SCL(FOL) Can Simulate Ground Nonredundant Ordered Resolution

Martin Desharnais

March 14, 2025

Abstract

SCL(FOL) (i.e., Simple Clause Learning for First-Order Logic without equality) is known to be able to simulate the derivation of nonredundant clauses by the ground ordered resolution calculus [1]. Due to the space constraints of a 16-pages paper, the published proof is monolithic and hard to comprehend. In this work, we reuse the existing strategy for ground ordered resolution and present a new, simpler strategy for SCL(FOL). We prove a stronger bisimulation theorem between these two strategies (i.e., they both simulate each other). Our proof is modular: it consists of ten refinement steps focusing on different aspects of the two strategies.

Contents

theory *Isabelle-2024-Compatibility*
imports
Main
HOL−Library.Multiset
begin

lemmas *wfP-def* = *wfp-def*
lemmas *wfP-if-convertible-to-wfP* = *wfp-if-convertible-to-wfp*
lemmas *wfP-imp-asymp* = *wfp-imp-asymp*
lemmas *wfP-induct-rule* = *wfp-induct-rule*
lemmas *wfP-multp* = *wfp-multp*

end
theory *Ground-Ordered-Resolution*
imports
Saturation-Framework.Calculus
Saturation-Framework-Extensions.Clausal-Calculus
Isabelle-2024-Compatibility
First-Order-Clause.Ground-Context
First-Order-Clause.HOL-Extra
First-Order-Clause.Transitive-Closure-Extra
Min-Max-Least-Greatest.Min-Max-Least-Greatest-FSet
Min-Max-Least-Greatest.Min-Max-Least-Greatest-Multiset
First-Order-Clause.Multiset-Extra
Superposition-Calculus.Relation-Extra
begin

hide-type *Inference-System.inference*
hide-const
Inference-System.Infer
Inference-System.prems-of
Inference-System.concl-of
Inference-System.main-prem-of

primrec *mset-lit* :: *'a literal* ⇒ *'a multiset* **where**
mset-lit (*Pos A*) = {#*A*#} |

mset-lit (*Neg A*) = {#*A*, *A*#}

type-synonym *'t atom* = *'t*

1 Ground Resolution Calculus

locale *ground-ordered-resolution-calculus* =
fixes
less-trm :: *'f gterm* ⇒ *'f gterm* ⇒ *bool* (**infix** ≺$_t$ *50*) **and**
select :: *'f gterm atom clause* ⇒ *'f gterm atom clause*
assumes
transp-less-trm[*simp*]: *transp* (≺$_t$) **and**
asymp-less-trm[*intro*]: *asymp* (≺$_t$) **and**
wfP-less-trm[*intro*]: *wfP* (≺$_t$) **and**
totalp-less-trm[*intro*]: *totalp* (≺$_t$) **and**
less-trm-compatible-with-gctxt[*simp*]: ⋀*ctxt t t'*. *t* ≺$_t$ *t'* ⟹ *ctxt*⟨*t*⟩$_G$ ≺$_t$ *ctxt*⟨*t'*⟩$_G$
and
less-trm-if-subterm[*simp*]: ⋀*t ctxt*. *ctxt* ≠ □$_G$ ⟹ *t* ≺$_t$ *ctxt*⟨*t*⟩$_G$ **and**
select-subset: ⋀*C*. *select C* ⊆# *C* **and**
select-negative-lits: ⋀*C L*. *L* ∈# *select C* ⟹ *is-neg L*
begin

lemma *irreflp-on-less-trm*[*simp*]: *irreflp-on A* (≺$_t$)
by (*metis asympD asymp-less-trm irreflp-onI*)

abbreviation *lesseq-trm* (**infix** ⪯$_t$ *50*) **where**
lesseq-trm ≡ (≺$_t$)$^{==}$

lemma *lesseq-trm-if-subtermeq*: *t* ⪯$_t$ *ctxt*⟨*t*⟩$_G$
using *less-trm-if-subterm*
by (*metis gctxt-ident-iff-eq-GHole reflclp-iff*)

definition *less-lit* ::
'f gterm atom literal ⇒ *'f gterm atom literal* ⇒ *bool* (**infix** ≺$_l$ *50*) **where**
less-lit L1 L2 ≡ *multp* (≺$_t$) (*mset-lit L1*) (*mset-lit L2*)

abbreviation *lesseq-lit* (**infix** ⪯$_l$ *50*) **where**
lesseq-lit ≡ (≺$_l$)$^{==}$

abbreviation *less-cls* ::
'f gterm atom clause ⇒ *'f gterm atom clause* ⇒ *bool* (**infix** ≺$_c$ *50*) **where**
less-cls ≡ *multp* (≺$_l$)

abbreviation *lesseq-cls* (**infix** ⪯$_c$ *50*) **where**
lesseq-cls ≡ (≺$_c$)$^{==}$

lemma *transp-on-less-lit*[*simp*]: *transp-on A* (≺$_l$)
by (*smt* (*verit*, *best*) *less-lit-def transpE transp-less-trm transp-multp transp-onI*)

corollary *transp-less-lit*: *transp* ($\prec_l$)
by *simp*

lemma *transp-less-cls*[*simp*]: *transp* ($\prec_c$)
by (*simp add*: *transp-multp*)

lemma *asymp-on-less-lit*[*simp*]: *asymp-on A* ($\prec_l$)
by (*metis asympD asymp-less-trm asymp-multp*$_{HO}$ *asymp-onI less-lit-def multp-eq-multp*$_{HO}$ *transp-less-trm*)

corollary *asymp-less-lit*[*simp*]: *asymp* ($\prec_l$)
by *simp*

lemma *asymp-less-cls*[*simp*]: *asymp* ($\prec_c$)
by (*simp add*: *asymp-multp*$_{HO}$ *multp-eq-multp*$_{HO}$)

lemma *irreflp-on-less-lit*[*simp*]: *irreflp-on A* ($\prec_l$)
by (*simp only*: *asymp-on-less-lit irreflp-on-if-asymp-on*)

lemma *wfP-less-lit*[*simp*]: *wfP* ($\prec_l$)
unfolding *less-lit-def*
using *wfP-less-trm wfp-multp wfp-if-convertible-to-wfp* **by** *meson*

lemma *wfP-less-cls*[*simp*]: *wfP* ($\prec_c$)
using *wfP-less-lit wfp-multp* **by** *blast*

lemma *totalp-on-less-lit*[*simp*]: *totalp-on A* ($\prec_l$)
proof (*rule totalp-onI*)
fix *L1 L2* :: *'f gterm atom literal*
assume *L1* $\neq$ *L2*

show *L1* $\prec_l$ *L2* $\vee$ *L2* $\prec_l$ *L1*
unfolding *less-lit-def*
proof (*rule totalp-multp*[*THEN totalpD*])
show *totalp* ($\prec_t$)
using *totalp-less-trm* **.**
next
show *transp* ($\prec_t$)
using *transp-less-trm* **.**
next
show *mset-lit L1* $\neq$ *mset-lit L2*
using ‹*L1* $\neq$ *L2*›
by (*cases L1*; *cases L2*) (*auto simp add*: *add-eq-conv-ex*)
qed
qed

corollary *totalp-less-lit*: *totalp* ($\prec_l$)
by *simp*

lemma *totalp-less-cls*[*simp*]: *totalp* ($\prec_c$)
proof (*rule totalp-multp*)
  show *totalp* ($\prec_l$)
    by *simp*
next
  show *transp* ($\prec_l$)
    by *simp*
qed

interpretation *term-order*: *linorder lesseq-trm less-trm*
proof *unfold-locales*
  show $\bigwedge x\ y.\ (x \prec_t y) = (x \preceq_t y \wedge \neg\ y \preceq_t x)$
    by (*metis asympD asymp-less-trm reflclp-iff*)
next
  show $\bigwedge x.\ x \preceq_t x$
    by *simp*
next
  show $\bigwedge x\ y\ z.\ x \preceq_t y \Longrightarrow y \preceq_t z \Longrightarrow x \preceq_t z$
    by (*meson transpE transp-less-trm transp-on-reflclp*)
next
  show $\bigwedge x\ y.\ x \preceq_t y \Longrightarrow y \preceq_t x \Longrightarrow x = y$
    by (*metis asympD asymp-less-trm reflclp-iff*)
next
  show $\bigwedge x\ y.\ x \preceq_t y \vee y \preceq_t x$
    by (*metis reflclp-iff totalpD totalp-less-trm*)
qed

interpretation *literal-order*: *linorder lesseq-lit less-lit*
proof *unfold-locales*
  show $\bigwedge x\ y.\ (x \prec_l y) = (x \preceq_l y \wedge \neg\ y \preceq_l x)$
    by (*metis asympD asymp-less-lit reflclp-iff*)
next
  show $\bigwedge x.\ x \preceq_l x$
    by *simp*
next
  show $\bigwedge x\ y\ z.\ x \preceq_l y \Longrightarrow y \preceq_l z \Longrightarrow x \preceq_l z$
    by (*meson transpE transp-less-lit transp-on-reflclp*)
next
  show $\bigwedge x\ y.\ x \preceq_l y \Longrightarrow y \preceq_l x \Longrightarrow x = y$
    by (*metis asympD asymp-less-lit reflclp-iff*)
next
  show $\bigwedge x\ y.\ x \preceq_l y \vee y \preceq_l x$
    by (*metis reflclp-iff totalpD totalp-less-lit*)
qed

interpretation *clause-order*: *linorder lesseq-cls less-cls*
proof *unfold-locales*
  show $\bigwedge x\ y.\ (x \prec_c y) = (x \preceq_c y \wedge \neg\ y \preceq_c x)$
    by (*metis asympD asymp-less-cls reflclp-iff*)

next
show $\bigwedge x.\ x \preceq_c x$
by *simp*
next
show $\bigwedge x\ y\ z.\ x \preceq_c y \Longrightarrow y \preceq_c z \Longrightarrow x \preceq_c z$
by (*meson transpE transp-less-cls transp-on-reflclp*)
next
show $\bigwedge x\ y.\ x \preceq_c y \Longrightarrow y \preceq_c x \Longrightarrow x = y$
by (*metis asympD asymp-less-cls reflclp-iff*)
next
show $\bigwedge x\ y.\ x \preceq_c y \vee y \preceq_c x$
by (*metis reflclp-iff totalpD totalp-less-cls*)
qed

lemma *less-lit-simps*[*simp*]:
$Pos\ A_1 \prec_l Pos\ A_2 \longleftrightarrow A_1 \prec_t A_2$
$Pos\ A_1 \prec_l Neg\ A_2 \longleftrightarrow A_1 \preceq_t A_2$
$Neg\ A_1 \prec_l Neg\ A_2 \longleftrightarrow A_1 \prec_t A_2$
$Neg\ A_1 \prec_l Pos\ A_2 \longleftrightarrow A_1 \prec_t A_2$
by (*auto simp add*: *less-lit-def*)

1.1 Ground Rules

abbreviation *is-maximal-lit* :: *'f gterm literal* ⇒ *'f gterm clause* ⇒ *bool* **where**
is-maximal-lit L M ≡ *is-maximal-in-mset-wrt* (≺$_l$) *M L*

abbreviation *is-strictly-maximal-lit* :: *'f gterm literal* ⇒ *'f gterm clause* ⇒ *bool*
where
is-strictly-maximal-lit L M ≡ *is-greatest-in-mset-wrt* (≺$_l$) *M L*

inductive *ground-resolution* ::
'f gterm atom clause ⇒ *'f gterm atom clause* ⇒ *'f gterm atom clause* ⇒ *bool*
where
ground-resolutionI:
$P_1 = add\text{-}mset\ (Neg\ t)\ P_1' \Longrightarrow$
$P_2 = add\text{-}mset\ (Pos\ t)\ P_2' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$select\ P_1 = \{\#\} \wedge is\text{-}maximal\text{-}lit\ (Neg\ t)\ P_1 \vee Neg\ t \in\# select\ P_1 \Longrightarrow$
$select\ P_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\text{-}lit\ (Pos\ t)\ P_2 \Longrightarrow$
$C = P_1' + P_2' \Longrightarrow$
$ground\text{-}resolution\ P_1\ P_2\ C$

inductive *ground-factoring* :: *'f gterm atom clause* ⇒ *'f gterm atom clause* ⇒ *bool*
where
ground-factoringI:
$P = add\text{-}mset\ (Pos\ t)\ (add\text{-}mset\ (Pos\ t)\ P') \Longrightarrow$
$select\ P = \{\#\} \Longrightarrow$
$is\text{-}maximal\text{-}lit\ (Pos\ t)\ P \Longrightarrow$

$C = \mathit{add\text{-}mset}\ (\mathit{Pos}\ t)\ P' \Longrightarrow$
$\mathit{ground\text{-}factoring}\ P\ C$

1.2 Ground Layer

definition *G-Inf* :: *'f gterm atom clause inference set* **where**
G-Inf =
{*Infer* [P_2, P_1] C | P_2 P_1 C. *ground-resolution* P_1 P_2 C} ∪
{*Infer* [P] C | P C. *ground-factoring* P C}

abbreviation *G-Bot* :: *'f gterm atom clause set* **where**
G-Bot ≡ {{#}}

definition *G-entails* :: *'f gterm atom clause set* ⇒ *'f gterm atom clause set* ⇒ *bool* **where**
G-entails N_1 N_2 ⟷ (∀ (I :: *'f gterm set*). I ⊨*s* N_1 ⟶ I ⊨*s* N_2)

1.3 Correctness

lemma *soundness-ground-resolution*:
assumes
step: *ground-resolution P1 P2 C*
shows *G-entails* {*P1*, *P2*} {*C*}
using *step*
proof (*cases P1 P2 C rule*: *ground-resolution.cases*)
case (*ground-resolutionI t* P_1' P_2')

show *?thesis*
unfolding *G-entails-def true-clss-singleton*
unfolding *true-clss-insert*
proof (*intro allI impI*, *elim conjE*)
fix I :: *'f gterm set*
assume I ⊨ *P1* **and** I ⊨ *P2*
then obtain *K1 K2* :: *'f gterm atom literal* **where**
K1 ∈# *P1* **and** I ⊨*l* *K1* **and** *K2* ∈# *P2* **and** I ⊨*l* *K2*
by (*auto simp*: *true-cls-def*)

show I ⊨ C
proof (*cases K1* = *Neg t*)
case *K1-def*: *True*
hence I ⊨*l* *Neg t*
using ‹I ⊨*l* *K1*› **by** *simp*

show *?thesis*
proof (*cases K2* = *Pos t*)
case *K2-def*: *True*
hence I ⊨*l* *Pos t*
using ‹I ⊨*l* *K2*› **by** *simp*
hence *False*
using ‹I ⊨*l* *Neg t*› **by** *simp*

```
      thus ?thesis ..
    next
      case False
      hence K2 ∈# P_2'
        using ‹K2 ∈# P2›
        unfolding ground-resolutionI by simp
      hence I ⊨ P_2'
        using ‹I ⊨l K2› by blast
      thus ?thesis
        unfolding ground-resolutionI by simp
    qed
  next
    case False
    hence K1 ∈# P_1'
      using ‹K1 ∈# P1›
      unfolding ground-resolutionI by simp
    hence I ⊨ P_1'
      using ‹I ⊨l K1› by blast
    thus ?thesis
      unfolding ground-resolutionI by simp
  qed
 qed
qed

lemma soundness-ground-factoring:
 assumes step: ground-factoring P C
 shows G-entails {P} {C}
 using step
proof (cases P C rule: ground-factoring.cases)
 case (ground-factoringI t P')
 show ?thesis
  unfolding G-entails-def true-clss-singleton
 proof (intro allI impI)
  fix I :: 'f gterm set
  assume I ⊨ P
  then obtain K :: 'f gterm atom literal where
    K ∈# P and I ⊨l K
    by (auto simp: true-cls-def)

  show I ⊨ C
  proof (cases K = Pos t)
    case True
    hence I ⊨l Pos t
      using ‹I ⊨l K› by metis
    thus ?thesis
      unfolding ground-factoringI
      by (metis true-cls-add-mset)
  next
    case False
```

```
      hence K ∈# P'
        using ‹K ∈# P›
        unfolding ground-factoringI
        by auto
      hence K ∈# C
        unfolding ground-factoringI
        by simp
      thus ?thesis
        using ‹I ⊨l K› by blast
    qed
  qed
qed

interpretation G: sound-inference-system G-Inf G-Bot G-entails
proof unfold-locales
  show G-Bot ≠ {}
    by simp
next
  show ⋀B N. B ∈ G-Bot ⟹ G-entails {B} N
    by (simp add: G-entails-def)
next
  show ⋀N2 N1. N2 ⊆ N1 ⟹ G-entails N1 N2
    by (auto simp: G-entails-def elim!: true-clss-mono[rotated])
next
  fix N1 N2 assume ball-G-entails: ∀ C ∈ N2. G-entails N1 {C}
  show G-entails N1 N2
    unfolding G-entails-def
  proof (intro allI impI)
    fix I :: 'f gterm set
    assume I ⊨s N1
    hence ∀ C ∈ N2. I ⊨s {C}
      using ball-G-entails by (simp add: G-entails-def)
    then show I ⊨s N2
      by (simp add: true-clss-def)
  qed
next
  show ⋀N1 N2 N3. G-entails N1 N2 ⟹ G-entails N2 N3 ⟹ G-entails N1 N3
    using G-entails-def by simp
next
  show ⋀ι. ι ∈ G-Inf ⟹ G-entails (set (prems-of ι)) {concl-of ι}
    unfolding G-Inf-def
    using soundness-ground-resolution
    using soundness-ground-factoring
    by (auto simp: G-entails-def)
qed
```

1.4 Redundancy Criterion

lemma *ground-resolution-smaller-conclusion*:

assumes
step: *ground-resolution P1 P2 C*
shows $C \prec_c P1$
using *step*
proof (*cases P1 P2 C rule*: *ground-resolution.cases*)
case (*ground-resolutionI t* $P_1{}'$ $P_2{}'$)
have $\forall k{\in}\#P_2{}'.\ k \prec_l Pos\ t$
using ‹*is-strictly-maximal-lit* (*Pos t*) *P2*› ‹*P2* = *add-mset* (*Pos t*) $P_2{}'$›
by (*simp add*: *literal-order.is-greatest-in-mset-iff*)
moreover have $\bigwedge A.\ Pos\ A \prec_l Neg\ A$
by (*simp add*: *less-lit-def*)
ultimately have $\forall k{\in}\#P_2{}'.\ k \prec_l Neg\ t$
by (*metis transp-def transp-less-lit*)
hence $P_2{}' \prec_c \{\#Neg\ t\#\}$
using *one-step-implies-multp*[*of* $\{\#Neg\ t\#\}$ $P_2{}'$ $(\prec_l)$ $\{\#\}$] **by** *simp*
hence $P_2{}' + P_1{}' \prec_c add\text{-}mset\ (Neg\ t)\ P_1{}'$
using *multp-cancel*[*of* $(\prec_l)$ $P_1{}'$ $P_2{}'$ $\{\#Neg\ t\#\}$] **by** *simp*
thus *?thesis*
unfolding *ground-resolutionI*
by (*simp only*: *add.commute*)
qed

lemma *ground-factoring-smaller-conclusion*:
assumes *step*: *ground-factoring P C*
shows $C \prec_c P$
using *step*
proof (*cases P C rule*: *ground-factoring.cases*)
case (*ground-factoringI t P'*)
then show *?thesis*
by (*metis add-mset-add-single mset-subset-eq-exists-conv multi-self-add-other-not-self multp-subset-supersetI totalpD totalp-less-cls transp-less-lit*)
qed

interpretation *G*: *calculus-with-finitary-standard-redundancy G-Inf G-Bot G-entails* $(\prec_c)$
proof *unfold-locales*
show *transp* $(\prec_c)$
using *transp-less-cls* **.**
next
show *wfP* $(\prec_c)$
using *wfP-less-cls* **.**
next
show $\bigwedge\iota.\ \iota \in G\text{-}Inf \Longrightarrow prems\text{-}of\ \iota \neq []$
by (*auto simp*: *G-Inf-def*)
next
fix ι
have *concl-of* $\iota \prec_c$ *main-prem-of* ι
if ι-*def*: ι = *Infer* $[P_2, P_1]$ C **and**
infer: *ground-resolution* P_1 P_2 C

for P_2 P_1 C
unfolding ι*-def*
using *infer*
using *ground-resolution-smaller-conclusion*
by *simp*

moreover have *concl-of* $\iota \prec_c$ *main-prem-of* ι
if ι*-def*: $\iota =$ *Infer* $[P]$ C **and**
infer: *ground-factoring* P C
for P C
unfolding ι*-def*
using *infer*
using *ground-factoring-smaller-conclusion*
by *simp*

ultimately show $\iota \in$ *G-Inf* $\Longrightarrow$ *concl-of* $\iota \prec_c$ *main-prem-of* ι
unfolding *G-Inf-def*
by *fast*
qed

