · σ}
by (*rule clss-lits-generalize-clss-lits-singleton-subst-cls*[*of N - σ*, *OF assms*])
next
show *clss-lits-generalize-clss-lits* {*add-mset L1* (*add-mset L2 C*) · σ}
{*add-mset* (*L1* ·*l* σ) (*C* · σ)}
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 ⟷ *clss-lits-generalize-clss-lits* (*fset N*)
(*fset* (*state-learned S* |∪| *clss-of-trail* (*state-trail S*) |∪|
(*case state-conflict S of None* ⇒ {||} | *Some* (*C*, -) ⇒ {|*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 β S S' ⟹ initial-lits-generalize-learned-trail-conflict N S ⟹
    initial-lits-generalize-learned-trail-conflict N S'
proof (induction S S' rule: propagate.induct)
  case (propagateI C U L C' γ C₀ C₁ Γ μ)

  from propagateI.prems have
    N-generalize: clss-lits-generalize-clss-lits (fset N) (fset (U |∪| clss-of-trail Γ))
    unfolding initial-lits-generalize-learned-trail-conflict-def by simp-all

  from propagateI.hyps have
    C-in: C |∈| N |∪| U and
    C-def: C = add-mset L C' and
    C₀-def: C₀ = {#K ∈# C'. K ·l γ ≠ L ·l γ#} 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 L_D where D |∈| N and L_D ∈# D and generalizes-lit L_D K'
        using K'-in C-in N-generalize[unfolded clss-lits-generalize-clss-lits-def]
        by (metis (mono-tags, opaque-lifting) UN-iff funion-iff generalizes-lit-refl)

      show ∃ K'∈⋃ (set-mset ‘ fset N). generalizes-lit K' K
      proof (rule bexI)
        show generalizes-lit L_D K
          using ‹generalizes-lit L_D K'›
          by (metis generalizes-lit-def K-def subst-lit-comp-subst)
      next
        show ‹L_D ∈ ⋃ (set-mset ‘ fset N)›
          using ‹D |∈| N› ‹L_D ∈# 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 β S S′ rule*: *conflict.cases*)
  case (*conflictI D U γ Γ*)
  from *assms*(*2*) **have** *clss-lits-generalize-clss-lits* (*fset N*) (*fset* (*U* |∪| *clss-of-trail* Γ))
    unfolding *conflictI*(*1*) **by** (*simp add*: *initial-lits-generalize-learned-trail-conflict-def*)
  hence *ball-U-Γ-generalize*:
    ⋀*L*. *L* ∈ ⋃ (*set-mset* ‘ *fset* (*U* |∪| *clss-of-trail* Γ)) ⟹
      ∃ *K* ∈ ⋃ (*set-mset* ‘ *fset N*). *generalizes-lit K L*
    unfolding *clss-lits-generalize-clss-lits-def* **by** *blast*

  have *clss-lits-generalize-clss-lits* (*fset N*) (*insert D* (*fset* (*U* |∪| *clss-of-trail* Γ)))
    unfolding *clss-lits-generalize-clss-lits-def*
  proof (*rule ballI*)
    fix *L* **assume** *L* ∈ ⋃ (*set-mset* ‘ *insert D* (*fset* (*U* |∪| *clss-of-trail* Γ)))
    hence *L* ∈ *set-mset D* ∨ *L* ∈ ⋃ (*set-mset* ‘ (*fset* (*U* |∪| *clss-of-trail* Γ)))
      by *simp*
    thus ∃ *K* ∈ ⋃ (*set-mset* ‘ *fset N*). *generalizes-lit K L*
    proof (*elim disjE*)
      assume *L-in*: *L* ∈# *D*
      show *?thesis*
        using ‹*D* |∈| *N* |∪| *U*›[*unfolded funion-iff*]
      proof (*elim disjE*)
        show *D* |∈| *N* ⟹ *?thesis*
          using *L-in*
          by (*meson UN-I generalizes-lit-refl*)
      next
        assume *D* |∈| *U*
        hence ∃ *K* ∈ ⋃ (*set-mset* ‘ *fset N*). *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* ∈ ⋃ (*set-mset* ‘ *fset* (*U* |∪| *clss-of-trail* Γ)) ⟹ *?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 β S S' ⟹ initial-lits-generalize-learned-trail-conflict N S ⟹
    initial-lits-generalize-learned-trail-conflict N S'
proof (induction S S' rule: skip.induct)
  case (skipI L D σ Cl Γ U)
  then show ?case
    unfolding initial-lits-generalize-learned-trail-conflict-def
     unfolding state-learned-simp state-conflict-simp state-trail-simp option.case
prod.case
    by (auto elim: clss-lits-generalize-clss-lits-subset)
qed

lemma factorize-preserves-initial-lits-generalize-learned-trail-conflict:
  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.prems 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 · ϱC · 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 · ϱD · 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 Are Ground

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

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

lemma *propagate-preserves-trail-lits-ground*:
assumes *propagate N* β *S S′* **and** *trail-lits-ground S*
shows *trail-lits-ground S′*
using *assms*(*1*)
proof (*cases N* β *S S′ rule*: *propagate.cases*)
case (*propagateI C U L C′* γ C_0 C_1 Γ μ)
hence *is-ground-lit* (*L* ·*l* γ)
by (*meson Melem-subst-cls is-ground-cls-def mset-subset-eqD mset-subset-eq-add-right union-single-eq-member*)

moreover have $\forall \tau$. *is-unifiers* τ {*atm-of* ‘ *set-mset* (*add-mset L* C_1)} $\longrightarrow$ τ = $\mu \odot \tau$
using ‹*is-imgu* μ {*atm-of* ‘ *set-mset* (*add-mset L* C_1)}›
by (*simp add*: *is-imgu-def*)

moreover have *is-unifiers* γ {*atm-of* ‘ *set-mset* (*add-mset L* C_1)}
by (*auto simp*: *is-unifiers-def is-unifier-alt* ‹C_1 = {#*K* $\in$# *C′*. *K* ·*l* γ = *L* ·*l* γ#}›
intro: *subst-atm-of-eqI*)

ultimately have *is-ground-lit* (*L* ·*l* μ ·*l* γ)
by (*metis subst-lit-comp-subst*)

moreover have $\forall$ *L* $\in$ *fst* ‘ *set* Γ. *is-ground-lit L*
using ‹*trail-lits-ground S*› **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* β *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 ∀ L ∈ fst ‘ set Γ. 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
```

6.3 Trail Literals Are Defined Only Once

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

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

lemma *propagate-preserves-trail-lits-consistent*:
assumes *propagate N β S S′* **and** *invar*: *trail-lits-consistent S*
shows *trail-lits-consistent S′*
using *assms*(*1*)
proof (*cases N β S S′ rule*: *propagate.cases*)
case (*propagateI C U L C′ γ C*$_0$ *C*$_1$ Γ *μ*)

have *L* ·*l* *μ* ·*l* *γ* = *L* ·*l* *γ*
proof −
have *is-unifiers γ* {*atm-of* ‘ *set-mset* (*add-mset L C*$_1$)}
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 *γ* = *μ* ⊙ *γ*
using ‹*is-imgu μ* {*atm-of* ‘ *set-mset* (*add-mset L C*$_1$)}›
by (*simp add*: *is-imgu-def*)
thus *?thesis*
by (*metis subst-lit-comp-subst*)
qed
hence ¬ *trail-defined-lit* Γ (*L* ·*l* *μ* ·*l* *γ*)
using ‹¬ *trail-defined-lit* Γ (*L* ·*l* *γ*)› **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 β 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 β 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 β 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 β 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 β 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 β S S′* **and** *invar*: *trail-lits-consistent S*
shows *trail-lits-consistent S′*
using *assms*
by (*auto simp*: *trail-lits-consistent-def decide-lit-def elim*!: *backtrack.cases*
elim!: *trail-consistent-if-suffix intro*: *suffixI*)

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

lemma *trail-consistent-iff*: *trail-consistent* Γ ⟷ (∀Γ′ *Ln* Γ″. Γ = Γ″ @ *Ln* # Γ′
⟶ ¬ *trail-defined-lit* Γ′ (*fst Ln*))
proof (*intro iffI allI impI*)
fix Γ′ *Ln* Γ″
assume *trail-consistent* Γ **and** Γ = Γ″ @ *Ln* # Γ′
thus ¬ *trail-defined-lit* Γ′ (*fst Ln*)
proof (*induction* Γ *arbitrary*: Γ″ *rule*: *trail-consistent.induct*)
case *Nil*
thus *?case*
by *simp*
next
case *ind-hyps*: (*Cons* Γ *L u*)

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

6.4 Trail Closures Are False In Subtrails

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

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

proof (*cases N β 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 β 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 β 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 β 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 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*)
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* $\longleftrightarrow$
($\forall$ *K D γ Γ′ Γ″*. *Γ* = *Γ″* @ *propagate-lit K D γ* # *Γ′* $\longrightarrow$ *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 $\forall$ *K D γ Γ′ Γ″*. *Γ* = *Γ″* @ *trail-propagate Γ′ K D γ* $\longrightarrow$ *trail-false-cls Γ′* (*D* · *γ*)
thus *trail-closures-false Γ*
by (*induction Γ*) (*simp-all add*: *trail-closures-false.Cons*)
qed

6.5 Trail Literals Were Propagated or Decided

inductive *trail-propagated-or-decided* **for** *N β U* **where**
Nil[*simp*]: *trail-propagated-or-decided N β U* [] |
Propagate:
C |∈| *N* |∪| *U* $\Longrightarrow$
C = *add-mset L C′* $\Longrightarrow$
is-ground-cls (*C* · *γ*) $\Longrightarrow$

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

lemma *trail-propagate-or-decide-suffixI*:
assumes *trail-propagated-or-decided N β U ys* **and** *suffix xs ys*
shows *trail-propagated-or-decided N β U xs*
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′ γ C₀ C₁ Γ μ*)
from *Propagate.prems* **obtain** *zs* **where**
tr-prop-eq: $trail\text{-}propagate\ \Gamma\ (L \cdot l\ \mu)\ (C_0 \cdot \mu)\ \gamma = zs\ @\ xs$
by (*auto simp*: *suffix-def*)
show *?case*
proof (*cases zs*)
case *Nil*
with *tr-prop-eq* **have** $xs = trail\text{-}propagate\ \Gamma\ (L \cdot l\ \mu)\ (C_0 \cdot \mu)\ \gamma$
by *simp*
then show *?thesis*
by (*simp add*: *trail-propagated-or-decided.Propagate*[*OF Propagate.hyps*])
next
case (*Cons Ln Γ′*)
with *tr-prop-eq* **have** *suffix xs Γ*
by (*simp add*: *suffix-def propagate-lit-def*)
thus *?thesis*
by (*rule Propagate.IH*)
qed
next
case (*Decide L γ Γ*)
from *Decide.prems* **obtain** *zs* **where**
tr-deci-eq: $trail\text{-}decide\ \Gamma\ (L \cdot l\ \gamma) = zs\ @\ xs$
by (*auto simp*: *suffix-def*)

```
  show ?case
  proof (cases zs)
    case Nil
    with tr-deci-eq have xs = trail-decide Γ (L ·l γ)
      by simp
    then show ?thesis
      by (simp add: trail-propagated-or-decided.Decide[OF Decide.hyps])
  next
    case (Cons Ln Γ')
    with tr-deci-eq have suffix xs Γ
      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 β S =
    trail-propagated-or-decided N β (state-learned S) (state-trail S)

lemma trail-propagated-or-decided-learned-finsert:
  assumes trail-propagated-or-decided N β U Γ
  shows trail-propagated-or-decided N β (finsert C U) Γ
  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' γ D0 D1 Γ μ)

  from Propagate.hyps have D-in: D |∈| N |∪| finsert C U
    by simp

  have IH: trail-propagated-or-decided N β (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 γ Γ)
  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 β U (Γ1 @ Γ2)
  shows trail-propagated-or-decided N β U Γ2
  using assms
```

proof (*induction* Γ_1 @ Γ_2 *arbitrary*: Γ_1 Γ_2 *rule*: *trail-propagated-or-decided.induct*)
case *Nil*
thus *?case*
by *simp*
next
case (*Propagate* C L C' γ C_0 C_1 Γ μ)
hence *tr-prop-eq-*Γ_1*-*Γ_2:
Γ_1 @ Γ_2 = *trail-propagate* Γ (L ·*l* μ) (C_0 · μ) γ
by *simp*
thus *?case*
unfolding *propagate-lit-def append-eq-Cons-conv*
proof (*elim disjE conjE exE*)
assume Γ_1 = [] **and** Γ_2*-def*: Γ_2 = (L ·*l* μ ·*l* γ, *Some* (C_0 · μ, L ·*l* μ, γ)) # Γ
show *?thesis*
unfolding Γ_2*-def*
by (*rule trail-propagated-or-decided.Propagate*[*unfolded propagate-lit-def*];
rule Propagate.hyps)
next
fix Γ_1'
assume Γ_1 = (L ·*l* μ ·*l* γ, *Some* (C_0 · μ, L ·*l* μ, γ)) # Γ_1' **and** Γ_1' @ Γ_2 = Γ
thus *?thesis*
using *Propagate.hyps* **by** *blast*
qed
next
case (*Decide* L γ Γ)
hence Γ_1 @ Γ_2 = *trail-decide* Γ (L ·*l* γ)
by *simp*
thus *?case*
unfolding *decide-lit-def append-eq-Cons-conv*
proof (*elim disjE conjE exE*)
assume Γ_1 = [] **and** Γ_2*-def*: Γ_2 = (L ·*l* γ, *None*) # Γ
show *?thesis*
unfolding Γ_2*-def*
by (*rule trail-propagated-or-decided.Decide*[*unfolded decide-lit-def*]; *rule Decide.hyps*)
next
fix Γ_1' **assume** Γ_1 = (L ·*l* γ, *None*) # Γ_1' **and** Γ_1' @ Γ_2 = Γ
then show *?thesis*
using *Decide.hyps* **by** *blast*
qed
qed

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

lemma *propagate-preserves-trail-propagated-or-decided*:
assumes *propagate* N β S S' **and** *trail-propagated-or-decided'* N β S
shows *trail-propagated-or-decided'* N β S'

```
  using assms(1)
proof (cases N β S S' rule: propagate.cases)
  case (propagateI C U L C' γ C₀ C₁ Γ μ)

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

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

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

  from decideI(1) assms(2) have IH: trail-propagated-or-decided N β U Γ
    by (simp add: trail-propagated-or-decided'-def)
  show ?thesis
    unfolding decideI(2)
    apply (simp add: trail-propagated-or-decided'-def)
    by (rule trail-propagated-or-decided.Decide[rotated -1, OF IH])
      (rule decideI)+
qed

lemma conflict-preserves-trail-propagated-or-decided:
  assumes conflict N β S S' and invar: trail-propagated-or-decided' N β S
  shows trail-propagated-or-decided' N β S'
  using assms by (auto simp: trail-propagated-or-decided'-def elim: conflict.cases)

lemma skip-preserves-trail-propagated-or-decided:
  assumes skip N β S S' and invar: trail-propagated-or-decided' N β S
  shows trail-propagated-or-decided' N β S'
  using assms(1)
proof (cases N β S S' rule: skip.cases)
  case (skipI L D σ n Γ U)

  from invar have trail-propagated-or-decided N β U ((L, n) # Γ)
    unfolding skipI(1) by (simp add: trail-propagated-or-decided'-def)
  hence trail-propagated-or-decided N β U Γ
    by (cases N β U (L, n) # Γ 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 $\Gamma \longleftrightarrow (\forall (L_\gamma,\ n) \in set\ \Gamma.$
case n of
None $\Rightarrow$ *True*
$\mid$ *Some* (-, L, γ) $\Rightarrow L_\gamma = L \cdot l\ \gamma$)

lemma *trail-propagated-wf-iff*:
  trail-propagated-wf Γ ⟷ (∀ *Ln* ∈ *set* Γ. ∀ *D K* γ. *snd Ln* = *Some* (*D*, *K*, γ) ⟶ *fst Ln* = *K* ·*l* γ)
  (**is** *?lhs* ⟷ *?rhs*)
proof (*rule iffI*)
  show *?lhs* ⟹ *?rhs*
    unfolding *trail-propagated-wf-def*
    by *fastforce*
next
  assume *?rhs*
  show *?lhs*
    unfolding *trail-propagated-wf-def*
  proof (*rule ballI*)
    fix *K* **assume** *K* ∈ *set* Γ
    show *case K of* (*L*$_\gamma$, *None*) ⇒ *True* | (*L*$_\gamma$, *Some* (*x*, *L*, γ)) ⇒ *L*$_\gamma$ = *L* ·*l* γ
      unfolding *case-prod-beta*
      using ‹*?rhs*›[*rule-format*, *OF* ‹*K* ∈ *set* Γ›]
      by (*cases snd K*) *auto*
  qed
qed

lemma *trail-propagated-wf-if-trail-propagated-or-decided*:
  trail-propagated-or-decided N U β Γ ⟹ *trail-propagated-wf* Γ
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'* γ *C*$_0$ *C*$_1$ Γ μ)
  then show *?case*
    by (*simp add*: *trail-propagated-wf-def propagate-lit-def*)
next
  case (*Decide L* γ Γ)
  then show *?case*
    by (*simp add*: *trail-propagated-wf-def decide-lit-def*)
qed

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

lemma *trail-propagated-lit-wf-initial-state*:
  ∀ *K*∈*set* (*state-trail initial-state*). ∀ *D K* γ. *snd K* = *Some* (*D*, *K*, γ) ⟶ *fst K* = *K* ·*l* γ
  by *simp*

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

assumes *step*: *scl N β S S′* **and**
inv: ∀ *K* ∈ *set* (*state-trail S*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
shows ∀ *K* ∈ *set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *step inv*
unfolding *scl-def*
proof (*elim disjE*)
assume *propagate N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv*
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 β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv*
by (*smt* (*verit*) *Pair-inject decide.simps decide-lit-def option.discI set-ConsD sndI state-simp*)
next
assume *conflict N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv conflict.simps* **by** *fastforce*
next
assume *skip N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv skip.simps* **by** *fastforce*
next
assume *factorize N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv factorize.simps* **by** *fastforce*
next
assume *resolve N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv*
by (*smt* (*verit*) *resolve.cases state-trail-simp*)
next
assume *backtrack N β S S′*
thus ∀ *K*∈*set* (*state-trail S′*). ∀ *D K γ*. *snd K* = *Some* (*D*, *K*, *γ*) ⟶ *fst K* = *K* ·*l γ*
using *inv*
by (*smt* (*verit*, *del-insts*) *Un-iff fst-conv insertCI list.simps*(15) *backtrack.cases set-append state-trail-def*)

qed

6.6 Trail Atoms Are Less Than Bound

definition *trail-atoms-lt* **where**
trail-atoms-lt β $S \longleftrightarrow (\forall A \in$ *atm-of* ‘ *fst* ‘ *set* (*state-trail* S). $A \preceq_B \beta$)

lemma *trail-atoms-lt-initial-state*[*simp*]: *trail-atoms-lt* β *initial-state*
by (*simp add*: *trail-atoms-lt-def*)

lemma *propagate-preserves-trail-atoms-lt*:
assumes *propagate* N β S S' **and** *trail-atoms-lt* β S
shows *trail-atoms-lt* β S'
using *assms*(*1*)
proof (*cases* N β S S' *rule*: *propagate.cases*)
case (*propagateI* C U L C' γ C_0 C_1 Γ μ)
hence *is-ground-lit* ($L \cdot l$ γ)
by (*meson Melem-subst-cls is-ground-cls-def mset-subset-eqD mset-subset-eq-add-right*
union-single-eq-member)

moreover have $\forall \tau$. *is-unifiers* τ {*atm-of* ‘ *set-mset* (*add-mset* L C_1)} $\longrightarrow \tau = \mu \odot \tau$
using ‹*is-imgu* μ {*atm-of* ‘ *set-mset* (*add-mset* L C_1)}›
by (*simp add*: *is-imgu-def*)

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

ultimately have *is-ground-lit* ($L \cdot l$ $\mu \cdot l$ γ)
by (*metis subst-lit-comp-subst*)

have *atm-of* $L \cdot a$ $\mu \cdot a$ γ = *atm-of* $L \cdot a$ γ
proof −
from *propagateI* **have** *is-unifiers* γ {*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 ‹*trail-atoms-lt* β *S*› **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* β *S S'* **and** *trail-atoms-lt* β *S*
shows *trail-atoms-lt* β *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* β *S S'* **and** *trail-atoms-lt* β *S*
shows *trail-atoms-lt* β *S'*
using *assms* **by** (*auto simp*: *trail-atoms-lt-def elim*!: *conflict.cases*)

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

lemma *factorize-preserves-trail-atoms-lt*:
assumes *factorize N* β *S S'* **and** *trail-atoms-lt* β *S*
shows *trail-atoms-lt* β *S'*
using *assms* **by** (*auto simp*: *trail-atoms-lt-def elim*!: *factorize.cases*)

lemma *resolve-preserves-trail-atoms-lt*:
assumes *resolve N* β *S S'* **and** *trail-atoms-lt* β *S*
shows *trail-atoms-lt* β *S'*
using *assms* **by** (*auto simp*: *trail-atoms-lt-def elim*!: *resolve.cases*)

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

6.7 Trail Resolved Literals Have Unique Polarity

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

$C \cdot \gamma$)

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 ‹$C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\}$›
by (*auto simp add*: *is-unifiers-def is-unifier-alt intro*: *subst-atm-of-eqI*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu μ* {*atm-of* ‘ *set-mset* (*add-mset L C₁*)}›
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 ‹$C_0 = \{\#K \in\# C'.\ K \cdot l\ \gamma \neq L \cdot l\ \gamma\#\}$› ‹¬ *trail-defined-lit Γ* ($L \cdot l\ \gamma$)›
‹*trail-false-cls Γ* ($C_0 \cdot \gamma$)›
by (*metis trail-defined-lit-iff-defined-uminus trail-defined-lit-iff-true-or-false*
trail-false-cls-def)

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

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

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

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

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

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

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

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

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

6.8 Trail And Conflict Closures Are Ground

definition *ground-closures* **where**
ground-closures S ⟷
(∀ *Ln* ∈ *set* (*state-trail S*). ∀ *C L γ*. *snd Ln* = *Some* (*C*, *L*, *γ*) ⟶ *is-ground-cls* (*add-mset L C* · *γ*)) ∧
(∀ *C γ*. *state-conflict S* = *Some* (*C*, *γ*) ⟶ *is-ground-cls* (*C* · *γ*))

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

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

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

have *C-def*: $C = \mathit{add\text{-}mset}\ L\ (C_0 + C_1)$
using *propagateI*(*3*−) **by** *auto*

have *is-unifiers γ* {*atm-of* ‘ *set-mset* (*add-mset L C₁*)}
unfolding ‹$C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\}$›
by (*auto simp add*: *is-unifiers-def is-unifier-alt intro*: *subst-atm-of-eqI*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu μ* {*atm-of* ‘ *set-mset* (*add-mset L C₁*)}›
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 *is-ground-cls* (*add-mset L C₀* · *μ* · *γ*)
using ‹*is-ground-cls* (*C* · *γ*)›
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 β S S′* **and** *invar*: *ground-closures S*
shows *ground-closures S′*
using *step*
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-imgu μ* {{*atm-of L*, *atm-of L′*}}›
by (*simp add*: *is-imgu-def is-unifiers-def*)

have *add-mset L D* · *μ* · *γ* = *add-mset L D* · *γ*
using ‹*μ* ⊙ *γ* = *γ*›
by (*metis subst-cls-comp-subst*)
hence *is-ground-cls* (*add-mset L D* · *μ* · *γ*)
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-ϱC: *is-renaming ϱC* **and**
ren-ϱD: *is-renaming ϱD* **and**
disjoint-vars: *vars-cls* (*C* · *ϱC*) ∩ *vars-cls* (*D* · *ϱD*) = {} **and**
ground-conf: *is-ground-cls* (*C* · *γC*) **and**
ground-prop: *is-ground-cls* (*D* · *γD*) **and**
merge-γ: *is-grounding-merge γ*
(*vars-cls* (*C* · *ϱC*)) (*rename-subst-domain ϱC γC*)
(*vars-cls* (*D* · *ϱD*)) (*rename-subst-domain ϱD γD*)
shows
∀ *L* ∈# *C*. *L* ·*l ϱC* ·*l γ* = *L* ·*l γC*
∀ *K* ∈# *D*. *K* ·*l ϱD* ·*l γ* = *K* ·*l γD*
proof −
have ∀ *x*∈*vars-cls* (*C* · *ϱC*). *vars-term* (*rename-subst-domain ϱC γC x*) = {}
using *ground-conf ren-ϱC*
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 ∀ *x*∈*vars-cls* (*D* · *ϱD*). *vars-term* (*rename-subst-domain ϱD γD*
x) = {}
using *ground-prop ren-ϱD*
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: $\forall\, x{\in}$*vars-cls* $(C \cdot \varrho_C)$. $\gamma\ x =$ *rename-subst-domain* $\varrho_C\ \gamma_C\ x$
and
ball-D-ϱ_D-apply-γ: $\forall\, x{\in}$*vars-cls* $(D \cdot \varrho_D)$. $\gamma\ x =$ *rename-subst-domain* $\varrho_D\ \gamma_D\ x$
using *disjoint-vars merge-γ*
unfolding *is-grounding-merge-def*
by *simp-all*

show $\forall\, L \in\# C.\ L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$
proof (*rule ballI*)
fix L **assume** *L-in*: $L \in\# C$
show $L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$
unfolding *subst-lit-comp-subst*[*symmetric*]
proof (*intro same-on-vars-lit ballI*)
fix x **assume** $x \in$ *vars-lit* L
moreover obtain x' **where** $\varrho_C\ x =$ *Var* x'
using *ren-ϱ_C*
by (*meson is-Var-def is-renaming-iff*)
ultimately have $x' \in$ *vars-lit* $(L \cdot l\ \varrho_C)$
using *vars-subst-lit-eq* **by** *fastforce*
hence $\gamma\ x' =$ *rename-subst-domain* $\varrho_C\ \gamma_C\ x'$
using *ball-C-ϱ_C-apply-γ L-in multi-member-split* **by** *force*
thus $(\varrho_C \odot \gamma)\ x = \gamma_C\ x$
apply (*simp add*: *subst-compose-def* ‹$\varrho_C\ x =$ *Var* x'›)
by (*metis* (*no-types*, *opaque-lifting*) *L-in Un-iff* ‹$\varrho_C\ x =$ *Var* x'› ‹$x \in$ *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 $\forall\, K \in\# D.\ K \cdot l\ \varrho_D \cdot l\ \gamma = K \cdot l\ \gamma_D$
proof (*rule ballI*)
fix K **assume** *K-in*: $K \in\# D$
show $K \cdot l\ \varrho_D \cdot l\ \gamma = K \cdot l\ \gamma_D$
unfolding *subst-lit-comp-subst*[*symmetric*]
proof (*intro same-on-vars-lit ballI*)
fix x **assume** $x \in$ *vars-lit* K
moreover obtain x' **where** $\varrho_D\ x =$ *Var* x'
using *ren-ϱ_D*
by (*meson is-Var-def is-renaming-iff*)
ultimately have $x' \in$ *vars-lit* $(K \cdot l\ \varrho_D)$
using *vars-subst-lit-eq* **by** *fastforce*
hence $\gamma\ x' =$ *rename-subst-domain* $\varrho_D\ \gamma_D\ x'$
using *ball-D-ϱ_D-apply-γ K-in multi-member-split* **by** *force*
thus $(\varrho_D \odot \gamma)\ x = \gamma_D\ x$
apply (*simp add*: *subst-compose-def* ‹$\varrho_D\ x =$ *Var* x'›)
by (*metis* (*no-types*, *opaque-lifting*) *K-in UnI1* ‹$\varrho_D\ x =$ *Var* x'› ‹$x \in$ *vars-lit*

K›
ground-prop image-eqI is-renaming-iff multi-member-split ren-ϱ_D rename-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 $\forall x.$ *is-Var* ($\varrho_C\ x$) **and** *inj* ϱ_C **and** $\forall x.$ *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: $\forall Ln \in$ *set* $\Gamma.\ \forall C\ L\ \gamma.$ *snd* Ln = *Some* $(C, L, \gamma) \longrightarrow$ *is-ground-cls* (*add-mset* $L\ C \cdot \gamma$)
unfolding *resolveI*(*1*,*2*) ‹Γ = *trail-propagate* $\Gamma'\ K\ D\ \gamma_D$›
by (*simp-all add*: *propagate-lit-def ground-closures-def*)
hence
$\forall L \in \#$ *add-mset* $L\ C.\ L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$
$\forall K \in \#$ *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* $L \cdot a\ \varrho_C \cdot a\ \gamma$ = *atm-of* $K \cdot a\ \varrho_D \cdot a\ \gamma$
using ‹$K \cdot l\ \gamma_D = -\ (L \cdot l\ \gamma_C)$›
‹$\forall L \in \#$ *add-mset* $L\ C.\ L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$›[*rule-format*, *of* L, *simplified*]
‹$\forall K \in \#$ *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* γ {{*atm-of* $L \cdot a\ \varrho_C$, *atm-of* $K \cdot a\ \varrho_D$}}
by (*simp add*: *is-unifiers-def is-unifier-alt*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu* μ {{*atm-of* $L \cdot a\ \varrho_C$, *atm-of* $K \cdot a\ \varrho_D$}}›
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 ‹$\forall L \in \#$ *add-mset* $L\ C.\ L \cdot l\ \varrho_C \cdot l\ \gamma = L \cdot l\ \gamma_C$› ‹$\forall K \in \#$ *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 $(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu \cdot \gamma = C \cdot \gamma_C + D \cdot \gamma_D$
by (*metis subst-cls-comp-subst subst-cls-union*)
thus *?thesis*
using *ground-conf ground-prop min-ground-clo*-Γ
unfolding *resolveI*
by (*simp add*: *ground-closures-def*)
qed

lemma *backtrack-preserves-ground-closures*:
assumes *step*: *backtrack* $N\ \beta\ S\ S'$ **and** *invar*: *ground-closures* S
shows *ground-closures* S'
using *assms*
by (*cases* $N\ \beta\ S\ S'$ *rule*: *backtrack.cases*)
(*simp add*: *ground-closures-def decide-lit-def ball-Un*)

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

6.9 Trail And Conflict Closures Are Ground And False

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

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

lemma *propagate-preserves-ground-false-closures*:
assumes *step*: *propagate* $N\ \beta\ S\ S'$ **and** *invar*: *ground-false-closures* S
shows *ground-false-closures* S'
using *step*
proof (*cases* $N\ \beta\ S\ S'$ *rule*: *propagate.cases*)
case *step-hyps*: (*propagateI* $C\ U\ L\ C'\ \gamma\ C_0\ C_1\ \Gamma\ \mu$)

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 ∀ *C γ*. *state-conflict S′* = *Some* (*C*, *γ*) ⟶ *trail-false-cls* (*state-trail S′*) (*C* · *γ*)
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 β S S′* **and** *invar*: *ground-false-closures S*
shows *ground-false-closures S′*
using *step*
proof (*cases N β S S′ rule*: *decide.cases*)
case *step-hyps*: (*decideI L γ Γ U*)

have *ground-closures S′*
using *invar decide-preserves-ground-closures*[*OF step*]
by (*metis ground-false-closures-def*)

moreover have *trail-closures-false* (*state-trail S′*)
using *invar decide-preserves-trail-closures-false′*[*OF step*]
by (*metis ground-false-closures-def trail-closures-false′-def*)

moreover have ∀ *C γ*. *state-conflict S′* = *Some* (*C*, *γ*) ⟶ *trail-false-cls* (*state-trail S′*) (*C* · *γ*)
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 β S S′* **and** *invar*: *ground-false-closures S*
shows *ground-false-closures S′*
using *step*
proof (*cases N β S S′ rule*: *conflict.cases*)
case *step-hyps*: (*conflictI D U γ Γ*)

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 ∀ C γ. state-conflict S' = Some (C, γ) ⟶ trail-false-cls (state-trail
S') (C · γ)
    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 ∀ C γ. state-conflict S' = Some (C, γ) ⟶ trail-false-cls (state-trail
S') (C · γ)
    using invar
    unfolding step-hyps(1,2)
    using ‹− L ∉# D · σ›
    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 ∀ *C γ*. *state-conflict S′* = *Some* (*C*, *γ*) ⟶ *trail-false-cls* (*state-trail S′*) (*C* · *γ*)
using *invar conflict-set-after-factorization*[*OF step*]
unfolding *step-hyps*(*1*,*2*) *ground-false-closures-def*
by (*auto simp del*: *set-mset-add-mset-insert dest*: *trail-false-cls-ignores-duplicates*)

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

lemma *resolve-preserves-ground-false-closures*:
assumes *step*: *resolve N β S S′* **and** *invar*: *ground-false-closures S*
shows *ground-false-closures S′*
using *step*
proof (*cases N β S S′ rule*: *resolve.cases*)
case *step-hyps*: (*resolveI* Γ Γ′ *K D γ*$_D$ *L γ*$_C$ *ϱ*$_C$ *ϱ*$_D$ *C μ γ U*)
hence Γ-*def*: Γ = *trail-propagate* Γ′ *K D γ*$_D$
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 ∀ *C γ*. *state-conflict S′* = *Some* (*C*, *γ*) ⟶ *trail-false-cls* (*state-trail S′*) (*C* · *γ*)
proof −
from ‹*ground-closures S*› **have**
ground-conf: *is-ground-cls* (*add-mset L C* · *γ*$_C$) **and**
ground-prop: *is-ground-cls* (*add-mset K D* · *γ*$_D$)
unfolding *step-hyps*(*1*,*2*)
by (*simp-all add*: *ground-closures-def* Γ-*def propagate-lit-def*)
hence
∀ *L*∈#*add-mset L C*. *L* ·*l ϱ*$_C$ ·*l γ* = *L* ·*l γ*$_C$
∀ *K*∈#*add-mset K D*. *K* ·*l ϱ*$_D$ ·*l γ* = *K* ·*l γ*$_D$
using *merge-of-renamed-groundings step-hyps*(*3*−)
by *metis*+

have *atm-of L ·a ϱ_C ·a γ = atm-of K ·a ϱ_D ·a γ*
using ‹*K ·l γ_D = − (L ·l γ_C)*›
‹∀ *L*∈#*add-mset L C*. *L ·l ϱ_C ·l γ = L ·l γ_C*›[*rule-format*, *of L*, *simplified*]
‹∀ *K*∈#*add-mset K D*. *K ·l ϱ_D ·l γ = K ·l γ_D*›[*rule-format*, *of K*, *simplified*]
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
hence *is-unifiers γ* {{*atm-of L ·a ϱ_C*, *atm-of K ·a ϱ_D*}}
by (*simp add*: *is-unifiers-def is-unifier-alt*)
hence *μ* ⊙ *γ = γ*
using ‹*is-imgu μ* {{*atm-of L ·a ϱ_C*, *atm-of K ·a ϱ_D*}}›
by (*auto simp*: *is-imgu-def*)
hence *C · ϱ_C · μ · γ = C · γ_C* **and** *D · ϱ_D · μ · γ = D · γ_D*
using ‹∀ *L*∈#*add-mset L C*. *L ·l ϱ_C ·l γ = L ·l γ_C*› ‹∀ *K*∈#*add-mset K D*.
K ·l ϱ_D ·l γ = K ·l γ_D›
by (*metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst*
subst-lit-comp-subst)+

moreover have *trail-false-cls* Γ (*C · γ_C*)
using *invar*
unfolding *step-hyps*(*1*,*2*)
by (*simp add*: *ground-false-closures-def trail-false-cls-add-mset*)

moreover have *trail-false-cls* Γ (*D · γ_D*)
proof (*rule trail-false-cls-if-trail-false-suffix*)
show *suffix* Γ′ Γ
unfolding ‹Γ = *trail-propagate* Γ′ *K D γ_D*›
by (*simp add*: *suffix-def*)
next
show *trail-false-cls* Γ′ (*D · γ_D*)
using *invar*
unfolding *step-hyps*(*1*,*2*) ‹Γ = *trail-propagate* Γ′ *K D γ_D*›
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 β S S′* **and** *invar*: *ground-false-closures S*
shows *ground-false-closures S′*
using *step*
proof (*cases N β S S′ rule*: *backtrack.cases*)
case *step-hyps*: (*backtrackI* Γ Γ′ Γ″ *K L σ 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 $\forall\, C\ \gamma.$ *state-conflict* $S' =$ *Some* $(C, \gamma) \longrightarrow$ *trail-false-cls* (*state-trail* S') $(C \cdot \gamma)$
unfolding *step-hyps*(*1*,*2*) **by** *simp*