1.5 Refutational Completeness

context
fixes N :: *'f gterm atom clause set*
begin

function *production* :: *'f gterm atom clause* $\Rightarrow$ *'f gterm set* **where**
production $C = \{A \mid A\ C'.$
$C \in N \wedge$
$C =$ *add-mset* (*Pos* A) $C' \wedge$
select $C = \{\#\} \wedge$
is-strictly-maximal-lit (*Pos* A) $C \wedge$
$\neg\ (\bigcup D \in \{D \in N.\ D \prec_c C\}.\ production\ D) \models C\}$
by *simp-all*

termination *production*
proof (*relation* $\{(x, y).\ x \prec_c y\}$)
show *wf* $\{(x, y).\ x \prec_c y\}$
using *wfP-less-cls*
by (*simp add*: *wfp-def*)
next
show $\bigwedge C\ D.\ D \in \{D \in N.\ D \prec_c C\} \Longrightarrow (D, C) \in \{(x, y).\ x \prec_c y\}$
by *simp*
qed

declare *production.simps*[*simp del*]

end

lemma *Uniq-striclty-maximal-lit-in-ground-cls*:
  $\exists_{\leq 1} L.$ *is-strictly-maximal-lit* L C
proof (*rule Uniq-is-greatest-in-mset-wrt*)
  show *transp-on* (*set-mset* C) ($\prec_l$)
    by (*auto intro*: *transp-on-subset transp-less-lit*)
next
  show *asymp-on* (*set-mset* C) ($\prec_l$)
    by (*auto intro*: *asymp-on-subset asymp-less-lit*)
next
  show *totalp-on* (*set-mset* C) ($\prec_l$)
    by (*auto intro*: *totalp-on-subset totalp-less-lit*)
qed

lemma *production-eq-empty-or-singleton*:
  production N $C = \{\} \vee (\exists A.$ *production* N $C = \{A\})$
proof −
  have $\exists_{\leq 1} A.$ *is-strictly-maximal-lit* (*Pos* A) C
    using *Uniq-striclty-maximal-lit-in-ground-cls*
    by (*metis* (*mono-tags*, *lifting*) *Uniq-def literal.inject*(*1*))
  hence $\exists_{\leq 1} A. \exists C'.$ $C =$ *add-mset* (*Pos* A) $C' \wedge$ *is-strictly-maximal-lit* (*Pos* A) C
    by (*simp add*: *Uniq-def*)
  hence *Uniq-production*: $\exists_{\leq 1} A. \exists C'.$
    $C \in N \wedge$
    $C =$ *add-mset* (*Pos* A) $C' \wedge$
    select $C = \{\#\} \wedge$
    is-strictly-maximal-lit (*Pos* A) $C \wedge$
    $\neg (\bigcup D \in \{D \in N.\ D \prec_c C\}.$ *production* N $D) \models C$
    using *Uniq-antimono'*
    by (*smt* (*verit*) *Uniq-def Uniq-prodI case-prod-conv*)
  show *?thesis*
    unfolding *production.simps*[*of* N C]
    using *Collect-eq-if-Uniq*[*OF Uniq-production*]
    by (*smt* (*verit*, *best*) *Collect-cong Collect-empty-eq Uniq-def Uniq-production case-prod-conv*
      insertCI mem-Collect-eq)
qed

lemma *production-eq-singleton-if-atom-in-production*:
  assumes $A \in$ *production* N C
  shows *production* N $C = \{A\}$
  using *assms production-eq-empty-or-singleton*
  by *force*

definition *interp* **where**
  interp N $C \equiv (\bigcup D \in \{D \in N.\ D \prec_c C\}.$ *production* N $D)$

lemma *interp-mempty*[*simp*]: *interp* N $\{\#\} = \{\}$
proof −

have $\nexists C.\ C \prec_c \{\#\}$
by (*metis clause-order.order.asym subset-implies-multp subset-mset.gr-zeroI*)
hence $\{D \in N.\ D \prec_c \{\#\}\} = \{\}$
by *simp*
thus *?thesis*
unfolding *interp-def* **by** *auto*
qed

lemma *production-unfold*: *production* $N\ C = \{A \mid A\ C'.$
$C \in N \wedge$
$C =$ *add-mset* (*Pos* A) $C' \wedge$
select $C = \{\#\} \wedge$
is-strictly-maximal-lit (*Pos* A) $C \wedge$
$\neg$ *interp* $N\ C \models C\}$
by (*simp add*: *production.simps*[*of* $N\ C$] *interp-def*)

lemma *production-unfold'*: *production* $N\ C = \{A \mid A.$
$C \in N \wedge$
select $C = \{\#\} \wedge$
is-strictly-maximal-lit (*Pos* A) $C \wedge$
$\neg$ *interp* $N\ C \models C\}$
unfolding *production-unfold*
by (*metis* (*mono-tags*, *lifting*) *literal-order.explode-greatest-in-mset*)

lemma *mem-productionE*:
assumes *C-prod*: $A \in$ *production* $N\ C$
obtains C' **where**
$C \in N$ **and**
$C =$ *add-mset* (*Pos* A) C' **and**
select $C = \{\#\}$ **and**
is-strictly-maximal-lit (*Pos* A) C **and**
$\neg$ *interp* $N\ C \models C$
using *C-prod*
unfolding *production.simps*[*of* $N\ C$] *mem-Collect-eq Let-def interp-def*
by (*metis* (*no-types*, *lifting*))

lemma *production-subset-if-less-cls*: $C \prec_c D \Longrightarrow$ *production* $N\ C \subseteq$ *interp* $N\ D$
unfolding *interp-def*
using *production-unfold* **by** *blast*

lemma *Uniq-production-eq-singleton*: $\exists_{\leq 1}\ C.$ *production* $N\ C = \{A\}$
proof (*rule Uniq-I*)
fix $C\ D$
assume *production* $N\ C = \{A\}$
hence $A \in$ *production* $N\ C$
by *simp*
then obtain C' **where**
$C \in N$
$C =$ *add-mset* (*Pos* A) C'

is-strictly-maximal-lit (*Pos A*) *C*
¬ *interp N C* ⊨ *C*
by (*auto elim*!: *mem-productionE*)

assume *production N D* = {*A*}
hence *A* ∈ *production N D*
by *simp*
then obtain *D*′ **where**
D ∈ *N*
D = *add-mset* (*Pos A*) *D*′
is-strictly-maximal-lit (*Pos A*) *D*
¬ *interp N D* ⊨ *D*
by (*auto elim*!: *mem-productionE*)

have ¬ (*C* ≺$_c$ *D*)
proof (*rule notI*)
assume *C* ≺$_c$ *D*
hence *production N C* ⊆ *interp N D*
using *production-subset-if-less-cls* **by** *metis*
hence *A* ∈ *interp N D*
unfolding ‹*production N C* = {*A*}› **by** *simp*
hence *interp N D* ⊨ *D*
unfolding ‹*D* = *add-mset* (*Pos A*) *D*′› **by** *simp*
with ‹¬ *interp N D* ⊨ *D*› **show** *False*
by *metis*
qed

moreover have ¬ (*D* ≺$_c$ *C*)
proof (*rule notI*)
assume *D* ≺$_c$ *C*
hence *production N D* ⊆ *interp N C*
using *production-subset-if-less-cls* **by** *metis*
hence *A* ∈ *interp N C*
unfolding ‹*production N D* = {*A*}› **by** *simp*
hence *interp N C* ⊨ *C*
unfolding ‹*C* = *add-mset* (*Pos A*) *C*′› **by** *simp*
with ‹¬ *interp N C* ⊨ *C*› **show** *False*
by *metis*
qed

ultimately show *C* = *D*
by *order*
qed

lemma *singleton-eq-CollectD*: {*x*} = {*y*. *P y*} ⟹ *P x*
by *blast*

lemma *subset-Union-mem-CollectI*: *P x* ⟹ *f x* ⊆ (⋃ *y* ∈ {*z*. *P z*}. *f y*)
by *blast*

lemma *interp-subset-if-less-cls*: $C \prec_c D \Longrightarrow$ *interp* $N\ C \subseteq$ *interp* $N\ D$
by (*smt* (*verit*, *best*) *UN-iff mem-Collect-eq interp-def subsetI transpD transp-less-cls*)

lemma *interp-subset-if-less-cls'*: $C \prec_c D \Longrightarrow$ *interp* $N\ C \subseteq$ *interp* $N\ D \cup$ *production* $N\ D$
using *interp-subset-if-less-cls* **by** *blast*

lemma *split-Union-production*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.\ production\ N\ C) =$
interp $N\ D \cup$ *production* $N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ production\ N\ C)$
proof −
have $N = \{C \in N.\ C \prec_c D\} \cup \{D\} \cup \{C \in N.\ D \prec_c C\}$
proof (*rule partition-set-around-element*)
show *totalp-on* N $(\prec_c)$
using *totalp-less-cls totalp-on-subset* **by** *blast*
next
show $D \in N$
using *D-in* **by** *simp*
qed
hence $(\bigcup C \in N.\ production\ N\ C) =$
$(\bigcup C \in \{C \in N.\ C \prec_c D\}.\ production\ N\ C) \cup production\ N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ production\ N\ C)$
by *auto*
thus $(\bigcup C \in N.\ production\ N\ C) =$
interp $N\ D \cup$ *production* $N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ production\ N\ C)$
by (*simp add*: *interp-def*)
qed

lemma *split-Union-production'*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.\ production\ N\ C) =$ *interp* $N\ D \cup (\bigcup C \in \{C \in N.\ D \preceq_c C\}.\ production\ N\ C)$
using *split-Union-production*[*OF D-in*] *D-in* **by** *auto*

lemma *split-interp*:
assumes $C \in N$ **and** *D-in*: $D \in N$ **and** $D \prec_c C$
shows *interp* $N\ C =$ *interp* $N\ D \cup (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}.\ production\ N\ C')$
proof −
have $\{D \in N.\ D \prec_c C\} =$
$\{y \in \{D \in N.\ D \prec_c C\}.\ y \prec_c D\} \cup \{D\} \cup \{y \in \{D \in N.\ D \prec_c C\}.\ D \prec_c y\}$
proof (*rule partition-set-around-element*)
show *totalp-on* $\{D \in N.\ D \prec_c C\}$ $(\prec_c)$
using *totalp-less-cls totalp-on-subset* **by** *blast*
next
from *D-in* ‹$D \prec_c C$› **show** $D \in \{D \in N.\ D \prec_c C\}$

by *simp*
qed
also have $\ldots = \{x \in N.\ x \prec_c C \wedge x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
by *auto*
also have $\ldots = \{x \in N.\ x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
using ‹$D \prec_c C$› *transp-less-cls*
by (*metis* (*no-types*, *opaque-lifting*) *transpD*)
finally have *Collect-N-lt-C*: $\{x \in N.\ x \prec_c C\} = \{x \in N.\ x \prec_c D\} \cup \{x \in N.\ D \preceq_c x \wedge x \prec_c C\}$
by *auto*

have *interp N C* $= (\bigcup C' \in \{D \in N.\ D \prec_c C\}.\ production\ N\ C')$
by (*simp add*: *interp-def*)
also have $\ldots = (\bigcup C' \in \{x \in N.\ x \prec_c D\}.\ production\ N\ C') \cup (\bigcup C' \in \{x \in N.\ D \preceq_c x \wedge x \prec_c C\}.\ production\ N\ C')$
unfolding *Collect-N-lt-C* **by** *simp*
finally show *interp N C* $=$ *interp N D* $\cup \bigcup$ (*production N* ‘ $\{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}$)
unfolding *interp-def* **by** *simp*
qed

lemma *less-imp-Interp-subseteq-interp*: $C \prec_c D \Longrightarrow$ *interp N C* $\cup$ *production N C* $\subseteq$ *interp N D*
using *interp-subset-if-less-cls production-subset-if-less-cls*
by (*simp add*: *interp-def*)

lemma *not-interp-to-Interp-imp-le*: $A \notin$ *interp N C* $\Longrightarrow A \in$ *interp N D* $\cup$ *production N D* $\Longrightarrow C \preceq_c D$
using *less-imp-Interp-subseteq-interp*
by (*metis* (*mono-tags*, *opaque-lifting*) *subsetD sup2CI totalpD totalp-less-cls*)

lemma *produces-imp-in-interp*:
assumes *Neg A* $\in\#$ *C* **and** *D-prod*: $A \in$ *production N D*
shows $A \in$ *interp N C*
proof −
from *D-prod* **have** *Pos A* $\in\#$ *D* **and** *is-strictly-maximal-lit* (*Pos A*) *D*
by (*auto elim*: *mem-productionE*)

have $D \prec_c C$
proof (*rule multp-if-maximal-of-lhs-is-less*)
show *Pos A* $\in\#$ *D*
using ‹*Pos A* $\in\#$ *D*› **.**
next
show *Neg A* $\in\#$ *C*
using ‹*Neg A* $\in\#$ *C*› **.**
next
show *Pos A* $\prec_l$ *Neg A*
by (*simp add*: *less-lit-def subset-implies-multp*)

```
  next
    show is-maximal-lit (Pos A) D
      using ‹is-strictly-maximal-lit (Pos A) D› by auto
  qed simp-all
  hence ¬ (≺c)== C D
    by (metis asympD asymp-less-cls reflclp-iff)
  thus ?thesis
  proof (rule contrapos-np)
    from D-prod show  A ∉ interp N C ⟹ (≺c)== C D
      using not-interp-to-Interp-imp-le by simp
  qed
qed

lemma neg-notin-Interp-not-produce:
  Neg A ∈# C ⟹ A ∉ interp N D ∪ production N D ⟹ C ⪯c D ⟹ A ∉
production N D''
  using interp-subset-if-less-cls'
  by (metis Un-iff produces-imp-in-interp reflclp-iff sup.orderE)

lemma lift-interp-entails:
  assumes
    D-in: D ∈ N and
    D-entailed: interp N D ⊨ D and
    C-in: C ∈ N and
    D-lt-C: D ≺c C
  shows interp N C ⊨ D
proof −
  from D-entailed obtain L A where
    L-in: L ∈# D and
    L-eq-disj-L-eq: L = Pos A ∧ A ∈ interp N D ∨ L = Neg A ∧ A ∉ interp N D
    unfolding true-cls-def true-lit-iff by metis

  have interp N D ⊆ interp N C
    using interp-subset-if-less-cls[OF D-lt-C] .

  from L-eq-disj-L-eq show interp N C ⊨ D
  proof (elim disjE conjE)
    assume L = Pos A and A ∈ interp N D
    thus interp N C ⊨ D
      using L-in ‹interp N D ⊆ interp N C› by auto
  next
    assume L = Neg A and A ∉ interp N D
    hence A ∉ interp N C
      using neg-notin-Interp-not-produce
      by (smt (verit, ccfv-threshold) L-in UN-E interp-def produces-imp-in-interp)
    thus interp N C ⊨ D
      using L-in ‹L = Neg A› by blast
  qed
qed
```

lemma *lift-interp-entails-to-interp-production-entails*:
assumes
C-in: $C \in N$ **and**
D-in: $D \in N$ **and**
C-lt-D: $D \prec_c C$ **and**
D-entailed: *interp* $N\ C \models D$
shows *interp* $N\ C \cup$ *production* $N\ C \models D$
proof −
from *D-entailed* **obtain** $L\ A$ **where**
L-in: $L \in\# D$ **and**
L-eq-disj-L-eq: $L = Pos\ A \wedge A \in$ *interp* $N\ C \vee L = Neg\ A \wedge A \notin$ *interp* $N\ C$
unfolding *true-cls-def true-lit-iff* **by** *metis*

from *L-eq-disj-L-eq* **show** *interp* $N\ C \cup$ *production* $N\ C \models D$
proof (*elim disjE conjE*)
assume $L = Pos\ A$ **and** $A \in$ *interp* $N\ C$
thus *interp* $N\ C \cup$ *production* $N\ C \models D$
using *L-in* **by** *blast*
next
assume $L = Neg\ A$ **and** $A \notin$ *interp* $N\ C$
hence $A \notin$ *interp* $N\ C \cup$ *production* $N\ C$
using *neg-notin-Interp-not-produce*
by (*metis* (*no-types*, *opaque-lifting*) *C-lt-D L-in Un-iff interp-subset-if-less-cls*
produces-imp-in-interp subsetD)
thus *interp* $N\ C \cup$ *production* $N\ C \models D$
using *L-in* ‹$L = Neg\ A$› **by** *blast*
qed
qed

lemma *lift-entailment-to-Union*:
fixes $N\ D$
assumes
D-in: $D \in N$ **and**
R_D*-entails-D*: *interp* $N\ D \models D$
shows
$(\bigcup C \in N.$ *production* $N\ C) \models D$
using *lift-interp-entails*
by (*smt* (*verit*, *best*) *D-in* R_D*-entails-D UN-iff produces-imp-in-interp split-Union-production′*
subsetD sup-ge1 true-cls-def true-lit-iff)

lemma
assumes
$D \preceq_c C$ **and**
C-prod: $A \in$ *production* $N\ C$ **and**
L-in: $L \in\# D$
shows
lesseq-trm-if-pos: *is-pos* $L \Longrightarrow$ *atm-of* $L \preceq_t A$ **and**

```
    less-trm-if-neg: is-neg L ⟹ atm-of L ≺t A
proof −
  from C-prod obtain C′ where
    C-def: C = add-mset (Pos A) C′ and
    C-max-lit: is-strictly-maximal-lit (Pos A) C
    by (auto elim: mem-productionE)

  have Pos A ≺l L if is-pos L and ¬ atm-of L ⪯t A
  proof −
    from that(2) have A ≺t atm-of L
      using totalp-less-trm[THEN totalpD] by auto
    hence multp (≺t) {#A#} {#atm-of L#}
    by (smt (verit, del-insts) add.right-neutral empty-iff insert-iff one-step-implies-multp
      set-mset-add-mset-insert set-mset-empty transpD transp-less-trm union-mset-add-mset-right)
    with that(1) show Pos A ≺l L
      by (cases L) (simp-all add: less-lit-def)
  qed

  moreover have Pos A ≺l L if is-neg L and ¬ atm-of L ≺t A
  proof −
    from that(2) have A ⪯t atm-of L
      using totalp-less-trm[THEN totalpD] by auto
    hence multp (≺t) {#A#} {#atm-of L, atm-of L#}
    by (smt (z3) add-mset-add-single add-mset-remove-trivial add-mset-remove-trivial-iff
      empty-not-add-mset insert-DiffM insert-noteq-member one-step-implies-multp
reflclp-iff
      transp-def transp-less-trm union-mset-add-mset-left union-mset-add-mset-right)
    with that(1) show Pos A ≺l L
      by (cases L) (simp-all add: less-lit-def)
  qed

  moreover have False if Pos A ≺l L
  proof −
    have C ≺c D
    proof (rule multp-if-maximal-of-lhs-is-less)
      show Pos A ∈# C
        by (simp add: C-def)
    next
      show L ∈# D
        using L-in by simp
    next
      show is-maximal-lit (Pos A) C
        using C-max-lit by auto
    next
      show Pos A ≺l L
        using that by simp
    qed simp-all
    with ‹D ⪯c C› show False
      by (metis asympD reflclp-iff wfp-imp-asymp wfP-less-cls)
```

```
  qed

  ultimately show is-pos L ⟹ atm-of L ≼t A and is-neg L ⟹ atm-of L ≺t A
    by metis+
qed

lemma less-trm-iff-less-cls-if-mem-production:
  assumes C-prod: AC ∈ production N C and D-prod: AD ∈ production N D
  shows AC ≺t AD ⟷ C ≺c D
proof −
  from C-prod obtain C′ where
    C ∈ N and
    C-def: C = add-mset (Pos AC) C′ and
    is-strictly-maximal-lit (Pos AC) C
    by (elim mem-productionE) simp
  hence ∀ L ∈# C′. L ≺l Pos AC
    by (simp add: literal-order.is-greatest-in-mset-iff)

  from D-prod obtain D′ where
    D ∈ N and
    D-def: D = add-mset (Pos AD) D′ and
    is-strictly-maximal-lit (Pos AD) D
    by (elim mem-productionE) simp
  hence ∀ L ∈# D′. L ≺l Pos AD
    by (simp add: literal-order.is-greatest-in-mset-iff)

  show ?thesis
  proof (rule iffI)
    assume AC ≺t AD
    hence Pos AC ≺l Pos AD
      by (simp add: less-lit-def)
    moreover hence ∀ L ∈# C′. L ≺l Pos AD
      using ‹∀ L ∈# C′. L ≺l Pos AC›
      by (meson transp-less-lit transpD)
    ultimately show C ≺c D
      using one-step-implies-multp[of D C - {#}]
      by (simp add: D-def C-def)
  next
    assume C ≺c D
    hence production N C ⊆ (⋃ (production N ‘ {x ∈ N. x ≺c D}))
      using ‹C ∈ N› by auto
    hence AC ∈ (⋃ (production N ‘ {x ∈ N. x ≺c D}))
      using C-prod by auto
    hence AC ≠ AD
      by (metis D-prod interp-def mem-productionE true-cls-add-mset true-lit-iff)
    moreover have ¬ (AD ≺t AC)
      by (metis C-def D-prod ‹C ≺c D› asympD asymp-less-trm lesseq-trm-if-pos
literal.disc(1)
          literal.sel(1) reflclp-iff union-single-eq-member)
```

```
    ultimately show A_C ≺t A_D
      using totalp-less-trm[THEN totalpD]
      using C-def D-def by auto
  qed
qed

lemma false-cls-if-productive-production:
  assumes C-prod: A ∈ production N C and D ∈ N and C ≺c D
  shows ¬ interp N D ⊨ C − {#Pos A#}
proof −
  from C-prod obtain C' where
    C-in: C ∈ N and
    C-def: C = add-mset (Pos A) C' and
    select C = {#} and
    Pox-A-max: is-strictly-maximal-lit (Pos A) C and
    ¬ interp N C ⊨ C
    by (rule mem-productionE) blast

  from ‹D ∈ N› ‹C ≺c D› have A ∈ interp N D
    using C-prod production-subset-if-less-cls by auto

  from ‹D ∈ N› have interp N D ⊆ (⋃ D ∈ N. production N D)
    by (auto simp: interp-def)

  have ¬ interp N D ⊨ C'
    unfolding true-cls-def Set.bex-simps
  proof (intro ballI)
    fix L assume L-in: L ∈# C'
    hence L ∈# C
      by (simp add: C-def)

    have C' ≺c C
      by (simp add: C-def subset-implies-multp)
    hence C' ⪯c C
      by order

    show ¬ interp N D ⊨l L
    proof (cases L)
      case (Pos A_L)
      moreover have A_L ∉ interp N D
      proof −
        have ∀ y∈#C'. y ≺l Pos A
          using Pox-A-max
          by (simp add: C-def literal-order.is-greatest-in-mset-iff)
        with Pos have A_L ∉ insert A (interp N C)
          using L-in ‹¬ interp N C ⊨ C› C-def
          by blast

        moreover have A_L ∉ (⋃ D' ∈ {D' ∈ N. C ≺c D' ∧ D' ≺c D}. production
```

$N\ D'$)
proof −
have $A_L \preceq_t A$
using *Pos lesseq-trm-if-pos*[*OF* ‹$C' \preceq_c C$› *C-prod* ‹$L \in\# C'$›]
by *simp*
thus *?thesis*
using *less-trm-iff-less-cls-if-mem-production*
using *C-prod calculation interp-def* **by** *fastforce*
qed

moreover have *interp* $N\ D =$
insert A (*interp* $N\ C$) ∪ ($\bigcup D' \in \{D' \in N.\ C \prec_c D' \wedge D' \prec_c D\}$. *production* $N\ D'$)
proof −
have *interp* $N\ D = (\bigcup D' \in \{D' \in N.\ D' \prec_c D\}.$ *production* $N\ D'$)
by (*simp only*: *interp-def*)
also have $\ldots = (\bigcup D' \in \{D' \in \{y \in N.\ y \prec_c C\} \cup \{C\} \cup \{y \in N.\ C \prec_c y\}.\ D' \prec_c D\}.$ *production* $N\ D'$)
using *partition-set-around-element*[*of* N ($\prec_c$), *OF* - ‹$C \in N$›]
totalp-on-subset[*OF totalp-less-cls*, *simplified*]
by *simp*
also have $\ldots = (\bigcup D' \in \{y \in N.\ y \prec_c C \wedge y \prec_c D\} \cup \{C\} \cup \{y \in N.\ C \prec_c y \wedge y \prec_c D\}.$ *production* $N\ D'$)
using ‹$C \prec_c D$› **by** *auto*
also have $\ldots = (\bigcup D' \in \{y \in N.\ y \prec_c C\} \cup \{C\} \cup \{y \in N.\ C \prec_c y \wedge y \prec_c D\}.$ *production* $N\ D'$)
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*3*) *transpD transp-less-cls*)
also have $\ldots =$ *interp* $N\ C \cup$ *production* $N\ C \cup (\bigcup D' \in \{y \in N.\ C \prec_c y \wedge y \prec_c D\}.$ *production* $N\ D'$)
by (*auto simp*: *interp-def*)
finally show *?thesis*
using *C-prod*
by (*metis* (*no-types*, *lifting*) *empty-iff insertE insert-is-Un*
production-eq-empty-or-singleton sup-commute)
qed

ultimately show *?thesis*
by *simp*
qed
ultimately show *?thesis*
by *simp*
next
case (*Neg* A_L)
moreover have $A_L \in$ *interp* $N\ D$
using *Neg* ‹$L \in\# C$› ‹$C \prec_c D$› ‹$\neg$ *interp* $N\ C \models C$› *interp-subset-if-less-cls*
by *blast*
ultimately show *?thesis*
by *simp*
qed

```
  qed
  thus ¬ interp N D ⊨ C − {#Pos A#}
    by (simp add: C-def)
qed

lemma production-subset-Union-production:
  ⋀C N. C ∈ N ⟹ production N C ⊆ (⋃ D ∈ N. production N D)
  by auto

lemma interp-subset-Union-production:
  ⋀C N. C ∈ N ⟹ interp N C ⊆ (⋃ D ∈ N. production N D)
  by (simp add: split-Union-production′)

lemma model-construction:
  fixes
    N :: 'f gterm atom clause set and
    C :: 'f gterm atom clause
  assumes G.saturated N and {#} ∉ N and C-in: C ∈ N
  shows
    production N C = {} ⟷ interp N C ⊨ C
    (⋃ D ∈ N. production N D) ⊨ C
    D ∈ N ⟹ C ≺c D ⟹ interp N D ⊨ C
  unfolding atomize-conj atomize-imp
  using wfP-less-cls C-in
proof (induction C arbitrary: D rule: wfp-induct-rule)
  case (less C)
  note IH = less.IH

  from ‹{#} ∉ N› ‹C ∈ N› have C ≠ {#}
    by metis

  define I :: 'f gterm set where
    I = interp N C

  have i: interp N C ⊨ C ⟷ (production N C = {})
  proof (rule iffI)
    show interp N C ⊨ C ⟹ production N C = {}
      by (smt (z3) Collect-empty-eq interp-def production.elims)
  next
    assume production N C = {}
    show interp N C ⊨ C
     proof (cases ∃ A. Neg A ∈# C ∧ (Neg A ∈# select C ∨ select C = {#} ∧
is-maximal-lit (Neg A) C))
      case ex-neg-lit-sel-or-max: True
      then obtain A where
        Neg A ∈# C and
        sel-or-max: Neg A ∈# select C ∨ select C = {#} ∧ is-maximal-lit (Neg A)
C
        by metis
```

then obtain C' **where**
C-def: $C = \mathit{add\text{-}mset}\ (\mathit{Neg}\ A)\ C'$
by (*metis mset-add*)

show *?thesis*
proof (*cases* $A \in \mathit{interp}\ N\ C$)
case *True*
then obtain D **where**
$A \in \mathit{production}\ N\ D$ **and** $D \in N$ **and** $D \prec_c C$
unfolding *interp-def* **by** *auto*
then obtain D' **where**
D-def: $D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$ **and**
sel-D: $\mathit{select}\ D = \{\#\}$ **and**
max-t-t': *is-strictly-maximal-lit* $(\mathit{Pos}\ A)\ D$ **and**
$\neg\ \mathit{interp}\ N\ D \models D$
by (*elim mem-productionE*) *fast*

have *reso*: *ground-resolution* $C\ D\ (C' + D')$
proof (*rule ground-resolutionI*)
show $C = \mathit{add\text{-}mset}\ (\mathit{Neg}\ A)\ C'$
by (*simp add*: *C-def*)
next
show $D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$
by (*simp add*: *D-def*)
next
show $D \prec_c C$
using ‹$D \prec_c C$› **.**
next
show $\mathit{select}\ C = \{\#\} \wedge \mathit{is\text{-}maximal\text{-}lit}\ (\mathit{Neg}\ A)\ C \vee \mathit{Neg}\ A \in\# \mathit{select}\ C$
using *sel-or-max* **by** *auto*
next
show $\mathit{select}\ D = \{\#\}$
using *sel-D* **by** *blast*
next
show *is-strictly-maximal-lit* $(\mathit{Pos}\ A)\ D$
using *max-t-t'* **.**
qed *simp-all*

define ι :: *'f gterm clause inference* **where**
$\iota = \mathit{Infer}\ [D,\ C]\ (C' + D')$

have $\iota \in$ *G-Inf*
using *reso*
by (*auto simp only*: ι*-def G-Inf-def*)

moreover have $\bigwedge t.\ t \in \mathit{set}\ (\mathit{prems\text{-}of}\ \iota) \Longrightarrow t \in N$
using ‹$C \in N$› ‹$D \in N$›
by (*auto simp add*: ι*-def*)

ultimately have $\iota \in G.Inf\text{-}from\ N$
by (*auto simp*: *G.Inf-from-def*)
hence $\iota \in G.Red\text{-}I\ N$
using ‹*G.saturated N*›
by (*auto simp*: *G.saturated-def*)
then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-CD: *G-entails* (*insert D DD*) $\{C' + D'\}$ **and**
ball-DD-lt-C: $\forall D{\in}DD.\ D \prec_c C$
unfolding *G.Red-I-def G.redundant-infer-def mem-Collect-eq*
by (*auto simp*: *ι-def*)

moreover have $\forall D{\in}$ *insert D DD*. *interp N C* $\models D$
using *IH*[*THEN conjunct2*, *THEN conjunct2*, *rule-format*, *of - C*]
using ‹$C \in N$› ‹$D \in N$› ‹$D \prec_c C$› *DD-subset ball-DD-lt-C*
by (*metis in-mono insert-iff*)

ultimately have *interp N C* $\models C' + D'$
using *DD-entails-CD*
unfolding *G-entails-def*
by (*simp add*: *I-def true-clss-def*)

moreover have $\neg$ *interp N D* $\models D'$
using ‹$\neg$ *interp N D* $\models D$›
using *D-def* **by** *force*

moreover have $D' \prec_c D$
unfolding *D-def*
by (*simp add*: *subset-implies-multp*)

moreover have $\neg$ *interp N C* $\models D'$
using *false-cls-if-productive-production*[*OF* - ‹$C \in N$› ‹$D \prec_c C$›]
by (*metis D-def* ‹$A \in$ *production N D*› *remove1-mset-add-mset-If*)

ultimately show *interp N C* $\models C$
unfolding *C-def* **by** *fast*
next
case *False*
thus *?thesis*
using ‹*Neg A* $\in\#$ *C*›
by (*auto simp add*: *true-cls-def*)
qed
next
case *False*
hence *select C* $= \{\#\}$
using *select-subset select-negative-lits*
by (*metis* (*no-types*, *opaque-lifting*) *Neg-atm-of-iff mset-subset-eqD multi-set-nonemptyE*)

```
      from False obtain A where Pos-A-in: Pos A ∈# C and max-Pos-A:
is-maximal-lit (Pos A) C
      using ex-maximal-in-mset-wrt[OF transp-on-less-lit asymp-on-less-lit ‹C ≠
{#}›]
      using is-maximal-in-mset-wrt-iff[OF transp-on-less-lit asymp-on-less-lit]
      by (metis Neg-atm-of-iff ‹select C = {#}› is-pos-def)
    then obtain C' where C-def: C = add-mset (Pos A) C'
      by (meson mset-add)

    show ?thesis
    proof (cases interp N C ⊨ C')
      case True
      then show ?thesis
        using C-def by force
    next
      case False
      show ?thesis
      proof (cases is-strictly-maximal-lit (Pos A) C)
        case strictly-maximal: True
        then show ?thesis
          using ‹production N C = {}› ‹select C = {#}› less.prems
          unfolding production-unfold[of N C] Collect-empty-eq
          using C-def by blast
      next
        case False
        hence count C (Pos A) ≥ 2
          using max-Pos-A
       by (simp add: literal-order.count-ge-2-if-maximal-in-mset-and-not-greatest-in-mset)
       then obtain C' where C-def: C = add-mset (Pos A) (add-mset (Pos A)
C')
          by (metis two-le-countE)

        define ι :: 'f gterm clause inference where
          ι = Infer [C] (add-mset (Pos A) C')

        have eq-fact: ground-factoring C (add-mset (Pos A) C')
        proof (rule ground-factoringI)
          show C = add-mset (Pos A) (add-mset (Pos A) C')
            by (simp add: C-def)
        next
          show select C = {#}
            using ‹select C = {#}› .
        next
          show is-maximal-lit (Pos A) C
            using max-Pos-A .
        qed simp-all
        hence ι ∈ G-Inf
          by (auto simp: ι-def G-Inf-def)
```

```
      moreover have ⋀t. t ∈ set (prems-of ι) ⟹ t ∈ N
        using ‹C ∈ N›
        by (auto simp add: ι-def)

      ultimately have ι ∈ G.Inf-from N
        by (auto simp: G.Inf-from-def)
      hence ι ∈ G.Red-I N
        using ‹G.saturated N›
        by (auto simp: G.saturated-def)
      then obtain DD where
        DD-subset: DD ⊆ N and
        finite DD and
        DD-entails-concl: G-entails DD {add-mset (Pos A) C'} and
        ball-DD-lt-C: ∀ D∈DD. D ≺c C
        unfolding G.Red-I-def G.redundant-infer-def mem-Collect-eq
        by (auto simp: ι-def)

      moreover have ∀ D∈DD. interp N C ⊨ D
        using IH[THEN conjunct2, THEN conjunct2, rule-format, of - C]
        using ‹C ∈ N› DD-subset ball-DD-lt-C
        by blast

      ultimately have interp N C ⊨ add-mset (Pos A) C'
        using DD-entails-concl
        unfolding G-entails-def
        by (simp add: I-def true-clss-def)
      then show ?thesis
        by (simp add: C-def joinI-right pair-imageI)
    qed
  qed
 qed
qed

moreover have iia: (⋃ (production N ' N)) ⊨ C
 using production-eq-empty-or-singleton[of N C]
proof (elim disjE exE)
 assume production N C = {}
 hence interp N C ⊨ C
  by (simp only: i)
 thus ?thesis
  using lift-entailment-to-Union[OF ‹C ∈ N›] by argo
next
 fix A
 assume production N C = {A}
 hence prod: A ∈ production N C
  by simp

 from prod have Pos A ∈# C
```

unfolding *production.simps*[*of N C*] *mem-Collect-eq*
by *force*

moreover from *prod* **have** $A \in \bigcup (\mathit{production}\ N\ `\ N)$
using ‹$C \in N$› *production-subset-Union-production*
by *fast*

ultimately show *?thesis*
by *blast*
qed

moreover have *iib*: *interp N D* $\models$ *C* **if** $D \in N$ **and** $C \prec_c D$
using *production-eq-empty-or-singleton*[*of N C*, *folded*]
proof (*elim disjE exE*)
assume *production N C* = {}
hence *interp N C* $\models$ *C*
unfolding *i* **by** *simp*
thus *?thesis*
using *lift-interp-entails*[*OF* ‹$C \in N$› *- that*] **by** *argo*
next
fix *A* **assume** *production N C* = {*A*}
thus *?thesis*
by (*metis Un-insert-right insertCI insert-subset less-imp-Interp-subseteq-interp mem-productionE pos-literal-in-imp-true-cls that*(*2*) *union-single-eq-member*)
qed

ultimately show *?case*
by *argo*
qed

lemma
assumes *clause-order.is-least-in-fset N C* **and** *A* ∈ *production* (*fset N*) *C*
shows $\bigwedge A'.\ A' \prec_t A \Longrightarrow A' \notin (\bigcup D \in \mathit{fset}\ N.\ \mathit{production}\ (\mathit{fset}\ N)\ D)$
using *assms less-trm-iff-less-cls-if-mem-production clause-order.is-least-in-fset-iff*
by *auto*

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-productive*:
assumes *A* ∈ *production* (*fset N*) *C*
shows $\bigwedge A'.\ A' \prec_t A \Longrightarrow A' \notin \mathit{interp}\ (\mathit{fset}\ N)\ C \Longrightarrow A' \notin (\bigcup D \in \mathit{fset}\ N.\ \mathit{production}\ (\mathit{fset}\ N)\ D)$
using *assms production-subset-if-less-cls less-trm-iff-less-cls-if-mem-production*
by *fastforce*

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-not-productive*:
assumes *literal-order.is-maximal-in-mset C L* **and** *production* (*fset N*) *C* = {}
shows $\bigwedge A'.\ A' \prec_t \mathit{atm\text{-}of}\ L \Longrightarrow A' \notin \mathit{interp}\ (\mathit{fset}\ N)\ C \Longrightarrow A' \notin (\bigcup D \in \mathit{fset}\ N.\ \mathit{production}\ (\mathit{fset}\ N)\ D)$
using *assms*
by (*metis* (*no-types*, *lifting*) *UN-E UnCI less-trm-if-neg lesseq-trm-if-pos*

literal-order.is-maximal-in-mset-iff term-order.less-imp-not-less term-order.not-le not-interp-to-Interp-imp-le)

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp*:
fixes *A*
assumes
L-max: *literal-order.is-maximal-in-mset C L* **and**
A-less: $A \prec_t$ *atm-of L* **and**
A-no-in: $A \notin$ *interp* (*fset N*) *C*
shows $A \notin (\bigcup D \in$ *fset N*. *production* (*fset N*) *D*)
proof (*cases production* (*fset N*) *C* = {})
case *True*
thus *?thesis*
using *L-max A-less A-no-in*
using *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-not-productive*
by *metis*
next
case *False*
then obtain A' **where** *C-produces-A'*: $A' \in$ *production* (*fset N*) *C*
by *auto*
hence *is-strictly-maximal-lit* (*Pos A'*) *C*
unfolding *production-unfold mem-Collect-eq* **by** *metis*
hence *literal-order.is-maximal-in-mset C* (*Pos A'*)
using *literal-order.is-maximal-in-mset-if-is-greatest-in-mset* **by** *metis*
hence *L* = *Pos A'*
using *L-max*
by (*metis Uniq-D literal-order.Uniq-is-maximal-in-mset*)
hence *atm-of L* ∈ *production* (*fset N*) *C*
using *C-produces-A'* **by** *simp*
thus *?thesis*
using *L-max A-less A-no-in*
using *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-productive*
by *metis*
qed

lemma *lesser-atoms-in-previous-interp-are-in-final-interp*:
fixes *A*
assumes
L-max: *literal-order.is-maximal-in-mset C L* **and**
A-less: $A \prec_t$ *atm-of L* **and**
A-in: $A \in$ *interp N C*
shows $A \in (\bigcup D \in N$. *production N D*)
using *A-in interp-def* **by** *fastforce*

lemma *interp-fixed-for-smaller-literals*:
fixes *A*
assumes
L-max: *literal-order.is-maximal-in-mset C L* **and**
A-less: $A \prec_t$ *atm-of L* **and**