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

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

6.10 Learned Clauses Are Non-empty

definition *learned-nonempty* **where**
learned-nonempty $S \longleftrightarrow \{\#\} \notin$ *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 β 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 β 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 β 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 β 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 β S S′* **and** *learned-nonempty S*
shows *learned-nonempty S′*
using *assms* **unfolding** *scl-def*
using *propagate-preserves-learned-nonempty decide-preserves-learned-nonempty*
conflict-preserves-learned-nonempty skip-preserves-learned-nonempty
factorize-preserves-learned-nonempty resolve-preserves-learned-nonempty
backtrack-preserves-learned-nonempty
by *metis*

6.11 Backtrack Follows Conflict Resolution

definition *conflict-resolution* **where**
conflict-resolution N β S ⟷ (*state-conflict S* ≠ *None* ⟶
(∃ *S0 S1*. *conflict N β S0 S1* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S1 S*))

lemma *conflict-resolution-initial-state*[*simp*]: *conflict-resolution N β initial-state*
by (*simp add*: *conflict-resolution-def*)

lemma *propagate-preserves-conflict-resolution*:
assumes *step*: *propagate N β S S′*
shows *conflict-resolution N β S′*
using *step* **by** (*auto simp*: *conflict-resolution-def elim*: *propagate.cases*)

lemma *decide-preserves-conflict-resolution*:
assumes *step*: *decide N β S S′*
shows *conflict-resolution N β S′*
using *step* **by** (*auto simp*: *conflict-resolution-def elim*: *decide.cases*)

```
lemma conflict-preserves-conflict-resolution:
  assumes step: conflict N β S S′
  shows conflict-resolution N β S′
  using step unfolding conflict-resolution-def by blast

lemma skip-preserves-conflict-resolution:
  assumes step: skip N β S S′ and invar: conflict-resolution N β S
  shows conflict-resolution N β S′
  using step
proof −
  from step have state-conflict S ≠ None
    by (auto elim: skip.cases)
  with invar obtain S0 S1 where
    conflict N β S0 S1 and (skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S
    by (auto simp: conflict-resolution-def)
  show ?thesis
    unfolding conflict-resolution-def
  proof (intro impI exI conjI)
    show conflict N β S0 S1
      by (rule ‹conflict N β S0 S1›)
  next
    show (skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S′
      using ‹(skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S› step
      by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl sup2CI)
  qed
qed

lemma factorize-preserves-conflict-resolution:
  assumes step: factorize N β S S′ and invar: conflict-resolution N β S
  shows conflict-resolution N β S′
  using step
proof −
  from step have state-conflict S ≠ None
    by (auto elim: factorize.cases)
  with invar obtain S0 S1 where
    conflict N β S0 S1 and (skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S
    by (auto simp: conflict-resolution-def)
  show ?thesis
    unfolding conflict-resolution-def
  proof (intro impI exI conjI)
    show conflict N β S0 S1
      by (rule ‹conflict N β S0 S1›)
  next
    show (skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S′
      using ‹(skip N β ⊔ factorize N β ⊔ resolve N β)** S1 S› step
      by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl sup2CI)
  qed
qed
```

lemma *resolve-preserves-conflict-resolution*:
  assumes *step*: *resolve N β S S′* **and** *invar*: *conflict-resolution N β S*
  shows *conflict-resolution N β S′*
  using *step*
proof −
  from *step* **have** *state-conflict S ≠ None*
    by (*auto elim*: *resolve.cases*)
  with *invar* **obtain** *S0 S1* **where**
    conflict N β S0 S1 **and** (*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S1 S*
    by (*auto simp*: *conflict-resolution-def*)
  show *?thesis*
    unfolding *conflict-resolution-def*
  proof (*intro impI exI conjI*)
    show *conflict N β S0 S1*
      by (*rule* ‹*conflict N β S0 S1*›)
  next
    show (*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S1 S′*
      using ‹(*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S1 S*› *step*
      by (*metis* (*no-types*, *opaque-lifting*) *rtranclp.rtrancl-into-rtrancl sup2CI*)
  qed
qed

lemma *backtrack-preserves-conflict-resolution*:
  assumes *step*: *backtrack N β S S′*
  shows *conflict-resolution N β S′*
  using *step* **by** (*auto simp*: *conflict-resolution-def elim*: *backtrack.cases*)

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

6.12 Miscellaneous Lemmas

lemma *before-conflict*:
  assumes *conflict N β S1 S2* **and**
    invars: *learned-nonempty S1 trail-propagated-or-decided′ N β S1*
  shows {#} |∈| *N* ∨ (∃ *S0*. *propagate N β S0 S1*) ∨ (∃ *S0*. *decide N β S0 S1*)
  using *assms*
proof (*cases N β S1 S2 rule*: *conflict.cases*)
  case (*conflictI D U γ* Γ)
  with *invars*(*2*) **have** *trail-propagated-or-decided N β U* Γ
    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 · γ = {#}
      using ‹trail-false-cls Γ (D · γ)› not-trail-false-Nil(2) by blast
    hence D = {#}
      by (simp add: local.conflictI(4))
    moreover from invars(1) have {#} |∉| U
      by (simp add: conflictI(1) learned-nonempty-def)
    ultimately have {#} |∈| N
      using ‹D |∈| N |∪| U› by simp
    thus ?thesis by simp
  next
    case (Propagate C L C' γ_C C_0 C_1 Γ' μ)
    hence ∃ S0. propagate N β S0 S1
      unfolding conflictI(1)
      using propagateI by blast
    thus ?thesis by simp
  next
    case (Decide L γ_L Γ')
    hence ∃ S0. decide N β S0 S1
      unfolding conflictI(1)
      using decideI by blast
    thus ?thesis by simp
  qed
qed

lemma before-backtrack:
  assumes backt: backtrack N β Sn Sm and
    invar: conflict-resolution N β Sn
  shows ∃ S0 S1. conflict N β S0 S1 ∧ (skip N β ⊔ factorize N β ⊔ resolve N
β)** S1 Sn
  using backt
proof (cases N β Sn Sm 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 ⟹ trail-atoms-lt β S ⟹ ∀ L ∈# C. atm-of L
≺_B β
  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-𝒢* (**infix** $\models_{\mathcal{G}e}$ *50*) **where**
entails-𝒢 N U ≡ *grounding-of-clss N* $\models_{e}$ *grounding-of-clss U*

definition *sound-trail* **where**
sound-trail N Γ ⟷
(∀ *Ln* ∈ *set* Γ. ∀ *D K* γ. *snd Ln* = *Some* (*D*, *K*, γ) ⟶ *fset N* $\models_{\mathcal{G}e}$ {*add-mset K D*})

lemma *sound-trail-Nil*[*simp*]: *sound-trail N* []
by (*simp add*: *sound-trail-def*)

lemma *entails-𝒢-mono*: *N* $\models_{\mathcal{G}e}$ *U* ⟹ *N* ⊆ *NN* ⟹ *NN* $\models_{\mathcal{G}e}$ *U*
by (*meson grounding-of-clss-mono true-clss-mono*)

lemma *sound-trail-supersetI*: *sound-trail N* Γ ⟹ *N* |⊆| *NN* ⟹ *sound-trail NN* Γ
unfolding *sound-trail-def*
by (*meson entails-𝒢-mono less-eq-fset.rep-eq*)

lemma *sound-trail-ConsD*: *sound-trail N* (*Ln* # Γ) ⟹ *sound-trail N* Γ
by (*simp add*: *sound-trail-def*)

lemma *sound-trail-appendD*: *sound-trail N* (Γ @ Γ′) ⟹ *sound-trail N* Γ′
by (*induction* Γ) (*auto intro*: *sound-trail-ConsD*)

lemma *sound-trail-propagate*:
assumes
*sound-*Γ: *sound-trail N* Γ **and**
N-entails-C-L: *fset N* $\models_{\mathcal{G}e}$ {*C* + {#*L*#}}
shows *sound-trail N* (*trail-propagate* Γ *L C* σ)
using *assms*
by (*simp add*: *sound-trail-def propagate-lit-def*)

lemma *sound-trail-decide*:
sound-trail N Γ ⟹ *sound-trail N* (*trail-decide* Γ *L*)
by (*simp add*: *sound-trail-def decide-lit-def*)

7.2 Sound State

definition *sound-state* :: ('*f*, '*v*) *term clause fset* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *state* ⇒ *bool* **where**
sound-state N β *S* ⟷
(∃Γ *U u*. *S* = (Γ, *U*, *u*) ∧ *sound-trail N* Γ ∧ *fset N* $\models_{\mathcal{G}e}$ *fset U* ∧
(*case u of None* ⇒ *True* | *Some* (*C*, γ) ⇒ *fset N* $\models_{\mathcal{G}e}$ {*C*}))

7.3 Initial State Is Sound

lemma *sound-initial-state*[*simp*]: *sound-state N β initial-state*
by (*simp add*: *sound-state-def trail-atoms-lt-def*)

7.4 SCL Rules Preserve Soundness

lemma *mem-vars-cls-subst-clsD*: $x' \in$ *vars-cls* $(C \cdot \varrho) \Longrightarrow \exists x \in$ *vars-cls* $C.\ x' \in$ *vars-term* $(\varrho\ x)$
unfolding *vars-subst-cls-eq*
using *subst-domain-def* **by** *force*

lemma *propagate-preserves-sound-state*:
assumes *step*: *propagate N β S S′* **and** *sound*: *sound-state N β S*
shows *sound-state N β S′*
using *assms*(*1*)
proof (*cases N β S S′ rule*: *propagate.cases*)
case (*propagateI C U L C′ γ C*$_0$ *C*$_1$ Γ *μ*)
hence
S-def: $S = (\Gamma,\ U,\ None)$ **and**
S′-def: $S' =$ (*trail-propagate* Γ $(L \cdot l\ \mu)$ $(C_0 \cdot \mu)$ $\gamma,\ U,\ None$) **and**
C-in: $C\ |\in|\ N\ |\cup|\ U$ **and**
C-def: $C =$ *add-mset L C′* **and**
gr-C-γ: *is-ground-cls* $(C \cdot \gamma)$ **and**
C_0*-def*: $C_0 = \{\#K \in\# C'.\ K \cdot l\ \gamma \neq L \cdot l\ \gamma\#\}$ **and**
C_1*-def*: $C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\}$ **and**
Γ*-false-*C_0*-γ*: *trail-false-cls* Γ $(C_0 \cdot \gamma)$ **and**
*undef-*Γ*-L-γ*: ¬ *trail-defined-lit* Γ $(L \cdot l\ \gamma)$ **and**
imgu-μ: *is-imgu* μ {*atom-of* ‘ *set-mset* (*add-mset L C*$_1$)}
by *simp-all*

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

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

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

have *fin-atm-*C_1: *finite* (*atom-of* ‘ (*set-mset C*$_1$))
by *blast*
hence *is-unifier* γ (*atom-of* ‘ (*set-mset* (*add-mset L C*$_1$)))
by (*auto simp add*: *is-unifier-alt* C_1*-def intro*: *subst-atm-of-eqI*)
hence *μ-γ-simp*: $\mu \odot \gamma = \gamma$
using *imgu-μ*[*unfolded is-imgu-def*, *THEN conjunct2*]

using *is-unifiers-def* **by** *fastforce*

have $L \cdot l\ \mu \cdot l\ \gamma = L \cdot l\ \gamma$
using μ-γ-*simp* **by** (*metis subst-lit-comp-subst*)

have $C_0 \cdot \mu \cdot \gamma = C_0 \cdot \gamma$
using μ-γ-*simp* **by** (*metis subst-cls-comp-subst*)

have *sound-trail N* (*trail-propagate* Γ $(L \cdot l\ \mu)$ $(C_0 \cdot \mu)$ γ)
proof (*rule sound-trail-propagate*)
show *sound-trail N* Γ
by (*rule sound-*Γ)
next
from *C-in N-entails-U* **have** *fset N* $\models\mathcal{G}e$ $\{C\}$
by (*metis* (*no-types*, *opaque-lifting*) *empty-subsetI funion-iff grounding-of-clss-mono insert-subset true-clss-mono*)
hence *fset N* $\models\mathcal{G}e$ $\{$*add-mset L* $(C_0 + C_1)\}$
by (*simp add*: *C-def* C_0-*def* C_1-*def*)
hence *fset N* $\models\mathcal{G}e$ $\{$*add-mset L* $(C_0 + C_1) \cdot \mu\}$
by (*metis* (*no-types*, *opaque-lifting*) *grounding-of-clss-singleton subst-cls-eq-grounding-of-cls-subset-eq true-clss-mono*)
hence *fset N* $\models\mathcal{G}e$ $\{$*add-mset L* $C_0 \cdot \mu\}$
proof (*rule entails-trans*)
have $\exists$ $C \in$ *grounding-of-clss* $\{(C_0 + C_1 + \{\#L\#\}) \cdot \mu\}$. *set-mset D* = *set-mset C*
if *D-in*: $D \in$ *grounding-of-clss* $\{(C_0 + \{\#L\#\}) \cdot \mu\}$ **for** *I D*
proof−
from *D-in* **obtain** σ **where**
D-def: D = *add-mset L* $C_0 \cdot \mu \cdot \sigma$ **and** *gr-*σ: *is-ground-subst* σ
by (*auto simp*: *grounding-of-cls-def*)
show *?thesis*
proof (*rule bexI*)
from *imgu-*μ **have** *is-unifier* μ (*atm-of* ‘ *set-mset* (*add-mset L* C_1))
by (*simp add*: *is-imgu-def is-unifiers-def*)
hence $\forall$ *A*$\in$*atm-of* ‘ *set-mset* (*add-mset L* C_1). $\forall$ *B*$\in$*atm-of* ‘ *set-mset* (*add-mset L* C_1).
A $\cdot a$ μ = *B* $\cdot a$ μ
using *is-unifier-alt*
by (*metis* (*mono-tags*, *opaque-lifting*) *finite-imageI finite-set-mset*)
hence $\forall$ *A*$\in$*atm-of* ‘ *set-mset* C_1. *A* $\cdot a$ μ = *atm-of L* $\cdot a$ μ
by (*metis image-insert insert-iff set-mset-add-mset-insert*)
hence $\forall$ *A*$\in$*set-mset* C_1. *A* $\cdot l$ μ = *L* $\cdot l$ μ
unfolding C_1-*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* $((C_0 + C_1 + \{\#L\#\}) \cdot \mu)$ = *set-mset* $((C_0 + \{\#L\#\}) \cdot$

μ)
        by *auto*
      thus *set-mset D* = *set-mset* ((*C*$_0$ + *C*$_1$ + {#*L*#}) · μ · σ)
        unfolding *D-def set-mset-subst-cls-conv*[*of* - σ]
        by *simp*
    next
      show (*C*$_0$ + *C*$_1$ + {#*L*#}) · μ · σ ∈ *grounding-of-clss* {(*C*$_0$ + *C*$_1$ + {#*L*#}) · μ}
      using *gr-*σ **by** (*auto simp*: *grounding-of-cls-def*)
    qed
  qed
  then show {*add-mset L* (*C*$_0$ + *C*$_1$) · μ} ⟹$\mathcal{G}e$ {*add-mset L C*$_0$ · μ}
    by (*auto elim*: *true-clss-if-set-mset-eq*[*rotated*])
 qed
 thus *fset N* ⟹$\mathcal{G}e$ {*C*$_0$ · μ + {#*L* ·*l* μ#}}
 by (*metis* (*no-types*, *opaque-lifting*) *add-mset-add-single grounding-of-clss-singleton*
     subst-cls-add-mset)
qed
thus *?thesis*
 unfolding *S'-def sound-state-def*
 using *N-entails-U* **by** *simp*
qed

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

 moreover have *sound-trail N* (*trail-decide* Γ (*L* ·*l* γ))
  by (*simp add*: *local.decideI*(*4*) *local.decideI*(*5*) *sound-*Γ *sound-trail-decide*)

 ultimately show *?thesis*
  unfolding *decideI sound-state-def* **by** *simp*
qed

lemma *conflict-preserves-sound-state*:
 assumes *step*: *conflict N* β *S S'* **and** *sound*: *sound-state N* β *S*
 shows *sound-state N* β *S'*
 using *assms*(*1*)
proof (*cases N* β *S S' rule*: *conflict.cases*)
 case (*conflictI D U* γ Γ)
 hence *D-in*: *D* |∈| *N* |∪| *U* **by** *simp*

```
  from conflictI(1) sound have
    sound-Γ: sound-trail N Γ and
    N-entails-U: fset N ⊨Ge fset U
    unfolding sound-state-def by auto

  have N-entails-D': fset N ⊨Ge {D}
  proof (intro allI impI)
    fix I
    assume valid-N: I ⊨s grounding-of-clss (fset N)
    then show I ⊨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 grounding-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 ⊨Ge fset U and
    N-entails-D-L-L': fset N ⊨Ge {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-eq-L'-γ: L ·l γ = L' ·l γ by simp

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

  from imgu-μ have is-unifier μ {atm-of L, atm-of L'}
   by (auto simp add: is-unifiers-def dest: is-imgu-is-mgu[THEN is-mgu-is-unifiers])
  hence L-eq-L'-μ: L ·l μ = L' ·l μ
    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 ⊨𝒢e {(D + {#L#}) · μ}
  proof (rule entails-trans)
    show fset N ⊨𝒢e {D + {#L, L'#}}
      by (rule N-entails-D-L-L')
  next
    have ∗: {(D + {#L, L'#}) · μ} = {D · μ + {#L ·l μ, L ·l μ#}}
      by (simp add: L-eq-L'-μ)

    have {D + {#L, L'#}} ⊨𝒢e {(D + {#L, L'#}) · μ}
      using subst-cls-eq-grounding-of-cls-subset-eq true-clss-mono
      by (metis (mono-tags, lifting) grounding-of-clss-singleton)

    moreover have {(D + {#L, L'#}) · μ} ⊨𝒢e {(D + {#L#}) · μ}
      apply (intro allI impI)
      by (erule true-clss-if-set-mset-eq[rotated]) (auto simp add: L-eq-L'-μ ground-
ing-of-cls-def)

    ultimately show {D + {#L, L'#}} ⊨𝒢e {(D + {#L#}) · μ}
      by simp
  qed
  thus ?thesis
    unfolding factorizeI sound-state-def
    using sound-Γ N-entails-U by simp
qed

lemma resolve-preserves-sound-state:
  assumes step: resolve N β S S' and sound: sound-state N β S
  shows sound-state N β S'
  using assms(1)
proof (cases N β S S' rule: resolve.cases)
  case (resolveI Γ Γ' K D γ_D L γ_C ϱ_C ϱ_D C μ γ U)
  hence
    Γ-def: Γ = trail-propagate Γ' K D γ_D and
    imgu-μ: is-imgu μ {{atm-of L ·a ϱ_C, atm-of K ·a ϱ_D}}
    by simp-all
```

from *sound* **have**
sound-Γ: *sound-trail N Γ* **and**
N-entails-U: *fset N* $\models\mathcal{G}e$ *fset U* **and**
N-entails-conf: *fset N* $\models\mathcal{G}e$ {*add-mset L C*}
unfolding *resolveI*(*1*,*2*) *sound-state-def* **by** *simp-all*

from *sound-Γ* **have**
N-entails-prop: *fset N* $\models\mathcal{G}e$ {*add-mset K D*}
unfolding *sound-trail-def Γ-def*
by (*simp add*: *propagate-lit-def*)

moreover have *fset N* $\models\mathcal{G}e$ $\{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$
proof −
have ∗: *fset N* $\models\mathcal{G}e$ {*add-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-cls-subset true-clss-mono*)

have ∗∗: *fset N* $\models\mathcal{G}e$ {*add-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-cls-subset true-clss-mono*)

show *fset N* $\models\mathcal{G}e$ $\{(C \cdot \varrho_C + D \cdot \varrho_D) \cdot \mu\}$
unfolding *true-clss-def*
proof (*intro allI impI ballI*)
fix *I E*
assume *I-entails*: $\forall E \in$ *grounding-of-clss* (*fset N*). *I* $\models$ *E* **and**
E-in: *E* $\in$ *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-cls-def* **by** *auto*

have *I* $\models l$ *L* $\cdot l$ ϱ_C $\cdot l$ μ $\cdot l$ γ_E $\vee$ $(\exists x \in\#$ *C* $\cdot$ $\varrho_C \cdot \mu \cdot \gamma_E$. *I* $\models l$ *x*)
proof −
have *add-mset L C* $\cdot$ $\varrho_C \cdot \mu \cdot \gamma_E \in$ *grounding-of-clss* {*add-mset L C* $\cdot$ ϱ_C $\cdot$ μ}
using *gr-*γ_E **unfolding** *grounding-of-clss-def grounding-of-cls-def* **by** *auto*
hence $\exists K \in\#$ *add-mset L C* $\cdot$ $\varrho_C \cdot \mu \cdot \gamma_E$. *I* $\models l$ *K*
using ∗[*rule-format*, *unfolded true-clss-def*, *OF I-entails*]
by (*metis true-cls-def*)
thus *?thesis*
by *simp*
qed

moreover have *I* $\models l$ *K* $\cdot l$ ϱ_D $\cdot l$ μ $\cdot l$ γ_E $\vee$ $(\exists x \in\#$ *D* $\cdot$ $\varrho_D \cdot \mu \cdot \gamma_E$. *I* $\models l$ *x*)
proof −
have *add-mset K D* $\cdot$ $\varrho_D \cdot \mu \cdot \gamma_E \in$ *grounding-of-clss* {*add-mset K D* $\cdot$ ϱ_D $\cdot$ μ}

```
      using gr-γ_E unfolding grounding-of-clss-def grounding-of-cls-def by auto
    hence ∃K ∈# add-mset K D · ϱ_D · μ · γ_E. I ⟨=l K
      using ∗∗[rule-format, unfolded true-clss-def, OF I-entails]
      by (metis true-cls-def)
    thus ?thesis
      by simp
  qed

  ultimately show I ⟨= E
  proof (elim disjE)
    assume I ⟨=l L ·l ϱ_C ·l μ ·l γ_E and I ⟨=l K ·l ϱ_D ·l μ ·l γ_E
    moreover have atm-of L ·a ϱ_C ·a μ = atm-of K ·a ϱ_D ·a μ
      using imgu-μ[unfolded is-imgu-def]
     by (meson finite.emptyI finite.insertI insertCI is-unifier-alt is-unifiers-def)
    ultimately have False
      using ‹K ·l γ_D = − (L ·l γ_C)›
      by (cases L; cases K; simp add: uminus-literal-def subst-lit-def)
    thus ?thesis ..
  qed (auto simp: E-def)
 qed
qed

lemma backtrack-preserves-sound-state:
 assumes step: backtrack 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: backtrack.cases)
 case (backtrackI Γ Γ' Γ'' K L σ D U)
 from backtrackI(1) sound have
   sound-Γ: sound-trail N Γ and
   N-entails-U: fset N ⟨=Ge fset U and
   N-entails-D-L-L': fset N ⟨=Ge {D + {#L#}}
   unfolding sound-state-def by simp-all

 from backtrackI have Γ-def: Γ = trail-decide (Γ' @ Γ'') (− (L ·l σ)) by simp

 moreover have sound-trail N Γ''
 proof −
   from sound-Γ have sound-trail N Γ
     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* ⟨=*Ge* (*fset U* ∪ {*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 β S S′* ⟹ *sound-state N β S* ⟹ *sound-state N β 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 β S S′ ⟷
scl N β S S′ ∧ (*decide N β S S′* ⟶ ¬(∃ *S″*. *conflict N β S′ S″*))

lemma *scl-if-reasonable*: *reasonable-scl N β S S′* ⟹ *scl N β S S′*
unfolding *reasonable-scl-def scl-def* **by** *simp*

definition *regular-scl* **where**
regular-scl N β S S′ ⟷
conflict N β S S′ ∨ ¬ (∃ *S″*. *conflict N β S S″*) ∧ *reasonable-scl N β S S′*

lemma *reasonable-if-regular*:
regular-scl N β S S′ ⟹ *reasonable-scl N β S S′*
unfolding *regular-scl-def*
proof (*elim disjE conjE*)
assume *conflict N β S S′*
hence *scl N β S S′*
by (*simp add*: *scl-def*)
moreover have *decide N β S S′* ⟶ ¬(∃ *S″*. *conflict N β S′ S″*)
by (*smt* (*verit*, *best*) ‹*conflict N β S S′*› *conflict.cases option.distinct*(*1*) *snd-conv*)
ultimately show *reasonable-scl N β S S′*
by (*simp add*: *reasonable-scl-def*)
next
assume ¬ (∃ *S″*. *conflict N β S S″*) **and** *reasonable-scl N β S S′*
thus *?thesis* **by** *simp*
qed

lemma *scl-if-regular*:
regular-scl N β S S′ ⟹ *scl N β 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 β S S′* ⟷ *reasonable-scl N β S S′* ∧
((∃ *S″*. *conflict N β S S″*) ⟶ *conflict N β S S′*)
(**is** *?lhs* = *?rhs*)
proof (*rule iffI*)
assume *?lhs*
thus *?rhs*
unfolding *regular-scl-def*
proof (*elim disjE conjE*)
assume *conf*: *conflict N β S S′*
show *?rhs*
proof (*rule conjI*)
from *conf* **have** ¬ *decide N β S S′*
by (*simp add*: *conflict-well-defined*(*2*))
with *conf* **show** *reasonable-scl N β S S′*
unfolding *reasonable-scl-def scl-def* **by** *simp*
next
show (∃ *S″*. *conflict N β S S″*) ⟶ *conflict N β S S′*
using *conf* **by** *simp*
qed
next
assume ¬ (∃ *S″*. *conflict N β S S″*) **and** *reasonable-scl N β S S′*
thus *?rhs*
by *simp*
qed
next
assume *?rhs*
thus *?lhs*
proof (*cases* ∃ *S″*. *conflict N β S S″*)
case *True*
with ‹*?rhs*› **have** *conflict N β S S′*
by *blast*
then show *?thesis*
unfolding *regular-scl-def*
by *simp*
next
case *False*
then show *?thesis*
unfolding *regular-scl-def*
using ‹*?rhs*› **by** *simp*
qed
qed

definition *ex-conflict* **where**
ex-conflict C Γ ⟷ (∃ *γ*. *is-ground-cls* (*C* · *γ*) ∧ *trail-false-cls Γ* (*C* · *γ*))

definition *is-shortest-backtrack* **where**
is-shortest-backtrack C Γ Γ$_0$ ⟷ *C* ≠ {#} ⟶ *suffix* Γ$_0$ *Γ* ∧ ¬ *ex-conflict C* Γ$_0$
∧
(∀ *Kn*. *suffix* (*Kn* # Γ$_0$) *Γ* ⟶ *ex-conflict C* (*Kn* # Γ$_0$))

definition *shortest-backtrack-strategy* **where**
shortest-backtrack-strategy R N β S S′ ⟷ *R N β S S′* ∧ (*backtrack N β S S′*
⟶
is-shortest-backtrack (*fst* (*the* (*state-conflict S*))) (*state-trail S*) (*state-trail S′*))

lemma *regular-scl-if-shortest-backtrack-strategy*:
shortest-backtrack-strategy regular-scl N β S S′ ⟹ *regular-scl N β S S′*
by (*simp add*: *shortest-backtrack-strategy-def*)

lemma *strategy-restrictions*:
shows
shortest-backtrack-strategy regular-scl N β S S′ ⟹ *regular-scl N β S S′* **and**
regular-scl N β S S′ ⟹ *reasonable-scl N β S S′* **and**
reasonable-scl N β S S′ ⟹ *scl N β S S′*
using *regular-scl-if-shortest-backtrack-strategy reasonable-if-regular scl-if-reasonable*
by *metis+*

primrec *shortest-backtrack* **where**
shortest-backtrack C [] = [] |
shortest-backtrack C (*Ln* # Γ) =
(*if ex-conflict C* (*Ln* # Γ) *then*
shortest-backtrack C Γ
else
Ln # Γ)

lemma *suffix-shortest-backtrack*: *suffix* (*shortest-backtrack C Γ*) *Γ*
by (*induction Γ*) (*simp-all add*: *suffix-Cons*)

lemma *ex-conflict-shortest-backtrack*: *ex-conflict C* (*shortest-backtrack C Γ*) ⟷
C = {#}
by (*induction Γ*) (*auto simp add*: *ex-conflict-def*)

lemma *is-shortest-backtrack-shortest-backtrack*:
C ≠ {#} ⟹ *is-shortest-backtrack C Γ* (*shortest-backtrack C Γ*)
proof (*induction Γ*)
case *Nil*
then show *?case*
by (*simp add*: *is-shortest-backtrack-def ex-conflict-def*)
next
case (*Cons Kn Γ*)
then show *?case*

apply (*simp del*:)
apply (*rule conjI*)
apply (*simp add*: *is-shortest-backtrack-def suffix-Cons*)
by (*meson is-shortest-backtrack-def not-Cons-self2 suffix-ConsD suffix-appendD suffix-order.dual-order.antisym suffix-order.dual-order.refl*)
qed

9 Monotonicity w.r.t. the Bounding Atom

lemma *scl-monotone-wrt-bound*:
assumes $\bigwedge A$. *is-ground-atm* $A \Longrightarrow A \preceq_B \beta \Longrightarrow A \preceq_B \beta'$ **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 β S₀ S₁*
with *assms(1)* **have** *backtrack N β′ S₀ S₁*
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* $\Longrightarrow$ $A \preceq_B \beta \Longrightarrow A \preceq_B \beta'$ **and** *reasonable-scl N β S₀ S₁*
shows *reasonable-scl N β′ S₀ S₁*
unfolding *reasonable-scl-def*
proof (*intro conjI impI*)
show *scl N β′ S₀ S₁*
using *assms scl-monotone-wrt-bound scl-if-reasonable* **by** *metis*
next
assume *decide N β′ S₀ S₁*
with *assms(2)* **have** *decide N β S₀ S₁*
using *decide-well-defined*
by (*simp add*: *reasonable-scl-def scl-def*)
with *assms(2)* **have** $\nexists S_2$. *conflict N β S₁ S₂*
by (*simp add*: *reasonable-scl-def*)
with *assms(1)* **show** $\nexists S_2$. *conflict N β′ S₁ S₂*
by (*simp add*: *conflict.simps*)
qed

lemma *regular-scl-monotone-wrt-bound*:
assumes $\bigwedge A$. *is-ground-atm A* $\Longrightarrow$ $A \preceq_B \beta \Longrightarrow A \preceq_B \beta'$ **and** *regular-scl N β S₀ S₁*
shows *regular-scl N β′ S₀ S₁*
using *assms(2)*[*unfolded regular-scl-def*]
proof (*elim disjE conjE*)
assume *conflict N β S₀ S₁*
hence *conflict N β′ S₀ S₁*
by (*simp add*: *conflict.simps*)
thus *regular-scl N β′ S₀ S₁*
by (*simp add*: *regular-scl-def*)
next
assume $\nexists S_1'$. *conflict N β S₀ S₁′* **and** *reasonable-scl N β S₀ S₁*
have $\nexists S_1'$. *conflict N β′ S₀ S₁′*
using ‹$\nexists S_1'$. *conflict N β S₀ S₁′*›
by (*simp add*: *conflict.simps*)
moreover have *reasonable-scl N β′ S₀ S₁*
using *assms(1)* ‹*reasonable-scl N β S₀ S₁*› *reasonable-scl-monotone-wrt-bound*
by *metis*
ultimately show *regular-scl N β′ S₀ S₁*

by (*simp add*: *regular-scl-def*)
qed

lemma *min-back-regular-scl-monotone-wrt-bound*:
assumes
$\bigwedge A$. *is-ground-atm* $A \Longrightarrow A \preceq_B \beta \Longrightarrow A \preceq_B \beta'$ **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 $\bigwedge A$. *is-ground-atm* $A \Longrightarrow A \preceq_B \beta \Longrightarrow A \preceq_B \beta'$
shows
scl N β S_0 S_1 $\Longrightarrow$ *scl* N β' S_0 S_1 **and**
reasonable-scl N β S_0 S_1 $\Longrightarrow$ *reasonable-scl* N β' S_0 S_1 **and**
regular-scl N β S_0 S_1 $\Longrightarrow$ *regular-scl* N β' S_0 S_1 **and**
shortest-backtrack-strategy regular-scl N β S_0 S_1 $\Longrightarrow$
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 β S₀ S₁ ⟹ scl N β′ S₀ S₁ **and**
reasonable-scl N β S₀ S₁ ⟹ reasonable-scl N β′ S₀ S₁ **and**
regular-scl N β S₀ S₁ ⟹ regular-scl N β′ S₀ S₁ **and**
shortest-backtrack-strategy regular-scl N β S₀ S₁ ⟹
shortest-backtrack-strategy regular-scl N β′ S₀ S₁
proof −
from ‹*transp-on* {*A*. *is-ground-atm A*} (≺$_B$)› **have** *transp-on* {*A*. *is-ground-atm A*} (⪯$_B$)
by (*metis transp-on-reflclp*)
moreover from ‹*β* ≺$_B$ *β′*› **have** *β* ⪯$_B$ *β′*
by *blast*
ultimately have ⋀*A*. *is-ground-atm A ⟹ A* ⪯$_B$ *β ⟹ A* ⪯$_B$ *β′*
using ‹*is-ground-atm β*› ‹*is-ground-atm β′*›
by (*meson CollectI transp-onD*)
thus
scl N β S₀ S₁ ⟹ scl N β′ S₀ S₁ **and**
reasonable-scl N β S₀ S₁ ⟹ reasonable-scl N β′ S₀ S₁ **and**
regular-scl N β S₀ S₁ ⟹ regular-scl N β′ S₀ S₁ **and**
shortest-backtrack-strategy regular-scl N β S₀ S₁ ⟹
shortest-backtrack-strategy regular-scl N β′ S₀ S₁
using *monotonicity-wrt-bound*
by *metis+*
qed

end

end
theory *Correct-Termination*
imports *SCL-FOL*
begin

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

lemma *not-satisfiable-if-sound-state-conflict-bottom*:
assumes *sound-S*: *sound-state N β S* **and** *conflict-S*: *state-conflict S = Some* ({#}, *γ*)
shows ¬ *satisfiable* (*grounding-of-clss* (*fset N*))
proof −
from *sound-S conflict-S* **have** *fset N* ⟙$\mathcal{G}e$ {{#}}
unfolding *sound-state-def state-conflict-def* **by** *auto*
thus *?thesis* **by** *simp*
qed

lemma *propagate-if-conflict-follows-decide*:
assumes
trail-lt-β: *trail-atoms-lt β S₂* **and**
no-conf: ∄ *S₁*. *conflict N β S₀ S₁* **and** *deci*: *decide N β S₀ S₂* **and** *conf*: *conflict N β S₂ S₃*
shows ∃ *S₄*. *propagate N β S₀ S₄*

proof −
from *deci* **obtain** $L\ \gamma\ \Gamma\ U$ **where**
S_0-*def*: $S_0 = (\Gamma,\ U,\ None)$ **and**
S_2-*def*: $S_2 = (trail\text{-}decide\ \Gamma\ (L \cdot l\ \gamma),\ U,\ None)$ **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*: $S_3 = (trail\text{-}decide\ \Gamma\ (L \cdot l\ \gamma),\ U,\ Some\ (D,\ \gamma_D))$ **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* $(trail\text{-}decide\ \Gamma\ (L \cdot l\ \gamma))\ (D \cdot \gamma_D)$
by (*auto elim*: *conflict.cases*)

have *vars-cls* $D \subseteq$ *subst-domain* γ_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*: S_0-*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*: $D =$ *add-mset* $L'\ D'$ **and** $-\ (L \cdot l\ \gamma) = L' \cdot l\ \gamma_D$
by (*metis Melem-subst-cls multi-member-split*)

define C_0 **where**
$C_0 = \{\#K \in\#\ D'.\ K \cdot l\ \gamma_D \neq L' \cdot l\ \gamma_D\#\}$

define C_1 **where**
$C_1 = \{\#K \in\#\ D'.\ K \cdot l\ \gamma_D = L' \cdot l\ \gamma_D\#\}$

have *ball-atms-lt-*β: $\forall K \in\#\ D \cdot \gamma_D.\ atm\text{-}of\ K \preceq_B \beta$
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 ‹$-\ (L \cdot l\ \gamma) = L' \cdot l\ \gamma_D$› ‹*atm-of* $L \cdot a\ \gamma \preceq_B \beta$›
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
next