$C \prec_c D$
shows $A \in \mathit{interp}\ N\ C \longleftrightarrow A \in \mathit{interp}\ N\ D$
proof (*rule iffI*)
show $A \in \mathit{interp}\ N\ C \Longrightarrow A \in \mathit{interp}\ N\ D$
using *assms*(*3*) *interp-subset-if-less-cls* **by** *auto*
next
assume $A \in \mathit{interp}\ N\ D$

then obtain E **where** $A \in \mathit{production}\ N\ E$ **and** $E \in N$ **and** $E \prec_c D$
unfolding *interp-def* **by** *auto*

hence *literal-order.is-greatest-in-mset* E (*Pos* A)
by (*auto elim*: *mem-productionE*)

moreover have *Pos* $A \prec_l L$
by (*metis A-less Neg-atm-of-iff less-lit-simps*(*1*) *less-lit-simps*(*2*)
literal-order.dual-order.strict-trans term-order.order-eq-iff literal.collapse(*1*))

ultimately have $E \prec_c C$
using *L-max*
using *literal-order.multp*$_{HO}$*-if-maximal-less-that-maximal multp*$_{DM}$*-imp-multp multp*$_{HO}$*-imp-multp*$_{DM}$
by *blast*

hence *production* $N\ E \subseteq \mathit{interp}\ N\ C$
by (*simp add*: *production-subset-if-less-cls*)

thus $A \in \mathit{interp}\ N\ C$
using ‹$A \in \mathit{production}\ N\ E$› **by** *auto*
qed

lemma *neg-lits-not-in-model-stay-out-of-model*:
assumes
L-in: $L \in\# C$ **and**
L-neg: *is-neg* L **and**
atm-L-not-in: *atm-of* $L \notin \mathit{interp}\ N\ C$
shows *atm-of* $L \notin (\bigcup D \in N.\ \mathit{production}\ N\ D)$
using *assms produces-imp-in-interp* **by** *force*

lemma *neg-lits-already-in-model-stay-in-model*:
assumes
L-in: $L \in\# C$ **and**
L-neg: *is-neg* L **and**
atm-L-not-in: *atm-of* $L \in \mathit{interp}\ N\ C$
shows *atm-of* $L \in (\bigcup D \in N.\ \mathit{production}\ N\ D)$
using *atm-L-not-in interp-def* **by** *auto*

lemma *image-eq-imageI*:
assumes $\bigwedge x.\ x \in X \Longrightarrow f\ x = g\ x$

shows $f \mathbin{`} X = g \mathbin{`} X$
using *assms* **by** *auto*

lemma *production-swap-clause-set*:
assumes
agree: $\{D \in N1.\ D \preceq_c C\} = \{D \in N2.\ D \preceq_c C\}$
shows *production N1 C* = *production N2 C*
using *agree*
proof (*induction C rule*: *wfp-induct-rule*[*OF wfP-less-cls*])
case *hyps*: (*1 C*)
from *hyps* **have** *AAA*: $C \in N1 \longleftrightarrow C \in N2$
by *auto*

from *hyps* **have** $\{D \in N1.\ D \prec_c C\} = \{D \in N2.\ D \prec_c C\}$
by *blast*

have *production N1* $\mathbin{`} \{D \in N2.\ D \prec_c C\}$ = *production N2* $\mathbin{`} \{D \in N2.\ D \prec_c C\}$
proof (*rule image-eq-imageI*)
fix x **assume** $x \in \{D \in N2.\ D \prec_c C\}$
hence $x \in N2$ **and** $x \prec_c C$
by *simp-all*
moreover have $\{D \in N1.\ (\prec_c)^{==}\ D\ x\} = \{D \in N2.\ (\prec_c)^{==}\ D\ x\}$
using *hyps.prems* $\langle x \prec_c C \rangle$ *clause-order.order.strict-trans1* **by** *blast*
ultimately show *production N1 x* = *production N2 x*
using *hyps.IH*[*rule-format*, *of x*] **by** *metis*
qed
hence *BBB*: *interp N1 C* = *interp N2 C*
unfolding *interp-def* $\langle \{D \in N1.\ D \prec_c C\} = \{D \in N2.\ D \prec_c C\} \rangle$
by *argo*

show *?case*
unfolding *production-unfold*
unfolding *AAA BBB*
by *simp*
qed

lemma *interp-swap-clause-set*:
assumes *agree*: $\{D \in N1.\ D \prec_c C\} = \{D \in N2.\ D \prec_c C\}$
shows *interp N1 C* = *interp N2 C*
proof −
have *BBB*: *production N1* $\mathbin{`} \{D \in N2.\ D \prec_c C\}$ = *production N2* $\mathbin{`} \{D \in N2.\ D \prec_c C\}$
proof (*intro image-eq-imageI production-swap-clause-set*)
fix x
assume $x \in \{D \in N2.\ D \prec_c C\}$
thus $\{D \in N1.\ (\prec_c)^{==}\ D\ x\} = \{D \in N2.\ (\prec_c)^{==}\ D\ x\}$
using *agree clause-order.le-less-trans* **by** *blast*
qed

show *?thesis*
unfolding *interp-def*
unfolding *agree BBB*
by *argo*
qed

definition *interp'* **where**
interp' N ≡ (⋃ *C* ∈ *N*. *production N C*)

lemma *interp-eq-interp'*: *interp N D* = *interp'* {*C* ∈ *N*. *C* ≺*c D*}
proof −
have *interp N D* = *interp* {*C* ∈ *N*. *C* ≺*c D*} *D*
proof (*rule interp-swap-clause-set*)
show {*Da* ∈ *N*. *Da* ≺*c D*} = {*Da* ∈ {*C* ∈ *N*. *C* ≺*c D*}. *Da* ≺*c D*}
by *blast*
qed

also have ... = *interp'* {*C* ∈ *N*. *C* ≺*c D*}
unfolding *interp-def interp'-def* **by** *blast*

finally show *?thesis* **.**
qed

lemma *production-unfold''*: *production N C* = {*A* | *A*.
C ∈ *N* ∧ *select C* = {#} ∧
is-strictly-maximal-lit (*Pos A*) *C* ∧
¬ *interp'* {*B* ∈ *N*. *B* ≺*c C*} |= *C*}
unfolding *production-unfold interp-eq-interp'*
using *literal-order.explode-greatest-in-mset*
by *metis*

lemma *Interp-swap-clause-set*:
assumes *agree*: {*D* ∈ *N1*. *D* ⪯*c C*} = {*D* ∈ *N2*. *D* ⪯*c C*}
shows *interp N1 C* ∪ *production N1 C* = *interp N2 C* ∪ *production N2 C*
using *production-swap-clause-set*[*OF agree*]
using *interp-swap-clause-set*
using *agree*
by *blast*

lemma *production-insert-greater-clause*:
assumes *C* ≺*c D*
shows *production* (*insert D N*) *C* = *production N C*
proof (*rule production-swap-clause-set*)
show {*Da* ∈ *insert D N*. (≺*c*)== *Da C*} = {*D* ∈ *N*. (≺*c*)== *D C*}
using ‹*C* ≺*c D*› **by** *auto*
qed

lemma *interp-insert-greater-clause-strong*:
assumes *C* ⪯*c D*

shows *interp* (*insert D N*) *C* = *interp N C*
proof (*rule interp-swap-clause-set*)
show {*x* ∈ *insert D N*. *x* ≺$_c$ *C*} = {*x* ∈ *N*. *x* ≺$_c$ *C*}
using ‹*C* ⪯$_c$ *D*› **by** *auto*
qed

lemma *interp-insert-greater-clause*:
assumes *C* ≺$_c$ *D*
shows *interp* (*insert D N*) *C* = *interp N C*
proof (*rule interp-swap-clause-set*)
show {*x* ∈ *insert D N*. *x* ≺$_c$ *C*} = {*x* ∈ *N*. *x* ≺$_c$ *C*}
using ‹*C* ≺$_c$ *D*› **by** *auto*
qed

lemma *Interp-insert-greater-clause*:
assumes *C* ≺$_c$ *D*
shows *interp* (*insert D N*) *C* ∪ *production* (*insert D N*) *C* = *interp N C* ∪ *production N C*
proof (*rule Interp-swap-clause-set*)
show {*Da* ∈ *insert D N*. (≺$_c$)$^{==}$ *Da C*} = {*D* ∈ *N*. (≺$_c$)$^{==}$ *D C*}
using ‹*C* ≺$_c$ *D*› **by** *auto*
qed

lemma *production-add-irrelevant-clause-to-set0*:
assumes
fin: *finite N* **and**
D-irrelevant: *E* ∈ *N E* ⊂# *D set-mset D* = *set-mset E* **and**
no-select: *select E* = {#}
shows *production* (*insert D N*) *D* = {}
proof −
from *D-irrelevant* **have** *E* ≺$_c$ *D*
using *subset-implies-multp* **by** *metis*
hence *prod-E-subset*: *production* (*insert D N*) *E* ⊆ *interp* (*insert D N*) *D*
using *production-subset-if-less-cls* **by** *metis*

show *?thesis*
proof (*cases production* (*insert D N*) *E* = {})
case *True*
hence (∄ *A*. *is-strictly-maximal-lit* (*Pos A*) *E*) ∨ *interp* (*insert D N*) *E* ⊨ *E*
unfolding *production-unfold*
using *no-select*
by (*smt* (*verit*, *del-insts*) *Collect-empty-eq* ‹*E* ∈ *N*› *diff-single-eq-union insertI2*
literal-order.is-greatest-in-mset-iff)
hence (∄ *A*. *is-strictly-maximal-lit* (*Pos A*) *D*) ∨ *interp* (*insert D N*) *D* ⊨ *D*
proof (*elim disjE*)
assume ∄ *A*. *is-strictly-maximal-lit* (*Pos A*) *E*
hence ∄ *A*. *is-strictly-maximal-lit* (*Pos A*) *D*
proof (*rule contrapos-nn*)
show ∃ *A*. *is-strictly-maximal-lit* (*Pos A*) *D* ⟹ ∃ *A*. *is-strictly-maximal-lit*

```
(Pos A) E
        using ‹E ⊂# D› ‹set-mset D = set-mset E›
        unfolding literal-order.is-greatest-in-mset-iff
        by (metis (no-types, opaque-lifting) add-mset-remove-trivial-eq
            insert-union-subset-iff)
    qed
    thus ?thesis ..
  next
    assume interp (insert D N) E ⊨ E
    hence interp (insert D N) D ⊨ D
      using lift-interp-entails ‹E ∈ N› ‹E ≺c D›
      by (metis ‹set-mset D = set-mset E› insert-iff true-cls-def)
    thus ?thesis ..
  qed
  thus ?thesis
    unfolding production-unfold by auto
next
  case False
  hence interp (insert D N) D ⊨ D
    using prod-E-subset
  by (metis ‹set-mset D = set-mset E› mem-productionE production-eq-empty-or-singleton
      insertCI insert-subset literal-order.is-greatest-in-mset-iff
      pos-literal-in-imp-true-cls)
  thus ?thesis
    unfolding production-unfold by simp
qed
qed

lemma production-add-irrelevant-clause-to-set:
  assumes
    fin: finite N and
    C-in: C ∈ N and
    D-irrelevant: ∃E ∈ N. E ⊂# D ∧ set-mset D = set-mset E and
    no-select: ⋀C. select C = {#}
  shows production (insert D N) C = production N C
    using wfP-less-cls C-in
proof (induction C rule: wfp-induct-rule)
  case (less C)
  hence C-in-iff: C ∈ insert D N ⟷ C ∈ N
    by simp

  have interp-insert-eq: interp (insert D N) C = interp N C
  proof (cases D ≺c C)
    case True
    hence {x ∈ insert D N. x ≺c C} = insert D {x ∈ N. x ≺c C}
      by auto
    hence ⋃ (production (insert D N) ‘ {x ∈ insert D N. x ≺c C}) =
      production (insert D N) D ∪ ⋃ (production (insert D N) ‘ {D ∈ N. D ≺c
C})
```

```
    by simp
  also have ... = ⋃ (production (insert D N) ` {D ∈ N. D ≺c C})
    using production-add-irrelevant-clause-to-set0
    using D-irrelevant fin no-select by force
  also have ... = ⋃ (production N ` {D ∈ N. D ≺c C})
    using less.IH by simp
  finally show ?thesis
    unfolding interp-def .
 next
  case False
  then show ?thesis
    unfolding interp-def
   by (smt (verit, best) Collect-cong image-eq-imageI insert-iff less.IH mem-Collect-eq)
 qed

 show ?case
  unfolding production-unfold C-in-iff interp-insert-eq by argo
qed

lemma production-add-irrelevant-clauses-to-set0:
 assumes
  fin: finite N finite N' and
  D-in: D ∈ N' and
  irrelevant: ∀ D ∈ N'. ∃ E ∈ N. E ⊂# D ∧ set-mset D = set-mset E and
  no-select: ⋀C. select C = {#}
 shows production (N ∪ N') D = {}
 using ‹finite N'› D-in irrelevant
proof (induction N' rule: finite-induct)
 case empty
 hence False
  by simp
 thus ?case ..
next
 case (insert x F)
 then show ?case
  using production-add-irrelevant-clause-to-set0
  by (metis UnCI fin(1) finite-Un finite-insert production-add-irrelevant-clause-to-set
no-select)
qed

lemma production-add-irrelevant-clauses-to-set:
 assumes
  fin: finite N finite N' and
  C-in: C ∈ N and
  irrelevant: ∀ D ∈ N'. ∃ E ∈ N. E ⊂# D ∧ set-mset D = set-mset E and
  no-select: ⋀C. select C = {#}
 shows production (N ∪ N') C = production N C
 using ‹finite N'› irrelevant
proof (induction N' rule: finite-induct)
```

```
  case empty
  show ?case
    by simp
next
  case (insert x F)
  then show ?case
    using production-add-irrelevant-clause-to-set
    by (metis C-in UnI1 Un-insert-right fin(1) finite-Un insertCI no-select)
qed

lemma interp-add-irrelevant-clauses-to-set:
  assumes
    fin: finite N finite N' and
    C-in: C ∈ N and
    irrelevant: ∀ D ∈ N'. ∃ E ∈ N. E ⊂# D ∧ set-mset D = set-mset E and
    no-select: ⋀C. select C = {#}
  shows interp (N ∪ N') C = interp N C
proof −
  have interp (N ∪ N') C = ⋃ (production (N ∪ N') ‘ {D ∈ N ∪ N'. D ≺c C})
    unfolding interp-def ..
  also have ... = ⋃ (production (N ∪ N') ‘ {D ∈ N. D ≺c C} ∪
    production (N ∪ N') ‘ {D ∈ N'. D ≺c C})
    by auto
  also have ... = ⋃ (production (N ∪ N') ‘ {D ∈ N. D ≺c C})
   using production-add-irrelevant-clauses-to-set0[OF fin - irrelevant no-select] by
simp
  also have ... = ⋃ (production N ‘ {D ∈ N. D ≺c C})
    using production-add-irrelevant-clauses-to-set[OF fin - - no-select] irrelevant
    using subset-mset.less-le by fastforce
  also have ... = interp N C
    unfolding interp-def ..
  finally show ?thesis .
qed

lemma interp-add-irrelevant-clauses-to-set':
  assumes
    fin: finite N finite N' and
    C-in: C ∈ N and
    irrelevant: ∀ D ∈ N'. ∃ E ∈ N. E ⊆# D ∧ set-mset D = set-mset E and
    no-select: ⋀C. select C = {#}
  shows interp (N ∪ N') C = interp N C
proof −
  define N'' where
    N'' = N' − N

  from fin(2) have finite N''
    unfolding N''-def by simp

  moreover from irrelevant have ∀ D ∈ N''. ∃ E ∈ N. E ⊂# D ∧ set-mset E =
```

set-mset D
unfolding *N''-def*
by (*metis Diff-iff subset-mset.le-less*)

moreover have $N \cup N' = N \cup N''$
unfolding *N''-def* **by** *simp*

ultimately show *?thesis*
using *assms interp-add-irrelevant-clauses-to-set* **by** *metis*
qed

lemma *lesser-entailed-clause-stays-entailed'*:
assumes $C \preceq_c D$ **and** *D-lt*: $D \prec_c E$ **and** *C-entailed*: *interp N D* $\cup$ *production N D* $\models C$
shows *interp N E* $\models C$
proof −
from *C-entailed* **obtain** *L* **where** $L \in\# C$ **and** *interp N D* $\cup$ *production N D* $\models l$ *L*
by (*auto simp*: *true-cls-def*)

show *?thesis*
proof (*cases L*)
case (*Pos A*)
hence $A \in$ *interp N D* $\cup$ *production N D*
using ‹*interp N D* $\cup$ *production N D* $\models l$ *L*› **by** *simp*
moreover from *D-lt* **have** *interp N D* $\cup$ *production N D* $\subseteq$ *interp N E*
using *less-imp-Interp-subseteq-interp* **by** *blast*
ultimately have $A \in$ *interp N E*
by *auto*
thus *?thesis*
using *Pos* ‹$L \in\# C$› **by** *auto*
next
case (*Neg A*)
then show *?thesis*
using *neg-lits-not-in-model-stay-out-of-model*
by (*smt* (*verit*, *best*) *UN-E Un-iff* ‹$L \in\# C$› ‹*interp N D* $\cup$ *production N D* $\models l$ *L*› *assms*(*1*)
interp-def clause-order.antisym-conv neg-literal-notin-imp-true-cls
not-interp-to-Interp-imp-le produces-imp-in-interp true-lit-simps(*2*))
qed
qed

lemma *lesser-entailed-clause-stays-entailed*:
assumes *C-le*: $C \preceq_c D$ **and** *D-lt*: $D \prec_c E$ **and** *C-entailed*: *interp N D* $\cup$ *production N D* $\models C$
shows *interp N E* $\cup$ *production N E* $\models C$
proof −
from *C-entailed* **obtain** *L* **where** $L \in\# C$ **and** *interp N D* $\cup$ *production N D* $\models l$ *L*

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

show *?thesis*
proof (*cases L*)
case (*Pos A*)
hence *A* ∈ *interp N D* ∪ *production N D*
using ‹*interp N D* ∪ *production N D* ⊨*l L*› **by** *simp*
moreover from *D-lt* **have** *interp N D* ∪ *production N D* ⊆ *interp N E* ∪ *production N E*
using *less-imp-Interp-subseteq-interp* **by** *blast*
ultimately have *A* ∈ *interp N E* ∪ *production N E*
by *auto*
thus *?thesis*
using *Pos* ‹*L* ∈# *C*› **by** *auto*
next
case (*Neg A*)
then show *?thesis*
using *neg-lits-not-in-model-stay-out-of-model*
by (*smt* (*verit*, *best*) *UN-E Un-iff* ‹*L* ∈# *C*› ‹*interp N D* ∪ *production N D* ⊨*l L*› *assms*(*1*)
interp-def clause-order.antisym-conv neg-literal-notin-imp-true-cls
not-interp-to-Interp-imp-le produces-imp-in-interp true-lit-simps(*2*))
qed
qed

lemma *entailed-clause-stays-entailed′*:
assumes *C-lt*: *C* ≺*c* *D* **and** *C-entailed*: *interp N C* ∪ *production N C* ⊨ *C*
shows *interp N D* ⊨ *C*
using *lesser-entailed-clause-stays-entailed′*[*OF clause-order.order-refl assms*] **.**

lemma *entailed-clause-stays-entailed*:
assumes *C-lt*: *C* ≺*c* *D* **and** *C-entailed*: *interp N C* ∪ *production N C* ⊨ *C*
shows *interp N D* ∪ *production N D* ⊨ *C*
using *lesser-entailed-clause-stays-entailed*[*OF clause-order.order-refl assms*] **.**

lemma *multp-if-all-left-smaller*: *M2* ≠ {#} ⟹ ∀ *k*∈#*M1*. ∃ *j*∈#*M2*. *R k j* ⟹ *multp R M1 M2*
using *one-step-implies-multp*
by (*metis add-0*)

lemma
fixes
P1 :: *'f gterm* **and**
C1 :: *'f gterm clause* **and**
N :: *'f gterm clause set*
defines
C1 ≡ {#*Neg P1*#} **and**
N ≡ {*C1*}
assumes

no-select: $\bigwedge C.$ *select* $C = \{\#\}$
shows
False
proof −
have *interp N C1* = {}
unfolding *interp-def N-def* **by** *simp*
have *production N C1* = {}
unfolding *production-unfold* ‹*interp N C1* = {}› *C1-def* **by** *simp*
hence *interp N C1* ∪ *production N C1* ⊨ *C1*
unfolding ‹*interp N C1* = {}› ‹*production N C1* = {}›
by (*simp add*: *C1-def*)
oops

lemma
fixes
P1 P2 :: *'f gterm* **and**
C1 :: *'f gterm clause* **and**
N :: *'f gterm clause set*
defines
C1 ≡ {#*Pos P1*, *Neg P2*#} **and**
N ≡ {*C1*}
assumes
term-order: *P1* $\prec_t$ *P2* **and**
no-select: $\bigwedge C.$ *select* $C = \{\#\}$
shows *False*
proof −
have *lit-order*: *Pos P1* $\prec_l$ *Neg P1 Neg P1* $\prec_l$ *Pos P2 Pos P2* $\prec_l$ *Neg P2*
using *term-order* **by** *simp-all*
have *interp N C1* = {}
unfolding *interp-def N-def* **by** *simp*
have *production N C1* = {}
unfolding *production-unfold*
using *C1-def* ‹*interp N C1* = {}› **by** *simp*
hence *interp N C1* ∪ *production N C1* ⊨ *C1*
unfolding ‹*interp N C1* = {}› ‹*production N C1* = {}›
by (*simp add*: *C1-def*)
oops