```
    have trail-lt-β': ∀ A ∈ atm-of ‘ fst ‘ set (trail-decide Γ (L ·l γ)). A ≼B β
      using trail-lt-β by (simp add: trail-atoms-lt-def S2-def)

    assume K-in: K ∈# D' · γD
    hence atm-of K ∈ atm-of ‘ fst ‘ set (trail-decide Γ (L ·l γ))
      using tr-Γ-L-false-D[unfolded D-def]
    by (metis D-def ‹K ∈# D · γD› atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
        trail-false-cls-def trail-false-lit-def)
   moreover from trail-lt-β have ∀ A ∈ atm-of ‘ fst ‘ set (trail-decide Γ (L ·l
γ)). A ≼B β
      by (simp add: trail-atoms-lt-def S2-def)
    ultimately show ?thesis
      by blast
  qed
 qed

 have tr-false-C1: trail-false-cls Γ (C0 · γD)
 proof (rule subtrail-falseI)
  show trail-false-cls ((L ·l γ, None) # Γ) (C0 · γD)
    unfolding C0-def trail-false-cls-def
    apply (rule ballI)
     apply (rule tr-Γ-L-false-D[unfolded D-def trail-false-cls-def decide-lit-def,
rule-format])
    by auto
 next
  show − (L ·l γ) ∉# C0 · γD
    unfolding C0-def
    using ‹− (L ·l γ) = L' ·l γD› by fastforce
 qed

 have not-def-L'-ϱ-σϱ: ¬ trail-defined-lit Γ (L' ·l γD)
  using ‹¬ trail-defined-lit Γ (L ·l γ)›
  by (metis ‹− (L ·l γ) = L' ·l γD› trail-defined-lit-iff-defined-uminus)

 obtain xs where mset xs = add-mset L' C1
  using ex-mset by auto
 hence set-xs-conv:
  set xs = set-mset (add-mset L' C1)
  by (metis set-mset-mset)

 have unifiers (set (pairs (map atm-of xs))) ≠ {}
 proof (rule not-empty-if-mem)
  have set (pairs (map atm-of xs)) =
    atm-of ‘ set-mset (add-mset L' C1) × atm-of ‘ set-mset (add-mset L' C1)
    unfolding set-pairs list.set-map set-xs-conv by simp
  also have ... =
    atm-of ‘ insert L' (set-mset C1) × atm-of ‘ insert L' (set-mset C1)
    unfolding set-mset-add-mset-insert by simp
  also have ... =
```

```
      atm-of ‘ insert L′ {K. K ∈# D′ ∧ K ·l γ_D = L′ ·l γ_D} ×
      atm-of ‘ insert L′ {K. K ∈# D′ ∧ K ·l γ_D = L′ ·l γ_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 ∈# D′ ∧ K ·l γ_D = L′ ·l γ_D} ×
      atm-of ‘ insert L′ {K. K ∈# D′ ∧ K ·l γ_D = L′ ·l γ_D}
      by assumption

    show γ_D ∈ unifiers (set (pairs (map atm-of xs)))
      unfolding unifiers-def mem-Collect-eq
    proof (rule ballI)
      fix p assume p-in: p ∈ set (pairs (map atm-of xs))
      then obtain K1 K2 where p-def: p = (atm-of K1, atm-of K2) and
        K1 = L′ ∨ K1 ∈ {K. K ∈# D′ ∧ K ·l γ_D = L′ ·l γ_D} and
        K2 = L′ ∨ K2 ∈ {K. K ∈# D′ ∧ K ·l γ_D = L′ ·l γ_D}
        by (auto simp: set-pairs-xs-simp)
      hence K1 ·l γ_D = L′ ·l γ_D ∧ K2 ·l γ_D = L′ ·l γ_D
        by auto
      thus fst p ·a γ_D = snd p ·a γ_D
        unfolding p-def prod.sel
        by (metis atm-of-subst-lit)
    qed
  qed
  then obtain ys where
    unify-pairs: unify (pairs (map atm-of xs)) [] = Some ys
    using ex-unify-if-unifiers-not-empty[OF - refl] by blast

  define μ where
    μ = subst-of ys

  have imgu-μ: is-imgu μ {atm-of ‘ set-mset (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])) []) =
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
imgu-μ]
    unfolding S_0-def by blast
qed

theorem correct-termination:
  fixes gnd-N and gnd-N-lt-β
```

assumes
sound-S: *sound-state N β S* **and**
invars: *trail-atoms-lt β S trail-propagated-wf* (*state-trail S*) *trail-lits-consistent S*
ground-false-closures S **and**
no-new-conflict: ∄ *S′*. *conflict N β S S′* **and**
no-new-propagate: ∄ *S′*. *propagate N β S S′* **and**
no-new-decide: ∄ *S′*. *decide N β S S′* ∧ (∄ *S″*. *conflict N β S′ S″*) **and**
no-new-skip: ∄ *S′*. *skip N β S S′* **and**
no-new-resolve: ∄ *S′*. *resolve N β S S′* **and**
no-new-backtrack: ∄ *S′*. *backtrack N β S S′* ∧
is-shortest-backtrack (*fst* (*the* (*state-conflict S*))) (*state-trail S*) (*state-trail S′*)
defines
gnd-N ≡ *grounding-of-clss* (*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-clss* (*state-trail S*) *gnd-N-lt-β*
proof −
obtain Γ *U u* **where** *S-def*: *S* = (Γ, *U*, *u*)
using *prod-cases3* **by** *blast*

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

from ‹*ground-false-closures S*› **have** *ground-closures S*
by (*simp add*: *ground-false-closures-def*)

have *trail-consistent*: *trail-consistent* (*state-trail S*)
using ‹*trail-lits-consistent S*› **by** (*simp add*: *trail-lits-consistent-def*)

show *?thesis*
proof (*cases u*)
case *u-def*: *None*
hence *state-conflict S* = *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* ∈ *grounding-of-clss* (*fset N*) **and** *C-lt-β*: ∀ *L* ∈# *C*. *atm-of L* ≺*B β*

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

from *C-in* **obtain** *C′ γ* **where** *C′-in*: *C′* |∈| *N* **and** *C-def*: *C* = *C′* · *γ*
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-np*)
assume ¬ *trail-defined-cls* Γ *C*
then obtain *L* **where** *L-in*: *L* ∈# *C* **and** ¬ *trail-defined-lit* Γ *L*
using *trail-defined-cls-def* **by** *blast*
then obtain *L′* **where** *L′-in*: *L′* ∈# *C′* **and** *L* = *L′* ·*l* γ
using *C-def Melem-subst-cls* **by** *blast*

have *deci*: *decide N* β (Γ, *U*, *None*) (*trail-decide* Γ (*L′* ·*l* γ), *U*, *None*)
proof (*rule decideI*)
show *is-ground-lit* (*L′* ·*l* γ)
using *L-in* ‹*L* = *L′* ·*l* γ› ‹*is-ground-cls C*› *is-ground-cls-def* **by** *blast*
next
show ¬ *trail-defined-lit* Γ (*L′* ·*l* γ)
using ‹*L* = *L′* ·*l* γ› ‹¬ *trail-defined-lit* Γ *L*› **by** *blast*
next
show *atm-of L′* ·*a* γ ≼*B* β
using ‹*L* = *L′* ·*l* γ› *C-lt-β L-in* **by** *fastforce*
qed

moreover have ∄*S″*. *conflict N* β (*trail-decide* Γ (*L′* ·*l* γ), *U*, *None*) *S″*
proof (*rule notI*, *elim exE*)
fix *S″*
assume *conf*: *conflict N* β (*trail-decide* Γ (*L′* ·*l* γ), *U*, *None*) *S″*
moreover have *trail-atoms-lt* β (*trail-decide* Γ (*L′* ·*l* γ), *U*, *None*)
using *decide-preserves-trail-atoms-lt*[*OF deci*] ‹*trail-atoms-lt* β *S*›
by (*simp add*: *S-def u-def*)
ultimately have ∃*S*4. *propagate N* β *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 ∃*S′*. *decide N* β *S S′* ∧ (∄*S″*. *conflict N* β *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* ⟹ *trail-true-cls* Γ *C*
by *assumption*
next
assume *trail-false-cls* Γ *C*

define ϱ :: *′v* ⇒ (*′f*, *′v*) *term* **where**
ϱ = *renaming-wrt* (*fset* (*N* |∪| *U* |∪| *clss-of-trail* Γ))

```
      define γ_ϱ where
        γ_ϱ = rename-subst-domain ϱ (restrict-subst-domain (vars-cls C′) γ)

       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
      next
        show is-ground-cls (C′ · restrict-subst-domain (vars-cls C′) γ)
          using ‹is-ground-cls C›[unfolded C-def]
          by (simp add: subst-cls-restrict-subst-domain)
      next
        show trail-false-cls Γ (C′ · restrict-subst-domain (vars-cls C′) γ)
          using ‹trail-false-cls Γ C›[unfolded C-def]
          by (simp add: subst-cls-restrict-subst-domain)
      qed
      with no-new-conflict have False
        by (simp add: S-def u-def)
      thus trail-true-cls Γ C ..
    qed
  qed

  moreover have satisfiable gnd-N-lt-β
    unfolding true-clss-def gnd-N-lt-β-def
  proof (intro exI ballI, unfold mem-Collect-eq, elim conjE)
    fix C
    have trail-consistent Γ
      using S-def trail-consistent by auto
    show C ∈ gnd-N ⟹ ∀ L ∈# C. atm-of L ≼_B β ⟹ trail-interp Γ ⊨ C
      using tr-true[unfolded gnd-N-lt-β-def]
      using trail-interp-cls-if-trail-true[OF ‹trail-consistent Γ›]
      by (simp add: trail-true-clss-def)
  qed

  ultimately show ?thesis
    by (simp add: S-def)
next
  case (Some Cl)
  then obtain C γ_C where u-def: u = Some (C, γ_C) by force

  from ‹ground-false-closures S› have Γ-false-C-γ_C: trail-false-cls Γ (C · γ_C)
    by (simp add: S-def u-def ground-false-closures-def)

  show ?thesis
  proof (cases C = {#})
    case True
    hence ¬ satisfiable gnd-N ∧ (∃γ. state-conflict S = Some ({#}, γ))
      using S-def u-def not-satisfiable-if-sound-state-conflict-bottom[OF sound-S]
```

```
      by (simp add: gnd-N-def)
    thus ?thesis by simp
  next
    case C-not-empty: False
    have False
    proof (cases Γ)
      case Nil
      with Γ-false-C-γ_C show False
        using C-not-empty by simp
    next
      case (Cons Ln Γ')
      then obtain K_Γ n where Γ-def: Γ = (K_Γ, n) # Γ'
        by fastforce

      show False
      proof (cases - K_Γ ∈# C · γ_C)
        case True — Literal cannot be skipped
        then obtain C' L where C-def: C = add-mset L C' and K-γ: L ·l γ_C
= - K_Γ
          by (metis Melem-subst-cls multi-member-split)
        hence L-eq-uminus-K-γ: K_Γ = - (L ·l γ_C)
          by simp

        show False
        proof (cases n)
          case None — Conflict clause can be backtracked
          hence Γ-def: Γ = trail-decide Γ' (- (L ·l γ_C))
            by (simp add: Γ-def L-eq-uminus-K-γ decide-lit-def)

          from suffix-shortest-backtrack[of add-mset L C' Γ'] obtain Γ'' where
            Γ'-def: Γ' = Γ'' @ shortest-backtrack (add-mset L C') Γ'
            using suffixE by blast

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

          have backtrack N β S S'
            unfolding S-def[unfolded u-def C-def] S'-def
          proof (rule backtrackI[OF - refl])
            show Γ = trail-decide (Γ'' @ shortest-backtrack (add-mset L C') Γ')
(- (L ·l γ_C))
              using Γ-def Γ'-def by simp
          next
            show ∄γ. is-ground-cls (add-mset L C' · γ) ∧
              trail-false-cls (shortest-backtrack (add-mset L C') Γ') (add-mset L C'
· γ)
              using ex-conflict-shortest-backtrack[of add-mset L C', simplified]
              by (simp add: ex-conflict-def)
```

```
    qed
    moreover have is-shortest-backtrack (fst (the (state-conflict S)))
      (state-trail S) (state-trail S')
      unfolding S-def[unfolded u-def C-def] S'-def
      apply simp
   using is-shortest-backtrack-shortest-backtrack[of add-mset L C', simplified]
    by (metis C-def Γ-def Γ-false-C-γ_C ‹S = (Γ, U, Some (add-mset L C',
γ_C))›
      ‹ground-closures S› ex-conflict-def ground-closures-def is-shortest-backtrack-def
          state-proj-simp(3) suffix-Cons)
    ultimately show False
      using no-new-backtrack by metis
  next
    case Some — Literal can be resolved
    then obtain D K γ_D where n-def: n = Some (D, K, γ_D)
      by (metis prod-cases3)
    with ‹trail-propagated-wf (state-trail S)› have L-def: K_Γ = K ·l γ_D
      by (simp add: Γ-def S-def trail-propagated-wf-def)
    hence 1: Γ = trail-propagate Γ' K D γ_D
      using Γ-def n-def
      by (simp add: propagate-lit-def)

    from ‹ground-closures S› have
      ground-conf: is-ground-cls (add-mset L C' · γ_C) and
      ground-prop: is-ground-cls (add-mset K D · γ_D)
      unfolding S-def ground-closures-def
      by (simp-all add: 1 C-def u-def ground-closures-def propagate-lit-def)

    define ϱ :: 'v ⇒ ('f, 'v) Term.term where
      ϱ = renaming-wrt {add-mset K D}

    have ren-ϱ: is-renaming ϱ
      unfolding ϱ-def
      by (rule is-renaming-renaming-wrt) simp
    hence ∀ x. is-Var (ϱ x) inj ϱ
      by (simp-all add: is-renaming-iff)

   have disjoint-vars: ⋀C. vars-cls (C · ϱ) ∩ vars-cls (add-mset K D · Var)
= {}
      by (simp add: ϱ-def vars-cls-subst-renaming-disj)

    have 2: K ·l γ_D = − (L ·l γ_C)
      using K-γ L-def by fastforce

    let ?γ_D' = restrict-subst-domain (vars-cls (add-mset K D)) γ_D

    have K ·l ?γ_D' = K ·l γ_D and D · ?γ_D' = D · γ_D
  by (simp-all add: subst-lit-restrict-subst-domain subst-cls-restrict-subst-domain)
  hence K ·l ?γ_D' = − (L ·l γ_C) and ground-prop': is-ground-cls (add-mset
```

```
K D · ?γ_D')
            using 2 ground-prop by simp-all

          have dom-γ_D': subst-domain ?γ_D' ⊆ vars-cls (add-mset K D)
            by simp

          let ?γ = λx.
            if x ∈ vars-cls (add-mset L C' · ϱ) then
              rename-subst-domain ϱ γ_C x
            else
              γ_D x
          have L ·l ϱ ·l ?γ = L ·l γ_C and K ·l ?γ = K ·l γ_D
              using merge-of-renamed-groundings[OF ren-ϱ 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 ϱ ·a ?γ = atm-of K ·a ?γ
           by (smt (verit, best) 2 atm-of-uminus subst-lit-id-subst atm-of-subst-lit)
           then obtain μ where Unification.mgu (atm-of L ·a ϱ) (atm-of K) =
Some μ
            using ex-mgu-if-subst-apply-term-eq-subst-apply-term
            by blast
          hence imgu-μ: is-imgu μ {{atm-of L ·a ϱ, atm-of K ·a Var}}
            by (simp add: is-imgu-if-mgu-eq-Some)

          have ∃S. resolve N β (Γ, U, Some (add-mset L C', γ_C)) S
            using resolveI[OF 1 2 ren-ϱ is-renaming-id-subst disjoint-vars imgu-μ
                is-grounding-merge-if-mem-then-else] ..
          with no-new-resolve show False
            by (metis C-def S-def u-def)
        qed
      next
        case False — Literal can be skipped
       hence skip N β ((K_Γ, n) # Γ', U, Some (C, γ_C)) (Γ', U, Some (C, γ_C))
          by (rule skipI[of K_Γ C γ_C N β n Γ' U])
        with no-new-skip show False
          by (metis S-def Γ-def u-def)
      qed
    qed
    thus ?thesis ..
  qed
 qed
qed

corollary correct-termination-strategy:
 fixes gnd-N and gnd-N-lt-β
 assumes
```

run: (*strategy N β*)** *initial-state S* **and**
no-step: ∄ *S'*. *strategy N β S S'* **and**
strategy-restricted-by-min-back:
⋀*S S'*. *shortest-backtrack-strategy regular-scl N β S S'* ⟹ *strategy N β S S'*
and
strategy-preserves-invars:
⋀*N β S S'*. *strategy N β S S'* ⟹ *sound-state N β S* ⟹ *sound-state N β S'*
⋀*N β S S'*. *strategy N β S S'* ⟹ *trail-atoms-lt β S* ⟹ *trail-atoms-lt β S'*
⋀*N β S S'*. *strategy N β S S'* ⟹ *trail-propagated-or-decided' N β S* ⟹ *trail-propagated-or-decided' N β S'*
⋀*N β S S'*. *strategy N β S S'* ⟹ *trail-lits-consistent S* ⟹ *trail-lits-consistent S'*
⋀*N β S S'*. *strategy N β S S'* ⟹ *ground-false-closures S* ⟹ *ground-false-closures S'*
defines
gnd-N ≡ *grounding-of-clss* (*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-clss* (*state-trail S*) *gnd-N-lt-β*
proof −
from *no-step* **have** *no-step'*: ∄ *S'*. *shortest-backtrack-strategy regular-scl N β S S'*
proof (*rule contrapos-nn*)
show ∃ *S'*. *shortest-backtrack-strategy regular-scl N β S S'* ⟹ ∃ *S'*. *strategy N β S S'*
using *strategy-restricted-by-min-back* **by** *metis*
qed

show *?thesis*
proof (*rule correct-termination*[*of N β S*, *folded gnd-N-def*, *folded gnd-N-lt-β-def*])
from *run* **show** *sound-state N β S*
by (*induction S rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(*1*))
next
from *run* **show** *trail-atoms-lt β S*
by (*induction S rule*: *rtranclp-induct*) (*auto intro*: *strategy-preserves-invars*(*2*))
next
from *run* **have** *trail-propagated-or-decided' N β 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** ∄ *S'*. *conflict N β S S'*

unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*
using *backtrack-well-defined*(3) **by** *blast*
next
from *no-step'* **show** $\nexists S'$. *propagate N β S S'*
unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*
using *backtrack-well-defined*(3) *propagate-well-defined*(1) *propagate-well-defined*(6) **by** *blast*
next
from *no-step'* **show** $\nexists S'$. *decide N β S S'* $\wedge$ ($\nexists S''$. *conflict N β S' S''*)
unfolding *shortest-backtrack-strategy-def regular-scl-def reasonable-scl-def scl-def*
using *backtrack-well-defined*(2) *backtrack-well-defined*(3) **by** *blast*
next
from *no-step'* **show** $\nexists S'$. *skip N β 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** $\nexists S'$. *resolve N β 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** $\nexists S'$. *backtrack N β 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: $\nexists S'$. *scl N β S S'*
defines
gnd-N $\equiv$ *grounding-of-clss* (*fset N*) **and**
gnd-N-lt-β $\equiv \{C \in$ *gnd-N*. $\forall L \in\#$ *C*. *atm-of L* $\preceq_B$ *β*$\}$
shows $\neg$ *satisfiable gnd-N* $\wedge$ ($\exists \gamma$. *state-conflict S* = *Some* ($\{\#\}$, γ)) $\vee$
satisfiable gnd-N-lt-β $\wedge$ *trail-true-clss* (*state-trail S*) *gnd-N-lt-β*
proof (*rule correct-termination-strategy*[*of - N β*, *folded gnd-N-def*, *folded gnd-N-lt-β-def*])
show (*scl N β*)** *initial-state S*
by (*rule run*)
next
show $\nexists S'$. *scl N β S S'*

by (*rule no-step*)
next
show $\bigwedge S\ S'$. *shortest-backtrack-strategy regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *scl* $N\ \beta\ S\ S'$
by (*simp add*: *regular-scl-if-shortest-backtrack-strategy scl-if-regular*)
next
show $\bigwedge N\ \beta\ S\ S'$. *scl* $N\ \beta\ S\ S' \Longrightarrow$ *sound-state* $N\ \beta\ S \Longrightarrow$ *sound-state* $N\ \beta\ S'$
using *scl-preserves-sound-state* **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-atoms-lt* $\beta\ S \Longrightarrow$ *trail-atoms-lt* $\beta\ S'$
using *scl-preserves-trail-atoms-lt* **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-propagated-or-decided'* $N\ \beta\ S \Longrightarrow$
trail-propagated-or-decided' $N\ \beta\ S'$
using *scl-preserves-trail-propagated-or-decided* **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-lits-consistent* $S \Longrightarrow$ *trail-lits-consistent* S'
using *scl-preserves-trail-lits-consistent* **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *scl* $N\ \beta\ S\ S' \Longrightarrow$ *ground-false-closures* $S \Longrightarrow$ *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: $(\textit{reasonable-scl}\ N\ \beta)^{**}$ *initial-state* S **and**
no-step: $\nexists S'$. *reasonable-scl* $N\ \beta\ S\ S'$
defines
$\textit{gnd-N} \equiv \textit{grounding-of-clss}\ (\textit{fset}\ N)$ **and**
$\textit{gnd-N-lt-}\beta \equiv \{C \in \textit{gnd-N}.\ \forall L \in\# C.\ \textit{atm-of}\ L \preceq_B \beta\}$
shows $\neg$ *satisfiable gnd-N* $\wedge$ $(\exists \gamma.\ \textit{state-conflict}\ S = \textit{Some}\ (\{\#\}, \gamma)) \vee$
satisfiable gnd-N-lt-β $\wedge$ *trail-true-clss* (*state-trail S*) *gnd-N-lt-β*
proof (*rule correct-termination-strategy*[*of - N β*, *folded gnd-N-def*, *folded gnd-N-lt-β-def*])
show $(\textit{reasonable-scl}\ N\ \beta)^{**}$ *initial-state S*
by (*rule run*)
next
show $\nexists S'$. *reasonable-scl N β S S'*
by (*rule no-step*)
next
show $\bigwedge S\ S'$. *shortest-backtrack-strategy regular-scl N β S S'* $\Longrightarrow$ *reasonable-scl N β S S'*
by (*simp add*: *reasonable-if-regular regular-scl-if-shortest-backtrack-strategy*)
next
show $\bigwedge N\ \beta\ S\ S'$. *reasonable-scl N β S S'* $\Longrightarrow$ *sound-state N β S* $\Longrightarrow$ *sound-state N β S'*
using *scl-preserves-sound-state*[*OF scl-if-reasonable*] **by** *simp*
next

show $\bigwedge N\ \beta\ S\ S'$. *reasonable-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-atoms-lt* $\beta\ S \Longrightarrow$ *trail-atoms-lt* $\beta\ S'$
using *scl-preserves-trail-atoms-lt*[*OF scl-if-reasonable*] **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *reasonable-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-propagated-or-decided'* $N\ \beta\ S \Longrightarrow$
trail-propagated-or-decided' $N\ \beta\ S'$
using *scl-preserves-trail-propagated-or-decided*[*OF scl-if-reasonable*] **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *reasonable-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-lits-consistent* $S \Longrightarrow$ *trail-lits-consistent* S'
using *scl-preserves-trail-lits-consistent*[*OF scl-if-reasonable*] **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *reasonable-scl* $N\ \beta\ S\ S' \Longrightarrow$ *ground-false-closures* $S \Longrightarrow$ *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* $N\ \beta$)** *initial-state* S **and**
no-step: $\nexists S'$. *regular-scl* $N\ \beta\ S\ S'$
defines
gnd-N $\equiv$ *grounding-of-clss* (*fset* N) **and**
gnd-N-lt-β $\equiv \{C \in$ *gnd-N*. $\forall L \in\# C$. *atm-of* $L \preceq_B \beta\}$
shows $\neg$ *satisfiable gnd-N* $\wedge$ ($\exists \gamma$. *state-conflict* S = *Some* ($\{\#\}$, γ)) $\vee$
satisfiable gnd-N-lt-β $\wedge$ *trail-true-clss* (*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 (*regular-scl* $N\ \beta$)** *initial-state* S
by (*rule run*)
next
show $\nexists S'$. *regular-scl* $N\ \beta\ S\ S'$
by (*rule no-step*)
next
show $\bigwedge S\ S'$. *shortest-backtrack-strategy regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *regular-scl* $N\ \beta\ S\ S'$
by (*simp add*: *reasonable-if-regular regular-scl-if-shortest-backtrack-strategy*)
next
show $\bigwedge N\ \beta\ S\ S'$. *regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *sound-state* $N\ \beta\ S \Longrightarrow$ *sound-state* $N\ \beta\ S'$
using *scl-preserves-sound-state*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-atoms-lt* $\beta\ S \Longrightarrow$ *trail-atoms-lt* $\beta\ S'$
using *scl-preserves-trail-atoms-lt*[*OF scl-if-regular*] **by** *simp*
next
show $\bigwedge N\ \beta\ S\ S'$. *regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-propagated-or-decided'* $N\ \beta\ S \Longrightarrow$

trail-propagated-or-decided′ N β S′
using *scl-preserves-trail-propagated-or-decided*[*OF scl-if-regular*] **by** *simp*
next
show ⋀*N β S S′*. *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** *simp*
next
show ⋀*N β S S′*. *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** *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: ∄*S′*. *shortest-backtrack-strategy regular-scl N β S S′*
defines
gnd-N ≡ *grounding-of-clss* (*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-clss* (*state-trail S*) *gnd-N-lt-β*
proof (*rule correct-termination-strategy*[*of - N β*, *folded gnd-N-def*, *folded gnd-N-lt-β-def*])
show (*shortest-backtrack-strategy regular-scl N β*)** *initial-state S*
by (*rule run*)
next
show ∄*S′*. *shortest-backtrack-strategy regular-scl N β S S′*
by (*rule no-step*)
next
show ⋀*S S′*. *shortest-backtrack-strategy regular-scl N β S S′* ⟹ *shortest-backtrack-strategy regular-scl N β S S′*
by *simp*
next
show ⋀*N β S S′*. *shortest-backtrack-strategy regular-scl N β S S′* ⟹ *sound-state N β S* ⟹ *sound-state N β S′*
using *scl-preserves-sound-state*[*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-atoms-lt β S* ⟹ *trail-atoms-lt β S′*
using *scl-preserves-trail-atoms-lt*[*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-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 $\bigwedge N\ \beta\ S\ S'$. *shortest-backtrack-strategy regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *trail-lits-consistent* $S \Longrightarrow$ *trail-lits-consistent* S'
using *scl-preserves-trail-lits-consistent*[*OF scl-if-regular*]
by (*auto simp*: *shortest-backtrack-strategy-def*)
next
show $\bigwedge N\ \beta\ S\ S'$. *shortest-backtrack-strategy regular-scl* $N\ \beta\ S\ S' \Longrightarrow$ *ground-false-closures* $S \Longrightarrow$ *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* $N\ \beta$)** *initial-state* $S \wedge (\nexists S'.$ *scl* $N\ \beta\ S\ S') \vee$
(*reasonable-scl* $N\ \beta$)** *initial-state* $S \wedge (\nexists S'.$ *reasonable-scl* $N\ \beta\ S\ S') \vee$
(*regular-scl* $N\ \beta$)** *initial-state* $S \wedge (\nexists S'.$ *regular-scl* $N\ \beta\ S\ S') \vee$
(*shortest-backtrack-strategy regular-scl* $N\ \beta$)** *initial-state* $S \wedge$
$(\nexists S'.$ *shortest-backtrack-strategy regular-scl* $N\ \beta\ S\ S')$
defines
gnd-N $\equiv$ *grounding-of-clss* (*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* ($\{\#\}$, γ)) $\vee$
satisfiable gnd-N-lt-$\beta \wedge$ *trail-true-clss* (*state-trail* S) *gnd-N-lt-*β
unfolding *gnd-N-def gnd-N-lt-*β*-def*
using *assms*(*1*)
correct-termination-scl-run[*of* $N\ \beta\ S$]
correct-termination-reasonable-scl-run[*of* $N\ \beta\ S$]
correct-termination-regular-scl-run[*of* $N\ \beta\ S$]
correct-termination-shortest-backtrack-strategy-regular-scl[*of* $N\ \beta\ S$]
by *argo*

end

end
theory *Trail-Induced-Ordering*
imports

Main

List−Index.List-Index
begin

lemma *wf-if-convertible-to-wf*:
fixes $r :: 'a\ rel$ **and** $s :: 'b\ rel$ **and** $f :: 'a \Rightarrow 'b$
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** $Q :: 'a\ set$
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 $\longleftrightarrow$
$(\exists i < length\ Ls.\ \exists j < length\ Ls.\ i > j \wedge L = Ls\ !\ i \wedge K = Ls\ !\ j)$

definition *trail-less-comp-id* **where**
trail-less-comp-id Ls L K $\longleftrightarrow$
$(\exists i < length\ Ls.\ \exists j < length\ Ls.\ i > j \wedge L = -\ (Ls\ !\ i) \wedge K = Ls\ !\ j)$

definition *trail-less-id-comp* **where**
trail-less-id-comp Ls L K $\longleftrightarrow$
$(\exists i < length\ Ls.\ \exists j < length\ Ls.\ i \geq j \wedge L = Ls\ !\ i \wedge K = -\ (Ls\ !\ j))$

definition *trail-less-comp-comp* **where**
trail-less-comp-comp Ls L K $\longleftrightarrow$
$(\exists i < length\ Ls.\ \exists j < length\ Ls.\ i > j \wedge L = -\ (Ls\ !\ i) \wedge K = -\ (Ls\ !\ j))$

definition *trail-less* **where**
trail-less Ls L K $\longleftrightarrow$ *trail-less-id-id Ls L K* $\vee$ *trail-less-comp-id Ls L K* $\vee$
trail-less-id-comp Ls L K $\vee$ *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* $\Longrightarrow$ *trail-less'* *Ls* *y* *z* $\Longrightarrow$ *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* $\Longrightarrow$ *i* < *length Ls* $\Longrightarrow$ *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* $\Longrightarrow$ *i* < *length Ls* $\Longrightarrow$ *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 < length Ls
  shows trail-less' Ls (Ls ! i) (- (Ls ! j))
proof -
  have 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 < length Ls
  shows 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 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 ‹trail-less' Ls (- Ls ! i) (Ls ! i')› by metis
  next
    case False
    hence 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 ‹trail-less' Ls (- Ls ! i) (Ls ! i')›
      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: ⋀x :: 'a. − x ≠ x and
    uminus-uminus-id: ⋀x :: 'a. − (− x) = x and
    pairwise-distinct:
      ∀ i < length Ls. ∀ j < length Ls. i ≠ j ⟶ Ls ! i ≠ Ls ! j ∧ Ls ! i ≠ − (Ls !
j)
  shows trail-less Ls = trail-less' Ls
proof (intro ext iffI)
  fix L K
  show trail-less Ls L K ⟹ 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 ⟹ ?thesis
      using trail-less'-comp-id by (metis trail-less-comp-id-def)
  next
    show trail-less-id-comp Ls L K ⟹ ?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 ⟹ ?thesis
      using trail-less'-comp-comp
      by (metis trail-less-comp-comp-def)
  qed
next
  fix L K
  show trail-less' Ls L K ⟹ trail-less Ls L K
    unfolding trail-less'-def
  proof (induction K rule: tranclp-induct)
    case (base K)
    then show ?case
    proof (elim exE conjE disjE)
      fix i assume i < length Ls and L = Ls ! i and K = − (Ls ! i)
      hence trail-less-id-comp Ls L K
        by (auto simp: trail-less-id-comp-def)
      thus trail-less Ls L K
        by (simp add: trail-less-def)
    next
      fix i assume Suc i < length Ls and L = − (Ls ! Suc i) and K = Ls ! i
      hence trail-less-comp-id Ls L K
        unfolding trail-less-comp-id-def
        using Suc-lessD by blast
      thus trail-less Ls L K
        by (simp add: trail-less-def)
    qed
```

```
  next
    case (step y z)
    from step.hyps(2) show ?case
    proof (elim exE conjE disjE)
      fix i assume i < length Ls and y = Ls ! i and z = - (Ls ! i)

      from step.IH[unfolded trail-less-def] show trail-less Ls L z
      proof (elim disjE)
        assume trail-less-id-id Ls L y
        hence trail-less-id-comp Ls L z
          unfolding trail-less-id-id-def trail-less-id-comp-def
          by (metis ‹y = Ls ! i› ‹z = - Ls ! i› less-or-eq-imp-le)
        thus trail-less Ls L z
          by (simp add: trail-less-def)
      next
        assume trail-less-comp-id Ls L y
        hence trail-less-comp-comp Ls L z
          unfolding trail-less-comp-id-def trail-less-comp-comp-def
          by (metis ‹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 ⟹ trail-less Ls L z
          by (smt (verit, best) ‹Suc i < length Ls› ‹y = - Ls ! Suc i› ‹z = Ls ! i›
                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 ‹Suc i < length Ls› ‹y = - Ls ! Suc i›
‹z = Ls ! i›
            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 ‹Suc i < length Ls› ‹y = - Ls ! Suc i› ‹z = Ls ! i›
pairwise-distinct
            uminus-uminus-id)
      thus trail-less Ls L z
        by (simp add: trail-less-def)
    qed
  qed
 qed
qed
```