lemma
fixes
P1 P2 P3 P4 :: *'f gterm* **and**
C1 C2 C3 C4 C5 :: *'f gterm clause* **and**
N :: *'f gterm clause set*
defines
C1 ≡ {#*Neg P1*, *Neg P2*#} **and**
C2 ≡ {#*Pos P2*, *Neg P3*#} **and**

```
    C3 ≡ {#Pos P1, Pos P2, Pos P4#} and
    C4 ≡ {#Pos P2, Pos P3, Pos P4#} and
    C5 ≡ {#Pos P2, Neg P4#} and
    N ≡ {C1, C2, C3, C4, C5}
  assumes
    term-order: P1 ≺t P2 P2 ≺t P3 P3 ≺t P4 and
    no-select: ⋀C. select C = {#}
  shows
    C1 ≺c C2 C2 ≺c C3 C3 ≺c C4 C4 ≺c C5
proof −
  have lit-order: Pos P1 ≺l Neg P1 Neg P1 ≺l Pos P2 Pos P2 ≺l Neg P2 Neg P2
≺l Pos P3
    Pos P3 ≺l Neg P3 Neg P3 ≺l Pos P4 Pos P4 ≺l Neg P4
    using term-order by simp-all

  show C1 ≺c C2
    unfolding C1-def C2-def
  proof (rule multp-if-all-left-smaller)
    show {#Pos P2, Neg P3#} ≠ {#}
      by simp
  next
    show ∀k∈#{#Neg P1, Neg P2#}. ∃j∈#{#Pos P2, Neg P3#}. k ≺l j
      using lit-order by simp
  qed

  moreover show C2 ≺c C3
    unfolding C2-def C3-def
  proof (rule multp-if-all-left-smaller)
    show {#Pos P1, Pos P2, Pos P4#} ≠ {#}
      by simp
  next
    show ∀k∈#{#Pos P2, Neg P3#}. ∃j∈#{#Pos P1, Pos P2, Pos P4#}. k ≺l
j
      using lit-order by auto
  qed

  moreover show C3 ≺c C4
  proof −
    have {#Pos P1, Pos P2#} ≺c {#Pos P2, Pos P3#}
    proof (rule multp-if-all-left-smaller)
      show {#Pos P2, Pos P3#} ≠ {#}
        by simp
    next
      show ∀k∈#{#Pos P1, Pos P2#}. ∃j∈#{#Pos P2, Pos P3#}. k ≺l j
        using lit-order by simp
    qed
    thus ?thesis
      unfolding C3-def C4-def
    by (smt (verit, ccfv-SIG) add-mset-commute irreflp-on-less-lit multp-cancel-add-mset
```

```
        transp-less-lit)
  qed

  moreover show C4 ≺c C5
    unfolding C4-def C5-def
  proof (rule multp-if-all-left-smaller)
    show {#Pos P2, Neg P4#} ≠ {#}
      by simp
  next
    show ∀k∈#{#Pos P2, Pos P3, Pos P4#}. ∃j∈#{#Pos P2, Neg P4#}. k ≺l
j
      using lit-order by auto
  qed

  note cls-order = calculation this

  have interp N C1 = {}
    unfolding interp-def N-def
    using cls-order
     by (smt (verit, best) Collect-empty-eq bot-fset.rep-eq ccSUP-empty finsertCI
fset-simps(2)
      clause-order.dual-order.strict-implies-not-eq clause-order.is-minimal-in-fset-finsertI
        clause-order.is-minimal-in-fset-iff singletonD)
  hence production N C1 = {}
    unfolding production-unfold C1-def by simp
  hence interp N C1 ∪ production N C1 ⊨ C1
    unfolding ‹interp N C1 = {}› ‹production N C1 = {}›
    unfolding C1-def
    unfolding true-cls-def true-lit-def
    by (simp add: C1-def)

  have {D ∈ N. D ≺c C2} = {C1}
    unfolding N-def
    using cls-order by auto
  hence interp N C2 = interp N C1 ∪ production N C1
    unfolding ‹interp N C1 = {}›
    unfolding interp-def
    by simp
  hence interp N C2 = {}
    unfolding ‹interp N C1 = {}› ‹production N C1 = {}› by simp
  hence production N C2 = {}
    unfolding production-unfold
    by (simp add: C2-def)
  hence interp N C2 ∪ production N C2 ⊨ C2
    using ‹interp N C2 = {}›
    by (simp add: C2-def)

  have {D ∈ N. D ≺c C3} = {C1, C2}
    unfolding N-def
```

using *cls-order* **by** *auto*
hence *interp N C3 = interp N C2 ∪ production N C2*
unfolding ‹*interp N C2 = {}*›
unfolding *interp-def*
by (*simp add*: ‹*production N C1 = {}*›)
hence *interp N C3 = {}*
unfolding ‹*interp N C2 = {}*› ‹*production N C2 = {}*› **by** *simp*
have *production N C3 = {P4}*
proof −
have *C3 ∈ N*
by (*simp add*: *N-def*)
moreover have ∃ *C3′*. *C3 = add-mset* (*Pos P4*) *C3′*
by (*auto simp*: *C3-def*)
moreover have *is-strictly-maximal-lit* (*Pos P4*) *C3*
unfolding *C3-def literal-order.is-greatest-in-mset-iff*
using *lit-order* **by** *auto*
moreover have ¬ *interp N C3 ⊨ C3*
unfolding ‹*interp N C3 = {}*›
unfolding *C3-def*
by *simp*
ultimately have *P4 ∈ production N C3*
unfolding *production-unfold*
using *no-select*
by *simp*
thus *?thesis*
using *production-eq-empty-or-singleton* **by** *fastforce*
qed
hence *interp N C3 ∪ production N C3 ⊨ C3*
using ‹*interp N C3 = {}*›
by (*simp add*: *C3-def*)

have {*D ∈ N*. *D* ≺*c C4*} = {*C1*, *C2*, *C3*}
unfolding *N-def*
using *cls-order* **by** *auto*
hence *interp N C4 = interp N C3 ∪ production N C3*
unfolding ‹*interp N C3 = interp N C2 ∪ production N C2*›
unfolding ‹*interp N C2 = interp N C1 ∪ production N C1*›
unfolding ‹*interp N C1 = {}*›
unfolding *interp-def*
by *auto*
hence *interp N C4 = {P4}*
using ‹*interp N C3 = {}*› ‹*production N C3 = {P4}*› **by** *simp*
hence *interp N C4 ⊨ C4*
using *C4-def* **by** *simp*
hence *production N C4 = {}*
unfolding *production-unfold* **by** *simp*
hence *interp N C4 ∪ production N C4 ⊨ C4*
using ‹*interp N C4 = {P4}*›
by (*simp add*: *C4-def*)

```
  have {D ∈ N. D ≺c C5} = {C1, C2, C3, C4}
    unfolding N-def
    using cls-order by auto
  hence interp N C5 = interp N C4 ∪ production N C4
    unfolding ‹interp N C4 = interp N C3 ∪ production N C3›
    unfolding ‹interp N C3 = interp N C2 ∪ production N C2›
    unfolding ‹interp N C2 = interp N C1 ∪ production N C1›
    unfolding ‹interp N C1 = {}›
    unfolding interp-def
    by auto
  hence interp N C5 = {P4}
    using ‹interp N C4 = {P4}› ‹production N C4 = {}› by simp
  have production N C5 = {}
  proof −
    have is-strictly-maximal-lit (Neg P4) C5
      unfolding C5-def literal-order.is-greatest-in-mset-iff
      using lit-order by auto
    hence ⋀A. ¬ is-strictly-maximal-lit (Pos A) C5
      by (meson Uniq-D literal-order.Uniq-is-greatest-in-mset literal.distinct(1))
    thus ?thesis
      unfolding production-unfold by simp
  qed
  hence ¬ interp N C5 ∪ production N C5 ⊨ C5
    unfolding ‹interp N C5 = {P4}› ‹production N C5 = {}›
    unfolding C5-def
    using term-order
    by simp
qed


interpretation G: statically-complete-calculus G-Bot G-Inf G-entails G.Red-I G.Red-F
proof unfold-locales
  fix B :: 'f gterm atom clause and N :: 'f gterm atom clause set
  assume B ∈ G-Bot and G.saturated N
  hence B = {#}
    by simp

  assume G-entails N {B}
  hence {#} ∈ N
    unfolding ‹B = {#}›
  proof (rule contrapos-pp)
    assume {#} ∉ N

    define I :: 'f gterm set where
      I = (⋃D ∈ N. production N D)

    show ¬ G-entails N G-Bot
      unfolding G-entails-def not-all not-imp
```

```
    proof (intro exI conjI)
      show I ⊨s N
        unfolding I-def
        using model-construction(2)[OF ‹G.saturated N› ‹{#} ∉ N›]
        by (simp add: true-clss-def)
    next
      show ¬ I ⊨s G-Bot
        by simp
    qed
  qed
  thus ∃ B'∈G-Bot. B' ∈ N
    by auto
qed

end

end
theory Lower-Set
  imports Main
begin

definition is-lower-set-wrt :: ('a ⇒ 'a ⇒ bool) ⇒ 'a set ⇒ 'a set ⇒ bool where
  transp-on X R ⟹ asymp-on X R ⟹
    is-lower-set-wrt R L X ⟷ L ⊆ X ∧ (∀ l ∈ L. ∀ x ∈ X. R x l ⟶ x ∈ L)

definition is-strict-lower-set-wrt :: ('a ⇒ 'a ⇒ bool) ⇒ 'a set ⇒ 'a set ⇒ bool
where
  transp-on X R ⟹ asymp-on X R ⟹
    is-strict-lower-set-wrt R L X ⟷ L ⊂ X ∧ (∀ l ∈ L. ∀ x ∈ X. R x l ⟶ x ∈ L)

lemma is-lower-set-wrt-empty:
  fixes X :: 'a set and R :: 'a ⇒ 'a ⇒ bool
  assumes transp-on X R and asymp-on X R
  shows is-lower-set-wrt R {} X
  unfolding is-lower-set-wrt-def[OF assms] by simp

lemma is-lower-set-wrt-refl:
  fixes X :: 'a set and R :: 'a ⇒ 'a ⇒ bool
  assumes transp-on X R and asymp-on X R
  shows is-lower-set-wrt R X X
  unfolding is-lower-set-wrt-def[OF assms] by simp

lemma is-lower-set-wrt-trans:
  fixes X Y Z :: 'a set and R :: 'a ⇒ 'a ⇒ bool
  assumes
    transp-on Z R and asymp-on Z R and
    is-lower-set-wrt R X Y and is-lower-set-wrt R Y Z
  shows is-lower-set-wrt R X Z
proof −
```

```
  have Y ⊆ Z and Y-lower: ∀ l∈Y. ∀ x∈Z. R x l ⟶ x ∈ Y
    using ‹is-lower-set-wrt R Y Z›
    unfolding is-lower-set-wrt-def[OF ‹transp-on Z R› ‹asymp-on Z R›]
    unfolding atomize-conj .

  have transp-on Y R and asymp-on Y R
    unfolding atomize-conj
    using ‹transp-on Z R› ‹asymp-on Z R› ‹Y ⊆ Z›
    by (metis asymp-on-subset transp-on-subset)

  have X ⊆ Y and X-lower: ∀ l∈X. ∀ x∈Y. R x l ⟶ x ∈ X
    using ‹is-lower-set-wrt R X Y›
    unfolding is-lower-set-wrt-def[OF ‹transp-on Y R› ‹asymp-on Y R›]
    unfolding atomize-conj .

  have X ⊆ Z
    using ‹X ⊆ Y› ‹Y ⊆ Z› by order

  moreover have (∀ l∈X. ∀ x∈Z. R x l ⟶ x ∈ X)
    using ‹X ⊆ Y› X-lower Y-lower by blast

  ultimately show ?thesis
    unfolding is-lower-set-wrt-def[OF ‹transp-on Z R› ‹asymp-on Z R›] ..
qed

lemma is-lower-set-wrt-antisym:
  fixes X Y :: 'a set and R :: 'a ⇒ 'a ⇒ bool
  assumes
    transp-on Y R and asymp-on Y R and
    is-lower-set-wrt R X Y and is-lower-set-wrt R Y X
  shows X = Y
proof −
  have X ⊆ Y
    using ‹is-lower-set-wrt R X Y›
    unfolding is-lower-set-wrt-def[OF ‹transp-on Y R› ‹asymp-on Y R›] ..

  have transp-on X R and asymp-on X R
    unfolding atomize-conj
    using ‹transp-on Y R› ‹asymp-on Y R› ‹X ⊆ Y›
    by (metis asymp-on-subset transp-on-subset)

  have Y ⊆ X
    using ‹is-lower-set-wrt R Y X›
    unfolding is-lower-set-wrt-def[OF ‹transp-on X R› ‹asymp-on X R›] ..

  show X = Y
    using ‹X ⊆ Y› ‹Y ⊆ X› by (metis subset-antisym)
qed
```

lemma *order-is-lower-set-wrt*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *transp R* **and** *asymp R*
shows *class.order* (*is-lower-set-wrt R*) (*is-strict-lower-set-wrt R*)
proof *unfold-locales*
have *trans*: $\bigwedge A.$ *transp-on A R*
using ‹*transp R*› **by** (*metis subset-UNIV transp-on-subset*)

have *asym*: $\bigwedge A.$ *asymp-on A R*
using ‹*asymp R*› **by** (*metis subset-UNIV asymp-on-subset*)

show $\bigwedge x\ y.$ *is-strict-lower-set-wrt R x y* = (*is-lower-set-wrt R x y* ∧ ¬ *is-lower-set-wrt R y x*)
unfolding *is-lower-set-wrt-def*[*OF trans asym*]
unfolding *is-strict-lower-set-wrt-def*[*OF trans asym*]
by (*metis order-le-less subset-not-subset-eq*)

show $\bigwedge x.$ *is-lower-set-wrt R x x*
using *is-lower-set-wrt-refl*[*OF trans asym*] **.**

show $\bigwedge x\ y\ z.$ *is-lower-set-wrt R x y* ⟹ *is-lower-set-wrt R y z* ⟹ *is-lower-set-wrt R x z*
using *is-lower-set-wrt-trans*[*OF trans asym*] **.**

show $\bigwedge x\ y.$ *is-lower-set-wrt R x y* ⟹ *is-lower-set-wrt R y x* ⟹ $x = y$
using *is-lower-set-wrt-antisym*[*OF trans asym*] **.**
qed

lemma *is-lower-set-wrt-insertI*:
assumes *transp-on* (*insert x X*) *R* **and** *asymp-on* (*insert x X*) *R* **and**
$x \in X$ **and** $\forall w \in X.\ R\ w\ x \longrightarrow w \in L$ **and** *is-lower-set-wrt R L X*
shows *is-lower-set-wrt R* (*insert x L*) *X*
proof −
have *transp-on X R* **and** *asymp-on X R*
unfolding *atomize-conj*
using ‹*transp-on* (*insert x X*) *R*› ‹*asymp-on* (*insert x X*) *R*›
by (*metis asymp-on-subset subset-insertI transp-on-subset*)

have *is-lower-set-wrt R* (*insert x L*) *X* ⟷
insert x L ⊆ *X* ∧ (∀ *l*∈*insert x L*. ∀ *xa*∈*X*. *R xa l* ⟶ *xa* ∈ *insert x L*)
unfolding *is-lower-set-wrt-def*[*OF* ‹*transp-on X R*› ‹*asymp-on X R*›] **..**

also have ... ⟷ $x \in X \wedge L \subseteq X \wedge$ (∀ *l*∈*insert x L*. ∀ *xa*∈*X*. *R xa l* ⟶ *xa* ∈ *insert x L*)
by *simp*

also have ... ⟷ $x \in X \wedge L \subseteq X \wedge (\forall\, w{\in}X.\ R\ w\ x \longrightarrow w \in$ *insert x L*$) \wedge$
(∀ *l*∈*L*. ∀ *xa*∈*X*. *R xa l* ⟶ *xa* ∈ *insert x L*)
by *simp*

also have ... $\longleftrightarrow$ $x \in X \wedge L \subseteq X \wedge (\forall\, w{\in}X.\ R\ w\ x \longrightarrow w \in L) \wedge$
$(\forall\, l{\in}L.\ \forall\, y{\in}X.\ R\ y\ l \longrightarrow y \in \mathit{insert}\ x\ L)$
using ‹*asymp-on* (*insert x X*) *R*› **by** (*smt* (*verit*) *asymp-onD insertCI insertE*)

also have ... $\longleftrightarrow$ $x \in X \wedge (\forall\, w{\in}X.\ R\ w\ x \longrightarrow w \in L) \wedge$ *is-lower-set-wrt R L X*
using ‹*is-lower-set-wrt R L X*›
unfolding *is-lower-set-wrt-def*[*OF* ‹*transp-on X R*› ‹*asymp-on X R*›]
by *blast*

finally show *?thesis*
using *assms*(*3*−) **by** *argo*
qed

lemma *lower-set-wrt-appendI*:
assumes
trans: *transp-on* (*set* (*xs* @ *ys*)) *R* **and**
asym: *asymp-on* (*set* (*xs* @ *ys*)) *R* **and**
sorted: *sorted-wrt R* (*xs* @ *ys*)
shows *is-lower-set-wrt R* (*set xs*) (*set* (*xs* @ *ys*))
unfolding *is-lower-set-wrt-def*[*OF trans asym*]
proof (*intro conjI*)
show *set xs* ⊆ *set* (*xs* @ *ys*)
by *simp*
next
show $\forall\, l{\in}\mathit{set}\ \mathit{xs}.\ \forall\, x{\in}\mathit{set}\ (\mathit{xs} \mathbin{@} \mathit{ys}).\ R\ x\ l \longrightarrow x \in \mathit{set}\ \mathit{xs}$
using *sorted*
by (*metis Un-iff asym asymp-on-def set-append sorted-wrt-append*)
qed

lemma *sorted-and-lower-set-wrt-appendD-left*:
assumes *transp-on A R* **and** *asymp-on A R* **and**
sorted-wrt R (*xs* @ *ys*) **and** *is-lower-set-wrt R* (*set* (*xs* @ *ys*)) *A*
shows *sorted-wrt R xs* **and** *is-lower-set-wrt R* (*set xs*) *A*
unfolding *atomize-conj*
using *assms*
by (*smt* (*verit*) *Un-iff asymp-on-def is-lower-set-wrt-def le-sup-iff set-append sorted-wrt-append*
subsetD)

lemma *sorted-and-lower-set-wrt-appendD-right*:
assumes *transp-on A R* **and** *asymp-on A R* **and**
sorted-wrt ($\lambda x\ y.\ R\ y\ x$) (*xs* @ *ys*) **and** *is-lower-set-wrt R* (*set* (*xs* @ *ys*)) *A*
shows *sorted-wrt* ($\lambda x\ y.\ R\ y\ x$) *ys* **and** *is-lower-set-wrt R* (*set ys*) *A*
unfolding *atomize-conj*
using *assms*
by (*smt* (*verit*, *ccfv-threshold*) *Un-iff asymp-onD is-lower-set-wrt-def le-sup-iff set-append*

sorted-wrt-append subsetD)

lemma *not-in-lower-set-wrtI*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *trans*: *transp-on Y R* **and** *asym*: *asymp-on Y R*
shows *is-lower-set-wrt R X Y* $\Longrightarrow y \notin X \Longrightarrow y \in Y \Longrightarrow R\ y\ z \Longrightarrow z \notin X$
unfolding *is-lower-set-wrt-def*[*OF trans asym*]
by *blast*

abbreviation (**in** *preorder*) *is-lower-set* **where**
is-lower-set $\equiv$ *is-lower-set-wrt* $(<)$

lemmas (**in** *preorder*) *is-lower-set-iff* =
is-lower-set-wrt-def[*OF transp-on-less asymp-on-less*]

context *linorder* **begin**

sublocale *is-lower-set*: *order is-lower-set-wrt* $(<)$ *is-strict-lower-set-wrt* $(<)$
proof (*rule order-is-lower-set-wrt*)
show *transp* $(<)$
using *transp-on-less* **.**
next
show *asymp* $(<)$
using *asymp-on-less* **.**
qed

end

lemmas (**in** *preorder*) *is-lower-set-empty*[*simp*] =
is-lower-set-wrt-empty[*OF transp-on-less asymp-on-less*]

lemmas (**in** *preorder*) *is-lower-set-insertI* =
is-lower-set-wrt-insertI[*OF transp-on-less asymp-on-less*]

lemmas (**in** *preorder*) *lower-set-appendI* =
lower-set-wrt-appendI[*OF transp-on-less asymp-on-less*]

lemmas (**in** *preorder*) *sorted-and-lower-set-appendD-left* =
sorted-and-lower-set-wrt-appendD-left[*OF transp-on-less asymp-on-less*]

lemmas (**in** *preorder*) *sorted-and-lower-set-appendD-right* =
sorted-and-lower-set-wrt-appendD-right[*OF transp-on-less asymp-on-less*]

lemmas (**in** *preorder*) *not-in-lower-setI* =
not-in-lower-set-wrtI[*OF transp-on-less asymp-on-less*]

end

theory *HOL-Extra-Extra*
imports *First-Order-Clause.HOL-Extra*
begin

no-notation *restrict-map* (**infixl** |‘ *110*)

lemma
assumes $\exists_{\leq 1} x.\ P\ x$
shows *finite* $\{x.\ P\ x\}$
using *assms Collect-eq-if-Uniq* **by** *fastforce*

lemma *finite-if-Uniq-Uniq*:
assumes
$\exists_{\leq 1} x.\ P\ x$
$\forall x.\ \exists_{\leq 1} y.\ Q\ x\ y$
shows *finite* $\{y.\ \exists x.\ P\ x \wedge Q\ x\ y\}$
using *assms*
by (*smt* (*verit*, *best*) *Collect-eq-if-Uniq UniqI Uniq-D finite.emptyI finite-insert*)

lemma *finite-if-finite-finite*:
assumes
finite $\{x.\ P\ x\}$
$\forall x.$ *finite* $\{y.\ Q\ x\ y\}$
shows *finite* $\{y.\ \exists x.\ P\ x \wedge Q\ x\ y\}$
using *assms* **by** *auto*

lemma (**in** *order*) *greater-wfp-on-finite-set*: *finite* $\mathcal{X} \Longrightarrow$ *Wellfounded.wfp-on* $\mathcal{X}$ $(>)$
using *strict-partial-order-wfp-on-finite-set*[*OF transp-on-greater asymp-on-greater*]
.