9.1 Examples

```
experiment
 fixes L0 L1 L2 :: 'a :: uminus
begin

lemma trail-less-id-comp [L2, L1, L0] L2 (- L2)
 unfolding trail-less-id-comp-def
proof (intro exI conjI)
 show (0 :: nat) ≤ 0 by presburger
qed simp-all

lemma trail-less-comp-id [L2, L1, L0] (- L1) L2
 unfolding trail-less-comp-id-def
proof (intro exI conjI)
 show (0 :: nat) < 1 by presburger
qed simp-all

lemma trail-less-id-comp [L2, L1, L0] L1 (- L1)
 unfolding trail-less-id-comp-def
proof (intro exI conjI)
 show (1 :: nat) ≤ 1 by presburger
qed simp-all

lemma trail-less-comp-id [L2, L1, L0] (- L0) L1
 unfolding trail-less-comp-id-def
```

proof (*intro exI conjI*)
  show (*1* :: *nat*) < *2* **by** *presburger*
qed *simp-all*

lemma *trail-less-id-comp* [*L2*, *L1*, *L0*] *L0* (− *L0*)
  unfolding *trail-less-id-comp-def*
proof (*intro exI conjI*)
  show (*2* :: *nat*) ≤ *2* **by** *presburger*
qed *simp-all*

lemma *trail-less-id-id* [*L2*, *L1*, *L0*] *L1 L2*
  unfolding *trail-less-id-id-def*
proof (*intro exI conjI*)
  show (*0* :: *nat*) < *1* **by** *presburger*
qed *simp-all*

lemma *trail-less-id-id* [*L2*, *L1*, *L0*] *L0 L1*
  unfolding *trail-less-id-id-def*
proof (*intro exI conjI*)
  show (*1* :: *nat*) < *2* **by** *presburger*
qed *simp-all*

lemma *trail-less-comp-comp* [*L2*, *L1*, *L0*] (− *L1*) (− *L2*)
  unfolding *trail-less-comp-comp-def*
proof (*intro exI conjI*)
  show (*0* :: *nat*) < *1* **by** *presburger*
qed *simp-all*

lemma *trail-less-comp-comp* [*L2*, *L1*, *L0*] (− *L0*) (− *L1*)
  unfolding *trail-less-comp-comp-def*
proof (*intro exI conjI*)
  show (*1* :: *nat*) < *2* **by** *presburger*
qed *simp-all*

end

9.2 Miscellaneous Lemmas

lemma *not-trail-less-Nil*: ¬ *trail-less* [] *L K*
  unfolding *trail-less-def trail-less-id-id-def trail-less-comp-id-def*
    trail-less-id-comp-def trail-less-comp-comp-def
  by *simp*

lemma *defined-if-trail-less*:
  assumes *trail-less Ls L K*
  shows *L* ∈ *set Ls* ∪ *uminus* ‘ *set Ls K* ∈ *set Ls* ∪ *uminus* ‘ *set Ls*
   apply (*atomize* (*full*))
  using *assms* **unfolding** *trail-less-def trail-less-id-id-def trail-less-comp-id-def*

trail-less-id-comp-def trail-less-comp-comp-def
by (*elim disjE exE conjE*) *simp-all*

lemma *not-less-if-undefined*:
fixes $L :: 'a :: uminus$
assumes
uminus-uminus-id: $\bigwedge x :: 'a. - (- x) = x$ **and**
$L \notin set\ Ls - L \notin set\ Ls$
shows $\neg$ *trail-less Ls L K* $\neg$ *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 $L :: 'a :: uminus$
assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $L \in set\ Ls \cup uminus\ `\ set\ Ls \longleftrightarrow L \in set\ Ls \vee - L \in set\ Ls$
by (*auto simp*: *rev-image-eqI uminus-uminus-id*)

lemma *trail-less-comp-rightI*: $L \in set\ Ls \Longrightarrow$ *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 $Ls :: ('a :: uminus)\ list$
assumes *uminus-uminus-id*: $\bigwedge x :: 'a. - (- x) = x$
shows $- L \in set\ Ls \Longrightarrow$ *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: $\forall x \in set\ Ls. \forall y \in set\ Ls.\ x \neq - y$ **and**
L-le-K: *trail-less-id-id Ls L K*
shows
$\neg$ *trail-less-id-comp Ls L K*
$\neg$ *trail-less-comp-id Ls L K*
$\neg$ *trail-less-comp-comp Ls L K*
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: $\forall x \in set\ Ls. \forall y \in set\ Ls.\ x \neq - y$ **and**
L-le-K: *trail-less-id-comp Ls L K*

shows
¬ *trail-less-id-id Ls L K*
¬ *trail-less-comp-id Ls L K*
¬ *trail-less-comp-comp Ls L K*
using *L-le-K*
unfolding *trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def*
trail-less-comp-comp-def
using *pairwise-distinct*[*rule-format*, *OF nth-mem nth-mem*]
by *metis+*

lemma *trail-less-comp-id-well-defined*:
assumes
pairwise-distinct: ∀ *x* ∈ *set Ls*. ∀ *y* ∈ *set Ls*. *x* ≠ − *y* **and**
L-le-K: *trail-less-comp-id Ls L K*
shows
¬ *trail-less-id-id Ls L K*
¬ *trail-less-id-comp Ls L K*
¬ *trail-less-comp-comp Ls L K*
using *L-le-K*
unfolding *trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def*
trail-less-comp-comp-def
using *pairwise-distinct*[*rule-format*, *OF nth-mem nth-mem*]
by *metis+*

lemma *trail-less-comp-comp-well-defined*:
assumes
pairwise-distinct: ∀ *x* ∈ *set Ls*. ∀ *y* ∈ *set Ls*. *x* ≠ − *y* **and**
L-le-K: *trail-less-comp-comp Ls L K*
shows
¬ *trail-less-id-id Ls L K*
¬ *trail-less-id-comp Ls L K*
¬ *trail-less-comp-id Ls L K*
using *L-le-K*
unfolding *trail-less-id-id-def trail-less-comp-id-def trail-less-id-comp-def*
trail-less-comp-comp-def
using *pairwise-distinct*[*rule-format*, *OF nth-mem nth-mem*]
by *metis+*

9.4 Strict Partial Order

lemma *irreflp-trail-less*:
fixes *Ls* :: ('*a* :: *uminus*) *list*
assumes
uminus-not-id: ⋀*x* :: '*a*. − *x* ≠ *x* **and**
uminus-uminus-id: ⋀*x* :: '*a*. − (− *x*) = *x* **and**
pairwise-distinct:
∀ *i* < *length Ls*. ∀ *j* < *length Ls*. *i* ≠ *j* ⟶ *Ls* ! *i* ≠ *Ls* ! *j* ∧ *Ls* ! *i* ≠ − (*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* ⟹ *False*
unfolding *trail-less-id-id-def*
using *pairwise-distinct* **by** *fastforce*
next
show *trail-less-comp-id Ls L L* ⟹ *False*
unfolding *trail-less-comp-id-def*
by (*metis pairwise-distinct uminus-not-id*)
next
show *trail-less-id-comp Ls L L* ⟹ *False*
unfolding *trail-less-id-comp-def*
by (*metis pairwise-distinct uminus-not-id*)
next
show *trail-less-comp-comp Ls L L* ⟹ *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: ⋀*x* :: *'a*. − *x* ≠ *x* **and**
uminus-uminus-id: ⋀*x* :: *'a*. − (− *x*) = *x* **and**
pairwise-distinct:
∀ *i* < *length Ls*. ∀ *j* < *length Ls*. *i* ≠ *j* ⟶ *Ls* ! *i* ≠ *Ls* ! *j* ∧ *Ls* ! *i* ≠ − (*Ls* !
j)
shows *transp* (*trail-less Ls*)
proof (*rule transpI*)
fix *L K H* :: *'a*
show *trail-less Ls L K* ⟹ *trail-less Ls K H* ⟹ *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* :: *uminus*) *list*
assumes
uminus-not-id: ⋀*x* :: *'a*. − *x* ≠ *x* **and**
uminus-uminus-id: ⋀*x* :: *'a*. − (− *x*) = *x* **and**
pairwise-distinct:

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

shows *asymp* (*trail-less Ls*)
using *irreflp-trail-less*[*OF assms*] *transp-trail-less*[*OF assms*]
using *asymp-on-iff-irreflp-on-if-transp-on*
by *auto*

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

lemma *totalp-on-trail-less*:
totalp-on (*set Ls* ∪ *uminus* ‘ *set Ls*) (*trail-less Ls*)
proof (*rule totalp-onI*, *unfold Un-iff*, *elim disjE*)
fix *L K*
assume *L* ∈ *set Ls* **and** *K* ∈ *set Ls* **and** *L* ≠ *K*
then obtain *i j* **where** *i* < *length Ls Ls* ! *i* = *L j* < *length Ls Ls* ! *j* = *K*
unfolding *in-set-conv-nth* **by** *auto*
thus *trail-less Ls L K* ∨ *trail-less Ls K L*
using ‹*L* ≠ *K*› *less-linear*[*of i j*]
by (*meson trail-less-def trail-less-id-id-def*)
next
fix *L K*
assume *L* ∈ *set Ls* **and** *K* ∈ *uminus* ‘ *set Ls* **and** *L* ≠ *K*
then obtain *i j* **where** *i* < *length Ls Ls* ! *i* = *L j* < *length Ls* − (*Ls* ! *j*) = *K*
unfolding *in-set-conv-nth image-set length-map* **by** *auto*
thus *trail-less Ls L K* ∨ *trail-less Ls K L*
using *less-linear*[*of i j*]
by (*metis le-eq-less-or-eq trail-less-comp-id-def trail-less-def trail-less-id-comp-def*)
next
fix *L K*
assume *L* ∈ *uminus* ‘ *set Ls* **and** *K* ∈ *set Ls* **and** *L* ≠ *K*
then obtain *i j* **where** *i* < *length Ls* − (*Ls* ! *i*) = *L j* < *length Ls Ls* ! *j* = *K*
unfolding *in-set-conv-nth image-set length-map* **by** *auto*
thus *trail-less Ls L K* ∨ *trail-less Ls K L*
using *less-linear*[*of i j*]
by (*metis le-eq-less-or-eq trail-less-comp-id-def trail-less-def trail-less-id-comp-def*)
next
fix *L K*
assume *L* ∈ *uminus* ‘ *set Ls* **and** *K* ∈ *uminus* ‘ *set Ls* **and** *L* ≠ *K*
then obtain *i j* **where** *i* < *length Ls* − (*Ls* ! *i*) = *L j* < *length Ls* − (*Ls* ! *j*) = *K*
unfolding *in-set-conv-nth image-set length-map* **by** *auto*
thus *trail-less Ls L K* ∨ *trail-less Ls K L*
using ‹*L* ≠ *K*› *less-linear*[*of i j*]
by (*metis trail-less-comp-comp-def trail-less-def*)
qed

9.6 Well-Founded

lemma *not-trail-less-Cons-id-comp*:
fixes *Ls* :: (′*a* :: *uminus*) *list*

assumes
uminus-not-id: $\bigwedge x :: 'a.\ -\ x \neq x$ **and**
uminus-uminus-id: $\bigwedge x :: 'a.\ -\ (-\ x) = x$ **and**
pairwise-distinct:
$\forall\, i < \mathit{length}\ (L\ \#\ Ls).\ \forall\, j < \mathit{length}\ (L\ \#\ Ls).\ i \neq j \longrightarrow$
$(L\ \#\ Ls)\ !\ i \neq (L\ \#\ Ls)\ !\ j \wedge (L\ \#\ Ls)\ !\ i \neq -\ ((L\ \#\ Ls)\ !\ j)$
shows $\neg$ *trail-less* $(L\ \#\ Ls)\ (-\ L)\ L$
proof (*rule notI*, *unfold trail-less-def*, *elim disjE*)
show *trail-less-id-id* $(L\ \#\ Ls)\ (-\ L)\ L \Longrightarrow \mathit{False}$
unfolding *trail-less-id-id-def*
using *pairwise-distinct uminus-not-id* **by** *metis*
next
show *trail-less-comp-id* $(L\ \#\ Ls)\ (-\ L)\ L \Longrightarrow \mathit{False}$
unfolding *trail-less-comp-id-def*
using *pairwise-distinct uminus-uminus-id*
by (*metis less-not-refl*)
next
show *trail-less-id-comp* $(L\ \#\ Ls)\ (-\ L)\ L \Longrightarrow \mathit{False}$
unfolding *trail-less-id-comp-def*
using *pairwise-distinct uminus-not-id*
by (*metis length-pos-if-in-set nth-Cons-0 nth-mem*)
next
show *trail-less-comp-comp* $(L\ \#\ Ls)\ (-\ L)\ L \Longrightarrow \mathit{False}$
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 $L :: 'a :: \mathit{uminus}$
assumes
undefined: $L \notin \mathit{set}\ Ls - L \notin \mathit{set}\ Ls$ **and**
uminus-uminus-id: $\bigwedge x :: 'a.\ -\ (-\ x) = x$
shows $\neg$ *trail-less Ls L K* $\neg$ *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 $L\ H\ K :: 'a :: \mathit{uminus}$
assumes *uminus-uminus-id*: $\bigwedge x :: 'a.\ -\ (-\ x) = x$ **and**
L-neq-K: $L \neq K$ **and** *L-neq-minus-K*: $L \neq -\ K$ **and**
less-Cons: *trail-less* $(L\ \#\ Ls)\ H\ K$
shows *trail-less Ls H K*
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:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -(Ls\ !\ j)$
shows $Q \subseteq set\ Ls \cup uminus\ `\ set\ Ls \Longrightarrow Q = \{\} \vee (\exists z \in Q.\ \forall y.\ trail\text{-}less\ Ls\ y\ z \longrightarrow y \notin Q)$
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*:
$\forall i{<}length\ (L\ \#\ Ls).\ \forall j{<}length\ (L\ \#\ Ls).\ i \neq j \longrightarrow$
$(L\ \#\ Ls)\ !\ i \neq (L\ \#\ Ls)\ !\ j \wedge (L\ \#\ Ls)\ !\ i \neq -(L\ \#\ Ls)\ !\ j$
by *simp*
hence *pairwise-distinct-Ls*:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -(Ls\ !\ j)$
by (*metis distinct.simps*(*2*) *distinct-conv-nth length-Cons not-less-eq nth-Cons-Suc*)

```
  show ?case
  proof (cases Q = {})
    case True
    thus ?thesis by simp
  next
    case Q-neq-empty: False
      have Q-minus-subset: Q − {L, − L} ⊆ set Ls ∪ uminus ‘ set Ls using
Cons-ind.prems(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 ∃ z∈Q. ∀ y. trail-less (L # Ls) y z ⟶ y ∉ Q
      using Cons-ind.IH[OF Q-minus-subset pairwise-distinct-Ls]
    proof (elim disjE bexE)
      assume Q − {L, −L} = {}
      with Q-neq-empty have Q ⊆ {L, −L} by simp
      have ?thesis if L ∈ Q
        apply (intro bexI[OF - ‹L ∈ Q›] allI impI)
        apply (erule contrapos-pn)
        apply (drule set-rev-mp[OF - ‹Q ⊆ {L, −L}›])
        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 ∉ Q
      proof −
        from ‹L ∉ Q› have Q = {− L}
          using Q-neq-empty ‹Q ⊆ {L, −L}› by auto
        thus ?thesis
          using irreflp-gt-L-Ls[THEN irreflpD, of − L] by auto
      qed
      ultimately show ?thesis by metis
    next
      fix K
      assume K-in-Q-minus: K ∈ Q − {L, −L} and ∀ y. trail-less Ls y K ⟶ y
∉ Q − {L, −L}
      from K-in-Q-minus have L ≠ K − L ≠ K by auto
      from K-in-Q-minus have L ≠ − K using ‹− L ≠ K› uminus-uminus-id by
blast
      show ?thesis
      proof (intro bexI allI impI)
        show K ∈ 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 ‹L ≠ K› ‹L ≠ - K›])
    hence H ∉ Q - {L, -L}
      using ‹∀ y. trail-less Ls y K ⟶ y ∉ Q - {L, -L}› by simp
    moreover have H ≠ L ∧  H ≠ - L
      using uminus-uminus-id pairwise-distinct-L-Ls ‹trail-less Ls H K›
    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 ∉ Q
      by simp
   qed
  qed
  thus ?thesis by simp
 qed
qed

lemma wfP-trail-less:
 fixes Ls :: ('a :: uminus) list
 assumes
  uminus-not-id: ⋀x :: 'a. - x ≠ x and
  uminus-uminus-id: ⋀x :: 'a. - (- x) = x and
  pairwise-distinct:
   ∀ i < length Ls. ∀ j < length Ls. i ≠ j ⟶ Ls ! i ≠ Ls ! j ∧ Ls ! i ≠ - (Ls !
j)
 shows wfP (trail-less Ls)
 unfolding wfP-eq-minimal
proof (intro allI impI)
 fix M :: 'a set and L :: 'a
 assume L ∈ M
 show ∃ z ∈ M. ∀ y. trail-less Ls y z ⟶ y ∉ M
 proof (cases M ∩ (set Ls ∪ uminus ‘ set Ls) = {})
  case True
  with ‹L ∈ M› have L-not-in-Ls: L ∉ set Ls ∧ - L ∉ set Ls
   unfolding disjoint-iff by (metis UnCI image-eqI uminus-uminus-id)
  then show ?thesis
  proof (intro bexI[OF - ‹L ∈ M›] 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 ∉ M ..
  qed
 next
  case False
  hence M ∩ (set Ls ∪ uminus ‘ set Ls) ⊆ set Ls ∪ uminus ‘ set Ls
   by simp
  with False obtain H where
   H-in: H ∈ M ∩ (set Ls ∪ uminus ‘ set Ls) and
   all-lt-H-no-in: ∀ y. trail-less Ls y H ⟶ y ∉ M ∩ (set Ls ∪ uminus ‘ 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 $\forall y.$ *trail-less Ls y H* $\longrightarrow y \notin M$
using *all-lt-H-no-in uminus-uminus-id*
by (*metis Int-iff Un-iff image-eqI not-trail-less-if-undefined*(*1*))
qed
qed
qed

9.7 Extension on All Literals

definition *trail-less-ex* **where**
trail-less-ex lt Ls L K $\longleftrightarrow$
(*if* $L \in$ *set Ls* $\vee -L \in$ *set Ls then*
if $K \in$ *set Ls* $\vee -K \in$ *set Ls then*
trail-less Ls L K
else
True
else
if $K \in$ *set Ls* $\vee -K \in$ *set Ls then*
False
else
lt L K)

lemma
fixes *Ls* :: (*'a* :: *uminus*) *list*
assumes
uminus-uminus-id: $\bigwedge x :: 'a. -(-x) = x$
shows $K \in$ *set Ls* $\vee -K \in$ *set Ls* $\Longrightarrow$ *trail-less-ex lt Ls L K* $\longleftrightarrow$ *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 *trail-less Ls L K* $\Longrightarrow$ *trail-less-ex lt Ls L K*
unfolding *trail-less-ex-def*
using *defined-if-trail-less*[*THEN defined-conv*[*OF uminus-uminus-id*, *THEN iffD1*]]
by *auto*

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

assumes
uminus-uminus-id: $\bigwedge x :: 'a.\ -\ (-\ x) = x$
shows $L \in set\ Ls \cup uminus\ `\ set\ Ls \Longrightarrow K \notin set\ Ls \cup uminus\ `\ set\ Ls \Longrightarrow$ *trail-less-ex lt Ls L K*
using *defined-conv uminus-uminus-id*
by (*auto simp add*: *trail-less-ex-def*)

lemma *irreflp-trail-ex-less*:
fixes $Ls :: ('a :: uminus)\ list$ **and** $lt :: 'a \Rightarrow 'a \Rightarrow bool$
assumes
uminus-not-id: $\bigwedge x :: 'a.\ -\ x \neq x$ **and**
uminus-uminus-id: $\bigwedge x :: 'a.\ -\ (-\ x) = x$ **and**
pairwise-distinct:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -\ (Ls\ !\ j)$ **and**
irreflp-lt: *irreflp lt*
shows *irreflp* (*trail-less-ex lt Ls*)
unfolding *trail-less-ex-def*
using *irreflp-trail-less*[*OF uminus-not-id uminus-uminus-id pairwise-distinct*] *irreflp-lt*
by (*simp add*: *irreflpD irreflpI*)

lemma *transp-trail-less-ex*:
fixes $Ls :: ('a :: uminus)\ list$
assumes
uminus-not-id: $\bigwedge x :: 'a.\ -\ x \neq x$ **and**
uminus-uminus-id: $\bigwedge x :: 'a.\ -\ (-\ x) = x$ **and**
pairwise-distinct:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -\ (Ls\ !\ j)$ **and**
transp-lt: *transp lt*
shows *transp* (*trail-less-ex lt Ls*)
unfolding *trail-less-ex-def*
using *transp-trail-less*[*OF uminus-not-id uminus-uminus-id pairwise-distinct*] *transp-lt*
by (*smt* (*verit*, *ccfv-SIG*) *transp-def*)

lemma *asymp-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:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -\ (Ls\ !\ j)$ **and**
asymp-lt: *asymp lt*
shows *asymp* (*trail-less-ex lt Ls*)
unfolding *trail-less-ex-def*
using *asymp-trail-less*[*OF uminus-not-id uminus-uminus-id pairwise-distinct*] *asymp-lt*

by (*auto intro*: *asympI dest*: *asympD*)

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* ∪ *set Ls* ∪ *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:
$\forall i < length\ Ls.\ \forall j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -(Ls\ !\ j)$ **and**
wfP-lt: *wfP lt*
shows *wfP* (*trail-less-ex lt Ls*)
unfolding *wfP-eq-minimal*
proof (*intro allI impI*)
fix *Q* :: '*a set* **and** *x* :: '*a*
assume *x* ∈ *Q*
show ∃ *z*∈*Q*. ∀ *y*. *trail-less-ex lt Ls y z* ⟶ *y* ∉ *Q*
proof (*cases Q* ∩ (*set Ls* ∪ *uminus* ‘ *set Ls*) = {})
case *True*
then show *?thesis*
using *wfP-lt*[*unfolded wfP-eq-minimal*, *rule-format*, *OF* ‹*x* ∈ *Q*›]
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* ∩ (*set Ls* ∪ *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 *asymp-trail-term-less*:
assumes *distinct ts*
shows *asymp* (*trail-term-less ts*)
by (*rule asympI*)
(*metis assms distinct-Ex1 dual-order.strict-trans nth-mem order-less-imp-not-less*
trail-term-less-def)

lemma *irreflp-trail-term-less*:
assumes *distinct ts*
shows *irreflp* (*trail-term-less ts*)
using *assms irreflp-on-if-asymp-on*[*OF asymp-trail-term-less*] **by** *metis*

lemma *totalp-on-trail-term-less*:
shows *totalp-on* (*set ts*) (*trail-term-less ts*)
by (*rule totalp-onI*) (*metis in-set-conv-nth nat-neq-iff trail-term-less-def*)

lemma *wfP-trail-term-less*:
assumes *distinct ts*
shows *wfP* (*trail-term-less ts*)
proof (*rule wfP-if-convertible-to-nat*)
fix *t1 t2* **assume** *trail-term-less ts t1 t2*
then obtain *i j* **where** *i*<*length ts* **and** *j*<*i* **and** *t1* $=$ *ts* ! *i* **and** *t2* $=$ *ts* ! *j*
unfolding *trail-term-less-def* **by** *auto*
then show *index* (*rev ts*) *t1* $<$ *index* (*rev ts*) *t2*
using *assms diff-commute index-nth-id index-rev* **by** *fastforce*
qed

lemma *trail-term-less-Cons-if-mem*:
assumes *y* $\in$ *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* ((*3*! (-/|:|-)./ -) [*0*, *0*, *10*] *10*)
-fBex :: *pttrn* ⇒ *'a fset* ⇒ *bool* ⇒ *bool* ((*3*? (-/|:|-)./ -) [*0*, *0*, *10*] *10*)

syntax
-fBall :: *pttrn* ⇒ *'a fset* ⇒ *bool* ⇒ *bool* ((*3*∀ (-/|∈|-)./ -) [*0*, *0*, *10*] *10*)
-fBex :: *pttrn* ⇒ *'a fset* ⇒ *bool* ⇒ *bool* ((*3*∃ (-/|∈|-)./ -) [*0*, *0*, *10*] *10*)

translations
∀ *x*|∈|*A*. *P* ⇌ *CONST fBall A* (λ*x*. *P*)
∃ *x*|∈|*A*. *P* ⇌ *CONST fBex A* (λ*x*. *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-mset* (*add-mset x M*) = *finsert x* (*fset-mset M*)
by (*simp add*: *fset-mset-def*)

lemma *fset-fset-mset*[*simp*]: *fset* (*fset-mset M*) = *set-mset M*
by (*induction M rule*: *multiset-induct*) *simp-all*

lemma *fmember-fset-mset-iff*[*simp*]: $x \mid\in\mid$ *fset-mset* $M \longleftrightarrow x \in\# M$
by (*induction M rule*: *multiset-induct*) *simp-all*

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

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

lemma *fmember-ffUnion-iff*: $a \mid\in\mid \textit{ffUnion}\ (f\ \mid`\mid\ A) \longleftrightarrow (\exists x \mid\in\mid A.\ a \mid\in\mid f\ x)$
unfolding *ffUnion.rep-eq* **by** *simp*

lemma *fBex-ffUnion-iff*: $(\exists z \mid\in\mid \textit{ffUnion}\ (f\ \mid`\mid\ A).\ P\ z) \longleftrightarrow (\exists x \mid\in\mid A.\ \exists z \mid\in\mid f\ x.\ P\ z)$
unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

lemma *fBall-ffUnion-iff*: $(\forall z \mid\in\mid \textit{ffUnion}\ (f\ \mid`\mid\ A).\ P\ z) \longleftrightarrow (\forall x \mid\in\mid A.\ \forall z \mid\in\mid f\ x.\ P\ z)$
unfolding *ffUnion.rep-eq fimage.rep-eq* **by** *blast*

abbreviation *grounding-lits-of-clss* **where**
$\textit{grounding-lits-of-clss}\ N \equiv \{L \cdot l\ \gamma \mid L\ \gamma.\ L \in \bigcup(\textit{set-mset}\ `\ N) \wedge \textit{is-ground-lit}\ (L \cdot l\ \gamma)\}$

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

corollary *grounding-lits-of-learned-subset-grounding-lits-of-initial*:
assumes *initial-lits-generalize-learned-trail-conflict N S*
shows $\textit{grounding-lits-of-clss}\ (\textit{fset}\ (\textit{state-learned}\ S)) \subseteq \textit{grounding-lits-of-clss}\ (\textit{fset}\ N)$
(**is** *?lhs* $\subseteq$ *?rhs*)
proof (*rule subsetI*)
from *assms*(*1*) **have** *N-lits-sup*: $\textit{clss-lits-generalize-clss-lits}\ (\textit{fset}\ N)\ (\textit{fset}\ (\textit{state-learned}\ S))$
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**
$L\text{-}def$: $L = L' \cdot l\ \gamma$ **and**
$L' \in \bigcup\ (\textit{set-mset}\ `\ \textit{fset}\ (\textit{state-learned}\ S))$ **and**
$\textit{is-ground-lit}\ (L' \cdot l\ \gamma)$
by *auto*
then obtain $L_N\ \sigma_N$ **where** $L_N \in \bigcup(\textit{set-mset}\ `\ \textit{fset}\ N)$ **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 ‹*is-ground-lit* $(L' \cdot l\ \gamma)$›
unfolding *L-def* ‹$L_N \cdot l\ \sigma_N = L'$›[*symmetric*]
by (*metis subst-lit-comp-subst*)
qed

lemma *grounding-lits-of-clss-conv*:
grounding-lits-of-clss $N = \{L \mid L\ C.$ *add-mset* $L\ C \in$ *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** $L' \in \bigcup$ (*set-mset* ‘ N) **and** *is-ground-lit* $(L' \cdot l\ \gamma)$
by *auto*

from ‹$L' \in \bigcup$ (*set-mset* ‘ N)› **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 ‹$L' \in\# C$› ‹$L = L' \cdot l\ \gamma$›
by (*metis Melem-subst-cls* ‹*is-ground-lit* $(L' \cdot l\ \gamma)$› *is-ground-subst-lit*)
then obtain C' **where** $C \cdot \gamma \cdot \gamma_C =$ *add-mset* $L\ C'$
using *multi-member-split* **by** *metis*

moreover have $C \cdot \gamma \cdot \gamma_C \in$ *grounding-of-clss* N
unfolding *grounding-of-clss-def*
proof (*rule UN-I*)
show $C \in N$
using ‹$C \in N$› **.**
next
show $C \cdot \gamma \cdot \gamma_C \in$ *grounding-of-cls* C
using ‹*is-ground-cls* $(C \cdot \gamma \cdot \gamma_C)$›
by (*metis grounding-of-cls-ground grounding-of-subst-cls-subset insert-absorb insert-subset*)
qed

ultimately show $L \in$ *?rhs*
by *auto*
next
fix L
assume $L \in$ *?rhs*
then obtain C **where** *add-mset* $L\ C \in$ *grounding-of-clss* N
by *auto*
then obtain $CC\ \gamma$ **where** $CC \in N$ **and** $CC \cdot \gamma =$ *add-mset* $L\ C$

```
    unfolding grounding-of-clss-def
    by (smt (verit, best) UN-iff grounding-of-cls-def mem-Collect-eq)
  then obtain L' C' where CC = add-mset L' C' and L = L' ·l γ and C = C'
· γ
    by (metis (no-types, lifting) msed-map-invR subst-cls-def)

  show L ∈ ?lhs
  proof (intro CollectI exI conjI)
    show L = L' ·l γ
      using ‹L = L' ·l γ› by simp
  next
    show L' ∈ ⋃ (set-mset ‘ N)
      using ‹CC ∈ N› ‹CC = add-mset L' C'›
      by (metis Union-iff image-eqI union-single-eq-member)
  next
    show is-ground-lit (L' ·l γ)
      using ‹add-mset L C ∈ grounding-of-clss N› ‹L = L' ·l γ›
      by (metis grounding-ground is-ground-cls-add-mset)
  qed
qed

corollary groundings-of-learned-subset-groundings-of-initial:
  assumes initial-lits-generalize-learned-trail-conflict N S
  defines U ≡ state-learned S
  shows {L | L C. add-mset L C ∈ grounding-of-clss (fset U)} ⊆
    {L | L C. add-mset L C ∈ grounding-of-clss (fset N)}
  using assms grounding-lits-of-learned-subset-grounding-lits-of-initial
  unfolding grounding-lits-of-clss-conv
  by simp

end

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

lemma strict-subset-implies-multp_HO: A ⊂# B ⟹ multp_HO r A B
  unfolding multp_HO-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 β S S′ ⟹ *sound-state N β S* ⟹ *sound-state N β S′*
using *scl-preserves-sound-state reasonable-scl-def* **by** *blast*

lemma *reasonable-run-sound-state*:
(*reasonable-scl N β*)** *S S′* ⟹ *sound-state N β S* ⟹ *sound-state N β 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 β U* **where**
Nil[*simp*]: *no-conflict-after-decide N β U* [] |
Cons: (*is-decision-lit Ln* ⟶ (∄ *S′*. *conflict N β* (*Ln* # Γ, *U*, *None*) *S′*)) ⟹
no-conflict-after-decide N β U Γ ⟹ *no-conflict-after-decide N β U* (*Ln* # Γ)

definition *no-conflict-after-decide′* **where**
no-conflict-after-decide′ N β S = *no-conflict-after-decide N β* (*state-learned S*) (*state-trail S*)

lemma *no-conflict-after-decide′-initial-state*[*simp*]: *no-conflict-after-decide′ N β 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 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 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 β S S′* **and** ∄ *S″*. *conflict 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 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 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*: *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 ∄*γ*. *is-ground-cls* (*C* · *γ*) ∧ *trail-false-cls* Γ (*C* · *γ*)
shows ∄*S′*. *conflict N β* (Γ, *U*, *None*) *S′* ⟹ ∄*S′*. *conflict N β* (Γ, *finsert C U*, *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 ∃*S′*. *conflict N β* (Γ, *U*, *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 β U* Γ″

```
    by (induction Γ') (auto elim: no-conflict-after-decide.cases)
  hence no-conflict-after-decide N β (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 Γ'')
   hence ∄γ. is-ground-cls (add-mset L D · γ) ∧ trail-false-cls (Ln # Γ'') (add-mset
L D · γ)
      by metis
    hence ∄γ. is-ground-cls (add-mset L D · γ) ∧ trail-false-cls Γ'' (add-mset L D
· γ)
     by (metis (no-types, opaque-lifting) image-insert insert-iff list.set(2) trail-false-cls-def
         trail-false-lit-def)
    hence 1: no-conflict-after-decide N β (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 ∄S'. conflict N β (Ln # Γ'', U, None) S'
        by simp
     then show ∄S'. conflict N β (Ln # Γ'', finsert (add-mset L D) U, None) S'
        using learning-clause-without-conflict-preserves-nex-conflict
          using ‹∄γ. is-ground-cls (add-mset L D · γ) ∧ trail-false-cls (Ln # Γ'')
(add-mset L D · γ)›
        by blast
    next
      show no-conflict-after-decide N β (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 β S S' and no-conflict-after-decide' N β S
  shows no-conflict-after-decide' N β 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 β S1 S2* **and**
invars: *learned-nonempty S1 trail-propagated-or-decided′ N β S1*
no-conflict-after-decide′ N β S1
shows {#} |∈| *N* ∨ (∃ *S0*. *propagate N β S0 S1*)
using *before-conflict*[*OF conf invars*(*1*,*2*)]
proof (*elim disjE exE*)
fix *S0* **assume** *decide N β S0 S1*
hence *False*
proof (*cases N β S0 S1 rule*: *decide.cases*)
case (*decideI L γ* Γ *U*)
with *invars*(*3*) **have** *no-conflict-after-decide N β U* (*trail-decide* Γ (*L* ·*l γ*))
by (*simp add*: *no-conflict-after-decide′-def*)
hence ∄ *S′*. *conflict N β* (*trail-decide* Γ (*L* ·*l γ*), *U*, *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 β S S′* ⟹ *regular-scl N β S S′*
by (*simp add*: *regular-scl-def*)

lemma *regular-scl-if-skip*[*simp*]: *skip N β S S′* ⟹ *regular-scl N β 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 β S S′* ⟹ *regular-scl N β 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 β S S′* ⟹ *regular-scl N β 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 β S S′* ⟹ *regular-scl N β 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 β S S′* ⟹ *sound-state N β S* ⟹ *sound-state N β S′*
by (*rule reasonable-scl-sound-state*[*OF reasonable-if-regular*])

lemma *regular-run-sound-state*:

(*regular-scl N β*)** *S S'* ⟹ *sound-state N β S* ⟹ *sound-state N β 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 β U* **where**
Nil: (∄*S'*. *conflict N β* ([], *U*, *None*) *S'*) ⟹ *no-conflict-with-trail N β U* [] |
Cons: (∄*S'*. *conflict N β* (*Ln* # Γ, *U*, *None*) *S'*) ⟹
no-conflict-with-trail N β U Γ ⟹ *no-conflict-with-trail N β U* (*Ln* # Γ)

lemma *nex-conflict-if-no-conflict-with-trail*:
assumes *no-conflict-with-trail N β U* Γ
shows ∄*S'*. *conflict N β* (Γ, *U*, *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 β U* Γ
shows ∄*S'*. *conflict N β* ([], *U*, *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 β U Γ ⟹ *no-conflict-after-decide N β U* Γ
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 β U Γ ⟹ *D* |∈| *N* |∪| *U* ⟹ *D* ≠ {#} ⟹ *is-ground-cls* (*D* · *γ*) ⟹
¬ *trail-false-cls* Γ (*D* · *γ*)
proof (*induction* Γ *rule*: *no-conflict-with-trail.induct*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons Ln* Γ)
hence ¬ *trail-false-cls* (*Ln* # Γ) (*D* · *γ*)
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 β S ⟷
{#} |∈| *N* ∧ *state-trail S* = [] ∨
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)

lemma *nex-conflict-if-no-conflict-with-trail''*:
assumes *no-conf*: *state-conflict S* = *None* **and** {#} |∉| *N* **and** *learned-nonempty S*
no-conflict-with-trail N β (*state-learned S*) (*state-trail S*)
shows ∄*S'*. *conflict N β S S'*
proof −
from *no-conf* **obtain** Γ *U* **where** *S-def*: *S* = (Γ, *U*, *None*)
by (*metis state-simp*)

from ‹*learned-nonempty S*› **have** {#} |∉| *U*
by (*simp add*: *S-def learned-nonempty-def*)

show *?thesis*
using *assms*(*4*)
unfolding *S-def state-proj-simp*
proof (*cases N β U* Γ *rule*: *no-conflict-with-trail.cases*)
case *Nil*
then show ∄*S'*. *conflict N β* (Γ, *U*, *None*) *S'*
using ‹{#} |∉| *N*› ‹{#} |∉| *U*›
by (*auto simp*: *trail-false-cls-def elim*: *conflict.cases*)
next
case (*Cons Ln* Γ')
then show ∄*S'*. *conflict N β* (Γ, *U*, *None*) *S'*
by (*auto intro*: *no-conflict-tail-trail*)
qed
qed

lemma *no-conflict-with-trail-if-nex-conflict*:
assumes *no-conf*: ∄*S'*. *conflict N β S S' state-conflict S* = *None*
shows *no-conflict-with-trail N β* (*state-learned S*) (*state-trail S*)
proof −
from *no-conf*(*2*) **obtain** Γ *U* **where** *S-def*: *S* = (Γ, *U*, *None*)
by (*metis state-simp*)

show *?thesis*
using *no-conf*(*1*)
unfolding *S-def state-proj-simp*
proof (*induction* Γ)
case *Nil*
thus *?case* **by** (*simp add*: *no-conflict-with-trail.Nil*)
next
case (*Cons Ln* Γ)
have ∄*a*. *conflict N β* (Γ, *U*, *None*) *a*
by (*rule no-conflict-tail-trail*[*OF Cons.prems*])
hence *no-conflict-with-trail N β U* Γ
by (*rule Cons.IH*)
then show *?case*
using *Cons.prems*

```
    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 β U Γ ⟹ almost-no-conflict-with-trail N β (Γ, 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 β initial-state
  by (cases {#} |∈| 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 β S S' and reg-step: regular-scl N β S S'
  shows almost-no-conflict-with-trail N β S'
  using reg-step[unfolded regular-scl-def]
proof (elim disjE conjE)
  assume conflict N β S S'
  with step have False
    using conflict-well-defined by blast
  thus ?thesis ..
next
  assume no-conf: ∄S'. conflict N β S S' and reasonable-scl N β S S'
  from step show ?thesis
  proof (cases N β S S' rule: propagate.cases)
    case step-hyps: (propagateI C U L C' γ C₀ C₁ Γ μ)
    have no-conflict-with-trail N β U Γ
      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 β S S' and reg-step: regular-scl N β S S'
  shows almost-no-conflict-with-trail N β S'
proof −
  from reg-step have res-step: reasonable-scl N β S S'
    by (rule reasonable-if-regular)

  from step obtain Γ U where S'-def: S' = (Γ, U, None)
    by (auto elim: decide.cases)

  have no-conflict-with-trail N β (state-learned S') (state-trail S')
  proof (rule no-conflict-with-trail-if-nex-conflict)
    show ∄S''. conflict N β 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 β (Γ, U, Cl1) ⟷
   almost-no-conflict-with-trail N β (Γ, U, Cl2)
  by (simp add: almost-no-conflict-with-trail-def)

lemma conflict-preserves-almost-no-conflict-with-trail:
  assumes step: conflict N β S S' and invar: almost-no-conflict-with-trail N β S
  shows almost-no-conflict-with-trail N β S'
proof −
  from step obtain Γ U Cl where S = (Γ, U, None) and S' = (Γ, 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 β S S' and invar: almost-no-conflict-with-trail N β S
  shows almost-no-conflict-with-trail N β S'
  using step
proof (cases N β S S' rule: skip.cases)
  case step-hyps: (skipI L D σ n Γ U)
  have no-conflict-with-trail N β 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 β U Γ
    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 β S S' and invar: almost-no-conflict-with-trail N β S
  shows almost-no-conflict-with-trail N β S'
proof −
  from step obtain Γ U Cl1 Cl2 where S = (Γ, U, Some Cl1) and S' = (Γ, 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 β S S′* **and** *invar*: *almost-no-conflict-with-trail N β S*
shows *almost-no-conflict-with-trail N β S′*
proof −
from *step* **obtain** Γ *U Cl1 Cl2* **where** $S = (\Gamma,\ U,\ \mathit{Some}\ \mathit{Cl1})$ **and** $S' = (\Gamma,\ U,\ \mathit{Some}\ \mathit{Cl2})$
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 β S S′* **and** *invar*: *almost-no-conflict-with-trail N β S*
shows *almost-no-conflict-with-trail N β S′*
using *step*
proof (*cases N β S S′ rule*: *backtrack.cases*)
case *step-hyps*: (*backtrackI* Γ Γ′ Γ″ *K L σ D U*)
from *invar* **have** *no-conflict-with-trail N β U* $((-\ (L \cdot l\ \sigma),\ \mathit{None})\ \#\ \Gamma' @ \Gamma'')$
by (*simp add*: *step-hyps almost-no-conflict-with-trail-def decide-lit-def is-decision-lit-def*)
hence *no-conflict-with-trail N β U* $(\Gamma' @ \Gamma'')$
by (*auto elim*: *no-conflict-with-trail.cases*)
hence *no-conflict-with-trail N β U* Γ″
by (*induction* Γ′) (*auto elim*: *no-conflict-with-trail.cases*)
then have *no-conflict-with-trail N β* (*finsert* (*add-mset L D*) *U*) Γ″
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 β S S′* **and** *almost-no-conflict-with-trail N β S*
shows *almost-no-conflict-with-trail N β S′*
using *assms*
using *propagate-preserves-almost-no-conflict-with-trail decide-preserves-almost-no-conflict-with-trail*
conflict-preserves-almost-no-conflict-with-trail skip-preserves-almost-no-conflict-with-trail
factorize-preserves-almost-no-conflict-with-trail resolve-preserves-almost-no-conflict-with-trail
backtrack-preserves-almost-no-conflict-with-trail
by (*metis scl-def reasonable-if-regular scl-if-reasonable*)

11.1.2 Backtrack Follows Regular Conflict Resolution

lemma *before-conflict-in-regular-run*:

```
  assumes
    reg-run: (regular-scl N β)** initial-state S1 and
    conf: conflict N β S1 S2 and
    {#} |∉| N
  shows ∃S0. (regular-scl N β)** initial-state S0 ∧ regular-scl N β S0 S1 ∧
(propagate N β S0 S1)
proof −
  from reg-run conf show ?thesis
  proof (induction S1 arbitrary: S2 rule: rtranclp-induct)
    case base
    with ‹{#} |∉| N› 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 β S0
      by (induction S0 rule: rtranclp-induct)
      (simp-all add: scl-preserves-trail-propagated-or-decided[OF scl-if-reasonable[OF
            reasonable-if-regular]])
    with step.hyps(2) have trail-propagated-or-decided′ N β S1
     by (simp add: scl-preserves-trail-propagated-or-decided[OF scl-if-reasonable[OF
            reasonable-if-regular]])

    from step.hyps(1) have almost-no-conflict-with-trail N β S0
      by (induction S0 rule: rtranclp-induct)
        (simp-all add: regular-scl-preserves-almost-no-conflict-with-trail)
    with step.hyps(2) have almost-no-conflict-with-trail N β S1
      by (simp add: regular-scl-preserves-almost-no-conflict-with-trail)

    show ?case
    proof (intro exI conjI)
      show (regular-scl N β)** initial-state S0
        using step.hyps by simp
    next
      show regular-scl N β S0 S1
        using step.hyps by simp
    next
      from step.prems obtain Γ U C γ where
        S1-def: S1 = (Γ, U, None) and
        S2-def: S2 = (Γ, U, Some (C, γ)) and
        C-in: C |∈| N |∪| U and
```

```
      ground-conf: is-ground-cls (C · γ) and
      tr-false-conf: trail-false-cls Γ (C · γ)
      unfolding conflict.simps by auto
    with step.hyps have ¬ conflict N β S0 S1 and reasonable-scl N β S0 S1
      unfolding regular-scl-def by (simp-all add: conflict.simps)
    with step.prems have scl N β S0 S1 and ¬ decide N β S0 S1
      unfolding reasonable-scl-def by blast+
    moreover from step.prems 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.prems
    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.prems)
      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.prems 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 β S ⟷ {#} |∉| *N* ⟶
(*case state-conflict S of*
None ⇒ (*regular-scl N β*)** *initial-state S* |
Some - ⇒ (∃ *S0 S1 S2 S3*. (*regular-scl N β*)** *initial-state S0* ∧
propagate N β S0 S1 ∧ *regular-scl N β S0 S1* ∧
conflict N β S1 S2 ∧ *regular-scl N β S1 S2* ∧
(*factorize N β*)** *S2 S3* ∧ (*regular-scl N β*)** *S2 S3* ∧
(*S3* = *S* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*))))

lemma *regular-conflict-resolution-initial-state*[*simp*]:
regular-conflict-resolution N β initial-state
by (*simp add*: *regular-conflict-resolution-def*)

lemma *propagate-preserves-regular-conflict-resolution*:
assumes *step*: *propagate 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* **have** *state-conflict S* = *None* **and** *state-conflict S′* = *None*
by (*auto elim*: *propagate.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* ‹*state-conflict S′* = *None*›
unfolding *option.case*
proof (*rule impI*)
assume {#} |∉| *N*
with *invar* **have** (*regular-scl N β*)** *initial-state S*
unfolding *regular-conflict-resolution-def* ‹*state-conflict S* = *None*› **by** *simp*
thus (*regular-scl N β*)** *initial-state S′*
using *reg-step* **by** (*rule rtranclp.rtrancl-into-rtrancl*)
qed
qed

lemma *decide-preserves-regular-conflict-resolution*:
assumes *step*: *decide 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* **have** *state-conflict S* = *None* **and** *state-conflict S′* = *None*
by (*auto elim*: *decide.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* ‹*state-conflict S′* = *None*›
unfolding *option.case*
proof (*rule impI*)
assume {#} |∉| *N*

```
    with invar have (regular-scl N β)** initial-state S
      unfolding regular-conflict-resolution-def ‹state-conflict S = None› by simp
    thus (regular-scl N β)** initial-state S'
      using reg-step by (rule rtranclp.rtrancl-into-rtrancl)
  qed
qed

lemma conflict-preserves-regular-conflict-resolution:
  assumes step: conflict 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 = None and state-conflict S' =
Some (C, γ)
    by (auto elim!: conflict.cases)

  show ?thesis
    unfolding regular-conflict-resolution-def ‹state-conflict S' = Some (C, γ)›
    unfolding option.cases
  proof (rule impI)
    assume {#} |∉| N
    with invar have reg-run: (regular-scl N β)** initial-state S
      unfolding regular-conflict-resolution-def ‹state-conflict S = None› by simp

    from ‹{#} |∉| N› obtain S0 where
      (regular-scl N β)** initial-state S0 propagate N β S0 S regular-scl N β S0 S
      using before-conflict-in-regular-run[OF reg-run step] by metis

    with step show ∃ S0 S1 S2 S3. (regular-scl N β)** initial-state S0 ∧
      propagate N β S0 S1 ∧ regular-scl N β S0 S1 ∧
      conflict N β S1 S2 ∧ regular-scl N β S1 S2 ∧
      (factorize N β)** S2 S3 ∧ (regular-scl N β)** S2 S3 ∧
      (S3 = S' ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S'))
      using regular-scl-if-conflict
      by blast
  qed
qed

lemma
  assumes almost-no-conflict-with-trail N β S and {#} |∉| N
  shows no-conflict-after-decide' N β S
proof −
  obtain U Γ Cl where S-def: S = (Γ, U, Cl)
    by (metis state-simp)

  show ?thesis
  proof (cases Γ)
    case Nil
```

```
    thus ?thesis
      by (simp add: S-def no-conflict-after-decide'-def)
  next
    case (Cons Ln Γ')
    with assms have no-conf-with-trail:
      no-conflict-with-trail N β U (if is-decision-lit Ln then Ln # Γ' else Γ')
      by (simp add: S-def almost-no-conflict-with-trail-def)

    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 β S ⟹ {#} |∉| N ⟹ {#} |∉| 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 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' = Some (C, γ)
    by (auto elim!: skip.cases)

  show ?thesis
    unfolding regular-conflict-resolution-def ‹state-conflict S' = Some (C, γ)›
    unfolding option.cases
  proof (intro 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 have (regular-scl N β)** initial-state S1
      using ‹regular-scl N β S0 S1› by simp
```

hence (*regular-scl N β*)** *initial-state S2*
using ‹*regular-scl N β S1 S2*› **by** *simp*
hence (*regular-scl N β*)** *initial-state S3*
using ‹(*regular-scl N β*)** *S2 S3*› **by** *simp*

from ‹(*factorize N β*)** *S2 S3*› **have** *state-trail S3 = state-trail S2*
by (*induction S3 rule*: *rtranclp-induct*) (*auto elim*: *factorize.cases*)
also from ‹*conflict N β S1 S2*› **have** *... = state-trail S1*
by (*auto elim*: *conflict.cases*)
finally have *state-trail S3 = state-trail S1*
by *assumption*

from ‹(*factorize N β*)** *S2 S3*› **have** *state-learned S3 = state-learned S2*
proof (*induction S3 rule*: *rtranclp-induct*)
case *base*
show *?case* **by** *simp*
next
case (*step y z*)
thus *?case*
by (*elim factorize.cases*) *simp*
qed
also from ‹*conflict N β S1 S2*› **have** *... = state-learned S1*
by (*auto elim*: *conflict.cases*)
finally have *state-learned S3 = state-learned S1*
by *assumption*

from ‹*propagate N β S0 S1*› **have** *state-learned S1 = state-learned S0*
by (*auto elim*: *propagate.cases*)

from ‹*propagate N β S0 S1*› **obtain** *L C γ* **where**
state-trail S1 = trail-propagate (*state-trail S0*) *L C γ*
by (*auto elim*: *propagate.cases*)

from ‹(*regular-scl N β*)** *initial-state S3*› **have** *almost-no-conflict-with-trail N β S3*
using *regular-scl-preserves-almost-no-conflict-with-trail*
by (*induction S3 rule*: *rtranclp-induct*) *simp-all*

show ∃ *S0 S1 S2 S3*. (*regular-scl N β*)** *initial-state S0* ∧
propagate N β S0 S1 ∧ *regular-scl N β S0 S1* ∧
conflict N β S1 S2 ∧ *regular-scl N β S1 S2* ∧
(*factorize N β*)** *S2 S3* ∧ (*regular-scl N β*)** *S2 S3* ∧
(*S3 = S′* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S′*))
using *reg-run propa confl facto*
proof (*intro impI exI conjI*)
show *S3 = S′* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S′*)
using *maybe-reso*

proof (*elim disjE exE conjE*)
fix *S4* **assume** *resolve N β S3 S4* **and** (*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S4 S*
with *step* **have** ∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S4 S'*
by (*meson rtranclp.rtrancl-into-rtrancl sup2CI*)
thus *?thesis* **..**
next
assume *S3 = S*
with ‹*almost-no-conflict-with-trail N β S3*› ‹{#} |∉| *N*›
have *no-conf-with-trail*: *no-conflict-with-trail N β* (*state-learned S*)
(*case state-trail S of* [] ⇒ [] | *Ln* # Γ ⇒ *if is-decision-lit Ln then Ln* # Γ *else* Γ)
by (*simp add*: *almost-no-conflict-with-trail-def*)
hence {#} |∉| *state-learned S*
using *nex-conflict-if-no-conflict-with-trail'*[*folded mempty-in-iff-ex-conflict*]
by *simp*

from *no-conf-with-trail*
have *no-conf-with-trail'*: *no-conflict-with-trail N β* (*state-learned S1*) (*state-trail S0*)
using ‹*S3 = S*› ‹*state-trail S3 = state-trail S1*›
‹*state-learned S3 = state-learned S1*›
‹*state-trail S1 = trail-propagate* (*state-trail S0*) *L C γ*›
by (*simp add*: *propagate-lit-def is-decision-lit-def*)

have ∃ *D γ*$_D$. *state-conflict S2 = Some* (*D*, *γ*$_D$) ∧ − (*L* ·*l γ*) ∈# *D* · *γ*$_D$
using ‹*conflict N β S1 S2*›
proof (*cases N β S1 S2 rule*: *conflict.cases*)
case (*conflictI D U γ*$_D$ Γ)
hence *trail-false-cls* (*trail-propagate* (*state-trail S0*) *L C γ*) (*D* · *γ*$_D$)
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* · *γ*$_D$)
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*) *local.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 β S′*
proof −
from *step* **obtain** *C γ C′ γ′* **where**
state-conflict S = Some (C, γ) **and** *state-conflict S′ = Some (C′, γ′)*
by (*auto elim*!: *factorize.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* ‹*state-conflict S′ = Some (C′, γ′)*›
unfolding *option.cases*
proof (*intro 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*

show ∃ *S0 S1 S2 S3*. (*regular-scl N β*)** *initial-state S0* ∧
propagate N β S0 S1 ∧ *regular-scl N β S0 S1* ∧
conflict N β S1 S2 ∧ *regular-scl N β S1 S2* ∧
(*factorize N β*)** *S2 S3* ∧ (*regular-scl N β*)** *S2 S3* ∧
(*S3 = S′* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S′*))
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 β*)** *S2 S′*
using ‹(*factorize N β*)** *S2 S3*› *step*
by (*simp add*: ‹*S3 = S*›)
next
show (*regular-scl N β*)** *S2 S′*
using ‹(*regular-scl N β*)** *S2 S3*› *reg-step*
by (*simp add*: ‹*S3 = S*›)
next
show *S′ = S′* ∨ (∃ *S4*. *resolve N β S′ S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S′*)
by *simp*
qed
next
fix *S4* **assume** *hyps*: *resolve N β S3 S4* (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*

show *?thesis*
using *reg-run propa confl facto*
proof (*intro exI conjI*)
show *S3* = *S'* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S'*)
using *hyps step*
by (*meson rtranclp.rtrancl-into-rtrancl sup2CI*)
qed
qed
qed
qed

lemma *resolve-preserves-regular-conflict-resolution*:
assumes *step*: *resolve 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 C' γ'* **where**
state-conflict S = *Some* (*C*, *γ*) **and** *state-conflict S'* = *Some* (*C'*, *γ'*)
by (*auto elim*!: *resolve.cases*)

show *?thesis*
unfolding *regular-conflict-resolution-def* ‹*state-conflict S'* = *Some* (*C'*, *γ'*)›
unfolding *option.cases*
proof (*intro impI*)
from *step* **have** *state-conflict S* ≠ *None*
by (*auto elim*: *resolve.cases*)

assume {#} |∉| *N*
with *invar* **obtain** *S0 S1 S2 S3* **where**
reg-run: (*regular-scl N β*)** *initial-state S0* **and**
propagate N β S0 S1 regular-scl N β S0 S1 **and**
conflict N β S1 S2 regular-scl N β S1 S2 **and**
(*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*

then show ∃ *S0 S1 S2 S3*. (*regular-scl N β*)** *initial-state S0* ∧
propagate N β S0 S1 ∧ *regular-scl N β S0 S1* ∧
conflict N β S1 S2 ∧ *regular-scl N β S1 S2* ∧
(*factorize N β*)** *S2 S3* ∧ (*regular-scl N β*)** *S2 S3* ∧
(*S3* = *S'* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S'*))
proof (*intro exI conjI*)
show *S3* = *S'* ∨ (∃ *S4*. *resolve N β S3 S4* ∧ (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S'*)

```
      using maybe-reso step
        by (metis (no-types, opaque-lifting) rtranclp.rtrancl-into-rtrancl rtran-
clp.rtrancl-refl
        sup2I2)
    qed
  qed
qed

lemma backtrack-preserves-regular-conflict-resolution:
  assumes step: backtrack N β S S′ and reg-step: regular-scl N β S S′ and
    invar: regular-conflict-resolution N β S
  shows regular-conflict-resolution N β S′
proof −
  from step obtain C γ where
    state-conflict S = Some (C, γ) and state-conflict S′ = None
    by (auto elim!: backtrack.cases)

  show ?thesis
    unfolding regular-conflict-resolution-def ‹state-conflict S′ = None›
    unfolding option.case
  proof (rule impI)
    assume {#} |∉| N
    with invar obtain S0 S1 S2 S3 where
      reg-run: (regular-scl N β)** initial-state S0 and
      propa: propagate N β S0 S1 regular-scl N β S0 S1 and
      confl: conflict N β S1 S2 regular-scl N β S1 S2 and
      facto: (factorize N β)** S2 S3 (regular-scl N β)** S2 S3 and
      maybe-reso: S3 = S ∨ (∃ S4. resolve N β S3 S4 ∧ (skip N β ⊔ factorize N β
⊔ resolve N β)** S4 S)
      unfolding regular-conflict-resolution-def ‹state-conflict S = Some (C, γ)›
      unfolding option.cases
      by metis

    from reg-run propa(2) confl(2) facto(2) have reg-run-S3: (regular-scl N β)**
initial-state S3
      by simp

    show (regular-scl N β)** initial-state S′
      using maybe-reso
    proof (elim disjE exE conjE)
      show S3 = S ⟹ (regular-scl N β)** initial-state S′
        using reg-run-S3 reg-step by simp
    next
      fix S4 assume hyps: resolve N β S3 S4 (skip N β ⊔ factorize N β ⊔ resolve
N β)** S4 S
      have (regular-scl N β)** initial-state S4
        using reg-run-S3 regular-scl-if-resolve[OF hyps(1)]
        by (rule rtranclp.rtrancl-into-rtrancl)
      also have (regular-scl N β)** S4 S
```

using *hyps*(*2*)
by (*rule mono-rtranclp*[*rule-format*, *rotated*]) *auto*
also have (*regular-scl N* β)** *S S'*
using *reg-step* **by** *simp*
finally show (*regular-scl N* β)** *initial-state S'*
by *assumption*
qed
qed
qed

lemma *regular-scl-preserves-regular-conflict-resolution*:
assumes *reg-step*: *regular-scl N* β *S S'* **and**
invars: *regular-conflict-resolution N* β *S*
shows *regular-conflict-resolution N* β *S'*
using *assms*
using *propagate-preserves-regular-conflict-resolution decide-preserves-regular-conflict-resolution conflict-preserves-regular-conflict-resolution skip-preserves-regular-conflict-resolution factorize-preserves-regular-conflict-resolution resolve-preserves-regular-conflict-resolution backtrack-preserves-regular-conflict-resolution*
by (*metis regular-scl-def reasonable-scl-def scl-def*)

11.2 Miscellaneous Lemmas

lemma *mempty-not-in-initial-clauses-if-non-empty-regular-conflict*:
assumes *state-conflict S* = *Some* (*C*, γ) **and** *C* $\neq$ {#} **and**
invars: *almost-no-conflict-with-trail N* β *S sound-state N* β *S ground-false-closures S*
shows {#} |$\notin$| *N*
proof −
from *assms*(*1*) **obtain** Γ *U* **where** *S-def*: *S* = (Γ, *U*, *Some* (*C*, γ))
by (*metis state-simp*)

from *assms*(*2*) **obtain** *L C'* **where** *C-def*: *C* = *add-mset L C'*
using *multi-nonempty-split* **by** *metis*

from *invars*(*3*) **have** *trail-false-cls* Γ (*C* · γ)
by (*simp add*: *S-def ground-false-closures-def*)
then obtain *Ln* Γ' **where** Γ = *Ln* # Γ'
by (*metis assms*(*2*) *neq-Nil-conv not-trail-false-Nil*(*2*) *subst-cls-empty-iff*)
with *invars*(*1*) **have** *no-conflict-with-trail N* β *U* (*if is-decision-lit Ln then Ln # Γ' else Γ'*)
by (*simp add*: *S-def almost-no-conflict-with-trail-def*)
hence ∄*S'*. *conflict N* β ([], *U*, *None*) *S'*
by (*rule nex-conflict-if-no-conflict-with-trail'*)
hence {#} |$\notin$| *N* |∪| *U*
unfolding *mempty-in-iff-ex-conflict*[*symmetric*] **by** *assumption*
thus *?thesis*
by *simp*
qed

lemma *mempty-not-in-initial-clauses-if-regular-run-reaches-non-empty-conflict*:
assumes (*regular-scl N β*)** *initial-state S* **and** *state-conflict S* = *Some* (*C*, *γ*)
and *C* ≠ {#}
shows {#} |∉| *N*
proof (*rule notI*)
from *assms*(*2*) **have** *initial-state* ≠ *S* **by** *fastforce*
then obtain *S′* **where**
reg-scl-init-S′: *regular-scl N β initial-state S′* **and** (*regular-scl N β*)** *S′ S*
by (*metis assms*(*1*) *converse-rtranclpE*)

assume {#} |∈| *N*
hence *conflict N β initial-state* ([], {||}, *Some* ({#}, *Var*))
by (*rule conflict-initial-state-if-mempty-in-intial-clauses*)
hence *conf-init*: *regular-scl N β initial-state* ([], {||}, *Some* ({#}, *Var*))
using *regular-scl-def* **by** *blast*
then obtain *γ* **where** *S′-def*: *S′* = ([], {||}, *Some* ({#}, *γ*))
using *reg-scl-init-S′*
unfolding *regular-scl-def*
using ‹*conflict N β initial-state* ([], {||}, *Some* ({#}, *Var*))›
conflict-initial-state-only-with-mempty
by *blast*

have ∄*S′*. *scl N β* ([], {||}, *Some* ({#}, *γ*)) *S′* **for** *γ*
using *no-more-step-if-conflict-mempty* **by** *simp*
hence ∄*S′*. *regular-scl N β* ([], {||}, *Some* ({#}, *γ*)) *S′* **for** *γ*
using *scl-if-reasonable*[*OF reasonable-if-regular*] **by** *blast*
hence *S* = *S′*
using ‹(*regular-scl N β*)** *S′ S*›
unfolding *S′-def*
by (*metis converse-rtranclpE*)
with *assms*(*2*,*3*) **show** *False* **by** (*simp add*: *S′-def*)
qed

lemma *before-regular-backtrack*:
assumes
backt: *backtrack N β S S′* **and**
invars: *sound-state N β S almost-no-conflict-with-trail N β S*
regular-conflict-resolution N β S ground-false-closures S
shows ∃*S0 S1 S2 S3 S4*. (*regular-scl N β*)** *initial-state S0* ∧
propagate N β S0 S1 ∧ *regular-scl N β S0 S1* ∧
conflict N β S1 S2 ∧ (*factorize N β*)** *S2 S3* ∧ *resolve N β S3 S4* ∧
(*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*
proof −
from *backt* **obtain** *L C γ* **where** *conflict-S*: *state-conflict S* = *Some* (*add-mset L C*, *γ*)
by (*auto elim*: *backtrack.cases*)

have {#} |∉| *N*

proof (*rule mempty-not-in-initial-clauses-if-non-empty-regular-conflict*)
show *state-conflict S = Some (add-mset L C, γ)*
by (*rule ‹state-conflict S = Some (add-mset L C, γ)›*)
next
show *add-mset L C ≠ {#}*
by *simp*
next
show *almost-no-conflict-with-trail N β S*
by (*rule ‹almost-no-conflict-with-trail N β S›*)
next
show *sound-state N β S*
by (*rule ‹sound-state N β S›*)
next
show *ground-false-closures S*
by (*rule ‹ground-false-closures S›*)
qed

then obtain *S0 S1 S2 S3* **where**
reg-run: (*regular-scl N β*)** *initial-state S0* **and**
propa: *propagate N β S0 S1 regular-scl N β S0 S1* **and**
confl: *conflict N β S1 S2* **and**
fact: (*factorize N β*)** *S2 S3* **and**
maybe-resolution: *S3 = S ∨*
(∃ *S4*. *resolve N β S3 S4 ∧* (*skip N β ⊔ factorize N β ⊔ resolve N β*)** *S4 S*)
using ‹*regular-conflict-resolution N β S*› ‹*state-conflict S = Some* (*add-mset L C, γ*)›
unfolding *regular-conflict-resolution-def conflict-S option.case*
by *metis*

have *S3 ≠ S*
proof (*rule notI*)
from ‹(*factorize N β*)** *S2 S3*› **have** *state-trail S3 = state-trail S2*
by (*induction S3 rule*: *rtranclp-induct*) (*auto elim*: *factorize.cases*)
also from ‹*conflict N β S1 S2*› **have** *... = state-trail S1*
by (*auto elim*: *conflict.cases*)
finally have *state-trail S3 = state-trail S1*
by *assumption*
from ‹*propagate N β S0 S1*› **obtain** *L C γ* **where**
state-trail S1 = trail-propagate (*state-trail S0*) *L C γ*
by (*auto elim*: *propagate.cases*)

from ‹(*factorize N β*)** *S2 S3*› **have** *state-learned S3 = state-learned S2*
proof (*induction S3 rule*: *rtranclp-induct*)
case *base*
show *?case* **by** *simp*
next
case (*step y z*)
thus *?case*
by (*elim factorize.cases*) *simp*

qed
also from ‹*conflict N β S1 S2*› **have** ... = *state-learned S1*
by (*auto elim*: *conflict.cases*)
finally have *state-learned S3* = *state-learned S1*
by *assumption*

from ‹*propagate N β S0 S1*› **have** *state-learned S1* = *state-learned S0*
by (*auto elim*: *propagate.cases*)

assume *S3* = *S*
hence *no-conf-with-trail*: *no-conflict-with-trail N β* (*state-learned S0*) (*state-trail S0*)
using ‹*almost-no-conflict-with-trail N β S*› ‹{#} |∉| *N*›
‹*state-trail S1* = *trail-propagate* (*state-trail S0*) *L C γ*› ‹*state-trail S3* = *state-trail S1*›
‹*state-learned S3* = *state-learned S1*› ‹*state-learned S1* = *state-learned S0*›
by (*simp add*: *almost-no-conflict-with-trail-def propagate-lit-def is-decision-lit-def*)
hence {#} |∉| *state-learned S0*
using *nex-conflict-if-no-conflict-with-trail'*[*folded mempty-in-iff-ex-conflict*] **by** *simp*

have ∃ *D γ*$_D$. *state-conflict S2* = *Some* (*D*, *γ*$_D$) ∧ − (*L* ·*l γ*) ∈# *D* · *γ*$_D$
using ‹*conflict N β S1 S2*›
proof (*cases N β S1 S2 rule*: *conflict.cases*)
case (*conflictI D U γ*$_D$ Γ)
hence *trail-false-cls* (*trail-propagate* (*state-trail S0*) *L C γ*) (*D* · *γ*$_D$)
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* · *γ*$_D$)
proof (*rule not-trail-false-cls-if-no-conflict-with-trail*)
show *D* |∈| *N* |∪| *state-learned S0*
using ‹*state-learned S1* = *state-learned S0*› *local.conflictI*(*1*) *local.conflictI*(*3*)
by *fastforce*
next
have {#} |∉| *U*
using ‹{#} |∉| *state-learned S0*› ‹*S3* = *S*› ‹*state-learned S3* = *state-learned S1*›
‹*state-learned S1* = *state-learned S0*›
unfolding *conflictI*(*1*,*2*)
by *simp*
thus *D* ≠ {#}
using ‹{#} |∉| *N*› ‹*D* |∈| *N* |∪| *U*›
by *auto*
next
show *is-ground-cls* (*D* · *γ*$_D$)
by (*rule* ‹*is-ground-cls* (*D* · *γ*$_D$)›)
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 backt ‹S3 = S› show 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: decide-lit-def propagate-lit-def elim!: backtrack.cases)
  qed
  with maybe-resolution obtain S4 where
    resolve N β S3 S4 and (skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S
    by metis
  show ?thesis
  proof (intro exI conjI)
    show (regular-scl N β)** initial-state S0
      by (rule ‹(regular-scl N β)** initial-state S0›)
  next
    show propagate N β S0 S1
      by (rule ‹propagate N β S0 S1›)
  next
    show regular-scl N β S0 S1
      by (rule propa(2))
  next
    show conflict N β S1 S2
      by (rule ‹conflict N β S1 S2›)
  next
```

show (*factorize N β*)** *S2 S3*
by (*rule ‹*(*factorize N β*)** *S2 S3*›*)
next
show *resolve N β S3 S4*
by (*rule ‹resolve N β S3 S4›*)
next
show (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*
by (*rule ‹*(*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*›*)
qed
qed

12 Resolve in Regular Runs

lemma *resolve-if-conflict-follows-propagate*:
assumes
no-conf: ∄ S_1. *conflict N β* S_0 S_1 **and**
propa: *propagate N β* S_0 S_1 **and**
conf: *conflict N β* S_1 S_2
shows ∃ S_3. *resolve N β* S_2 S_3
using *propa*
proof (*cases N β* S_0 S_1 *rule*: *propagate.cases*)
case (*propagateI C U L C′ γ* C_0 C_1 Γ *μ*)
hence S_0-*def*: S_0 = (Γ, *U*, *None*)
by *simp*

from *conf* **obtain** γ_D *D* **where**
S_2-*def*: S_2 = (*trail-propagate* Γ (*L* ·*l μ*) (C_0 · *μ*) *γ*, *U*, *Some* (*D*, γ_D)) **and**
D-in: *D* |∈| *N* |∪| *U* **and**
gr-D-γ_D: *is-ground-cls* (*D* · γ_D) **and**
*tr-false-*Γ*-L-μ*: *trail-false-cls* (*trail-propagate* Γ (*L* ·*l μ*) (C_0 · *μ*) *γ*) (*D* · γ_D)
by (*elim conflict.cases*) (*unfold propagateI*(*1*,*2*), *blast*)

from *no-conf* **have** ¬ *trail-false-cls* Γ (*D* · γ_D)
using *gr-D-*γ_D *D-in not-trail-false-ground-cls-if-no-conflict*[*of N β - D* γ_D]
using S_0-*def* **by** *force*
with *tr-false-*Γ*-L-μ* **have** − (*L* ·*l μ* ·*l γ*) ∈# *D* · γ_D
unfolding *propagate-lit-def* **by** (*metis subtrail-falseI*)
then obtain *D′ L′* **where** *D-def*: *D* = *add-mset L′ D′* **and** *1*: *L* ·*l μ* ·*l γ* = − (*L′* ·*l* γ_D)
by (*metis Melem-subst-cls multi-member-split uminus-of-uminus-id*)

define *ϱ* **where**
ϱ = *renaming-wrt* {*add-mset L* C_0 · *μ*}

have *is-renaming ϱ*
by (*metis ϱ-def finite.emptyI finite.insertI is-renaming-renaming-wrt*)
hence ∀ *x*. *is-Var* (*ϱ x*) **and** *inj ϱ*
by (*simp-all add*: *is-renaming-iff*)

have *disjoint-vars*: $\bigwedge C.$ *vars-cls* $(C \cdot \varrho) \cap$ *vars-cls* $(\mathit{add\text{-}mset}\ L\ C_0 \cdot \mu) = \{\}$
by (*simp add*: *ϱ-def vars-cls-subst-renaming-disj*)

have $\exists \mu'.$ *Unification.mgu* $(\mathit{atm\text{-}of}\ L' \cdot a\ \varrho)\ (\mathit{atm\text{-}of}\ L \cdot a\ \mu) = \mathit{Some}\ \mu'$
proof (*rule ex-mgu-if-subst-eq-subst-and-disj-vars*)
have *vars-lit* $L' \subseteq$ *subst-domain* γ_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* $L' \cdot a\ \varrho \cdot a$ *rename-subst-domain* $\varrho\ \gamma_D =$ *atm-of* $L' \cdot a\ \gamma_D$
by (*rule renaming-cancels-rename-subst-domain*[*OF* ‹$\forall x.$ *is-Var* $(\varrho\ x)$› ‹*inj* *ϱ*›])
then show *atm-of* $L' \cdot a\ \varrho \cdot a$ *rename-subst-domain* $\varrho\ \gamma_D =$ *atm-of* $L \cdot a\ \mu \cdot a\ \gamma$
using *1* **by** (*metis atm-of-subst-lit atm-of-uminus*)
next
show *vars-term* $(\mathit{atm\text{-}of}\ L' \cdot a\ \varrho) \cap$ *vars-term* $(\mathit{atm\text{-}of}\ L \cdot a\ \mu) = \{\}$
using *disjoint-vars*[*of* $\{\#L'\#\}$] **by** *auto*
qed
then obtain μ' **where** *imgu-*μ': *is-imgu* μ' $\{\{\mathit{atm\text{-}of}\ L' \cdot a\ \varrho,\ \mathit{atm\text{-}of}\ (L \cdot l\ \mu)\}\}$
using *is-imgu-if-mgu-eq-Some* **by** *auto*

let *?*Γ*prop* = *trail-propagate* $\Gamma\ (L \cdot l\ \mu)\ (C_0 \cdot \mu)\ \gamma$
let *?*γ*reso* = $\lambda x.$ *if* $x \in$ *vars-cls* $(\mathit{add\text{-}mset}\ L'\ D' \cdot \varrho)$ *then rename-subst-domain* $\varrho\ \gamma_D\ x$ *else* $\gamma\ x$

have *resolve* $N\ \beta$
$(?\Gamma prop,\ U,\ \mathit{Some}\ (\mathit{add\text{-}mset}\ L'\ D',\ \gamma_D))$
$(?\Gamma prop,\ U,\ \mathit{Some}\ ((D' \cdot \varrho + C_0 \cdot \mu \cdot \mathit{Var}) \cdot \mu',\ ?\gamma reso))$
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* $(\mathit{add\text{-}mset}\ L'\ D' \cdot \varrho) \cap$ *vars-cls* $(\mathit{add\text{-}mset}\ (L \cdot l\ \mu)\ (C_0 \cdot \mu) \cdot \mathit{Var}) = \{\}$
using *disjoint-vars*[*of add-mset* $L'\ D'$] **by** *simp*
next
show *is-imgu* μ' $\{\{\mathit{atm\text{-}of}\ L' \cdot a\ \varrho,\ \mathit{atm\text{-}of}\ (L \cdot l\ \mu) \cdot a\ \mathit{Var}\}\}$
using *imgu-*μ' **by** *simp*
next
show *is-grounding-merge* *?*γ*reso*
$(\mathit{vars\text{-}cls}\ (\mathit{add\text{-}mset}\ L'\ D' \cdot \varrho))\ (\mathit{rename\text{-}subst\text{-}domain}\ \varrho\ \gamma_D)$
$(\mathit{vars\text{-}cls}\ (\mathit{add\text{-}mset}\ (L \cdot l\ \mu)\ (C_0 \cdot \mu) \cdot \mathit{Var}))\ (\mathit{rename\text{-}subst\text{-}domain}\ \mathit{Var}\ \gamma)$
using *is-grounding-merge-if-mem-then-else*
by (*metis rename-subst-domain-Var-lhs*)
qed *simp-all*
thus *?thesis*
unfolding *S*$_2$*-def D-def* **by** *metis*

qed

lemma *factorize-preserves-resolvability*:
assumes *reso*: *resolve* $N\ \beta\ S_1\ S_2$ **and** *fact*: *factorize* $N\ \beta\ S_1\ S_3$ **and**
invar: *ground-closures* S_1
shows $\exists S_4.$ *resolve* $N\ \beta\ S_3\ S_4$
using *reso*
proof (*cases* $N\ \beta\ S_1\ S_2$ *rule*: *resolve.cases*)
case (*resolveI* $\Gamma\ \Gamma'\ K\ D\ \gamma_D\ L\ \gamma_C\ \varrho_C\ \varrho_D\ C\ \mu\ \gamma\ U$)

from *fact*[*unfolded resolveI*(*1*,*2*)] **obtain** $LL'\ LL\ CC\ \mu_L$ **where**
S_1-*def*: $S_1 = (\Gamma,\ U,\ \mathit{Some}\ (\mathit{add\text{-}mset}\ LL'\ (\mathit{add\text{-}mset}\ LL\ CC),\ \gamma_C))$ **and**
S_3-*def*: $S_3 = (\Gamma,\ U,\ \mathit{Some}\ (\mathit{add\text{-}mset}\ LL\ CC \cdot \mu_L,\ \gamma_C))$ **and**
$LL \cdot l\ \gamma_C = LL' \cdot l\ \gamma_C$ **and**
imgu-μ_L: *is-imgu* $\mu_L\ \{\{\mathit{atm\text{-}of}\ LL,\ \mathit{atm\text{-}of}\ LL'\}\}$
by (*auto simp*: ‹$S_1 = (\Gamma,\ U,\ \mathit{Some}\ (\mathit{add\text{-}mset}\ L\ C,\ \gamma_C))$› *elim*: *factorize.cases*)

from *invar* **have**
ground-conf: *is-ground-cls* $(\mathit{add\text{-}mset}\ L\ C \cdot \gamma_C)$
unfolding *resolveI*(*1*,*2*)
by (*simp-all add*: *ground-closures-def*)

have *add-mset* $L\ C = \mathit{add\text{-}mset}\ LL'\ (\mathit{add\text{-}mset}\ LL\ CC)$
using *resolveI*(*1*) S_1-*def* **by** *simp*

from *imgu*-μ_L **have** $\mu_L \odot \gamma_C = \gamma_C$
using ‹$LL \cdot l\ \gamma_C = LL' \cdot l\ \gamma_C$›
by (*auto simp*: *is-imgu-def is-unifiers-def is-unifier-alt intro*!: *subst-atm-of-eqI*)

have $L \cdot l\ \mu_L \in\#$ *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 ‹$LL \cdot l\ \gamma_C = LL' \cdot l\ \gamma_C$›
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 ‹*add-mset* $L\ C =$ *add-mset* LL' (*add-mset* $LL\ CC$)›
by (*metis insert-iff set-mset-add-mset-insert*)
thus *?thesis*

by *auto*
qed
then obtain *CCC* **where** *add-mset LL CC* $\cdot$ μ_L = *add-mset L CCC* $\cdot$ μ_L
by (*smt* (*verit*, *best*) *Melem-subst-cls mset-add subst-cls-add-mset*)