lemma (**in** *order*) *less-wfp-on-finite-set*: *finite* $\mathcal{X} \Longrightarrow$ *Wellfounded.wfp-on* $\mathcal{X}$ $(<)$
using *strict-partial-order-wfp-on-finite-set*[*OF transp-on-less asymp-on-less*] .

lemma *distinct-if-sorted-wrt-asymp*:
assumes *asymp-on* (*set xs*) *R* **and** *sorted-wrt R xs*
shows *distinct xs*
using *assms*
proof (*induction xs*)
case *Nil*
show *?case*
unfolding *distinct.simps* ..
next
case (*Cons x xs*)

have *R-x-asym*: $\forall y \in$ *set xs*. *R x y* $\longrightarrow \neg$ *R y x* **and** *asymp-on* (*set xs*) *R*
using *Cons.prems*(*1*)
unfolding *atomize-conj*
by (*metis asymp-on-def list.set-intros*(*1*) *list.set-intros*(*2*))

```
  have R-x: ∀ y ∈ set xs. R x y and sorted-wrt R xs
    using Cons.prems(2)
    unfolding atomize-conj sorted-wrt.simps
    by argo

  have x ∉ set xs
  proof (intro notI)
    assume x-in: x ∈ set xs

    have R x x
      using R-x x-in by metis

    moreover hence ¬ R x x
      using R-x-asym x-in by metis

    ultimately show False
      by contradiction
  qed

  moreover have distinct xs
    using Cons.IH ‹asymp-on (set xs) R› ‹sorted-wrt R xs› by argo

  ultimately show ?case
    unfolding distinct.simps by argo
qed

lemma dropWhile-append-eq-rhs:
  fixes xs ys :: 'a list and P :: 'a ⇒ bool
  assumes
    ⋀x. x ∈ set xs ⟹ P x and
    ⋀y. y ∈ set ys ⟹ ¬ P y
  shows dropWhile P (xs @ ys) = ys
  using assms by simp

lemma mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone:
  fixes R :: 'a ⇒ 'a ⇒ bool and xs :: 'a list and P :: 'a ⇒ bool
  assumes sorted-wrt R xs and monotone-on (set xs) R (≥) P
  shows x ∈ set (dropWhile P xs) ⟷ ¬ P x ∧ x ∈ set xs
  using assms
proof (induction xs)
  case Nil
  show ?case
    by simp
next
  case (Cons y xs)
  have ∀ z ∈ set xs. R y z and sorted-wrt R xs
    using ‹sorted-wrt R (y # xs)› by simp-all
```

moreover have *monotone-on* (*set xs*) *R* (≥) *P*
using ‹*monotone-on* (*set* (*y* # *xs*)) *R* (≥) *P*›
by (*metis monotone-on-subset set-subset-Cons*)

ultimately have *IH*: (*x* ∈ *set* (*dropWhile P xs*)) = (¬ *P x* ∧ *x* ∈ *set xs*)
using *Cons.IH* ‹*sorted-wrt R xs*› **by** *metis*

show *?case*
proof (*cases P y*)
case *True*
thus *?thesis*
unfolding *dropWhile.simps*
unfolding *if-P*[*OF True*]
using *IH* **by** *auto*
next
case *False*
then show *?thesis*
unfolding *dropWhile.simps*
unfolding *if-not-P*[*OF False*]
by (*metis* (*full-types*) *Cons.prems*(*1*) *Cons.prems*(*2*) *le-boolD list.set-intros*(*1*) *monotone-on-def*
set-ConsD sorted-wrt.simps(*2*))
qed
qed

lemma *ball-set-dropWhile-if-sorted-wrt-and-monotone-on*:
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool* **and** *xs* :: *'a list* **and** *P* :: *'a* ⇒ *bool*
assumes *sorted-wrt R xs* **and** *monotone-on* (*set xs*) *R* (≥) *P*
shows ∀ *x* ∈ *set* (*dropWhile P xs*). ¬ *P x*
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*[*OF assms*] **by** *metis*

lemma *filter-set-eq-filter-set-minus-singleton*:
assumes ¬ *P y*
shows {*x* ∈ *X*. *P x*} = {*x* ∈ *X* − {*y*}. *P x*}
using *assms* **by** *blast*

lemma *ex1-subset-eq-image-if-bij-betw*:
fixes *f* :: *'a* ⇒ *'b* **and** *X* :: *'a set* **and** *Y* :: *'b set*
assumes *bij-betw f X Y* **and** *Y'* ⊆ *Y*
shows ∃!*X'*. *X'* ⊆ *X* ∧ *Y'* = *f* ‘ *X'*
using *assms*
by (*metis bij-betw-def inv-into-image-cancel subset-image-iff*)

lemma *Collect-eq-image-filter-Collect-if-bij-betw*:
fixes *f* :: *'a* ⇒ *'b* **and** *X* :: *'a set* **and** *Y* :: *'b set*
assumes *bij*: *bij-betw f X Y* **and** *sub*: {*y*. *P y*} ⊆ *Y*
shows {*y*. *P y*} = *f* ‘ {*x*. *x* ∈ *X* ∧ *P* (*f x*)}
using *ex1-subset-eq-image-if-bij-betw*[*OF bij sub*]

by (*smt* (*verit*, *best*) *Collect-cong image-def in-mono mem-Collect-eq*)

lemma *restrict-map-ident-if-dom-subset*: *dom* $\mathcal{M} \subseteq A \Longrightarrow$ *restrict-map* $\mathcal{M}$ $A = \mathcal{M}$
by (*metis domIff ext in-mono restrict-map-def*)

1.6 Move to *HOL.Transitive-Closure*

lemma *relpowp-right-unique*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $n :: nat$ **and** $x\ y\ z :: 'a$
assumes *runique*: $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ x\ z \Longrightarrow y = z$
shows $(R \frown\frown n)\ x\ y \Longrightarrow (R \frown\frown n)\ x\ z \Longrightarrow y = z$
proof (*induction n arbitrary*: $x\ y\ z$)
case *0*
thus *?case*
by *simp*
next
case (*Suc n'*)
then obtain $x' :: 'a$ **where**
$(R \frown\frown n')\ x\ x'$ **and** $R\ x'\ y$ **and** $R\ x'\ z$
by *auto*
thus $y = z$
using *runique* **by** *simp*
qed

lemma *Uniq-relpowp*:
fixes $n :: nat$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$
assumes *runiq*: $\forall x.\ \exists_{\leq 1} y.\ R\ x\ y$
shows $\exists_{\leq 1} y.\ (R \frown\frown n)\ x\ y$
proof (*rule Uniq-I*)
fix $y\ z$
assume $(R \frown\frown n)\ x\ y$ **and** $(R \frown\frown n)\ x\ z$
show $y = z$
proof (*rule relpowp-right-unique*)
show $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ x\ z \Longrightarrow y = z$
using *runiq* **by** (*auto dest*: *Uniq-D*)
next
show $(R \frown\frown n)\ x\ y$
using ‹$(R \frown\frown n)\ x\ y$› **.**
next
show $(R \frown\frown n)\ x\ z$
using ‹$(R \frown\frown n)\ x\ z$› **.**
qed
qed

lemma *relpowp-plus-of-right-unique*:
assumes
right-unique R
$(R \frown\frown m)\ x\ y$ **and**

```
    (R ^^ (m + n)) x z
  shows (R ^^ n) y z
  using assms(2,3)
proof (induction m arbitrary: x)
  case 0
  thus ?case
    by simp
next
  case (Suc m)
  then show ?case
    by (metis add-Suc assms(1) relpowp-Suc-E2 right-uniqueD)
qed

lemma relpowp-plusD:
  assumes (R ^^ (m + n)) x z
  shows ∃ y. (R ^^ m) x y ∧ (R ^^ n) y z
  using assms
proof (induction m arbitrary: x)
  case 0
  thus ?case
    by simp
next
  case (Suc m)

  obtain y where R x y and (R ^^ (m + n)) y z
    using Suc.prems by (metis add-Suc relpowp-Suc-D2)

  obtain y' where (R ^^ m) y y' and (R ^^ n) y' z
    using Suc.IH[OF ‹(R ^^ (m + n)) y z›] by metis

  show ?case
  proof (intro exI conjI)
    show (R ^^ Suc m) x y'
      using ‹R x y› ‹(R ^^ m) y y'› by (metis relpowp-Suc-I2)
  next
    show (R ^^ n) y' z
      using ‹(R ^^ n) y' z› .
  qed
qed

lemma relpowp-Suc-of-right-unique:
  assumes
    right-unique R
    R x y and
    (R ^^ Suc n) x z
  shows (R ^^ n) y z
  using assms
  by (metis relpowp-Suc-D2 right-uniqueD)
```

lemma *tranclp-if-relpowp*: $n \neq 0 \Longrightarrow (R \mathbin{\frown\!\frown} n)\ x\ y \Longrightarrow R^{++}\ x\ y$
by (*meson bot-nat-0.not-eq-extremum tranclp-power*)

lemma *transp-on-singleton*[*simp*]: *transp-on* $\{x\}$ *R*
by (*simp add*: *transp-on-def*)

lemma *rtranclp-rtranclp-compose-if-right-unique*:
assumes *runique*: *right-unique R* **and** $R^{**}\ a\ b$ **and** $R^{**}\ a\ c$
shows $R^{**}\ a\ b \wedge R^{**}\ b\ c \vee R^{**}\ a\ c \wedge R^{**}\ c\ b$
using *assms*(*2*,*3*)
proof (*induction b arbitrary*: *c rule*: *rtranclp-induct*)
case *base*
thus *?case*
by *simp*
next
case (*step a' b*)
with *runique* **show** *?case*
by (*metis converse-rtranclpE right-uniqueD rtranclp.rtrancl-into-rtrancl*)
qed

lemma *right-unique-terminating-rtranclp*:
assumes *right-unique R*
shows *right-unique* $(\lambda x\ y.\ R^{**}\ x\ y \wedge (\nexists z.\ R\ y\ z))$
unfolding *right-unique-def*
using *rtranclp-rtranclp-compose-if-right-unique*[*OF* ‹*right-unique R*›]
by (*metis converse-rtranclpE*)

end
theory *The-Optional*
imports *Main*
begin

definition *The-optional* :: $('a \Rightarrow bool) \Rightarrow 'a\ option$ **where**
The-optional P = (*if* $\exists!x.\ P\ x$ *then Some* (*THE x. P x*) *else None*)

lemma *The-optional-eq-SomeD*: *The-optional P* = *Some x* $\Longrightarrow$ *P x*
unfolding *The-optional-def*
by (*metis option.discI option.inject theI-unique*)

lemma *Some-eq-The-optionalD*: *Some x* = *The-optional P* $\Longrightarrow$ *P x*
using *The-optional-eq-SomeD* **by** *metis*

lemma *The-optional-eq-NoneD*: *The-optional P* = *None* $\Longrightarrow \nexists!x.\ P\ x$
unfolding *The-optional-def*
by (*metis option.discI*)

lemma *None-eq-The-optionalD*: *None* = *The-optional P* $\Longrightarrow \nexists!x.\ P\ x$
unfolding *The-optional-def*
by (*metis option.discI*)

lemma *The-optional-eq-SomeI*:
assumes $\exists_{\leq 1} x.\ P\ x$ **and** $P\ x$
shows *The-optional* P = *Some* x
using *assms* **by** (*metis The-optional-def the1-equality'*)

end
theory *Full-Run*
imports
VeriComp.Transfer-Extras
HOL-Extra-Extra
begin

definition *full-run* **where**
full-run $\mathcal{R}\ x\ y \longleftrightarrow \mathcal{R}^{**}\ x\ y \wedge (\nexists z.\ \mathcal{R}\ y\ z)$

lemma *Uniq-full-run*:
assumes *Uniq-R*: $\bigwedge x.\ \exists_{\leq 1} y.\ R\ x\ y$
shows $\exists_{\leq 1} y.$ *full-run* $R\ x\ y$
unfolding *full-run-def*
using *assms*
by (*smt* (*verit*, *best*) *Uniq-I right-unique-iff rtranclp-complete-run-right-unique*)

lemma *ex1-full-run*:
assumes *Uniq-R*: $\bigwedge x.\ \exists_{\leq 1} y.\ R\ x\ y$ **and** *wfP-R*: *wfP* R^{-1-1}
shows $\exists! y.$ *full-run* $R\ x\ y$
proof −
have $\exists_{\leq 1}\ y.$ *full-run* $R\ x\ y$
using *Uniq-full-run*[*of* $R\ x$] *Uniq-R* **by** *argo*

moreover have $\exists\, y.$ *full-run* $R\ x\ y$
using *ex-terminating-rtranclp*[*OF wfP-R*, *of* x, *folded full-run-def*] **.**

ultimately show *?thesis*
using *Uniq-implies-ex1* **by** *metis*
qed

lemma *full-run-preserves-invariant*:
assumes
run: *full-run* $R\ x\ y$ **and**
P-init: $P\ x$ **and**
R-preserves-P: $\bigwedge x\ y.\ R\ x\ y \Longrightarrow P\ x \Longrightarrow P\ y$
shows $P\ y$
proof −
from *run* **have** $R^{**}\ x\ y$
unfolding *full-run-def* **by** *simp*
thus $P\ y$
using *P-init*
proof (*induction* x *rule*: *converse-rtranclp-induct*)

case *base*
thus *?case*
by *assumption*
next
case (*step x x'*)
then show *?case*
using *R-preserves-P* **by** *metis*
qed
qed

end
theory *Background*
imports
Simple-Clause-Learning.SCL-FOL
Simple-Clause-Learning.Correct-Termination
Simple-Clause-Learning.Initial-Literals-Generalize-Learned-Literals
Simple-Clause-Learning.Termination
Ground-Ordered-Resolution
Min-Max-Least-Greatest.Min-Max-Least-Greatest-FSet
First-Order-Clause.Multiset-Extra
VeriComp.Compiler
HOL−ex.Sketch-and-Explore
HOL−Library.FuncSet
Lower-Set
HOL-Extra-Extra
The-Optional
Full-Run
begin

lemma $I \models l\ L \longleftrightarrow (\textit{is-pos}\ L \longleftrightarrow \textit{atm-of}\ L \in I)$
by (*cases L*) *simp-all*

2 Move to *HOL−Library.Multiset*

lemmas *strict-subset-implies-multp* = *subset-implies-multp*
hide-fact *subset-implies-multp*

lemma *subset-implies-reflclp-multp*: $A \subseteq\# B \Longrightarrow (\textit{multp}\ R)^{==}\ A\ B$
by (*metis reflclp-iff strict-subset-implies-multp subset-mset.le-imp-less-or-eq*)

lemma *member-mset-repeat-msetD*: $L \in\# \textit{repeat-mset}\ n\ M \Longrightarrow L \in\# M$
by (*induction n*) *auto*

lemma *member-mset-repeat-mset-Suc*[*simp*]: $L \in\# \textit{repeat-mset}\ (\textit{Suc}\ n)\ M \longleftrightarrow L \in\# M$
by (*metis member-mset-repeat-msetD repeat-mset-Suc union-iff*)

lemma *image-msetI*: $x \in\# M \Longrightarrow f\ x \in\# \textit{image-mset}\ f\ M$
by (*metis imageI in-image-mset*)

lemma *inj-image-mset-mem-iff*: *inj f* $\Longrightarrow$ *f x* $\in\#$ *image-mset f M* $\longleftrightarrow$ *x* $\in\#$ *M*
by (*simp add*: *inj-image-mem-iff*)

3 Move to *HOL−Library.FSet*

declare *wfP-pfsubset*[*intro*]

syntax
-*FFilter* :: *pttrn* $\Rightarrow$ *'a fset* $\Rightarrow$ *bool* $\Rightarrow$ *'a fset* ((*1*{|- |∈| -./ -|}))

translations
{|*x* |∈| *X*. *P*|} == *CONST ffilter* (λ*x*. *P*) *X*

lemma *fimage-ffUnion*: *f* |‘| *ffUnion SS* = *ffUnion* ((|‘|) *f* |‘| *SS*)
proof (*intro fsubset-antisym fsubsetI*)
fix *x* **assume** *x* |∈| *f* |‘| *ffUnion SS*
then obtain *y* **where** *y* |∈| *ffUnion SS* **and** *x* = *f y*
by *auto*
thus *x* |∈| *ffUnion* ((|‘|) *f* |‘| *SS*)
unfolding *fmember-ffUnion-iff*
by (*metis UN-E ffUnion.rep-eq fimage-eqI*)
next
fix *x* **assume** *x* |∈| *ffUnion* ((|‘|) *f* |‘| *SS*)
then obtain *S* **where** *S* |∈| *SS* **and** *x* |∈| *f* |‘| *S*
unfolding *fmember-ffUnion-iff* **by** *auto*
then show *x* |∈| *f* |‘| *ffUnion SS*
by (*metis ffUnion-fsubset-iff fimage-mono fin-mono fsubsetI*)
qed

lemma *ffilter-eq-ffilter-minus-singleton*:
assumes ¬ *P y*
shows {|*x* |∈| *X*. *P x*|} = {|*x* |∈| *X* − {|*y*|}. *P x*|}
using *assms* **by** (*induction X*) *auto*

lemma *fun-upd-fimage*: *f*(*x* := *y*) |‘| *A* = (*if x* |∈| *A then finsert y* (*f* |‘| (*A* − {|*x*|})) *else f* |‘| *A*)
using *fun-upd-image*
by (*smt* (*verit*) *bot-fset.rep-eq finsert.rep-eq fset.set-map fset-cong minus-fset.rep-eq*)

lemma *ffilter-fempty*[*simp*]: *ffilter P* {||} = {||}
by (*metis ex-fin-conv ffmember-filter*)

lemma *fstrict-subset-iff-fset-strict-subset-fset*:
fixes $\mathcal{X}$ $\mathcal{Y}$ :: - *fset*
shows $\mathcal{X}$ |⊂| $\mathcal{Y}$ $\longleftrightarrow$ *fset* $\mathcal{X}$ ⊂ *fset* $\mathcal{Y}$
by *blast*

lemma (**in** *linorder*) *ex1-sorted-list-for-fset*:

$\exists!xs.\ sorted\text{-}wrt\ (<)\ xs \wedge fset\text{-}of\text{-}list\ xs = X$
using *ex1-sorted-list-for-set-if-finite*
by (*metis finite-fset fset-cong fset-of-list.rep-eq*)

lemma (**in** *linorder*) *is-least-in-fset-ffilterD*:
assumes *is-least-in-fset-wrt* (<) (*ffilter P X*) *x*
shows $x \mid\in\mid X\ P\ x$
using *assms*
by (*simp-all add*: *is-least-in-fset-wrt-iff*)