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

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

have *disjoint-vars*: $\bigwedge C$. *vars-cls* (*C* $\cdot$ $\varrho\varrho$) $\cap$ *vars-cls* (*add-mset K D*) = {}
by (*simp add*: $\varrho\varrho$*-def vars-cls-subst-renaming-disj*)

have *K* $\cdot l$ γ_D = $-$ (*L* $\cdot l$ μ_L $\cdot l$ γ_C)
proof $-$
have *L* $\cdot l$ μ_L $\cdot l$ γ_C = *L* $\cdot l$ γ_C
using ‹$\mu_L \odot \gamma_C = \gamma_C$›
by (*metis subst-lit-comp-subst*)
thus *?thesis*
using *resolveI* **by** *simp*
qed

have $\exists \mu$. *Unification.mgu* (*atm-of L* $\cdot a$ μ_L $\cdot a$ $\varrho\varrho$) (*atm-of K*) = *Some* μ
proof (*rule ex-mgu-if-subst-eq-subst-and-disj-vars*)
show *vars-term* (*atm-of L* $\cdot a$ μ_L $\cdot a$ $\varrho\varrho$) $\cap$ *vars-lit K* = {}
using *disjoint-vars*[*of* {#*L* $\cdot l$ μ_L#}] **by** *auto*
next
have *vars-term* (*atm-of L* $\cdot a$ μ_L) $\subseteq$ *subst-domain* γ_C
by (*metis* ‹$\mu_L \odot \gamma_C = \gamma_C$› *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-of L* $\cdot a$ μ_L $\cdot a$ $\varrho\varrho$ $\cdot a$ *rename-subst-domain* $\varrho\varrho$ γ_C = *atm-of L* $\cdot a$ μ_L $\cdot a$ γ_C
using *ren-*$\varrho\varrho$
by (*simp add*: *is-renaming-iff renaming-cancels-rename-subst-domain*)
thus *atm-of L* $\cdot a$ μ_L $\cdot a$ $\varrho\varrho$ $\cdot a$ *rename-subst-domain* $\varrho\varrho$ γ_C = *atm-of K* $\cdot a$ γ_D
using ‹*K* $\cdot l$ γ_D = $-$ (*L* $\cdot l$ μ_L $\cdot l$ γ_C)›
by (*metis atm-of-subst-lit atm-of-uminus*)
qed
then obtain $\mu\mu$ **where** *imgu-*$\mu\mu$: *is-imgu* $\mu\mu$ {{*atm-of L* $\cdot a$ μ_L $\cdot a$ $\varrho\varrho$, *atm-of K*}}
using *is-imgu-if-mgu-eq-Some* **by** *auto*

show *?thesis*
unfolding S_3*-def* ‹*add-mset LL CC* $\cdot$ μ_L = *add-mset L CCC* $\cdot$ μ_L›
using *resolve.resolveI*[*OF* ‹Γ = *trail-propagate* Γ' *K D* γ_D› ‹*K* $\cdot l$ γ_D = $-$ (*L* $\cdot l$ μ_L $\cdot l$ γ_C)› *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*

$CCC \cdot \mu_L$]
  by *auto*
qed

The following lemma corresponds to Lemma 7 in the paper.

lemma *no-backtrack-after-conflict-if*:
 assumes *conf*: *conflict N β S1 S2* **and** *trail-S2*: *state-trail S1 = trail-propagate* Γ *L C γ*
 shows ∄*S4*. *backtrack N β S2 S4*
proof −
 from *trail-S2 conf* **have** *state-trail S2 = trail-propagate* Γ *L C γ*
  unfolding *conflict.simps* **by** *auto*
 then show *?thesis*
  unfolding *backtrack.simps propagate-lit-def decide-lit-def*
  by *auto*
qed

lemma *skip-state-trail*: *skip N β S S′* ⟹ *suffix* (*state-trail S′*) (*state-trail S*)
 by (*auto simp*: *suffix-def elim*: *skip.cases*)

lemma *factorize-state-trail*: *factorize N β S S′* ⟹ *state-trail S′ = state-trail S*
 by (*auto elim*: *factorize.cases*)

lemma *resolve-state-trail*: *resolve N β S S′* ⟹ *state-trail S′ = state-trail S*
 by (*auto elim*: *resolve.cases*)

lemma *mempty-not-in-initial-clauses-if-run-leads-to-trail*:
 assumes
  reg-run: (*regular-scl N β*)** *initial-state S1* **and**
  trail-lit: *state-trail S1 = Lc # Γ*
 shows {#} |∉| *N*
proof (*rule notI*)
 assume {#} |∈| *N*
 then obtain *γ* **where** *conflict N β initial-state* ([], {||}, *Some* ({#}, *γ*))
  using *conflict-initial-state-if-mempty-in-intial-clauses* **by** *auto*
 moreover hence ∄*S′*. *local.scl N β* ([], {||}, *Some* ({#}, *γ*)) *S′* **for** *γ*
  using *no-more-step-if-conflict-mempty* **by** *simp*
 ultimately show *False*
  using *trail-lit*
  by (*metis* (*no-types*, *opaque-lifting*) *conflict-initial-state-only-with-mempty converse-rtranclpE*
    list.discI prod.sel(*1*) *reasonable-if-regular reg-run regular-scl-def scl-if-reasonable state-trail-def*)
qed

lemma *conflict-with-literal-gets-resolved*:
 assumes

```
    trail-lit: state-trail S1 = Lc # Γ and
    conf: conflict N β S1 S2 and
    resolution: (skip N β ⊔ factorize N β ⊔ resolve N β)** S2 Sn and
    backtrack: ∃ Sn'. backtrack N β Sn Sn' and
    mempty-not-in-init-clss: {#} |∉| N and
   invars: learned-nonempty S1 trail-propagated-or-decided' N β S1 no-conflict-after-decide'
N β S1
  shows ¬ is-decision-lit Lc ∧ strict-suffix (state-trail Sn) (state-trail S1)
proof −
  obtain S0 where propa: propagate N β S0 S1
   using mempty-not-in-init-clss before-reasonable-conflict[OF conf ‹learned-nonempty
S1›
       ‹trail-propagated-or-decided' N β S1› ‹no-conflict-after-decide' N β S1›] by
metis

  from trail-lit propa have ¬ is-decision-lit Lc
    by (auto simp: propagate-lit-def is-decision-lit-def elim!: propagate.cases)

  show ?thesis
  proof (rule conjI)
    show ¬ is-decision-lit Lc
      by (rule ‹¬ is-decision-lit Lc›)
  next
    show strict-suffix (state-trail Sn) (state-trail S1)
      unfolding strict-suffix-def
    proof (rule conjI)
      from conf have state-trail S2 = 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 ‹¬ is-decision-lit Lc› show state-trail Sn ≠ state-trail S1
        unfolding trail-lit
        by (auto simp: decide-lit-def is-decision-lit-def elim!: backtrack.cases)
    qed
  qed
```

qed

13 Clause Redundancy

definition *ground-redundant* **where**
ground-redundant lt N C $\longleftrightarrow$ $\{D \in N.\ lt\ D\ C\} \Vdash e\ \{C\}$

definition *redundant* **where**
redundant lt N C $\longleftrightarrow$
$(\forall\, C' \in$ *grounding-of-cls C*. *ground-redundant lt* (*grounding-of-clss N*) $C')$

lemma *redundant lt N C* $\longleftrightarrow$ $(\forall\, C' \in$ *grounding-of-cls C*. $\{D' \in$ *grounding-of-clss N*. *lt D' C'*$\} \Vdash e\ \{C'\})$
by (*simp add*: *redundant-def ground-redundant-def*)

lemma *ground-redundant-iff*:
ground-redundant lt N C $\longleftrightarrow$ $(\exists\, M \subseteq N.\ M \Vdash e\ \{C\} \wedge (\forall\, D \in M.\ lt\ D\ C))$
proof (*rule iffI*)
assume *red*: *ground-redundant lt N C*
show $\exists\, M \subseteq N.\ M \Vdash e\ \{C\} \wedge (\forall\, D \in M.\ lt\ D\ C)$
proof (*intro exI conjI*)
show $\{D \in N.\ lt\ D\ C\} \subseteq N$
by *simp*
next
show $\{D \in N.\ lt\ D\ C\} \Vdash e\ \{C\}$
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 $\exists\, M \subseteq N.\ M \Vdash e\ \{C\} \wedge (\forall\, D \in M.\ lt\ D\ C)$
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*$_{\mathcal{G}}$ $\equiv \lambda N.\ \{C.$ *ground-redundant lt N C*$\}$
shows *Red-F*$_{\mathcal{G}}$ $N = \{C.\ \exists\, M \subseteq N.\ M \Vdash e\ \{C\} \wedge (\forall\, D \in M.\ lt\ D\ C)\}$
by (*auto simp*: *Red-F*$_{\mathcal{G}}$*-def ground-redundant-iff*)

lemma *redundant-is-standard-redundancy*:
fixes *lt* $\mathcal{G}_F$ $\mathcal{G}_{Fs}$ *Red-F*$_{\mathcal{G}}$ *Red-F*
defines
$\mathcal{G}_F \equiv$ *grounding-of-cls* **and**
$\mathcal{G}_{Fs} \equiv$ *grounding-of-clss* **and**
Red-F$_{\mathcal{G}} \equiv \lambda N.\ \{C.$ *ground-redundant lt N C*$\}$ **and**

Red-F $\equiv \lambda N.\ \{C.\ \textit{redundant lt}\ N\ C\}$
shows *Red-F* $N = \{C.\ \forall D \in \mathcal{G}_F\ C.\ D \in \textit{Red-F}_{\mathcal{G}}\ (\mathcal{G}_{Fs}\ N)\}$
using *Red-F-def Red-F*$_{\mathcal{G}}$*-def* $\mathcal{G}_{Fs}$*-def* $\mathcal{G}_F$*-def redundant-def* **by** *auto*

lemma *ground-redundant-if-strict-subset*:
assumes $D \in N$ **and** $D \subset\# C$
shows *ground-redundant* (*multp*$_{HO}$ *R*) *N C*
using *assms*
unfolding *ground-redundant-def*
by (*metis* (*mono-tags*, *lifting*) *CollectI strict-subset-implies-multp*$_{HO}$ *subset-mset.less-le true-clss-def true-clss-singleton true-clss-subclause*)

lemma *redundant-if-strict-subset*:
assumes $D \in N$ **and** $D \subset\# C$
shows *redundant* (*multp*$_{HO}$ *R*) *N C*
unfolding *redundant-def*
proof (*rule ballI*)
fix C' **assume** $C' \in$ *grounding-of-cls C*
then obtain γ **where** $C' = C \cdot \gamma$ **and** *is-ground-subst* γ
by (*auto simp*: *grounding-of-cls-def*)

show *ground-redundant* (*multp*$_{HO}$ *R*) (*grounding-of-clss N*) C'
proof (*rule ground-redundant-if-strict-subset*)
from ‹$D \in N$› **show** $D \cdot \gamma \in$ *grounding-of-clss N*
using ‹*is-ground-subst* γ›
by (*auto simp*: *grounding-of-clss-def grounding-of-cls-def*)
next
from ‹$D \subset\# C$› **show** $D \cdot \gamma \subset\# C'$
by (*simp add*: ‹$C' = C \cdot \gamma$› *subst-subset-mono*)
qed
qed

lemma *redundant-if-strict-subsumes*:
assumes $D \cdot \sigma \subset\# C$ **and** $D \in N$
shows *redundant* (*multp*$_{HO}$ *R*) *N C*
unfolding *redundant-def*
proof (*rule ballI*)
fix C' **assume** $C' \in$ *grounding-of-cls C*
then obtain γ **where** $C' = C \cdot \gamma$ **and** *is-ground-subst* γ
by (*auto simp*: *grounding-of-cls-def*)

show *ground-redundant* (*multp*$_{HO}$ *R*) (*grounding-of-clss N*) C'
proof (*rule ground-redundant-if-strict-subset*)
from ‹$D \in N$› **show** $D \cdot \sigma \cdot \gamma \in$ *grounding-of-clss N*
using ‹*is-ground-subst* γ›
by (*metis* (*no-types*, *lifting*) *UN-iff ground-subst-ground-cls grounding-of-cls-ground grounding-of-clss-def insert-subset subst-cls-comp-subst subst-cls-eq-grounding-of-cls-subset-eq*)
next

from ‹$D \cdot \sigma \subset\# C$› **show** $D \cdot \sigma \cdot \gamma \subset\# C'$
by (*simp add*: ‹$C' = C \cdot \gamma$› *subst-subset-mono*)
qed
qed

lemma *ground-redundant-mono-strong*:
ground-redundant R N C $\Longrightarrow$ ($\bigwedge x.\ x \in N \Longrightarrow R\ x\ C \Longrightarrow S\ x\ C$) $\Longrightarrow$ *ground-redundant S N C*
unfolding *ground-redundant-def*
by (*simp add*: *true-clss-def*)

lemma *redundant-mono-strong*:
redundant R N C $\Longrightarrow$
($\bigwedge x\ y.\ x \in$ *grounding-of-clss N* $\Longrightarrow\ y \in$ *grounding-of-cls C* $\Longrightarrow R\ x\ y \Longrightarrow S\ x\ y$) $\Longrightarrow$
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* $\Longrightarrow$ *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* $\Longrightarrow$ *redundant* (*multp* (*trail-less-ex lt Ls*)) *N C*
by (*auto intro*: *subset-implies-multp elim*!: *redundant-mono-strong*)

lemma *not-bex-subset-mset-if-not-ground-redundant*:
assumes *is-ground-cls C* **and** *is-ground-clss N*
shows $\neg$ *ground-redundant* ($\subset\#$) *N C* $\Longrightarrow \neg\ (\exists D \in N.\ D \subset\# C)$
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* ≡ (*map fst* Γ)
shows *trail-consistent* Γ $\Longrightarrow$
$\forall\, i < length\ Ls.\ \forall\, j < length\ Ls.\ i \neq j \longrightarrow Ls\ !\ i \neq Ls\ !\ j \wedge Ls\ !\ i \neq -\ (Ls\ !\ j)$

```
  unfolding Ls-def
proof (induction Γ rule: trail-consistent.induct)
  case Nil
  show ?case by simp
next
  case (Cons Γ L u)
  from Cons.hyps have L-distinct:
    ∀ x < length (map fst Γ). map fst Γ ! x ≠ L
    ∀ x < length (map fst Γ). map fst Γ ! x ≠ - 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 < length (L # map fst Γ) and j-lt: j < length (L # map fst Γ)
and i ≠ j
    show
      (L # map fst Γ) ! i ≠ (L # map fst Γ) ! j ∧
       (L # map fst Γ) ! i ≠ - (L # map fst Γ) ! j
    proof (cases i)
      case 0
      thus ?thesis
        using L-distinct ‹i ≠ j› ‹j < length (L # map fst Γ)›
        using gr0-conv-Suc by auto
    next
      case (Suc k)
      then show ?thesis
        apply simp
        using i-lt j-lt ‹i ≠ j› 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:
  trail-consistent Γ ⟹ irreflp (trail-less (map fst Γ))
  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:
  trail-consistent Γ ⟹ transp (trail-less (map fst Γ))
  using transp-trail-less[OF
```

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

lemma *asymp-trail-less-if-trail-consistant*:
trail-consistent Γ ⟹ *asymp* (*trail-less* (*map fst* Γ))
using *asymp-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* ∈ *set* (*map* (*atm-of o fst*) Γ) **and** *atm-of K* ∈ *set* (*map* (*atm-of o fst*) Γ)
by (*auto simp*: *trail-term-less-def*)
hence *atm-of L* ∈ *atm-of* ‘ *fst* ‘ *set* Γ **and** *atm-of K* ∈ *atm-of* ‘ *fst* ‘ *set* Γ
by *auto*
hence *L* ∈ *fst* ‘ *set* Γ ∨ − *L* ∈ *fst* ‘ *set* Γ **and** *K* ∈ *fst* ‘ *set* Γ ∨ − *K* ∈ *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: ⋀*R L1 L2*. *lit-less R L1 L2* ⟹ $R^{==}$ (*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* ≠ {#} **and**
∗: ∀ *k*∈#*K*. ∃ *x*∈#*J*. *lit-less* (*trail-term-less* (*map* (*atm-of o fst*) Γ)) *k x*
by *auto*

show *?thesis*
unfolding *trail-defined-cls-def*
proof (*rule ballI*)
fix *L* **assume** *L-in*: *L* ∈# *C*

show *trail-defined-lit* Γ *L*
proof (*cases L* ∈# *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* ∈# *K* **by** *fastforce*
then obtain *L′* **where** *L′-in*: *L′*∈#*J* **and** *lit-less* (*trail-term-less* (*map* (*atm-of* o *fst*) Γ)) *L L′*
using ∗ **by** *blast*
hence (*trail-term-less* (*map* (*atm-of o fst*) Γ))== (*atm-of L*) (*atm-of L′*)
using *lit-less-preserves-term-order* **by** *metis*
thus *?thesis*
using *trail-defined-lit-if-trail-term-less*(*1*)
by (*metis* (*mono-tags*, *lifting*) *D-def L′-in atm-of-eq-atm-of sup2E tr-def-D trail-defined-cls-def trail-defined-lit-iff-defined-uminus union-iff*)
qed
qed
qed

15 Dynamic Non-Redundancy

lemma *regular-run-if-skip-factorize-resolve-run*:
assumes (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S S′*
shows (*regular-scl N β*)** *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 β S′ S″*
unfolding *scl-def* **by** *blast*
with *step.hyps*(*2*) **have** *reasonable-scl N β 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** ¬ *Ex* (*conflict N β 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 β 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 $\Gamma \Longrightarrow \neg$ (*trail-true-lit* Γ $L \wedge$ *trail-false-lit* Γ 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 $\Gamma \Longrightarrow \neg$ (*trail-true-cls* Γ $C \wedge$ *trail-false-cls* Γ C)
using *not-trail-true-and-false-lit*
by (*metis trail-false-cls-def trail-true-cls-def*)

fun *standard-lit-less* **where**
standard-lit-less R (*Pos t1*) (*Pos t2*) = R *t1 t2* |
standard-lit-less R (*Pos t1*) (*Neg t2*) = $R^{==}$ *t1 t2* |
standard-lit-less R (*Neg t1*) (*Pos t2*) = R *t1 t2* |
standard-lit-less R (*Neg t1*) (*Neg t2*) = R *t1 t2*

lemma *standard-lit-less-preserves-term-less*:
shows *standard-lit-less R L1 L2* $\Longrightarrow$ $R^{==}$ (*atm-of L1*) (*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* β *S0* **and**
invars: *learned-nonempty S0 trail-propagated-or-decided' N* β *S0*
no-conflict-after-decide' N β *S0 almost-no-conflict-with-trail N* β *S0*
trail-lits-consistent S0 trail-closures-false' S0 ground-false-closures S0 **and**
conflict: *conflict N* β *S0 S1* **and**
resolution: (*skip N* $\beta \sqcup$ *factorize N* $\beta \sqcup$ *resolve N* β)$^{++}$ *S1 Sn* **and**
backtrack: *backtrack N* β *Sn Sn'* **and**
lit-less-preserves-term-order: $\bigwedge R$ *L1 L2*. *lit-less R L1 L2* $\Longrightarrow$ $R^{==}$ (*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 ($\exists$ C γ. *state-conflict Sn* = *Some* (C, γ) $\wedge$
$C \cdot \gamma \notin$ *grounding-of-clss* (*fset N* $\cup$ *fset U*) $\wedge$
set-mset ($C \cdot \gamma$) $\notin$ *set-mset* ‘ *grounding-of-clss* (*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 −
from *conflict* **have** *almost-no-conflict-with-trail N* β *S1*
using ‹*almost-no-conflict-with-trail N* β *S0*›
by (*rule conflict-preserves-almost-no-conflict-with-trail*)

from *conflict* **obtain** *C1* γ*1* **where** *conflict-S1*: *state-conflict S1* = *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.prems 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.prems
      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 ⊨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.prems step.hyps(2) ‹suffix (state-trail Sm') (state-trail Sm)›
        ‹suffix (state-trail Sm') (state-trail S1)›
```

```
    unfolding suffix-def
    by auto

  from step.hyps(2) show ?case
  proof (elim sup2E)
    assume skip N β Sm Sm'
    thus ?thesis
      using ‹suffix (state-trail Sm') (state-trail S1)›
      using conflict-Sm skip.simps trail-false-Cm-γm by auto
  next
    assume factorize N β Sm Sm'
    thus ?thesis
    proof (cases N β Sm Sm' rule: factorize.cases)
      case (factorizeI L γ L' μ Γ U D)
       with conflict-Sm have Cm-def: Cm = add-mset L' (add-mset L D) and
γm-def: γm = γ
        by simp-all
      have trail-false-cls (state-trail S1) ((D + {#L#}) · μ · γ)
      proof (rule trail-false-cls-subst-mgu-before-grounding[of - - L L'])
        show trail-false-cls (state-trail S1) ((D + {#L, L'#}) · γ)
       by (metis Cm-def γm-def empty-neutral(1) trail-false-Cm-γm union-commute
            union-mset-add-mset-right)
      next
        show is-imgu μ {{atm-of L, atm-of L'}}
          using factorizeI(4) by fastforce
      next
        have is-unifier γ {atm-of L, atm-of L'}
          unfolding is-unifier-alt[of {atm-of L, atm-of L'}, simplified]
          by (metis atm-of-subst-lit factorizeI(3))
        thus is-unifiers γ {{atm-of L, atm-of L'}}
          by (simp add: is-unifiers-def)
      qed
      with factorizeI(2) show ?thesis
        using ‹suffix (state-trail Sm') (state-trail S1)›
        by (metis add-mset-add-single state-conflict-simp)
    qed
  next
    assume resolve N β Sm Sm'
    thus ?thesis
    proof (cases N β Sm Sm' rule: resolve.cases)
      case (resolveI Γ Γ' K D γD L γC ϱC ϱD C μ γ U)
       with conflict-Sm have Cm-def: Cm = add-mset L C and γm-def: γm =
γC
        by simp-all
      hence tr-false-S1-conf: trail-false-cls (state-trail S1) (add-mset L C · γC)
        using trail-false-Cm-γm by simp

      from sound-Sm have sound-trail N Γ
        unfolding resolveI(1,2) sound-state-def
```

by *simp*

from ‹*ground-false-closures Sm*› **have**
ground-conf: *is-ground-cls* (*add-mset L C* ⋅ γ_C) **and**
ground-prop: *is-ground-cls* (*add-mset K D* ⋅ γ_D)
unfolding *resolveI*(*1*,*2*) ‹Γ = *trail-propagate* Γ' *K D* γ_D›
by (*simp-all add*: *ground-false-closures-def ground-closures-def propagate-lit-def*)
hence
$\forall$ *L*∈#*add-mset L C*. *L* ⋅*l* ϱ_C ⋅*l* γ = *L* ⋅*l* γ_C
$\forall$ *K*∈#*add-mset K D*. *K* ⋅*l* ϱ_D ⋅*l* γ = *K* ⋅*l* γ_D
using *resolveI merge-of-renamed-groundings* **by** *metis*+

have *atm-of L* ⋅*a* ϱ_C ⋅*a* γ = *atm-of K* ⋅*a* ϱ_D ⋅*a* γ
using ‹*K* ⋅*l* γ_D = − (*L* ⋅*l* γ_C)›
‹$\forall$ *L*∈#*add-mset L C*. *L* ⋅*l* ϱ_C ⋅*l* γ = *L* ⋅*l* γ_C›[*rule-format*, *of L*, *simplified*]
‹$\forall$ *K*∈#*add-mset K D*. *K* ⋅*l* ϱ_D ⋅*l* γ = *K* ⋅*l* γ_D›[*rule-format*, *of K*, *simplified*]
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
hence *is-unifiers* γ {{*atm-of L* ⋅*a* ϱ_C, *atm-of K* ⋅*a* ϱ_D}}
by (*simp add*: *is-unifiers-def is-unifier-alt*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu* μ {{*atm-of L* ⋅*a* ϱ_C, *atm-of K* ⋅*a* ϱ_D}}›
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 ‹$\forall$ *L*∈#*add-mset L C*. *L* ⋅*l* ϱ_C ⋅*l* γ = *L* ⋅*l* γ_C› ‹$\forall$ *K*∈#*add-mset K D*. *K* ⋅*l* ϱ_D ⋅*l* γ = *K* ⋅*l* γ_D›
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* Γ' (*state-trail S1*)
using *resolveI* ‹*suffix* (*state-trail Sm'*) (*state-trail S1*)›
by (*metis* (*no-types*, *opaque-lifting*) *state-trail-simp suffix-order.trans suffix-trail-propagate*)
next
from ‹*trail-closures-false' Sm*› **have** *trail-closures-false* Γ
unfolding *resolveI*(*1*,*2*)
by (*simp add*: *trail-closures-false'-def*)
thus *trail-false-cls* Γ' ($D \cdot \gamma_D$)
using *resolveI*(*3*−)
by (*auto simp*: *propagate-lit-def elim*: *trail-closures-false.cases*)
qed

ultimately have *trail-false-cls* (*state-trail S1*) (($C \cdot \varrho_C + D \cdot \varrho_D$) ⋅ μ ⋅ γ)
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* ∈ *grounding-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 conflict-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 = state-learned S0*
using ‹*propagate N β S S0*› **by** (*auto simp add*: *propagate.simps*)
also from *conflict* **have** *... = state-learned S1*
by (*auto simp add*: *conflict.simps*)
finally have *state-learned S = state-learned S1*
by *assumption*

have *state-conflict S = None*
using ‹*propagate N β S S0*› **by** (*auto simp add*: *propagate.simps*)

from *conflict* **have** *state-trail S1 = state-trail S0*
by (*smt* (*verit*) *conflict.cases state-trail-simp*)

obtain *L C γ* **where** *trail-S0-eq*: *state-trail S0 = trail-propagate* (*state-trail S*) *L C γ*
using ‹*propagate N β S S0*› **unfolding** *propagate.simps* **by** *auto*

with *backtrack* **have** *strict-suffix* (*state-trail Sn*) (*state-trail S0*)
using *conflict-with-literal-gets-resolved*[*OF - conflict resolution*[*THEN tranclp-into-rtranclp*] -
‹{#} |∉| *N*› ‹*learned-nonempty S0*› ‹*trail-propagated-or-decided′ N β S0*›
‹*no-conflict-after-decide′ N β S0*›]
by (*metis* (*no-types*, *lifting*) *propagate-lit-def*)
hence *suffix* (*state-trail Sn*) (*state-trail S*)
unfolding *trail-S0-eq propagate-lit-def*
by (*metis suffix-Cons suffix-order.le-less suffix-order.less-irrefl*)

moreover have ¬ *trail-defined-lit* (*state-trail S*) (*L ·l γ*)
proof −
have *trail-consistent* (*state-trail S0*)
using ‹*state-trail S1 = state-trail S0*› ‹*trail-consistent* (*state-trail S1*)› **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 ¬ *trail-defined-lit* (*state-trail Sn*) (*L ·l γ*)
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* (*state-trail Sn*) (*Cn · γn*)
using ‹*ground-false-closures Sn*› *conflict-Sn* **by** (*auto simp add*: *ground-false-closures-def*)

ultimately have *L ·l γ ∉# Cn · γn ∧ − (L ·l γ) ∉# Cn · γn*
unfolding *trail-false-cls-def trail-false-lit-def trail-defined-lit-def*
by (*metis uminus-of-uminus-id*)

have *no-conf-at-S*: ∄*S′. conflict N β S S′*

```
  proof (rule nex-conflict-if-no-conflict-with-trail'')
    show state-conflict S = None
      using ‹propagate N β S S0› by (auto elim: propagate.cases)
  next
    show {#} |∉| N
      by (rule ‹{#} |∉| 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 β (state-learned S) (state-trail S)
      using ‹almost-no-conflict-with-trail N β S0›
      using ‹propagate N β 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: ∃S'. conflict N β S S'
    if D-in: D ∈ grounding-of-clss (fset N ∪ 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-γn:
    ¬ ground-redundant trail-ord (grounding-of-clss (fset N ∪ fset U)) (Cn · γn)
  proof (rule notI)
    define gnds-le-Cn-γn where
      gnds-le-Cn-γn = {D ∈ grounding-of-clss (fset N ∪ fset U). trail-ord D (Cn ·
γn)}

    assume ground-redundant trail-ord (grounding-of-clss (fset N ∪ fset U)) (Cn ·
γn)
    hence gnds-le-Cn-γn ‖=e {Cn · γn}
      unfolding ground-redundant-def gnds-le-Cn-γn-def by simp
    hence ¬ trail-interp (state-trail S1) ‖=s gnds-le-Cn-γn
      using not-trail-S1-entails-Cn-γn by auto
    then obtain D where D-in: D ∈ gnds-le-Cn-γn and ¬ trail-interp (state-trail
S1) ‖= D
      by (auto simp: true-clss-def)