4 Move to *VeriComp.Simulation*

locale *forward-simulation-with-measuring-function* =
L1: *semantics step1 final1* +
L2: *semantics step2 final2*
for
step1 :: $'state1 \Rightarrow 'state1 \Rightarrow bool$ **and**
step2 :: $'state2 \Rightarrow 'state2 \Rightarrow bool$ **and**
final1 :: $'state1 \Rightarrow bool$ **and**
final2 :: $'state2 \Rightarrow bool$ +
fixes
match :: $'state1 \Rightarrow 'state2 \Rightarrow bool$ **and**
measure :: $'state1 \Rightarrow 'index$ **and**
order :: $'index \Rightarrow 'index \Rightarrow bool$ (**infix** $\sqsubset$ *70*)
assumes
wfp-order:
wfp ($\sqsubset$) **and**
match-final:
match s1 s2 $\Longrightarrow$ *final1 s1* $\Longrightarrow$ *final2 s2* **and**
simulation:
match s1 s2 $\Longrightarrow$ *step1 s1 s1'* $\Longrightarrow$
($\exists$ *s2'*. *step2*$^{++}$ *s2 s2'* $\wedge$ *match s1' s2'*) $\vee$ (*match s1' s2* $\wedge$ *measure s1'* $\sqsubset$ *measure s1*)
begin

sublocale *forward-simulation* **where**
step1 = *step1* **and** *step2* = *step2* **and** *final1* = *final1* **and** *final2* = *final2* **and** *order* = *order* **and**
match = $\lambda i\ x\ y.\ i = measure\ x \wedge match\ x\ y$
proof *unfold-locales*
show $\bigwedge i\ s1\ s2.\ i = measure\ s1 \wedge match\ s1\ s2 \Longrightarrow final1\ s1 \Longrightarrow final2\ s2$
using *match-final* **by** *metis*
next
show $\bigwedge i\ s1\ s2\ s1'.\ i = measure\ s1 \wedge match\ s1\ s2 \Longrightarrow step1\ s1\ s1' \Longrightarrow$
($\exists i'\ s2'.\ step2^{++}\ s2\ s2' \wedge i' = measure\ s1' \wedge match\ s1'\ s2'$) $\vee$
($\exists i'.\ (i' = measure\ s1' \wedge match\ s1'\ s2) \wedge i' \sqsubset i$)
using *simulation* **by** *metis*
next
show *wfp* ($\sqsubset$)

using *wfp-order* **.**
qed

end

locale *backward-simulation-with-measuring-function* =
L1: *semantics step1 final1* +
L2: *semantics step2 final2*
for
step1 :: *'state1* ⇒ *'state1* ⇒ *bool* **and**
step2 :: *'state2* ⇒ *'state2* ⇒ *bool* **and**
final1 :: *'state1* ⇒ *bool* **and**
final2 :: *'state2* ⇒ *bool* +
fixes
match :: *'state1* ⇒ *'state2* ⇒ *bool* **and**
measure :: *'state2* ⇒ *'index* **and**
order :: *'index* ⇒ *'index* ⇒ *bool* (**infix** ⊏ *70*)
assumes
wfp-order:
wfp (⊏) **and**
match-final:
match s1 s2 ⟹ *final2 s2* ⟹ *final1 s1* **and**
simulation:
match s1 s2 ⟹ *step2 s2 s2'* ⟹
(∃ *s1'*. *step1*$^{++}$ *s1 s1'* ∧ *match s1' s2'*) ∨ (*match s1 s2'* ∧ *measure s2'* ⊏ *measure s2*)
begin

sublocale *backward-simulation* **where**
step1 = *step1* **and** *step2* = *step2* **and** *final1* = *final1* **and** *final2* = *final2* **and** *order* = *order* **and**
match = λ*i x y*. *i* = *measure y* ∧ *match x y*
proof *unfold-locales*
show ⋀*i s1 s2*. *i* = *measure s2* ∧ *match s1 s2* ⟹ *final2 s2* ⟹ *final1 s1*
using *match-final* **by** *metis*
next
show ⋀*i1 s1 s2 s2'*. *i1* = *measure s2* ∧ *match s1 s2* ⟹ *step2 s2 s2'* ⟹
(∃ *i2 s1'*. *step1*$^{++}$ *s1 s1'* ∧ *i2* = *measure s2'* ∧ *match s1' s2'*) ∨
(∃ *i2*. (*i2* = *measure s2'* ∧ *match s1 s2'*) ∧ *i2* ⊏ *i1*)
using *simulation* **by** *metis*
next
show *wfp* (⊏)
using *wfp-order* **.**
qed

end

5 Move to *Simple-Clause-Learning.SCL-FOL*

definition *trail-true-lit* :: (- *literal* × - *option*) *list* ⇒ - *literal* ⇒ *bool* **where**
trail-true-lit Γ *L* ⟷ *L* ∈ *fst* ‘ *set* Γ

definition *trail-false-lit* :: (- *literal* × - *option*) *list* ⇒ - *literal* ⇒ *bool* **where**
trail-false-lit Γ *L* ⟷ − *L* ∈ *fst* ‘ *set* Γ

definition *trail-true-cls* :: (- *literal* × - *option*) *list* ⇒ - *clause* ⇒ *bool* **where**
trail-true-cls Γ *C* ⟷ (∃ *L* ∈# *C*. *trail-true-lit* Γ *L*)

definition *trail-false-cls* :: (- *literal* × - *option*) *list* ⇒ - *clause* ⇒ *bool* **where**
trail-false-cls Γ *C* ⟷ (∀ *L* ∈# *C*. *trail-false-lit* Γ *L*)

lemma *trail-false-cls-mempty*[*simp*]: *trail-false-cls* Γ {#}
by (*simp add*: *trail-false-cls-def*)

definition *trail-defined-lit* :: (- *literal* × - *option*) *list* ⇒ - *literal* ⇒ *bool* **where**
trail-defined-lit Γ *L* ⟷ (*L* ∈ *fst* ‘ *set* Γ ∨ − *L* ∈ *fst* ‘ *set* Γ)

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*)

definition *trail-defined-cls* :: (- *literal* × - *option*) *list* ⇒ - *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 *subtrail-falseI*:
assumes *tr-false*: *trail-false-cls* ((*L*, *Cl*) # Γ) *C* **and** *L-not-in*: −*L* ∉# *C*
shows *trail-false-cls* Γ *C*
unfolding *trail-false-cls-def*
proof (*rule ballI*)
fix *M*
assume *M-in*: *M* ∈# *C*

from *M-in L-not-in* **have** *M-neq-L*: *M* ≠ −*L* **by** *auto*

from *M-in tr-false* **have** *tr-false-lit-M*: *trail-false-lit* ((*L*, *Cl*) # Γ) *M*
unfolding *trail-false-cls-def* **by** *simp*

thus *trail-false-lit* Γ *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

inductive *trail-consistent* :: ('*a literal* × '*b option*) *list* ⇒ *bool* **where**
Nil[*simp*]: *trail-consistent* [] |
Cons: ¬ *trail-defined-lit* Γ *L* ⟹ *trail-consistent* Γ ⟹ *trail-consistent* ((*L*, *u*) # Γ)

lemma *distinct-lits-if-trail-consistent*:
trail-consistent Γ ⟹ *distinct* (*map fst* Γ)
by (*induction* Γ *rule*: *trail-consistent.induct*)
(*simp-all add*: *image-comp trail-defined-lit-def*)

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

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

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-defined-lit-if-trail-defined-suffix*:
suffix Γ' Γ ⟹ *trail-defined-lit* Γ' *K* ⟹ *trail-defined-lit* Γ *K*
unfolding *trail-defined-lit-def*
by (*metis* (*no-types*) *Un-iff image-Un set-append suffix-def*)

lemma *trail-defined-cls-if-trail-defined-suffix*:
suffix Γ' Γ ⟹ *trail-defined-cls* Γ' *C* ⟹ *trail-defined-cls* Γ *C*
unfolding *trail-defined-cls-def* **by** (*metis trail-defined-lit-if-trail-defined-suffix*)

lemma *not-trail-true-lit-and-trail-false-lit*:
fixes Γ :: ('*a literal* × '*b option*) *list* **and** *L* :: '*a literal*
shows *trail-consistent* Γ ⟹ ¬ (*trail-true-lit* Γ *L* ∧ *trail-false-lit* Γ *L*)
proof (*induction* Γ *rule*: *trail-consistent.induct*)
case *Nil*
show *?case*
by (*simp add*: *trail-true-cls-def trail-false-cls-def trail-true-lit-def trail-false-lit-def*)
next

case (*Cons* Γ *K annot*)
then show *?case*
unfolding *trail-defined-lit-def trail-false-lit-def trail-true-lit-def*
by (*metis* (*no-types*, *opaque-lifting*) *fst-conv image-insert insertE list.simps*(*15*) *uminus-not-id′* *uminus-of-uminus-id*)
qed

lemma *not-trail-true-cls-and-trail-false-cls*:
fixes Γ :: (*′a literal* × *′b option*) *list* **and** *C* :: *′a clause*
shows *trail-consistent* Γ ⟹ ¬ (*trail-true-cls* Γ *C* ∧ *trail-false-cls* Γ *C*)
proof (*induction* Γ *rule*: *trail-consistent.induct*)
case *Nil*
show *?case*
by (*simp add*: *trail-true-cls-def trail-false-cls-def trail-true-lit-def trail-false-lit-def*)
next
case (*Cons* Γ *L u*)
thus *?case*
using *not-trail-true-lit-and-trail-false-lit*
by (*metis trail-consistent.Cons trail-false-cls-def trail-true-cls-def*)
qed

lemma *not-lit-and-comp-lit-false-if-trail-consistent*:
assumes *trail-consistent* Γ
shows ¬ (*trail-false-lit* Γ *L* ∧ *trail-false-lit* Γ (− *L*))
using *assms*
proof (*induction* Γ)
case *Nil*
show *?case*
by (*simp add*: *trail-false-lit-def*)
next
case (*Cons* Γ *K u*)
show *?case*
proof (*cases K* = *L* ∨ *K* = − *L*)
case *True*
thus *?thesis*
unfolding *trail-false-lit-def uminus-of-uminus-id*
unfolding *de-Morgan-conj list.set image-insert prod.sel*
by (*metis Cons.hyps*(*1*) *insertE trail-defined-lit-def uminus-not-id′ uminus-of-uminus-id*)
next
case *False*
thus *?thesis*
unfolding *trail-false-lit-def uminus-of-uminus-id*
by (*metis* (*no-types*, *lifting*) *Cons.IH fst-conv image-iff set-ConsD trail-false-lit-def uminus-of-uminus-id*)
qed
qed

lemma *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*:
assumes Γ-*consistent*: *trail-consistent* Γ **and** *C-false*: *trail-false-cls* Γ *C*
shows ¬ (*L* ∈# *C* ∧ − *L* ∈# *C*)
proof (*rule notI*)
assume *L* ∈# *C* ∧ − *L* ∈# *C*

hence *trail-false-lit* Γ *L* ∧ *trail-false-lit* Γ (− *L*)
using *C-false* **unfolding** *trail-false-cls-def* **by** *metis*

thus *False*
using Γ-*consistent not-lit-and-comp-lit-false-if-trail-consistent* **by** *metis*
qed

6 Move to ground ordered resolution

lemma (**in** *ground-ordered-resolution-calculus*) *unique-ground-resolution*:
shows ∃$_{\leq 1}$*C*. *ground-resolution P1 P2 C*
proof (*intro Uniq-I*)
fix *C C′*
assume *ground-resolution P1 P2 C* **and** *ground-resolution P1 P2 C′*
thus *C* = *C′*
unfolding *ground-resolution.simps*
apply (*elim exE conjE*)
apply *simp*
by (*metis asymp-on-less-lit is-maximal-in-mset-wrt-if-is-greatest-in-mset-wrt is-maximal-in-mset-wrt-iff literal.inject*(1) *totalpD totalp-on-less-lit transp-on-less-lit union-single-eq-diff*)
qed

lemma (**in** *ground-ordered-resolution-calculus*) *unique-ground-factoring*:
shows ∃$_{\leq 1}$*C*. *ground-factoring P C*
proof (*intro Uniq-I*)
fix *P C C′*
assume *ground-factoring P C* **and** *ground-factoring P C′*
thus *C* = *C′*
unfolding *ground-factoring.simps*
by (*metis asymp-on-less-lit is-maximal-in-mset-wrt-iff totalpD totalp-less-lit transp-on-less-lit union-single-eq-diff*)
qed

lemma (**in** *ground-ordered-resolution-calculus*) *termination-ground-factoring*:
shows *wfP ground-factoring*$^{-1-1}$
proof (*rule wfp-if-convertible-to-wfp*)
show ⋀*x y*. *ground-factoring*$^{-1-1}$ *x y* ⟹ *x* ≺$_c$ *y*
using *ground-factoring-smaller-conclusion* **by** *simp*
next
show *wfP* (≺$_c$)
by *simp*
qed

lemma (**in** *ground-ordered-resolution-calculus*) *atms-of-concl-subset-if-ground-resolution*:
assumes *ground-resolution* P_1 P_2 C
shows *atms-of* $C \subseteq$ *atms-of* $P_1 \cup$ *atms-of* P_2
using *assms* **by** (*cases* P_1 P_2 C *rule*: *ground-resolution.cases*) (*auto simp add*: *atms-of-def*)

lemma (**in** *ground-ordered-resolution-calculus*) *strict-subset-mset-if-ground-factoring*:
assumes *ground-factoring* P C
shows $C \subset\# P$
using *assms* **by** (*cases* P C *rule*: *ground-factoring.cases*) *simp*

lemma (**in** *ground-ordered-resolution-calculus*) *set-mset-eq-set-mset-if-ground-factoring*:
assumes *ground-factoring* P C
shows *set-mset* $P =$ *set-mset* C
using *assms* **by** (*cases* P C *rule*: *ground-factoring.cases*) *simp*

lemma (**in** *ground-ordered-resolution-calculus*) *atms-of-concl-eq-if-ground-factoring*:
assumes *ground-factoring* P C
shows *atms-of* $C =$ *atms-of* P
using *assms* **by** (*cases* P C *rule*: *ground-factoring.cases*) *simp*

lemma (**in** *ground-ordered-resolution-calculus*) *ground-factoring-preserves-maximal-literal*:
assumes *ground-factoring* P C
shows *is-maximal-lit* L $P =$ *is-maximal-lit* L C
using *assms* **by** (*cases* P C *rule*: *ground-factoring.cases*) (*simp add*: *is-maximal-in-mset-wrt-iff*)

lemma (**in** *ground-ordered-resolution-calculus*) *ground-factorings-preserves-maximal-literal*:
assumes *ground-factoring*** P C
shows *is-maximal-lit* L $P =$ *is-maximal-lit* L C
using *assms*
by (*induction* P *rule*: *converse-rtranclp-induct*)
(*simp-all add*: *ground-factoring-preserves-maximal-literal*)

lemma (**in** *ground-ordered-resolution-calculus*) *ground-factoring-reduces-maximal-pos-lit*:
assumes *ground-factoring* P C **and** *is-pos* L **and**
is-maximal-lit L P **and** *count* P $L =$ *Suc* (*Suc* n)
shows *is-maximal-lit* L C **and** *count* C $L =$ *Suc* n
unfolding *atomize-conj*
using ‹*ground-factoring* P C›
proof (*cases* P C *rule*: *ground-factoring.cases*)
case (*ground-factoringI* t P')
then show *is-maximal-lit* L $C \wedge$ *count* C $L =$ *Suc* n
by (*metis assms*(*1*) *assms*(*3*) *assms*(*4*) *asymp-on-less-lit count-add-mset*
ground-factoring-preserves-maximal-literal is-maximal-in-mset-wrt-iff nat.inject
totalpD
totalp-less-lit transp-on-less-lit)
qed

lemma (**in** *ground-ordered-resolution-calculus*) *ground-factorings-reduces-maximal-pos-lit*:
assumes (*ground-factoring* ^^ *m*) *P C* **and** $m \leq Suc\ n$ **and** *is-pos L* **and**
is-maximal-lit L P **and** *count P L* = *Suc* (*Suc n*)
shows *is-maximal-lit L C* **and** *count C L* = *Suc* (*Suc n* − *m*)
unfolding *atomize-conj*
using ‹(*ground-factoring* ^^ *m*) *P C*› ‹$m \leq Suc\ n$›
proof (*induction m arbitrary*: *C*)
case *0*
hence *P* = *C*
by *simp*
then show *?case*
using *assms*(*4*,*5*) **by** *simp*
next
case (*Suc m'*)
then show *?case*
by (*metis Suc-diff-le Suc-leD assms*(*3*) *diff-Suc-Suc ground-factoring-reduces-maximal-pos-lit*(*2*)
ground-factorings-preserves-maximal-literal relpowp-Suc-E relpowp-imp-rtranclp)
qed

lemma (**in** *ground-ordered-resolution-calculus*) *full-ground-factorings-reduces-maximal-pos-lit*:
assumes *steps*: (*ground-factoring* ^^ *Suc n*) *P C* **and** *L-pos*: *is-pos L* **and**
L-max: *is-maximal-lit L P* **and** *L-count*: *count P L* = *Suc* (*Suc n*)
shows *is-maximal-lit L C* **and** *count C L* = *Suc 0*
unfolding *atomize-conj*
using *steps L-max L-count*
proof (*induction n arbitrary*: *P*)
case *0*
then show *?case*
using *L-pos ground-factorings-reduces-maximal-pos-lit*[*of Suc 0 P C 0*] **by** *simp*
next
case (*Suc n*)
from *Suc.prems* **obtain** *P'* **where**
ground-factoring P P' **and** (*ground-factoring* ^^ *Suc n*) *P' C*
by (*metis relpowp-Suc-D2*)

from *Suc.prems* **have** *is-maximal-lit L P'* **and** *count P' L* = *Suc* (*Suc n*)
using *ground-factoring-reduces-maximal-pos-lit*[*OF* ‹*ground-factoring P P'*›
L-pos] **by** *simp-all*

thus *?case*
using *Suc.IH*[*OF* ‹(*ground-factoring* ^^ *Suc n*) *P' C*›] **by** *metis*
qed

7 Move somewhere?

lemma *true-cls-imp-neq-mempty*: $\mathcal{I} \models C \Longrightarrow C \neq \{\#\}$
by *blast*

lemma *lift-tranclp-to-pairs-with-constant-fst*:

$(R\ x)^{++}\ y\ z \Longrightarrow (\lambda(x,\ y)\ (x',\ z).\ x = x' \wedge R\ x\ y\ z)^{++}\ (x,\ y)\ (x,\ z)$
by (*induction z arbitrary*: *rule*: *tranclp-induct*) (*auto simp*: *tranclp.trancl-into-trancl*)

abbreviation (**in** *preorder*) *is-lower-fset* **where**
is-lower-fset X Y ≡ *is-lower-set-wrt* (<) (*fset X*) (*fset Y*)

lemma *lower-set-wrt-prefixI*:
assumes
trans: *transp-on* (*set zs*) *R* **and**
asym: *asymp-on* (*set zs*) *R* **and**
sorted: *sorted-wrt R zs* **and**
prefix: *prefix xs zs*
shows *is-lower-set-wrt R* (*set xs*) (*set zs*)
proof −
obtain *ys* **where** *zs* = *xs* @ *ys*
using *prefix* **by** (*auto elim*: *prefixE*)

show *?thesis*
using *trans asym sorted*
unfolding ‹*zs* = *xs* @ *ys*›
by (*metis lower-set-wrt-appendI*)
qed

lemmas (**in** *preorder*) *lower-set-prefixI* =
lower-set-wrt-prefixI[*OF transp-on-less asymp-on-less*]

lemma *lower-set-wrt-suffixI*:
assumes
trans: *transp-on* (*set zs*) *R* **and**
asym: *asymp-on* (*set zs*) *R* **and**
sorted: *sorted-wrt* R^{-1-1} *zs* **and**
suffix: *suffix ys zs*
shows *is-lower-set-wrt R* (*set ys*) (*set zs*)
proof −
obtain *xs* **where** *zs* = *xs* @ *ys*
using *suffix* **by** (*auto elim*: *suffixE*)

show *?thesis*
using *trans asym sorted*
unfolding ‹*zs* = *xs* @ *ys*›
by (*smt* (*verit*, *del-insts*) *Un-iff* ‹*zs* = *xs* @ *ys*› *asymp-onD asymp-on-conversep conversepI*
is-lower-set-wrt-def set-append set-mono-suffix sorted-wrt-append suffix)
qed

lemmas (**in** *preorder*) *lower-set-suffixI* =
lower-set-wrt-suffixI[*OF transp-on-less asymp-on-less*]

lemma *true-cls-repeat-mset-Suc*[*simp*]: $I \models$ *repeat-mset* (*Suc n*) $C \longleftrightarrow I \models C$
by (*induction n*) *simp-all*

lemma (**in** *backward-simulation*)
assumes *match i S1 S2* **and** ¬ *L1.inf-step S1*
shows ¬ *L2.inf-step S2*
using *assms match-inf* **by** *metis*

lemma (**in** *scl-fol-calculus*) *grounding-of-clss-ground*:
assumes *is-ground-clss N*
shows *grounding-of-clss N* = *N*
proof −
have *grounding-of-clss N* = ($\bigcup$ *C* ∈ *N*. *grounding-of-cls C*)
unfolding *grounding-of-clss-def* **by** *simp*
also have ... = ($\bigcup$ *C* ∈ *N*. {*C*})
using ‹*is-ground-clss N*›
by (*simp add*: *is-ground-clss-def grounding-of-cls-ground*)
also have ... = *N*
by *simp*
finally show *?thesis* **.**
qed