    from D-in have
      D-in: D ∈ grounding-of-clss (fset N ∪ fset U) and
      D-le-Cn-γn: trail-ord D (Cn · γn)
      unfolding gnds-le-Cn-γn-def by simp-all
```

from *D-le-Cn-γn* **have** *trail-defined-cls* (*state-trail S1*) *D*
using ‹*trail-defined-cls* (*state-trail S1*) (*Cn* · *γn*)›
using *trail-defined-cls-if-lt-defined*
using Γ*-def lit-less-preserves-term-order tr-consistent-S1 trail-ord-def*
by *metis*
hence *trail-false-cls* (*state-trail S1*) *D*
using ‹¬ *trail-interp* (*state-trail S1*) ⊨ *D*›
using ‹*trail-consistent* (*state-trail S1*)› *trail-interp-cls-if-trail-true*
trail-true-or-false-cls-if-defined **by** *blast*

have *L* ·*l* *γ* ∉# *D* ∧ − (*L* ·*l* *γ*) ∉# *D*
proof −
from *D-le-Cn-γn* **have** *D-lt-Cn-γn′*:
multp$_{HO}$ (*lit-less* (*trail-term-less* (*map* (*atm-of* ∘ *fst*) (*state-trail S1*)))) *D* (*Cn* · *γn*)
unfolding *trail-ord-def* Γ*-def* **.**

have ∀ *K*∈#*Cn* · *γn*. − *K* ∈ *fst* ‘ *set* (*state-trail S1*)
by (*rule* ‹*trail-false-cls* (*state-trail S1*) (*Cn* · *γn*)›[*unfolded trail-false-cls-def trail-false-lit-def*])
hence ∀ *K*∈#*Cn* · *γn*. − *K* ∈ *insert* (*L* ·*l* *γ*) (*fst* ‘ *set* (*state-trail S*))
unfolding ‹*state-trail S1* = *state-trail S0*›
‹*state-trail S0* = *trail-propagate* (*state-trail S*) *L C γ*›
propagate-lit-def list.set image-insert prod.sel
by *simp*
hence ∗: ∀ *K*∈#*Cn* · *γn*. − *K* ∈ *fst* ‘ *set* (*state-trail S*)
by (*metis* ‹*L* ·*l* *γ* ∉# *Cn* · *γn* ∧ − (*L* ·*l* *γ*) ∉# *Cn* · *γn*› *insert-iff uminus-lit-swap*)

have ∗∗: ∀ *K* ∈# *Cn* · *γn*. *trail-term-less* (*map* (*atm-of o fst*) (*state-trail S1*)) (*atm-of K*) (*atm-of* (*L* ·*l* *γ*))
unfolding ‹*state-trail S1* = *state-trail S0*›
‹*state-trail S0* = *trail-propagate* (*state-trail S*) *L C γ*›
propagate-lit-def comp-def prod.sel list.map
proof (*rule ballI*)
fix *K* **assume** *K* ∈# *Cn* · *γn*
with ∗ **have** − *K* ∈ *fst* ‘ *set* (*state-trail S*)
by *metis*
hence *atm-of K* ∈ *set* (*map* (λ*x*. *atm-of* (*fst x*)) (*state-trail S*))
by (*metis* (*no-types*, *lifting*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set comp-eq-dest-lhs image-comp image-cong list.set-map*)
thus *trail-term-less* (*atm-of* (*L* ·*l* *γ*) # *map* (λ*x*. *atm-of* (*fst x*)) (*state-trail S*)) (*atm-of K*) (*atm-of* (*L* ·*l* *γ*))
using *trail-term-less-Cons-if-mem* **by** *metis*
qed

have ¬ (*L* ·*l* *γ* ∈# *D* ∨ − (*L* ·*l* *γ*) ∈# *D*)

proof (*rule notI*)
obtain *I J K* **where**
Cn · *γn* = *I* + *J* **and** *D-def*: *D* = *I* + *K* **and** *J* ≠ {#} **and**
∀ *k*∈#*K*. ∃ *x*∈#*J*. *lit-less* (*trail-term-less* (*map* (*atm-of* ∘ *fst*) (*state-trail S1*))) *k x*
using *multp*$_{HO}$*-implies-one-step*[*OF D-lt-Cn-γn'*]
by *auto*
assume *L* ·*l γ* ∈# *D* ∨ − (*L* ·*l γ*) ∈# *D*
then show *False*
unfolding *D-def Multiset.union-iff*
proof (*elim disjE*)
show *L* ·*l γ* ∈# *I* ⟹ *False*
using ‹*L* ·*l γ* ∉# *Cn* · *γn* ∧ − (*L* ·*l γ*) ∉# *Cn* · *γn*› ‹*Cn* · *γn* = *I* + *J*› **by** *simp*
next
show − (*L* ·*l γ*) ∈# *I* ⟹ *False*
using ‹*L* ·*l γ* ∉# *Cn* · *γn* ∧ − (*L* ·*l γ*) ∉# *Cn* · *γn*› ‹*Cn* · *γn* = *I* + *J*› **by** *simp*
next
show *L* ·*l γ* ∈# *K* ⟹ *False*
using ‹*L* ·*l γ* ∉# *Cn* · *γn* ∧ − (*L* ·*l γ*) ∉# *Cn* · *γn*›[*THEN conjunct1*]
∗∗[*unfolded* ‹*Cn* · *γn* = *I* + *J*›] ‹∀ *k*∈#*K*. ∃ *x*∈#*J*. *lit-less* (*trail-term-less* (*map* (*atm-of* ∘ *fst*) (*state-trail S1*))) *k x*›
by (*metis* (*no-types*, *lifting*) *D-def Un-insert-right*
‹¬ *trail-interp* (*state-trail S1*) ⊨ *D*›
‹*state-trail S0* = *trail-propagate* (*state-trail S*) *L C γ*›
‹*state-trail S1* = *state-trail S0*› ‹*trail-consistent* (*state-trail S1*)› *image-insert*
insert-iff list.set(*2*) *mk-disjoint-insert prod.sel*(*1*) *set-mset-union*
trail-interp-cls-if-trail-true propagate-lit-def trail-true-cls-def
trail-true-lit-def)
next
assume − (*L* ·*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* ∘ *fst*) (*state-trail S1*))
using *distinct-atm-of-trail-if-trail-consistent* **by** *metis*

ultimately show *False*
using *asymp-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* ‹*trail-false-cls* (*state-trail S1*) *D*›
unfolding ‹*state-trail S1* = *state-trail S0*›
‹*state-trail S0* = *trail-propagate* (*state-trail S*) *L C* γ›
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 ¬ *redundant trail-ord* (*fset N* ∪ *fset U*) *Cn*
unfolding *redundant-def*
using *Cn-γn-in* **by** *auto*

moreover have *Cn* · γ*n* ∉ *grounding-of-clss* (*fset N* ∪ *fset U*)
proof −
have *is-ground-cls* (*Cn* · γ*n*)
using *Cn-γn-in is-ground-cls-if-in-grounding-of-cls* **by** *blast*

moreover have *is-ground-clss* (*grounding-of-clss* (*fset N* ∪ *fset U*))
by *simp*

ultimately have ¬ ({*D* ∈ *grounding-of-clss* (*fset N* ∪ *fset U*). *trail-ord D* (*Cn* · γ*n*)} ⟙=*e* {*Cn* · γ*n*})
using *not-gr-red-Cn-γn*
by (*simp add*: *ground-redundant-def*)
thus *?thesis*
using ‹*suffix* (*state-trail Sn*) (*state-trail S*)› *conf-at-S-if no-conf-at-S*
trail-Sn-false-Cn-γn trail-false-cls-if-trail-false-suffix **by** *blast*
qed

moreover have *set-mset* (*Cn* · γ*n*) ∉ *set-mset* ‘ *grounding-of-clss* (*fset N* ∪ *fset U*)
proof (*rule notI*)
assume *set-mset* (*Cn* · γ*n*) ∈ *set-mset* ‘ *grounding-of-clss* (*fset N* ∪ *fset U*)
then obtain *D* **where**
D-in: *D* ∈ *grounding-of-clss* (*fset N* ∪ *fset U*) **and**
set-mset-eq-D-Cn-γn: *set-mset D* = *set-mset* (*Cn* · γ*n*)

by (*auto simp add*: *image-iff*)

have ¬ *trail-interp* (*state-trail S1*) ⊨ *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 ‹∀ *L*∈#*Cn* · *γn*. *trail-defined-lit* (*state-trail S1*) *L*›
unfolding *set-mset-eq-D-Cn-γn*[*symmetric*]
by (*simp add*: *trail-defined-cls-def*)
hence *trail-false-cls* (*state-trail S1*) *D*
using ‹¬ *trail-interp* (*state-trail S1*) ⊨ *D*›
using *tr-consistent-S1 trail-interp-cls-if-trail-true trail-true-or-false-cls-if-defined*
by *blast*

have *L* ·*l* *γ* ∉# *D* ∧ − (*L* ·*l* *γ*) ∉# *D*
using ‹*L* ·*l* *γ* ∉# *Cn* · *γn* ∧ − (*L* ·*l* *γ*) ∉# *Cn* · *γn*›
unfolding *set-mset-eq-D-Cn-γn*[*symmetric*]
by *assumption*
hence *trail-false-cls* (*state-trail S*) *D*
using *D-in* ‹*trail-false-cls* (*state-trail S1*) *D*›
unfolding ‹*state-trail S1* = *state-trail S0*›
‹*state-trail S0* = *trail-propagate* (*state-trail S*) *L C γ*›
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 ¬(∃ *D* ∈ *fset N* ∪ *fset U*. ∃ *σ*. *D* · *σ* = *Cn*)
by (*metis* (*no-types*, *lifting*) *Cn-γn-in Set.set-insert UnCI calculation*(2)
grounding-of-clss-insert grounding-of-subst-cls-subset subsetD)

moreover hence *Cn* ∉ *fset N* ∪ *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 β* ⊔ *factorize N β* ⊔ *resolve N β*)^++ *S1 Sn* **and**
backtrack: *backtrack N β Sn Sn′* **and**
lit-less-preserves-term-order: ⋀*R L1 L2*. *lit-less R L1 L2* ⟹ *R*^== (*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* β)** *initial-state Sn'* $\wedge$
($\exists$ *C* γ. *state-conflict Sn* = *Some* (*C*, γ) $\wedge$
C $\cdot$ γ $\notin$ *grounding-of-clss* (*fset N* $\cup$ *fset U*) $\wedge$
set-mset (*C* $\cdot$ γ) $\notin$ *set-mset* ‘ *grounding-of-clss* (*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* $\cdot$ σ = *C*) $\wedge$
$\neg$ *redundant trail-ord* (*fset N* $\cup$ *fset U*) *C*)
proof −
have *sound-state N* β *initial-state*
by (*rule sound-initial-state*)
with *regular-run* **have** *sound-S0*: *sound-state N* β *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* β *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* β *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* β *S0*
by (*induction S0 rule*: *rtranclp-induct*)
(*simp-all add*: *regular-scl-preserves-almost-no-conflict-with-trail*)

from *regular-run* **have** *trail-lits-consistent S0*
by (*induction S0 rule*: *rtranclp-induct*)
(*auto intro*: *scl-preserves-trail-lits-consistent reasonable-if-regular scl-if-reasonable*)

from *regular-run* **have** *trail-lits-consistent S0*
by (*induction S0 rule*: *rtranclp-induct*)
(*auto intro*: *scl-preserves-trail-lits-consistent reasonable-if-regular scl-if-reasonable*)

from *regular-run* **have** *trail-closures-false' S0*
by (*induction S0 rule*: *rtranclp-induct*)
(*auto intro*: *scl-preserves-trail-closures-false' reasonable-if-regular scl-if-reasonable*)

from *regular-run* **have** *ground-false-closures S0*
by (*induction S0 rule*: *rtranclp-induct*)
(*auto intro*: *scl-preserves-ground-false-closures reasonable-if-regular scl-if-reasonable*)

from *regular-run conflict* **have** (*regular-scl N β*)** *initial-state S1*
by (*meson regular-scl-def rtranclp.simps*)
also from *resolution* **have** *reg-run-S1-Sn*: (*regular-scl N β*)** ... *Sn*
using *regular-run-if-skip-factorize-resolve-run tranclp-into-rtranclp* **by** *metis*
also have (*regular-scl N β*)** ... *Sn'*
proof (*rule r-into-rtranclp*)
from *backtrack* **have** *scl N β Sn Sn'*
by (*simp add*: *scl-def*)
with *backtrack* **have** *reasonable-scl N β Sn Sn'*
using *reasonable-scl-def decide-well-defined*(*6*) **by** *blast*
with *backtrack* **show** *regular-scl N β Sn Sn'*
unfolding *regular-scl-def*
by (*smt* (*verit*) *conflict.simps option.simps*(*3*) *backtrack.cases state-conflict-simp*)
qed
finally have (*regular-scl N β*)** *initial-state Sn'* **by** *assumption*
thus *?thesis*
using *learned-clauses-in-regular-runs-invars*[*OF sound-S0* ‹*learned-nonempty S0*›
‹*trail-propagated-or-decided' N β S0*›
‹*no-conflict-after-decide' N β S0*› ‹*almost-no-conflict-with-trail N β S0*›
‹*trail-lits-consistent S0*› ‹*trail-closures-false' S0*› ‹*ground-false-closures S0*›
conflict resolution backtrack]
using *lit-less-preserves-term-order*
using *U-def* Γ*-def trail-ord-def* **by** *presburger*
qed

theorem *dynamic-non-redundancy-strategy*:
fixes Γ
assumes
run: (*strategy N β*)** *initial-state S0* **and**
conflict: *conflict N β S0 S1* **and**
resolution: (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)^++ *S1 Sn* **and**
backtrack: *backtrack N β Sn Sn'* **and**
strategy-imp-regular-scl: ⋀*S S'*. *strategy N β S S'* ⟹ *regular-scl N β S S'* **and**
lit-less-preserves-term-order: ⋀*R L1 L2*. *lit-less R L1 L2* ⟹ *R*^== (*atm-of L1*) (*atm-of L2*)
defines
Γ ≡ *state-trail S1* **and**
U ≡ *state-learned S1* **and**
trail-ord ≡ *multp*$_{HO}$ (*lit-less* (*trail-term-less* (*map* (*atm-of o fst*) Γ)))
shows (∃ *C γ*. *state-conflict Sn* = *Some* (*C*, *γ*) ∧
C · *γ* ∉ *grounding-of-clss* (*fset N* ∪ *fset U*) ∧
set-mset (*C* · *γ*) ∉ *set-mset* ‘ *grounding-of-clss* (*fset N* ∪ *fset U*) ∧
C ∉ *fset N* ∪ *fset U* ∧
¬ (∃ *D* ∈ *fset N* ∪ *fset U*. ∃*σ*. *D* · *σ* = *C*) ∧
¬ *redundant trail-ord* (*fset N* ∪ *fset U*) *C*)
proof −
from *backtrack* **have** *backtrack'*: *backtrack N β Sn Sn'*

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

have ($\exists$ *C* γ. *state-conflict Sn* = *Some* (*C*, γ)) $\wedge$
C $\cdot$ γ $\notin$ *grounding-of-clss* (*fset N* $\cup$ *fset U*) $\wedge$
set-mset (*C* $\cdot$ γ) $\notin$ *set-mset* ‘ *grounding-of-clss* (*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* $\cdot$ σ = *C*) $\wedge$
$\neg$ *redundant* (*multp*$_{HO}$ (*lit-less*
(*trail-term-less* (*map* (*atm-of* $\circ$ *fst*) (*state-trail S1*)))))
(*fset N* $\cup$ *fset U*) *C*)
unfolding *U-def*
proof (*rule dynamic-non-redundancy-regular-scl*[*THEN conjunct2*])
from *run* **show** (*regular-scl N* β)** *initial-state S0*
by (*induction S0 rule*: *rtranclp-induct*) (*auto dest*: *strategy-imp-regular-scl*)
next
from *assms* **show** *conflict N* β *S0 S1*
by *simp*
next
from *assms* **show** (*skip N* β $\sqcup$ *factorize N* β $\sqcup$ *resolve N* β)$^{++}$ *S1 Sn*
by *simp*
next
from *assms* **show** *backtrack N* β *Sn Sn*$'$
by (*simp add*: *shortest-backtrack-strategy-def*)
next
from *assms* **show** $\bigwedge$*R L1 L2*. *lit-less R L1 L2* $\Longrightarrow$ *R*$^{==}$ (*atm-of L1*) (*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* β)** *initial-state S* **and**
step: *backtrack N* β *S S*$'$
shows $\exists$ *S0 S1 S2 S3 S4*. (*regular-scl N* β)** *initial-state S0* $\wedge$
propagate N β *S0 S1* $\wedge$ *regular-scl N* β *S0 S1* $\wedge$
conflict N β *S1 S2* $\wedge$ (*factorize N* β)** *S2 S3* $\wedge$ *resolve N* β *S3 S4* $\wedge$
(*skip N* β $\sqcup$ *factorize N* β $\sqcup$ *resolve N* β)** *S4 S*
proof $-$
from *run* **have** *sound-state N* β *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* β *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 β 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 β)** initial-state S0 and
    propa: propagate N β S0 S1 regular-scl N β S0 S1 and
    confl: conflict N β S1 S2 and
    facto: (factorize N β)** S2 S3 and
    resol: resolve N β S3 S4 and
    reg-res: (skip N β ⊔ factorize N β ⊔ resolve N β)** 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 β)** initial-state S and
    step: backtrack N β S S'
  defines
    U ≡ state-learned S
  shows ∃ C γ. state-conflict S = Some (C, γ) ∧ ¬ (∃ D ∈ fset N ∪ fset U.
subsumes D C)
proof −
  from before-regular-backtrack'[OF run step] obtain S0 S1 S2 S3 S4 where
    run-S0: (regular-scl N β)** initial-state S0 and
    propa: propagate N β S0 S1 regular-scl N β S0 S1 and
    confl: conflict N β S1 S2 and
    facto: (factorize N β)** S2 S3 and
    resol: resolve N β S3 S4 and
    reg-res: (skip N β ⊔ factorize N β ⊔ resolve N β)** S4 S
    using before-regular-backtrack[OF step] by blast

  have (regular-scl N β)** initial-state S1
    using run-S0 propa(2) by simp

  moreover have reg-res': (skip N β ⊔ factorize N β ⊔ resolve N β)++ S2 S
  proof −
    have (skip N β ⊔ factorize N β ⊔ resolve N β)** S2 S3
      using facto
      by (rule mono-rtranclp[rule-format, rotated]) simp
```

```
  also have (skip N β ⊔ factorize N β ⊔ resolve N β)^++ S3 S4
    using resol by auto
  finally show ?thesis
    using reg-res by simp
qed

ultimately obtain C γ lt where
  reg-run: (regular-scl N β)** initial-state S' and
  conf: state-conflict S = Some (C, γ) and
  not-gen: ¬ (∃ D ∈ fset N ∪ fset (state-learned S2). ∃ σ. D · σ = C) and
  not-red: ¬ redundant (multp_HO (standard-lit-less
              (trail-term-less (map (atm-of ∘ fst) (state-trail S2)))))
      (fset N ∪ fset (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 ¬ (∃ D∈fset N ∪ fset (state-learned S2). ∃ σ. D · σ ⊂# C)
  using redundant-if-strict-subsumes
  by (metis union-fset)
with not-gen have ¬ (∃ D∈fset N ∪ fset (state-learned S2). ∃ σ. D · σ ⊆# C)
  using subset-mset.order-iff-strict by blast
hence not-sub: ¬ (∃ D∈fset N ∪ fset (state-learned S2). subsumes D C)
  by (simp add: subsumes-def)

from reg-res' have learned-S2: state-learned S2 = 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 = 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: (strategy N β)** initial-state S and
    step: backtrack N β S S' and
    strategy-imp-regular-scl: ⋀S S'. strategy N β S S' ⟹ regular-scl N β S S'
```

defines
$U \equiv$ *state-learned* S
shows $\exists\, C\ \gamma.$ *state-conflict* S = *Some* $(C, \gamma) \wedge \neg\ (\exists\, D \in$ *fset* $N \cup$ *fset* $U.$ *subsumes* $D\ C)$
unfolding *U-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

lemma *wf-onI*:
$(\bigwedge P\ x.\ (\bigwedge y.\ y \in A \Longrightarrow (\bigwedge z.\ z \in A \Longrightarrow (z, y) \in r \Longrightarrow P\ z) \Longrightarrow P\ y) \Longrightarrow x \in A \Longrightarrow P\ x) \Longrightarrow$ *wf-on* $A\ r$
unfolding *wf-on-def* **by** *metis*

lemma *wfI*: $(\bigwedge P\ x.\ (\bigwedge y.\ (\bigwedge z.\ (z, y) \in r \Longrightarrow P\ z) \Longrightarrow P\ y) \Longrightarrow P\ x) \Longrightarrow$ *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**
$\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow (y, x) \in r \Longrightarrow P\ y) \Longrightarrow P\ x$ **and**
$x \in A$
shows $P\ x$
using *assms* **unfolding** *wf-on-def* **by** *metis*

16.1 Basic Results

16.1.1 Minimal-element characterization of well-foundedness

lemma *minimal-if-wf-on*:
assumes *wf*: *wf-on* $A\ R$ **and** $B \subseteq A$ **and** $B \neq \{\}$
shows $\exists\, z \in B.\ \forall\, y.\ (y, z) \in R \longrightarrow y \notin B$
using *wf-onE-pf*[*OF wf* ‹$B \subseteq A$›]
by (*metis Image-iff assms*(*3*) *subsetI*)

lemma *wfE-min*:

assumes *wf*: *wf R* **and** *Q*: $x \in Q$
obtains *z* **where** $z \in Q$ $\bigwedge y.\ (y,\ z) \in R \Longrightarrow y \notin Q$
using *Q wfE-pf*[*OF wf*, *of Q*] **by** *blast*

lemma *wfE-min'*:
wf R $\Longrightarrow$ *Q* $\neq$ {} $\Longrightarrow$ ($\bigwedge z.\ z \in Q \Longrightarrow (\bigwedge y.\ (y,\ z) \in R \Longrightarrow y \notin Q) \Longrightarrow$ *thesis*) $\Longrightarrow$ *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 *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 *ex-trans-min-element-if-wf-on*:
assumes *wf*: *wf-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-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*])

Well-foundedness of the empty relation

definition *inv-imagep-on* :: *'a set* $\Rightarrow$ (*'b* $\Rightarrow$ *'b* $\Rightarrow$ *bool*) $\Rightarrow$ (*'a* $\Rightarrow$ *'b*) $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool* **where**
inv-imagep-on A R f = ($\lambda x\ y.\ x \in A \wedge y \in A \wedge R\ (f\ x)\ (f\ y)$)

lemma *wfp-on-inv-imagep*:
assumes *wf*: *wfp-on* (*f* ` *A*) *R*
shows *wfp-on A* (*inv-imagep R f*)
unfolding *wfp-on-iff-ex-minimal*
proof (*intro allI impI*)
fix *B* **assume** $B \subseteq A$ **and** $B \neq \{\}$
hence $\exists\, z{\in}f\ `\ B.\ \forall\, y.\ R\ y\ z \longrightarrow y \notin f\ `\ B$
using *wf*[*unfolded wfp-on-iff-ex-minimal*, *rule-format*, *of f* ` *B*] **by** *blast*

thus $\exists z{\in}B.\ \forall y.\ \mathit{inv\text{-}imagep}\ R\ f\ y\ z \longrightarrow y \notin B$
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: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ (f\ x)\ (f\ y))$
shows *wfp-on* $A\ R$
unfolding *wfp-on-iff-ex-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** $\bigwedge z.\ Q\ (f\ z)\ (f\ y) \Longrightarrow z \notin B$
unfolding *wfp-on-iff-ex-minimal in-inv-imagep* **by** *metis*
thus $\exists z \in B.\ \forall y.\ R\ y\ z \longrightarrow y \notin B$
using ‹$B \subseteq A$› *convertible* **by** *blast*
qed

definition *lex-prodp* **where**
lex-prodp $R_A\ R_B\ x\ y \longleftrightarrow R_A\ (\mathit{fst}\ x)\ (\mathit{fst}\ y) \vee \mathit{fst}\ x = \mathit{fst}\ y \wedge R_B\ (\mathit{snd}\ x)\ (\mathit{snd}\ y)$

lemma *lex-prodp-lex-prod-iff*[*pred-set-conv*]:
lex-prodp $R_A\ R_B\ x\ y \longleftrightarrow (x,\ y) \in \mathit{lex\text{-}prod}\ \{(x,\ y).\ R_A\ x\ y\}\ \{(x,\ y).\ R_B\ x\ y\}$
unfolding *lex-prodp-def lex-prod-def* **by** *auto*

lemma *lex-prod-lex-prodp-iff*:
lex-prod $\{(x,\ y).\ R_A\ x\ y\}\ \{(x,\ y).\ R_B\ x\ y\} = \{(x,\ y).\ \mathit{lex\text{-}prodp}\ R_A\ R_B\ x\ y\}$
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* $B\ r_B$ **and** *AB-subset*: $AB \subseteq A \times B$
shows *wf-on* $AB\ (r_A <\!\!*\mathit{lex}\!\!*\!\!> r_B)$
unfolding *wf-on-iff-ex-minimal*
proof (*intro allI impI*)
fix AB' **assume** $AB' \subseteq AB$ **and** $AB' \neq \{\}$
hence $\mathit{fst}\ `\ AB' \subseteq A$ **and** $\mathit{snd}\ `\ AB' \subseteq B$
using *AB-subset* **by** *auto*

from ‹$\mathit{fst}\ `\ AB' \subseteq A$› ‹$AB' \neq \{\}$› **obtain** a **where**
a-in: $a \in \mathit{fst}\ `\ AB'$ **and**
a-minimal: $(\forall y.\ (y,\ a) \in r_A \longrightarrow y \notin \mathit{fst}\ `\ AB')$
using *wfA*[*unfolded wf-on-iff-ex-minimal*, *rule-format*, *of fst* $`\ AB'$]
by *auto*

from ‹$\mathit{snd}\ `\ AB' \subseteq B$› ‹$AB' \neq \{\}$› *a-in* **obtain** b **where**
b-in: $b \in \mathit{snd}\ `\ \{p \in AB'.\ \mathit{fst}\ p = a\}$ **and**

b-minimal: $(\forall\, y.\ (y,\ b) \in r_B \longrightarrow y \notin snd\ `\ \{p \in AB'.\ fst\ p = a\})$
using *wfB*[*unfolded wf-on-iff-ex-minimal*, *rule-format*, *of snd* ‘ $\{p \in AB'.\ fst\ p = a\}$]
by *blast*

show $\exists\, z{\in}AB'.\ \forall\, y.\ (y,\ z) \in r_A <*lex*> r_B \longrightarrow y \notin AB'$
proof (*rule bexI*)
show $(a,\ b) \in AB'$
using *b-in* **by** (*simp add*: *image-iff*)
next
show $\forall\, y.\ (y,\ (a,\ b)) \in r_A <*lex*> r_B \longrightarrow y \notin AB'$
proof (*intro allI impI*)
fix *p* **assume** $(p,\ (a,\ b)) \in r_A <*lex*> r_B$
hence $(fst\ p,\ a) \in r_A \vee fst\ p = a \wedge (snd\ p,\ b) \in r_B$
unfolding *lex-prod-def* **by** *auto*
thus $p \notin AB'$
proof (*elim disjE conjE*)
assume $(fst\ p,\ a) \in r_A$
hence $fst\ p \notin fst\ `\ AB'$
using *a-minimal* **by** *simp*
thus *?thesis*
by (*rule contrapos-nn*) *simp*
next
assume $fst\ p = a$ **and** $(snd\ p,\ b) \in r_B$
hence $snd\ p \notin snd\ `\ \{p \in AB'.\ fst\ p = a\}$
using *b-minimal* **by** *simp*
thus $p \notin AB'$
by (*rule contrapos-nn*) (*simp add*: ‹$fst\ p = a$›)
qed
qed
qed
qed

lemma *wfp-on-lex-prodp*: *wfp-on A* $R_A \Longrightarrow$ *wfp-on B* $R_B \Longrightarrow AB \subseteq A \times B \Longrightarrow$ *wfp-on AB* (*lex-prodp* R_A R_B)
using *wf-on-lex-prod*[*of A - B - AB*, *to-pred*, *unfolded lex-prod-lex-prodp-iff*, *to-pred*] **.**

corollary *wfp-lex-prodp*: *wfp* $R_A \Longrightarrow$ *wfp* $R_B \Longrightarrow$ *wfp* (*lex-prodp* R_A R_B)
using *wfp-on-lex-prodp*[*of UNIV - UNIV*, *simplified*] **.**

lemma *wfp-on-sup-if-convertible-to-wfp*:
includes *lattice-syntax*
assumes
wf-S: *wfp-on A S* **and**
wf-Q: *wfp-on* $(f\ `\ A)$ *Q* **and**
convertible-R: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ (f\ x)\ (f\ y)$ **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-on A* $(R \sqcup S)$

proof (*rule wfp-on-if-convertible-to-wfp*)
show *wfp-on* ((λ*x*. (*f x*, *x*)) ‘ *A*) (*lex-prodp Q S*)
proof (*rule wfp-on-subset*)
show *wfp-on* (*f* ‘ *A* × *A*) (*lex-prodp Q S*)
by (*rule wfp-on-lex-prodp*[*OF wf-Q wf-S subset-refl*])
next
show (λ*x*. (*f x*, *x*)) ‘ *A* ⊆ *f* ‘ *A* × *A*
by *auto*
qed
next
fix *x y*
show *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ (*R* ⊔ *S*) *x y* ⟹ *lex-prodp Q S* (*f x*, *x*) (*f y*, *y*)
using *convertible-R convertible-S*
by (*auto simp add*: *lex-prodp-def*)
qed

lemma *wfp-on-iff-wfp*: *wfp-on A R* ⟷ *wfp* (λ*x y*. *R x y* ∧ *x* ∈ *A* ∧ *y* ∈ *A*)
by (*smt* (*verit*, *del-insts*) *UNIV-I subsetI wfp-on-def wfp-on-antimono-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 llist*
assumes *Lazy-List-Chain.chain* (λ*x y*. *R x y*) (*LCons init xs*)
shows *Lazy-List-Chain.chain* (λ*x y*. *R x y* ∧ *R*** *init x*) (*LCons init xs*)
proof (*intro lnth-rel-chain allI impI conjI*)
show ¬ *lnull* (*LCons init xs*)
by *simp*
next
fix *j*
assume *hyp*: *enat* (*j* + *1*) < *llength* (*LCons init xs*)

```
  from hyp show R (lnth (LCons init xs) j) (lnth (LCons init xs) (j + 1))
    using assms[THEN chain-lnth-rel, of j] by simp

  from hyp show R** init (lnth (LCons init xs) j)
    using assms[THEN chain-lnth-rtranclp, of j]
    by (simp add: Suc-ile-eq)
qed

lemma rtranclp-implies-ex-lfinite-chain:
  assumes run: R** x0 x
  shows ∃ xs. lfinite xs ∧ chain (λy z. R y z ∧ R** x0 y) (LCons x0 xs) ∧ llast
(LCons x0 xs) = x
  using run
proof (induction x rule: rtranclp-induct)
  case base
  then show ?case
    by (meson chain.chain-singleton lfinite-code(1) llast-singleton)
next
  case (step y z)
  from step.IH obtain xs where
    lfinite xs and chain (λy z. R y z ∧ R** x0 y) (LCons x0 xs) and llast (LCons
x0 xs) = y
    by auto
  let ?xs = lappend xs (LCons z LNil)
  show ?case
  proof (intro exI conjI)
    show lfinite ?xs
      using ‹lfinite xs› by simp
  next
    show chain (λy z. R y z ∧ R** x0 y) (LCons x0 ?xs)
      using ‹chain (λy z. R y z ∧ R** x0 y) (LCons x0 xs)› ‹llast (LCons x0 xs)
= y›
        chain.chain-singleton chain-lappend step.hyps(1) step.hyps(2)
      by fastforce
  next
    show llast (LCons x0 ?xs) = z
      by (simp add: ‹lfinite xs› llast-LCons)
  qed
qed

lemma chain-conj-rtranclpD:
  fixes xs :: 'a llist
  assumes inf: ¬ lfinite xs and chain: chain (λy z. R y z ∧ R** x0 y) xs
  shows ∃ ys. lfinite ys ∧ chain (λy z. R y z ∧ R** x0 y) (lappend ys xs) ∧ lhd
(lappend ys xs) = x0
  using chain
proof (cases λy z. R y z ∧ R** x0 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: rtranclp.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: rtranclp-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-rtranclp-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-antimono-strong*)
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 ‹$\neg$ *lfinite xs*› *lfinite-lappend lfinite-ltl*
by *blast*
next
show *chain* $(\lambda y\ z.\ R\ y\ z \wedge R^{**}\ x_0\ y)$ (*LCons* x_0 (*ltl* (*lappend ys xs*)))
using ‹*chain* $(\lambda y\ z.\ R\ y\ z \wedge R^{**}\ x_0\ y)$ (*lappend ys xs*)› ‹*lhd* (*lappend ys xs*) $= x_0$›
chain-not-lnull lhd-LCons-ltl
by *fastforce*
qed
with *no-inf-chain* **show** *False*
by *argo*
qed
hence *Wellfounded.wfP* $(\lambda z\ y.\ R\ y\ z \wedge y \in \{x.\ R^{**}\ x_0\ x\})$
unfolding *wfP-iff-no-infinite-down-chain-llist*
using *Lazy-List-Chain.chain-mono* **by** *fastforce*
hence *wfp* $(\lambda z\ y.\ R^{-1-1}\ z\ y \wedge z \in \{x.\ R^{**}\ x_0\ x\} \wedge y \in \{x.\ R^{**}\ x_0\ x\})$
by (*auto elim*: *wfp-on-antimono-strong*)
thus *wfp-on* $\{x.\ R^{**}\ x_0\ x\}\ R^{-1-1}$
unfolding *wfp-on-iff-wfp*[*of* $\{x.\ R^{**}\ x_0\ x\}\ R^{-1-1}$] **by** *argo*
qed

end
theory *Termination*
imports
SCL-FOL
Non-Redundancy
Wellfounded-Extra
HOL−Library.Monad-Syntax
begin

17 Extra Lemmas

17.1 Set Extra

lemma *minus-psubset-minusI*:
assumes $C \subset B$ **and** $B \subseteq A$
shows $(A - B \subset A - C)$
proof (*rule Set.psubsetI*)
show $A - B \subseteq A - C$
using *assms*(*1*) **by** *blast*
next
show $A - B \neq A - C$
using *assms* **by** *blast*
qed

17.2 Prod Extra

lemma *lex-prod-lex-prodp-eq*:
lex-prod $\{(x, y).\ RA\ x\ y\}\ \{(x, y).\ RB\ x\ y\} = \{(x, y).\ lex\text{-}prodp\ RA\ RB\ x\ y\}$
unfolding *lex-prodp-def lex-prod-def*
by *auto*

lemma *reflp-on-lex-prodp*:
assumes *reflp-on A RA*
shows *reflp-on* $(A \times B)$ (*lex-prodp RA RB*)
proof (*rule reflp-onI*)
fix x **assume** $x \in A \times B$
hence *fst* $x \in A$
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 *asymp-lex-prodp*:
assumes *asymp RA* **and** *asymp RB*
shows *asymp* (*lex-prodp RA RB*)
proof (*rule asympI*)
fix *x y* **assume** *lex-prodp RA RB x y*
thus ¬ *lex-prodp RA RB y x*
using *assms* **by** (*metis* (*full-types*, *opaque-lifting*) *asympD lex-prodp-def*)
qed

lemma *totalp-on-lex-prodp*:
assumes *totalp-on A RA* **and** *totalp-on B RB*
shows *totalp-on* (*A* × *B*) (*lex-prodp RA RB*)
proof (*rule totalp-onI*)
fix *x y* **assume** *x* ∈ *A* × *B* **and** *y* ∈ *A* × *B* **and** *x* ≠ *y*
then show *lex-prodp RA RB x y* ∨ *lex-prodp RA RB y x*
using *assms*
by (*metis* (*full-types*) *lex-prodp-def mem-Times-iff prod-eq-iff totalp-on-def*)
qed

17.3 Wellfounded Extra

17.4 FSet Extra

lemma *finsert-Abs-fset*: *finite A* ⟹ *finsert a* (*Abs-fset A*) = *Abs-fset* (*insert a A*)
by (*simp add*: *eq-onp-same-args finsert.abs-eq*)

lemma *minus-pfsubset-minusI*:
assumes *C* |⊂| *B* **and** *B* |⊆| *A*
shows (*A* |−| *B* |⊂| *A* |−| *C*)
proof (*rule FSet.pfsubsetI*)
show *A* |−| *B* |⊆| *A* |−| *C*
using *assms*(*1*) **by** *blast*
next
show *A* |−| *B* ≠ *A* |−| *C*
using *assms* **by** *blast*
qed

lemma *Abs-fset-minus*: *finite A* ⟹ *finite B* ⟹ *Abs-fset* (*A* − *B*) = *Abs-fset A* |−| *Abs-fset B*
by (*metis Abs-fset-inverse fset-inverse mem-Collect-eq minus-fset*)

lemma *fminus-conv*: *A* |⊂| *B* ⟷ *fset A* ⊂ *fset B* ∧ *finite* (*fset A*) ∧ *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 $\mathcal{M}$*-prop-deci* :: - ⇒ - ⇒ (-, -) *Term.term literal fset* **where**
$\mathcal{M}$*-prop-deci* β Γ = *Abs-fset* {*L*. *atm-of L* $\preceq_B$ β} |−| (*fst* |`| *fset-of-list* Γ)

primrec $\mathcal{M}$*-skip-fact-reso* **where**
$\mathcal{M}$*-skip-fact-reso* [] *C* = [] |
$\mathcal{M}$*-skip-fact-reso* (*Ln* # Γ) *C* =
(*let n* = *count C* (− (*fst Ln*)) *in*
(*case snd Ln of None* ⇒ *0* | *Some* - ⇒ *n*) #
$\mathcal{M}$*-skip-fact-reso* Γ (*C* + (*case snd Ln of None* ⇒ {#} | *Some* (*D*, -, γ) ⇒ *repeat-mset n* (*D* · γ))))

fun $\mathcal{M}$*-skip-fact-reso′* **where**
$\mathcal{M}$*-skip-fact-reso′ C* [] = [] |
$\mathcal{M}$*-skip-fact-reso′ C* ((-, *None*) # Γ) = *0* # $\mathcal{M}$*-skip-fact-reso′ C* Γ |
$\mathcal{M}$*-skip-fact-reso′ C* ((*K*, *Some* (*D*, -, γ)) # Γ) =
(*let n* = *count C* (− *K*) *in n* # $\mathcal{M}$*-skip-fact-reso′* (*C* + *repeat-mset n* (*D* · γ)) Γ)

lemma $\mathcal{M}$*-skip-fact-reso* Γ *C* = $\mathcal{M}$*-skip-fact-reso′ C* Γ
proof (*induction* Γ *arbitrary*: *C*)
case *Nil*
show *?case*
by *simp*
next
case (*Cons Kn* Γ)
then show *?case*
apply (*cases Kn*)
apply (*cases snd Kn*)
by (*auto simp add*: *Let-def*)
qed

lemma $\mathcal{M}$*-skip-fact-reso′ C* (*decide-lit K* # Γ) = *0* # $\mathcal{M}$*-skip-fact-reso′ C* Γ
by (*simp add*: *decide-lit-def*)

lemma $\mathcal{M}$*-skip-fact-reso′ C* (*propagate-lit K D* γ # Γ) =
(*let n* = *count C* (− (*K* ·*l* γ)) *in n* # $\mathcal{M}$*-skip-fact-reso′* (*C* + *repeat-mset n* (*D* · γ)) Γ)
by (*simp add*: *propagate-lit-def*)

fun $\mathcal{M}$ :: - ⇒ - ⇒ ('*f*, '*v*) *state* ⇒
bool × ('*f*, '*v*) *Term.term literal fset* × *nat list* × *nat* **where**
$\mathcal{M}$ *N* β (Γ, *U*, *None*) = (*True*, $\mathcal{M}$*-prop-deci* β Γ, [], *0*) |
$\mathcal{M}$ *N* β (Γ, *U*, *Some* (*C*, γ)) = (*False*, {||}, $\mathcal{M}$*-skip-fact-reso* Γ (*C* · γ), *size C*)

lemma *length-$\mathcal{M}$-skip-fact-reso*[*simp*]: *length* (*$\mathcal{M}$-skip-fact-reso* Γ *C*) = *length* Γ
  by (*induction* Γ *arbitrary*: *C*) (*simp-all add*: *Let-def*)

lemma *$\mathcal{M}$-skip-fact-reso-add-mset*:
  (*$\mathcal{M}$-skip-fact-reso* Γ *C*, *$\mathcal{M}$-skip-fact-reso* Γ (*add-mset L C*)) ∈ (*List.lenlex* {(*x*, *y*). *x* < *y*})$^=$
proof (*induction* Γ *arbitrary*: *C*)
  case *Nil*
  show *?case* **by** *simp*
next
  case (*Cons Ln* Γ)
  show *?case*
  proof (*cases snd Ln*)
    case *None*
    then show *?thesis*
      using *Cons.IH*[*of C*]
      by (*simp add*: *Cons-lenlex-iff*)
  next
    case (*Some cl*)
    show *?thesis*
    proof (*cases L* = − *fst Ln*)
      case *True*
      then show *?thesis*
        by (*simp add*: *Let-def Some Cons-lenlex-iff*)
    next
      case *False*
      then show *?thesis*
        using *Cons.IH*
        by (*auto simp add*: *Let-def Some Cons-lenlex-iff*)
    qed
  qed
qed

lemma *termination-scl-without-back-invars*:
  fixes *N* β
  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* ⊓
      initial-lits-generalize-learned-trail-conflict N ⊓ *ground-closures*
  shows *wfp-on* {*S*. *invars S*} *scl-without-backtrack*$^{-1-1}$
proof −
  let *?less* =
    lex-prodp ((<) :: *bool* ⇒ *bool* ⇒ *bool*)
      (*lex-prodp* (|⊂|)
        (*lex-prodp* (λ*x y*. (*x*, *y*) ∈ *List.lenlex* {(*x* :: - :: *wellorder*, *y*). *x* < *y*})
          ((<) :: *nat* ⇒ *nat* ⇒ *bool*)))

```
  show wfp-on {S. invars S} scl-without-backtrack^-1-1
  proof (rule wfp-on-if-convertible-to-wfp)
    fix S' S :: ('f, 'v) state
   assume S' ∈ {S. invars S} and S ∈ {S. 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
      initial-lits-generalize-learned-trail-conflict N S and
      ground-closures S
      initial-lits-generalize-learned-trail-conflict N S'
      by (simp-all add: invars-def)

    from step show ?less (M N β S') (M N β S)
      unfolding conversep-iff scl-without-backtrack-def sup-apply sup-bool-def
    proof (elim disjE)
      assume decide N β S S'
      thus ?less (M N β S') (M N β S)
      proof (cases N β S S' rule: decide.cases)
        case (decideI L γ Γ U)
        have M-prop-deci β ((L ·l γ, None) # Γ) |⊂| M-prop-deci β Γ
          unfolding M-prop-deci-def fset-of-list-simps fimage-finsert prod.sel
        proof (rule minus-pfsubset-minusI)
          show fst |‘| fset-of-list Γ |⊂| finsert (L ·l γ) (fst |‘| fset-of-list Γ)
            using ‹¬ trail-defined-lit Γ (L ·l γ)›[unfolded trail-defined-lit-def]
            by (metis (no-types, lifting) finsertCI fset-of-list-elem fset-of-list-map
                fsubset-finsertI list.set-map nless-le)
        next
          have L ·l γ ∈ {L. atm-of L ≼_B β}
            using ‹atm-of L ·a γ ≼_B β›
            by simp
          moreover have fst ‘ set Γ ⊆ {L. atm-of L ≼_B β}
            using ‹trail-atoms-lt β S›
            by (auto simp: trail-atoms-lt-def decideI(1))
          ultimately have insert (L ·l γ) (fst ‘ set Γ) ⊆ {L. atm-of L ≼_B β}
            by simp
         then show finsert (L ·l γ) (fst |‘| fset-of-list Γ) |⊆| Abs-fset {L. atm-of L
≼_B β}
            using finite-lits-less-eq-B
            by (simp add: less-eq-fset.rep-eq Abs-fset-inverse fset-of-list.rep-eq)
        qed
        then show ?thesis
          unfolding decideI(1,2) decide-lit-def
          unfolding lex-prodp-def
          by simp
      qed
    next
      assume propagate N β S S'
```

thus *?less* $(\mathcal{M}\ N\ \beta\ S')\ (\mathcal{M}\ N\ \beta\ S)$
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* γ {*atm-of* ‘ *set-mset* (*add-mset* $L\ C_1$)}
unfolding ‹$C_1 = \{\#K \in\# C'.\ K \cdot l\ \gamma = L \cdot l\ \gamma\#\}$›
by (*auto simp*: *is-unifiers-def is-unifier-alt intro*: *subst-atm-of-eqI*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu* μ {*atm-of* ‘ *set-mset* (*add-mset* $L\ C_1$)}›[*unfolded is-imgu-def*, *THEN conjunct2*]
by *simp*
thus *?thesis*
by (*metis subst-lit-comp-subst*)
qed

have $\mathcal{M}$*-prop-deci* β (($L \cdot l\ \gamma$, *Some* ($C_0 \cdot \mu$, $L \cdot l\ \mu$, γ)) # Γ) |⊂| $\mathcal{M}$*-prop-deci* $\beta\ \Gamma$
unfolding $\mathcal{M}$*-prop-deci-def fset-of-list-simps fimage-finsert prod.sel*
proof (*rule minus-pfsubset-minusI*)
show *fst* |‘| *fset-of-list* Γ |⊂| *finsert* ($L \cdot l\ \gamma$) (*fst* |‘| *fset-of-list* Γ)
using ‹¬ *trail-defined-lit* Γ ($L \cdot l\ \gamma$)›[*unfolded trail-defined-lit-def*]
by (*metis* (*no-types*, *lifting*) *finsertCI fset-of-list-elem fset-of-list-map fsubset-finsertI list.set-map nless-le*)
next
have *insert* ($L \cdot l\ \gamma$) (*fst* ‘ *set* Γ) ⊆ {L. *atm-of* $L \preceq_B \beta$}
proof (*intro Set.subsetI Set.CollectI*)
fix K **assume** $K \in$ *insert* ($L \cdot l\ \gamma$) (*fst* ‘ *set* Γ)
thus *atm-of* $K \preceq_B \beta$
using ‹*trail-atoms-lt* $\beta\ S$›
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* ($L \cdot l\ \gamma$) (*fst* |‘| *fset-of-list* Γ) |⊆| *Abs-fset* {L. *atm-of* $L \preceq_B \beta$}
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 ‹$L \cdot l\ \mu \cdot l\ \gamma = L \cdot l\ \gamma$›
unfolding *lex-prodp-def*
by *simp*
qed
next
assume *conflict* $N\ \beta\ S\ S'$
thus *?less* $(\mathcal{M}\ N\ \beta\ S')\ (\mathcal{M}\ N\ \beta\ S)$

```
    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-imgu μ {{atm-of L, atm-of L'}}›
        by (simp add: is-imgu-def is-unifiers-def)

      have add-mset L D · μ · γ = add-mset L D · γ
        using ‹μ ⊙ γ = γ›
        by (metis subst-cls-comp-subst)
      hence (M-skip-fact-reso Γ (add-mset L D · μ · γ),
        M-skip-fact-reso Γ (add-mset L' (add-mset L D) · γ)) ∈ (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 β S S'
    thus ?less (M N β S') (M N β S)
    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)
      from ‹ground-closures S› have
        ground-conf: is-ground-cls (add-mset L C · γ_C) and
```

ground-prop: *is-ground-cls* (*add-mset K D* $\cdot \gamma_D$)
unfolding *resolveI*(*1*,*2*) ‹Γ = *trail-propagate* Γ' *K D* γ_D›
by (*simp-all add*: *ground-closures-def propagate-lit-def*)
hence
∀ *L*∈#*add-mset L C*. *L* ·*l* ϱ_C ·*l* γ = *L* ·*l* γ_C
∀ *K*∈#*add-mset K D*. *K* ·*l* ϱ_D ·*l* γ = *K* ·*l* γ_D
using *resolveI merge-of-renamed-groundings* **by** *metis*+

have *atm-of L* ·*a* ϱ_C ·*a* γ = *atm-of K* ·*a* ϱ_D ·*a* γ
using ‹*K* ·*l* γ_D = − (*L* ·*l* γ_C)›
‹∀ *L*∈#*add-mset L C*. *L* ·*l* ϱ_C ·*l* γ = *L* ·*l* γ_C›[*rule-format*, *of L*, *simplified*]
‹∀ *K*∈#*add-mset K D*. *K* ·*l* ϱ_D ·*l* γ = *K* ·*l* γ_D›[*rule-format*, *of K*, *simplified*]
by (*metis atm-of-eq-uminus-if-lit-eq atm-of-subst-lit*)
hence *is-unifiers* γ {{*atm-of L* ·*a* ϱ_C, *atm-of K* ·*a* ϱ_D}}
by (*simp add*: *is-unifiers-def is-unifier-alt*)
hence $\mu \odot \gamma = \gamma$
using ‹*is-imgu* μ {{*atm-of L* ·*a* ϱ_C, *atm-of K* ·*a* ϱ_D}}›
by (*auto simp*: *is-imgu-def*)
hence *C* · ϱ_C · μ · γ = *C* · γ_C **and** *D* · ϱ_D · μ · γ = *D* · γ_D
using ‹∀ *L*∈#*add-mset L C*. *L* ·*l* ϱ_C ·*l* γ = *L* ·*l* γ_C› ‹∀ *K*∈#*add-mset K D*. *K* ·*l* ϱ_D ·*l* γ = *K* ·*l* γ_D›
by (*metis insert-iff same-on-lits-clause set-mset-add-mset-insert subst-cls-comp-subst subst-lit-comp-subst*)+
hence (*C* · ϱ_C + *D* · ϱ_D) · μ · γ = *C* · γ_C + *D* · γ_D
by (*metis subst-cls-comp-subst subst-cls-union*)

have *L* ·*l* γ_C ∉# *D* · γ_D
using ‹*trail-resolved-lits-pol S*› ‹*K* ·*l* γ_D = − (*L* ·*l* γ_C)›
unfolding *resolveI*(*1*,*2*) ‹Γ = *trail-propagate* Γ' *K D* γ_D›
by (*simp add*: *trail-resolved-lits-pol-def propagate-lit-def*)

have ($\mathcal{M}$-*skip-fact-reso* Γ (*C* · γ_C + *D* · γ_D), $\mathcal{M}$-*skip-fact-reso* Γ (*add-mset L C* · γ_C)) ∈
lex {(*x*, *y*). *x* < *y*}
unfolding ‹Γ = *trail-propagate* Γ' *K D* γ_D› *propagate-lit-def*
unfolding $\mathcal{M}$-*skip-fact-reso.simps Let-def prod.sel option.case prod.case*
unfolding *lex-conv mem-Collect-eq prod.case*
apply (*rule conjI*)
apply *simp*
apply (*rule exI*[*of* - []])
apply *simp*
using ‹*K* ·*l* γ_D = − (*L* ·*l* γ_C)›
apply *simp*
unfolding *count-eq-zero-iff*
by (*rule* ‹*L* ·*l* γ_C ∉# *D* · γ_D›)
hence ($\mathcal{M}$-*skip-fact-reso* Γ (*C* · γ_C + *D* · γ_D), $\mathcal{M}$-*skip-fact-reso* Γ (*add-mset L C* · γ_C)) ∈
lenlex {(*x*, *y*). *x* < *y*}

```
        unfolding lenlex-conv by simp
      thus ?thesis
        unfolding lex-prodp-def resolveI(1,2)
        unfolding M.simps state-proj-simp option.case prod.case prod.sel
        unfolding ‹(C · ϱC + D · ϱD) · μ · γ = C · γC + D · γD›
        by simp
    qed
  qed
 next
  show wfp-on (M N β ‘ {S. invars S}) ?less
  proof (rule wfp-on-subset)
    show M N β ‘ {S. invars S} ⊆ UNIV
      by simp
  next
    show wfp ?less
    proof (intro wfp-lex-prodp)
      show wfp ((<) :: bool ⇒ bool ⇒ bool)
        by (simp add: Wellfounded.wfPUNIVI)
    next
      show wfp (|⊂|)
        by (rule wfP-pfsubset)
    next
      show wfp (λx y. (x, y) ∈ lenlex {(x :: - :: wellorder, y). x < y})
        unfolding Wellfounded.wfP-wf-eq
        using wf-lenlex
        using wf by blast
    next
      show wfp ((<) :: nat ⇒ nat ⇒ bool)
        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 ⊓
    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* $\Longrightarrow$ *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* $\Longrightarrow$ *invars S'*
unfolding *invars-def*
using
scl-preserves-trail-atoms-lt
scl-preserves-trail-resolved-lits-pol
scl-preserves-trail-lits-ground
scl-preserves-initial-lits-generalize-learned-trail-conflict
scl-preserves-ground-closures
by *simp-all*
qed
ultimately have *scl-without-backtrack*** *initial-state S* $\Longrightarrow$ *invars S* **for** *S*
by (*auto elim*: *rtranclp-induct*)
thus {*S*. *scl-without-backtrack*** *initial-state S*} ⊆ {*S*. *invars S*}
by *auto*
qed

corollary *termination-stragegy-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*
assumes *strategy-stronger*: $\bigwedge$*S S'*. *strategy S S'* $\Longrightarrow$ *scl-without-backtrack S S'*
shows *wfp-on* {*S*. *strategy*** *initial-state S*} *strategy*$^{-1-1}$
proof (*rule wfp-on-antimono-strong*)
show *wfp-on* {*S*. *strategy*** *initial-state S*} *scl-without-backtrack*$^{-1-1}$
proof (*rule wfp-on-subset*)
show *wfp-on* {*S*. *scl-without-backtrack*** *initial-state S*} *scl-without-backtrack*$^{-1-1}$
unfolding *scl-without-backtrack-def*
using *termination-scl-without-back* **by** *metis*
next
show {*S*. *strategy*** *initial-state S*} ⊆ {*S*. *scl-without-backtrack*** *initial-state S*}
using *strategy-stronger*
by (*metis* (*no-types*, *opaque-lifting*) *Collect-mono mono-rtranclp*)
qed
next

show $\bigwedge S'\ S.\ strategy^{-1-1}\ S'\ S \Longrightarrow scl\text{-}without\text{-}backtrack^{-1-1}\ S'\ S$
using *strategy-stronger* **by** *simp*
qed *simp*

18.2 Backtracking can only be done finitely often

definition *fclss-no-dup* :: $('f, 'v)$ *Term.term* $\Rightarrow$ $('f, 'v)$ *Term.term literal fset fset*
where
fclss-no-dup β = *fPow* (*Abs-fset* $\{L.$ *atm-of* $L \preceq_B \beta\}$)

lemma *image-fset-fset-fPow-eq*: *fset* ‘ *fset* (*fPow A*) = *Pow* (*fset A*)
proof (*rule Set.equalityI*)
show *fset* ‘ *fset* (*fPow A*) $\subseteq$ *Pow* (*fset A*)
by (*meson PowI fPowD image-subset-iff less-eq-fset.rep-eq*)
next
show *Pow* (*fset A*) $\subseteq$ *fset* ‘ *fset* (*fPow A*)
proof (*rule Set.subsetI*)
fix *x* **assume** $x \in$ *Pow* (*fset A*)
moreover hence *finite x*
by (*metis PowD finite-fset rev-finite-subset*)
ultimately show $x \in$ *fset* ‘ *fset* (*fPow A*)
unfolding *image-iff*
by (*metis PowD fPowI fset-cases less-eq-fset.rep-eq mem-Collect-eq*)
qed
qed

lemma
assumes $\forall L \in\# C.$ *count C L* = *1*
shows $\exists C'.$ C = *mset-set* C'
using *assms*
by (*metis count-eq-zero-iff count-mset-set*(*1*) *count-mset-set*(*3*) *finite-set-mset multiset-eqI*)

lemma *fmember-fclss-no-dup-if*:
assumes $\forall L \mid\in\mid C.$ *atm-of* $L \preceq_B \beta$
shows $C \mid\in\mid$ *fclss-no-dup* β
proof −
show *?thesis*
unfolding *fclss-no-dup-def fPow-iff*
proof (*rule fsubsetI*)
fix K **assume** $K \mid\in\mid C$
with *assms* **show** $K \mid\in\mid$ *Abs-fset* $\{L.$ *atm-of* $L \preceq_B \beta\}$
using *Abs-fset-inverse*[*simplified*, *OF finite-lits-less-eq-B*]
by *simp*
qed
qed

definition $\mathcal{M}$*-back* :: - $\Rightarrow$ $('f, 'v)$ *state* $\Rightarrow$ $('f, 'v)$ *Term.term literal fset fset*
where

```
  M-back β S = Abs-fset (fset (fclss-no-dup β) −
    Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned S)))

lemma M-back-after-regular-backtrack:
  assumes
    regular-run: (regular-scl N β)** initial-state S0 and
    conflict: conflict N β S0 S1 and
    resolution: (skip N β ⊔ factorize N β ⊔ resolve N β)++ S1 Sn and
    backtrack: backtrack N β Sn Sn′
  defines U ≡ state-learned Sn
  shows
    ∃ C γ. state-conflict Sn = Some (C, γ) ∧
      set-mset (C · γ) ∉ set-mset ‘ grounding-of-clss (fset N ∪ fset U) and
    M-back β Sn′ |⊂| M-back β Sn
proof −
  from regular-run have (scl N β)** initial-state S0
    by (induction S0 rule: rtranclp-induct)
      (auto intro: scl-if-regular rtranclp.rtrancl-into-rtrancl)
  with conflict have (scl N β)** initial-state S1
    by (meson regular-scl-if-conflict rtranclp.rtrancl-into-rtrancl scl-if-regular)
  with resolution have scl-run: (scl N β)** initial-state Sn
    by (metis (no-types, lifting) Nitpick.rtranclp-unfold mono-rtranclp
      regular-run-if-skip-factorize-resolve-run rtranclp-tranclp-tranclp scl-if-regular)

  from scl-run have ground-false-closures Sn
    by (induction Sn rule: rtranclp-induct)
      (auto intro: scl-preserves-ground-false-closures)
  hence ground-closures Sn
    using ground-false-closures-def by blast

  from scl-run have trail-atoms-lt β Sn
    by (induction Sn rule: rtranclp-induct)
      (auto intro: scl-preserves-trail-atoms-lt)

  obtain C γ where
    conf: state-conflict Sn = Some (C, γ) and
    set-conf-not-in-set-groundings:
      set-mset (C · γ) ∉ set-mset ‘ grounding-of-clss (fset N ∪ 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 ∃ C γ. state-conflict Sn = Some
(C, γ) ∧
      set-mset (C · γ) ∉ set-mset ‘ grounding-of-clss (fset N ∪ fset U)
    by (simp add: U-def)

  have Diff-strict-subsetI: x ∈ A ⟹ x ∈ B ⟹ A − B ⊂ A for x A B
    by auto

  have fset (fclss-no-dup β) − Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned
Sn′)) =
    fset (fclss-no-dup β) − 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 ... ⊂
    fset (fclss-no-dup β) − Abs-fset ‘ set-mset ‘ grounding-of-clss (fset (state-learned
Sn))
  proof (rule Diff-strict-subsetI)
    from ‹ground-closures Sn› have C · γ ∈ 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)) ∈ Abs-fset ‘ set-mset ‘ grounding-of-cls C
      by blast
  next
    have Abs-fset-in-image-Abs-fset-iff: Abs-fset A ∈ Abs-fset ‘ AA ⟷ A ∈ AA
      if finite A ∧ (∀ B ∈ 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) ∉ set-mset ‘ grounding-of-clss (fset (state-learned S1))
      using set-conf-not-in-set-groundings
      by (auto simp: grounding-of-clss-union)
    then have Abs-fset (set-mset (C · γ)) ∉
```

Abs-fset ‘ set-mset ‘ grounding-of-clss (*fset* (*state-learned Sn*))
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*)) ∈ *fset* (*fclss-no-dup β*)
proof (*intro fmember-fclss-no-dup-if ballI*)
fix *L* **assume** *L* |∈| *Abs-fset* (*set-mset* (*C* · *γ*))
hence *L* ∈# *C* · *γ*
by (*metis fset-fset-mset fset-inverse*)
moreover have *trail-false-cls* (*state-trail Sn*) (*C* · *γ*)
using ‹*ground-false-closures Sn*› *conf* **by** (*auto simp*: *ground-false-closures-def*)
ultimately show *atm-of L* ≼*B β*
using *ball-less-B-if-trail-false-and-trail-atoms-lt*[*OF* - ‹*trail-atoms-lt β Sn*›]
by *metis*
qed

ultimately show *Abs-fset* (*set-mset* (*C* · *γ*)) ∈ *fset* (*fclss-no-dup β*) −
Abs-fset ‘ set-mset ‘ grounding-of-clss (*fset* (*state-learned Sn*))
by *simp*
qed

finally show *M-back β Sn′* |⊂| *M-back β Sn*
unfolding *M-back-def*
unfolding *fminus-conv*
by (*simp add*: *Abs-fset-inverse*[*simplified*])
qed

18.3 Regular SCL terminates

theorem *termination-regular-scl-invars*:
fixes
N :: (′*f*, ′*v*) *Term.term clause fset* **and**
β :: (′*f*, ′*v*) *Term.term*
defines
invars ≡ *trail-atoms-lt β* ⊓ *trail-resolved-lits-pol* ⊓ *trail-lits-ground* ⊓
initial-lits-generalize-learned-trail-conflict N ⊓ *ground-closures* ⊓ *ground-false-closures*
⊓
sound-state N β ⊓ *almost-no-conflict-with-trail N β* ⊓ *regular-conflict-resolution*
N β
shows
wfp-on {*S*. *invars S*} (*regular-scl N β*)$^{-1-1}$
proof (*rule wfp-on-antimono-strong*)
fix *S S′* **assume** (*regular-scl N β*)$^{-1-1}$ *S S′*
thus (*backtrack N β* ⊔ (*propagate N β* ⊔ *decide N β* ⊔ *conflict N β* ⊔ *skip N β*
⊔ *factorize N β* ⊔
resolve N β))$^{-1-1}$ *S S′*
by (*auto simp*: *regular-scl-def reasonable-scl-def scl-def*)
next

show *wfp-on* {*S*. *invars S*} (*backtrack N β* ⊔ (*propagate N β* ⊔ *decide N β* ⊔ *conflict N β* ⊔
skip N β ⊔ *factorize N β* ⊔ *resolve N β*))$^{-1-1}$
unfolding *converse-join*[*of backtrack N β*]
proof (*rule wfp-on-sup-if-convertible-to-wfp*, *unfold mem-Collect-eq*)
show *wfp-on* {*S*. *invars S*} (*propagate N β* ⊔ *decide N β* ⊔ *conflict N β* ⊔ *skip N β* ⊔
factorize N β ⊔ *resolve N β*)$^{-1-1}$
using *termination-scl-without-back-invars*(*1*)[*of β N*]
by (*auto simp*: *invars-def inf-assoc elim*: *wfp-on-subset*)
next
show *wfp-on* (*M-back β* ‘ {*S*. *invars S*}) (|⊂|)
proof (*rule wfp-on-subset*)
show *wfp* (|⊂|)
by (*rule wfP-pfsubset*)
qed *simp*
next
fix *S′ S*
assume *invars S′* **and** *invars S* **and** (*backtrack N β*)$^{-1-1}$ *S′ S*
moreover from ‹*invars S*› **have** *sound-state N β S*
by (*simp add*: *invars-def*)

moreover from ‹*invars S*› **have** *almost-no-conflict-with-trail N β S*
by (*simp add*: *invars-def*)

moreover from ‹*invars S*› **have** *regular-conflict-resolution N β S*
by (*simp add*: *invars-def*)

moreover from ‹*invars S*› **have** *ground-false-closures S*
by (*simp add*: *invars-def*)

ultimately obtain *S0 S1 S2 S3 S4* **where**
reg-run: (*regular-scl N β*)** *initial-state S0* **and**
propa: *propagate N β S0 S1 regular-scl N β S0 S1* **and**
confl: *conflict N β S1 S2* **and**
facto: (*factorize N β*)** *S2 S3* **and**
resol: *resolve N β S3 S4* **and**
reg-res: (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S4 S*
using *before-regular-backtrack* **by** *blast*

show *M-back β S′* |⊂| *M-back β S*
proof (*rule M-back-after-regular-backtrack*)
show (*regular-scl N β*)** *initial-state S1*
using *reg-run propa*(*2*) **by** *simp*
next
show *conflict N β S1 S2*
by (*rule confl*)
next
have (*skip N β* ⊔ *factorize N β* ⊔ *resolve N β*)** *S2 S3*

```
      using facto
      by (rule mono-rtranclp[rule-format, rotated]) simp
    also have (skip N β ⊔ factorize N β ⊔ resolve N β)^++ S3 S4
      using resol by auto
    finally show (skip N β ⊔ factorize N β ⊔ resolve N β)^++ S2 S
      using reg-res by simp
  next
    from ‹(backtrack N β)^-1-1 S' S› show backtrack N β S S'
      by simp
  qed
 next
  fix S' S
  assume invars S' and invars S and
    (propagate N β ⊔ decide N β ⊔ conflict N β ⊔ skip N β ⊔ factorize N β ⊔
      resolve N β)^-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 M-back β S' = M-back β S
    by (simp add: M-back-def)
  thus M-back β S' |⊂| M-back β S ∨ M-back β S' = M-back β S ..
 qed
qed simp

corollary termination-regular-scl:
 fixes
  N :: ('f, 'v) Term.term clause fset and
  β :: ('f, 'v) Term.term
 defines
  invars ≡ trail-atoms-lt β ⊓ trail-resolved-lits-pol ⊓ trail-lits-ground ⊓
  initial-lits-generalize-learned-trail-conflict N ⊓ ground-closures ⊓ ground-false-closures
⊓
  sound-state N β ⊓ almost-no-conflict-with-trail N β ⊓ regular-conflict-resolution
N β
 shows wfp-on {S. (regular-scl N β)** initial-state S} (regular-scl N β)^-1-1
proof (rule wfp-on-subset)
 show wfp-on {S. invars S} (regular-scl N β)^-1-1
  by (rule termination-regular-scl-invars(1)[of β N, folded invars-def])
next
 note rea-to-scl = scl-if-reasonable
 note reg-to-rea = reasonable-if-regular
 note reg-to-scl = reg-to-rea[THEN rea-to-scl]

 have invars initial-state
  by (simp add: invars-def)

 moreover have ⋀S S'. regular-scl N β S S' ⟹ invars S ⟹ invars S'
  unfolding invars-def
```

using
reg-to-scl[*THEN scl-preserves-trail-atoms-lt*]
reg-to-scl[*THEN scl-preserves-trail-resolved-lits-pol*]
reg-to-scl[*THEN scl-preserves-trail-lits-ground*]
reg-to-scl[*THEN scl-preserves-initial-lits-generalize-learned-trail-conflict*]
reg-to-scl[*THEN scl-preserves-ground-closures*]
reg-to-scl[*THEN scl-preserves-ground-false-closures*]
reg-to-scl[*THEN scl-preserves-sound-state*]
regular-scl-preserves-almost-no-conflict-with-trail
regular-scl-preserves-regular-conflict-resolution
by *simp*
ultimately have (*regular-scl N* β)** *initial-state S* $\Longrightarrow$ *invars S* **for** *S*
by (*auto elim*: *rtranclp-induct*)
thus {*S*. (*regular-scl N* β)** *initial-state S*} $\subseteq$ {*S*. *invars S*}
by *auto*
qed

corollary *termination-strategy*:
fixes
N :: ('*f*, '*v*) *Term.term clause fset* **and**
β :: ('*f*, '*v*) *Term.term*
assumes *strategy-restricts-regular-scl*: $\bigwedge$*S S'*. *strategy S S'* $\Longrightarrow$ *regular-scl N* β *S S'*
shows *wfp-on* {*S*. *strategy*** *initial-state S*} *strategy*$^{-1-1}$
proof (*rule wfp-on-antimono-strong*)
show *wfp-on* {*S*. *strategy*** *initial-state S*} (*regular-scl N* β)$^{-1-1}$
proof (*rule wfp-on-subset*)
show *wfp-on* {*S*. (*regular-scl N* β)** *initial-state S*} (*regular-scl N* β)$^{-1-1}$
using *termination-regular-scl* **by** *metis*
next
show {*S*. *strategy*** *initial-state S*} $\subseteq$ {*S*. (*regular-scl N* β)** *initial-state S*}
using *strategy-restricts-regular-scl*
by (*metis* (*no-types*, *opaque-lifting*) *Collect-mono mono-rtranclp*)
qed
next
show $\bigwedge$*S' S*. *strategy*$^{-1-1}$ *S' S* $\Longrightarrow$ (*regular-scl N* β)$^{-1-1}$ *S' S*
using *strategy-restricts-regular-scl* **by** *simp*
qed *simp*

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 β *gnd-N*
defines
gnd-N $\equiv$ *grounding-of-clss* (*fset* N) **and**
gnd-N-lt-β $\equiv \{C \in$ *gnd-N*. $\forall L \in\# C$. *atm-of* $L \preceq_B \beta\}$
assumes
unsat: $\neg$ *satisfiable gnd-N-lt-*β **and**
run: (*regular-scl* N β)** *initial-state* S **and**
no-more-step: $\nexists S'$. *regular-scl* N β S S'
shows $\exists\gamma$. *state-conflict* S = *Some* ($\{\#\}$, γ)
using *unsat correct-termination-regular-scl-run*[*OF run no-more-step*]
by (*simp add*: *gnd-N-lt-*β*-def gnd-N-def*)

theorem (**in** *scl-fol-calculus*)
fixes N β *gnd-N*
defines
gnd-N $\equiv$ *grounding-of-clss* (*fset* N) **and**
gnd-N-lt-β $\equiv \{C \in$ *gnd-N*. $\forall L \in\# C$. *atm-of* $L \preceq_B \beta\}$
assumes *unsat*: $\neg$ *satisfiable gnd-N-lt-*β
shows $\exists S$. (*regular-scl* N β)** *initial-state* $S \wedge$
($\nexists S'$. *regular-scl* N β S S') $\wedge$
($\exists\gamma$. *state-conflict* S = *Some* ($\{\#\}$, γ))
proof −
obtain S **where**
run: (*regular-scl* N β)** *initial-state* S **and**
no-more-step: ($\nexists S'$. *regular-scl* N β 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 $\exists\gamma$. *state-conflict* S = *Some* ($\{\#\}$, γ)
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*:
$\nexists Ss$. $\neg$ *lfinite Ss* $\wedge$ *Lazy-List-Chain.chain* (λS S'. *regular-scl* N β S S') (*LCons initial-state Ss*)
using *termination-regular-scl wfp-on-rtranclp-conversep-iff-no-infinite-down-chain-llist*
by *metis*

theorem (**in** *scl-fol-calculus*) *completeness-wrt-bound*:
fixes N β *gnd-N*
defines
gnd-N $\equiv$ *grounding-of-clss* (*fset* N) **and**

gnd-N-lt-β ≡ {*C* ∈ *gnd-N*. ∀ *L* ∈# *C*. *atm-of L* ≺*B* *β*}
assumes *unsat*: ¬ *satisfiable gnd-N-lt-β*
shows
∄ *Ss*. ¬ *lfinite Ss* ∧ *Lazy-List-Chain.chain* (λ*S S′*. *regular-scl N β S S′*)
(*LCons initial-state Ss*) **and**
∀ *S*. (*regular-scl N β*)** *initial-state S* ⟶ (∄ *S′*. *regular-scl N β S S′*) ⟶
(∃ *γ*. *state-conflict S* = *Some* ({#}, *γ*))
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* :: *weighted*, *'v*) *term* ⇒ (*'f*, *'v*) *term* ⇒ *bool*
for *renaming-vars* :: *'v set* ⇒ *'v* ⇒ *'v*
begin

theorem *ex-bound-if-unsat*:
fixes *N* :: (*'f*, *'v*) *term clause fset*
defines
gnd-N ≡ *grounding-of-clss* (*fset N*)
assumes *unsat*: ¬ *satisfiable gnd-N*
shows ∃ *β*. ¬ *satisfiable* {*C* ∈ *gnd-N*. ∀ *L* ∈# *C*. *atm-of L* ≤ *β*}
proof −
from *unsat* **obtain** *gnd-N′* **where**
gnd-N′ ⊆ *gnd-N* **and** *finite gnd-N′* **and** *unsat′*: ¬ *satisfiable gnd-N′*
using *clausal-logic-compact*[*of gnd-N*] **by** *metis*

have *gnd-N′* ≠ {}
using ‹¬ *satisfiable gnd-N′*› **by** *force*

obtain *C* **where** *C-in*: *C* ∈ *gnd-N′* **and** *C-min*: ∀ *x*∈*gnd-N′*. *x* ≤ *C*
using *finite-has-maximal*[*OF* ‹*finite gnd-N′*› ‹*gnd-N′* ≠ {}›] **by** *force*

show *?thesis*
proof (*cases C*)
case *empty*
let *?S* = ([], {||}, *Some* ({#}, *Var*))

show *?thesis*
proof (*rule exI*)
have {#} |∈| *N*
using *C-in* ‹*gnd-N′* ⊆ *gnd-N*›
unfolding *empty gnd-N-def*
by (*smt* (*verit*, *del-insts*) *SCL-FOL.grounding-of-clss-def*
SCL-FOL.subst-cls-empty-iff UN-E mem-Collect-eq subset-iff
substitution-ops.grounding-of-cls-def)
hence {#} ∈ *gnd-N*
using *C-in* ‹*gnd-N′* ⊆ *gnd-N*› *local.empty* **by** *blast*

hence $\{\#\} \in \{C \in$ *gnd-N*. $\forall L \in \# C$. *atm-of* $L <$ *undefined*$\}$
by *force*
thus $\neg$ *satisfiable* $\{C \in$ *gnd-N*. $\forall L \in \# C$. *atm-of* $L \leq$ *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** $\forall D \in$ *gnd-N'*. $\forall K \in \# D$. *atm-of K* $\leq$ *atm-of L*
by (*meson dual-order.trans ex-gt-imp-less-multiset leq-imp-less-eq-atm-of verit-comp-simplify1*(*3*))
hence *gnd-N'* $\subseteq \{D \in$ *gnd-N*. $\forall K \in \#$ *D*. (*atm-of K*) $\leq$ (*atm-of L*)$\}$
using ‹*gnd-N'* $\subseteq$ *gnd-N*› *subset-Collect-iff* **by** *auto*
hence $\neg$ *satisfiable* $\{D \in$ *gnd-N*. $\forall K \in \#$ *D*. (*atm-of K*) $\leq$ (*atm-of L*)$\}$
using ‹$\neg$ *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
scl N β *S S'* $\Longrightarrow$ *trail-lits-ground S* $\Longrightarrow$ *trail-lits-ground S'* **and**
inv-trail-atoms-lt:
trail-atoms-lt β *initial-state*
scl N β *S S'* $\Longrightarrow$ *trail-atoms-lt* β *S* $\Longrightarrow$ *trail-atoms-lt* β *S'* **and**
inv-undefined-trail-lits:
$\forall \Gamma'$ *Ln* Γ''. *state-trail initial-state* $= \Gamma'' @$ *Ln* $\# \Gamma' \longrightarrow \neg$ *trail-defined-lit* Γ' (*fst Ln*)
scl N β *S S'* $\Longrightarrow$

$(\forall \Gamma'\ Ln\ \Gamma''.\ state\text{-}trail\ S = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg\ trail\text{-}defined\text{-}lit\ \Gamma'\ (fst\ Ln)) \Longrightarrow$
$(\forall \Gamma'\ Ln\ \Gamma''.\ state\text{-}trail\ S' = \Gamma'' @ Ln \# \Gamma' \longrightarrow \neg\ trail\text{-}defined\text{-}lit\ \Gamma'\ (fst\ Ln))$ **and**
inv-ground-closures:
ground-closures initial-state
scl N β S S' $\Longrightarrow$ *ground-closures S* $\Longrightarrow$ *ground-closures S'* **and**
inv-ground-false-closures:
ground-false-closures initial-state
scl N β S S' $\Longrightarrow$ *ground-false-closures S* $\Longrightarrow$ *ground-false-closures S'* **and**
inv-trail-propagated-lits-wf:
$\forall \mathcal{K} \in set\ (state\text{-}trail\ initial\text{-}state).\ \forall D\ K\ \gamma.\ snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K\ \cdot l\ \gamma$
scl N β S S' $\Longrightarrow$
$(\forall \mathcal{K} \in set\ (state\text{-}trail\ S).\ \forall D\ K\ \gamma.\ snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K\ \cdot l\ \gamma) \Longrightarrow$
$(\forall \mathcal{K} \in set\ (state\text{-}trail\ S').\ \forall D\ K\ \gamma.\ snd\ \mathcal{K} = Some\ (D, K, \gamma) \longrightarrow fst\ \mathcal{K} = K\ \cdot l\ \gamma)$ **and**
inv-trail-resolved-lits-pol:
trail-resolved-lits-pol initial-state
scl N β S S' $\Longrightarrow$ *trail-resolved-lits-pol S* $\Longrightarrow$ *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' $\Longrightarrow$ *initial-lits-generalize-learned-trail-conflict N S* $\Longrightarrow$ *initial-lits-generalize-learned-trail-conflict N S'* **and**
inv-sound-state:
sound-state N β initial-state
scl N β S S' $\Longrightarrow$ *sound-state N β S* $\Longrightarrow$ *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 *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.