lemma (**in** *scl-fol-calculus*) *propagateI'*:
C |∈| *N* |∪| *U* ⟹ *C* = *add-mset L C'* ⟹ *is-ground-cls* (*C* · γ) ⟹
∀ *K* ∈# *C* · γ. *atm-of K* ≼*B* β ⟹
*C*0 = {#*K* ∈# *C'*. *K* ·*l* γ ≠ *L* ·*l* γ#} ⟹ *C*1 = {#*K* ∈# *C'*. *K* ·*l* γ = *L* ·*l* γ#} ⟹
SCL-FOL.trail-false-cls Γ (*C*0 · γ) ⟹ ¬ *SCL-FOL.trail-defined-lit* Γ (*L* ·*l* γ) ⟹
is-imgu μ {*atm-of* ‘ *set-mset* (*add-mset L C*1)} ⟹
Γ' = *trail-propagate* Γ (*L* ·*l* μ) (*C*0 · μ) γ ⟹
propagate N β (Γ, *U*, *None*) (Γ', *U*, *None*)
using *propagateI* **by** *metis*

lemma (**in** *scl-fol-calculus*) *decideI'*:
is-ground-lit (*L* ·*l* γ) ⟹ ¬ *SCL-FOL.trail-defined-lit* Γ (*L* ·*l* γ) ⟹ *atm-of L* ·*a* γ ≼*B* β ⟹
Γ' = *trail-decide* Γ (*L* ·*l* γ) ⟹
decide N β (Γ, *U*, *None*) (Γ', *U*, *None*)
by (*auto intro*!: *decideI*)

lemma *ground-iff-vars-term-empty*: *ground t* ⟷ *vars-term t* = {}
proof (*rule iffI*)
show *ground t* ⟹ *vars-term t* = {}
by (*rule ground-vars-term-empty*)
next
show *vars-term t* = {} ⟹ *ground t*
by (*induction t*) *simp-all*
qed

lemma *is-ground-atm-eq-ground*[*iff*]: *is-ground-atm* = *ground*
proof (*rule ext*)
fix *t* :: ('*v*, '*f*) *Term.term*
show *is-ground-atm t* = *ground t*
by (*simp only*: *is-ground-atm-iff-vars-empty ground-iff-vars-term-empty*)
qed

definition *lit-of-glit* :: '*f gterm literal* ⇒ ('*f*, '*v*) *term literal* **where**
lit-of-glit = *map-literal term-of-gterm*

definition *glit-of-lit* **where**
glit-of-lit = *map-literal gterm-of-term*

definition *cls-of-gcls* **where**
cls-of-gcls = *image-mset lit-of-glit*

definition *gcls-of-cls* **where**
gcls-of-cls = *image-mset glit-of-lit*

lemma *inj-lit-of-glit*: *inj lit-of-glit*
proof (*rule injI*)
fix *L K*
show *lit-of-glit L* = *lit-of-glit K* ⟹ *L* = *K*
by (*metis lit-of-glit-def literal.expand literal.map-disc-iff literal.map-sel term-of-gterm-inv*)
qed

lemma *atm-of-lit-of-glit-conv*: *atm-of* (*lit-of-glit L*) = *term-of-gterm* (*atm-of L*)
by (*cases L*) (*simp-all add*: *lit-of-glit-def*)

lemma *ground-atm-of-lit-of-glit*[*simp*]: *Term-Context.ground* (*atm-of* (*lit-of-glit L*))
by (*simp add*: *atm-of-lit-of-glit-conv*)

lemma *is-ground-lit-lit-of-glit*[*simp*]: *is-ground-lit* (*lit-of-glit L*)
by (*simp add*: *is-ground-lit-def atm-of-lit-of-glit-conv*)

lemma *is-ground-cls-cls-of-gcls*[*simp*]: *is-ground-cls* (*cls-of-gcls C*)
by (*auto simp add*: *is-ground-cls-def cls-of-gcls-def*)

lemma *glit-of-lit-lit-of-glit*[*simp*]: *glit-of-lit* (*lit-of-glit L*) = *L*
by (*simp add*: *glit-of-lit-def lit-of-glit-def literal.map-comp comp-def literal.map-ident*)

lemma *gcls-of-cls-cls-of-gcls*[*simp*]: *gcls-of-cls* (*cls-of-gcls L*) = *L*
by (*simp add*: *gcls-of-cls-def cls-of-gcls-def multiset.map-comp comp-def multiset.map-ident*)

lemma *lit-of-glit-glit-of-lit-ident*[*simp*]: *is-ground-lit L* ⟹ *lit-of-glit* (*glit-of-lit L*) = *L*
by (*simp add*: *is-ground-lit-def lit-of-glit-def glit-of-lit-def literal.map-comp lit-*

eral.expand
literal.map-sel)

lemma *cls-of-gcls-gcls-of-cls-ident*[*simp*]: *is-ground-cls D* ⟹ *cls-of-gcls* (*gcls-of-cls D*) = *D*
by (*simp add*: *is-ground-cls-def cls-of-gcls-def gcls-of-cls-def*)

lemma *vars-lit-lit-of-glit*[*simp*]: *vars-lit* (*lit-of-glit L*) = {}
by *simp*

lemma *vars-cls-cls-of-gcls*[*simp*]: *vars-cls* (*cls-of-gcls C*) = {}
by (*metis is-ground-cls-cls-of-gcls is-ground-cls-iff-vars-empty*)

definition *atms-of-cls* :: *'a clause* ⇒ *'a fset* **where**
atms-of-cls C = *atm-of* |‘| *fset-mset C*

definition *atms-of-clss* :: *'a clause fset* ⇒ *'a fset* **where**
atms-of-clss N = *ffUnion* (*atms-of-cls* |‘| *N*)

lemma *atms-of-clss-fempty*[*simp*]: *atms-of-clss* {||} = {||}
unfolding *atms-of-clss-def* **by** *simp*

lemma *atms-of-clss-finsert*[*simp*]:
atms-of-clss (*finsert C N*) = *atms-of-cls C* |∪| *atms-of-clss N*
unfolding *atms-of-clss-def* **by** *simp*

definition *lits-of-clss* :: *'a clause fset* ⇒ *'a literal fset* **where**
lits-of-clss N = *ffUnion* (*fset-mset* |‘| *N*)

definition *lit-occures-in-clss* **where**
lit-occures-in-clss L N ⟷ *fBex N* (λ*C*. *L* ∈# *C*)

inductive *constant-context* **for** *R* **where**
R C D D' ⟹ *constant-context R* (*C*, *D*) (*C*, *D'*)

lemma *rtranclp-constant-context*: (*R C*)** *D D'* ⟹ (*constant-context R*)** (*C*, *D*) (*C*, *D'*)
by (*induction D' rule*: *rtranclp-induct*) (*auto intro*: *constant-context.intros rtranclp.intros*)

lemma *tranclp-constant-context*: (*R C*)++ *D D'* ⟹ (*constant-context R*)++ (*C*, *D*) (*C*, *D'*)
by (*induction D' rule*: *tranclp-induct*) (*auto intro*: *constant-context.intros tranclp.intros*)

lemma *right-unique-constant-context*:
assumes *R-ru*: ⋀*C*. *right-unique* (*R C*)
shows *right-unique* (*constant-context R*)
proof (*rule right-uniqueI*)

```
  fix x y z
  show constant-context R x y ⟹ constant-context R x z ⟹ y = z
    using R-ru[THEN right-uniqueD]
    by (elim constant-context.cases) (metis prod.inject)
qed

lemma safe-state-constant-context-if-invars:
  fixes N s
  assumes
    R-preserves-I:
      ⋀N s s'. R N s s' ⟹ I N s ⟹ I N s' and
    ex-R-if-not-final:
      ⋀N s. ¬ F (N, s) ⟹ I N s ⟹ ∃ s'. R N s s'
  assumes invars: I N s
  shows safe-state (constant-context R) F (N, s)
proof −
  {
    fix S'
   assume (constant-context R)** (N, s) S' and stuck-state (constant-context R)
S'
    then obtain s' where S' = (N, s') and (R N)** s s' and I N s'
      using invars
    proof (induction (N, s) arbitrary: N s rule: converse-rtranclp-induct)
      case base
      thus ?case by simp
    next
      case (step z)
      thus ?case
      by (metis (no-types, opaque-lifting) Pair-inject R-preserves-I constant-context.cases
          converse-rtranclp-into-rtranclp)
    qed
    hence ¬ F (N, s') ⟹ ∃ s''. R N s' s''
      using ex-R-if-not-final[of N s'] by argo
    hence ¬ F S' ⟹ ∃ S''. constant-context R S' S''
      unfolding ‹S' = (N, s')› using constant-context.intros by metis
    hence F S'
      by (metis ‹stuck-state (constant-context R) S'› stuck-state-def)
  }
  thus ?thesis
    by (metis (no-types) safe-state-def stuck-state-def)
qed

primrec trail-atms :: (- literal × -) list ⇒ - fset where
  trail-atms [] = {||} |
  trail-atms (Ln # Γ) = finsert (atm-of (fst Ln)) (trail-atms Γ)

lemma fset-trail-atms: fset (trail-atms Γ) = atm-of ‘ fst ‘ set Γ
  by (induction Γ) simp-all
```

lemma *trail-defined-lit-iff-trail-defined-atm*:
  trail-defined-lit Γ *L* ⟷ *atm-of L* |∈| *trail-atms* Γ
proof (*induction* Γ)
  case *Nil*
  show *?case*
    by (*simp add*: *trail-defined-lit-def*)
next
  case (*Cons Ln* Γ)

  have *trail-defined-lit* (*Ln* # Γ) *L* ⟷ *L* = *fst Ln* ∨ − *L* = *fst Ln* ∨ *trail-defined-lit* Γ *L*
    unfolding *trail-defined-lit-def* **by** *auto*

  also have ... ⟷ *atm-of L* = *atm-of* (*fst Ln*) ∨ *trail-defined-lit* Γ *L*
    by (*cases L*; *cases fst Ln*) *simp-all*

  also have ... ⟷ *atm-of L* = *atm-of* (*fst Ln*) ∨ *atm-of L* |∈| *trail-atms* Γ
    unfolding *Cons.IH* **..**

  also have ... ⟷ *atm-of L* |∈| *trail-atms* (*Ln* # Γ)
    by *simp*

  finally show *?case* **.**
qed

lemma *trail-atms-subset-if-suffix*:
  assumes *suffix* Γ′ Γ
  shows *trail-atms* Γ′ |⊆| *trail-atms* Γ
proof −
  obtain Γ$_0$ **where** Γ = Γ$_0$ @ Γ′
    using *assms* **unfolding** *suffix-def* **by** *metis*

  show *?thesis*
    unfolding ‹Γ = Γ$_0$ @ Γ′›
    by (*induction* Γ$_0$) *auto*
qed

lemma *dom-model-eq-trail-interp*:
  assumes
    ∀ *A C*. *M A* = *Some C* ⟷ *map-of* Γ (*Pos A*) = *Some* (*Some C*) **and**
    ∀ *Ln* ∈ *set* Γ. ∀ *L*. *Ln* = (*L*, *None*) ⟶ *is-neg L*
  shows *dom M* = *trail-interp* Γ
proof −
  have *dom M* = {*A*. ∃ *C*. *M A* = *Some C*}
    unfolding *dom-def* **by** *simp*
  also have ... = {*A*. ∃ *C*. *map-of* Γ (*Pos A*) = *Some* (*Some C*)}
    using *assms*(*1*) **by** *metis*
  also have ... = {*A*. ∃ *opt*. *map-of* Γ (*Pos A*) = *Some opt*}
  proof (*rule Collect-cong*)

show $\bigwedge A.\ (\exists\, C.\ \mathit{map\text{-}of}\ \Gamma\ (\mathit{Pos}\ A) = \mathit{Some}\ (\mathit{Some}\ C)) \longleftrightarrow (\exists\, \mathit{opt}.\ \mathit{map\text{-}of}\ \Gamma\ (\mathit{Pos}\ A) = \mathit{Some}\ \mathit{opt})$
using *assms*(2)
by (*metis literal.disc*(1) *map-of-SomeD option.exhaust*)
qed
also have $\ldots = \mathit{trail\text{-}interp}\ \Gamma$
proof (*induction* Γ)
case *Nil*
thus *?case*
by (*simp add*: *trail-interp-def*)
next
case (*Cons Ln* Γ)

have $\{A.\ \exists\, \mathit{opt}.\ \mathit{map\text{-}of}\ (\mathit{Ln}\ \#\ \Gamma)\ (\mathit{Pos}\ A) = \mathit{Some}\ \mathit{opt}\} =$
$\{A.\ \exists\, \mathit{opt}.\ \mathit{map\text{-}of}\ [\mathit{Ln}]\ (\mathit{Pos}\ A) = \mathit{Some}\ \mathit{opt}\} \cup \{A.\ \exists\, \mathit{opt}.\ \mathit{map\text{-}of}\ \Gamma\ (\mathit{Pos}\ A) = \mathit{Some}\ \mathit{opt}\}$
by *auto*

also have $\ldots = \{A.\ \mathit{Pos}\ A = \mathit{fst}\ \mathit{Ln}\} \cup \mathit{trail\text{-}interp}\ \Gamma$
unfolding *Cons.IH* **by** *simp*

also have $\ldots = \mathit{trail\text{-}interp}\ [\mathit{Ln}] \cup \mathit{trail\text{-}interp}\ \Gamma$
by (*cases fst Ln*) (*simp-all add*: *trail-interp-def*)

also have $\ldots = \mathit{trail\text{-}interp}\ (\mathit{Ln}\ \#\ \Gamma)$
unfolding *trail-interp-Cons*[*of Ln* Γ] **..**

finally show *?case* **.**
qed
finally show *?thesis* **.**
qed

type-synonym $'f\ \mathit{gliteral} = 'f\ \mathit{gterm}\ \mathit{literal}$
type-synonym $'f\ \mathit{gclause} = 'f\ \mathit{gterm}\ \mathit{clause}$

locale *simulation-SCLFOL-ground-ordered-resolution* =
renaming-apart renaming-vars
for *renaming-vars* :: $'v\ \mathit{set} \Rightarrow 'v \Rightarrow 'v$ +
fixes
less-trm :: $'f\ \mathit{gterm} \Rightarrow 'f\ \mathit{gterm} \Rightarrow \mathit{bool}$ (**infix** $\prec_t$ 50)
assumes
transp-less-trm[*simp*]: *transp* $(\prec_t)$ **and**
asymp-less-trm[*intro*]: *asymp* $(\prec_t)$ **and**
wfP-less-trm[*intro*]: *wfP* $(\prec_t)$ **and**
totalp-less-trm[*intro*]: *totalp* $(\prec_t)$ **and**

finite-less-trm: $\bigwedge\beta.$ *finite* $\{x.\ x \prec_t \beta\}$ **and**
less-trm-compatible-with-gctxt[*simp*]: $\bigwedge ctxt\ t\ t'.\ t \prec_t t' \Longrightarrow ctxt\langle t\rangle_G \prec_t ctxt\langle t'\rangle_G$
and
less-trm-if-subterm[*simp*]: $\bigwedge t\ ctxt.\ ctxt \neq \Box_G \Longrightarrow t \prec_t ctxt\langle t\rangle_G$

8 Ground ordered resolution for ground terms

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

sublocale *ord-res*: *ground-ordered-resolution-calculus* $(\prec_t)$ λ-. $\{\#\}$
by *unfold-locales auto*

sublocale *linorder-trm*: *linorder* $(\preceq_t)$ $(\prec_t)$
proof *unfold-locales*
show $\bigwedge x\ y.\ (x \prec_t y) = (x \preceq_t y \wedge \neg\ y \preceq_t x)$
by (*metis asympD asymp-less-trm reflclp-iff*)
next
show $\bigwedge x.\ x \preceq_t x$
by *simp*
next
show $\bigwedge x\ y\ z.\ x \preceq_t y \Longrightarrow y \preceq_t z \Longrightarrow x \preceq_t z$
by (*meson transpE transp-less-trm transp-on-reflclp*)
next
show $\bigwedge x\ y.\ x \preceq_t y \Longrightarrow y \preceq_t x \Longrightarrow x = y$
by (*metis asympD asymp-less-trm reflclp-iff*)
next
show $\bigwedge x\ y.\ x \preceq_t y \vee y \preceq_t x$
by (*metis reflclp-iff totalpD totalp-less-trm*)
qed

sublocale *linorder-lit*: *linorder* $(\preceq_l)$ $(\prec_l)$
proof *unfold-locales*
show $\bigwedge x\ y.\ (x \prec_l y) = (x \preceq_l y \wedge \neg\ y \preceq_l x)$
by (*metis asympD ord-res.asymp-less-lit reflclp-iff*)
next
show $\bigwedge x.\ x \preceq_l x$
by *simp*
next
show $\bigwedge x\ y\ z.\ x \preceq_l y \Longrightarrow y \preceq_l z \Longrightarrow x \preceq_l z$
by (*meson transpE ord-res.transp-less-lit transp-on-reflclp*)
next
show $\bigwedge x\ y.\ x \preceq_l y \Longrightarrow y \preceq_l x \Longrightarrow x = y$
by (*metis asympD ord-res.asymp-less-lit reflclp-iff*)
next
show $\bigwedge x\ y.\ x \preceq_l y \vee y \preceq_l x$
by (*metis reflclp-iff totalpD ord-res.totalp-less-lit*)
qed

sublocale *linorder-cls*: *linorder* $(\preceq_c)$ $(\prec_c)$

proof *unfold-locales*
show $\bigwedge x\ y.\ (x \prec_c y) = (x \preceq_c y \wedge \neg\ y \preceq_c x)$
by (*metis asympD ord-res.asymp-less-cls reflclp-iff*)
next
show $\bigwedge x.\ x \preceq_c x$
by *simp*
next
show $\bigwedge x\ y\ z.\ x \preceq_c y \Longrightarrow y \preceq_c z \Longrightarrow x \preceq_c z$
by (*meson transpE ord-res.transp-less-cls transp-on-reflclp*)
next
show $\bigwedge x\ y.\ x \preceq_c y \Longrightarrow y \preceq_c x \Longrightarrow x = y$
by (*metis asympD ord-res.asymp-less-cls reflclp-iff*)
next
show $\bigwedge x\ y.\ x \preceq_c y \vee y \preceq_c x$
by (*metis reflclp-iff totalpD ord-res.totalp-less-cls*)
qed

declare *linorder-trm.is-least-in-fset-ffilterD*[*no-atp*]
declare *linorder-lit.is-least-in-fset-ffilterD*[*no-atp*]
declare *linorder-cls.is-least-in-fset-ffilterD*[*no-atp*]

end

9 Common definitions and lemmas

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

abbreviation *ord-res-Interp* **where**
ord-res-Interp N C ≡ *ord-res.interp N C* ∪ *ord-res.production N C*

definition *is-least-false-clause* **where**
is-least-false-clause N C ⟷
linorder-cls.is-least-in-fset {|*C* |∈| *N*. ¬ *ord-res-Interp* (*fset N*) *C* ⊨ *C*|} *C*

lemma *is-least-false-clause-finsert-smaller-false-clause*:
assumes
D-least: *is-least-false-clause N D* **and**
C ≺$_c$ *D* **and**
C-false: ¬ *ord-res-Interp* (*fset* (*finsert C N*)) *C* ⊨ *C*
shows *is-least-false-clause* (*finsert C N*) *C*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *C* |∈| *finsert C N*
by *simp*
next
show ¬ *ord-res-Interp* (*fset* (*finsert C N*)) *C* ⊨ *C*
using *assms* **by** *metis*
next
fix *y*

```
  assume y |∈| finsert C N and y ≠ C and y-false: ¬ ord-res-Interp (fset (finsert
C N)) y ⊨ y
  hence y |∈| N
    by simp

  have ¬ (y ≺c C)
  proof (rule notI)
    assume y ≺c C
    hence ord-res-Interp (fset (finsert C N)) y = ord-res-Interp (fset N) y
      using ord-res.Interp-insert-greater-clause by simp

    hence ¬ ord-res-Interp (fset N) y ⊨ y
      using y-false by argo

    moreover have y ≺c D
      using ‹y ≺c C› ‹C ≺c D› by order

    ultimately show False
      using D-least
      by (metis (mono-tags, lifting) ‹y |∈| N› linorder-cls.is-least-in-ffilter-iff
          linorder-cls.less-asym' is-least-false-clause-def)
  qed
  thus C ≺c y
    using ‹y ≠ C› by order
qed

lemma is-least-false-clause-swap-swap-compatible-fsets:
  assumes {|x |∈| N1. x ⪯c C|} = {|x |∈| N2. x ⪯c C|}
  shows is-least-false-clause N1 C ⟷ is-least-false-clause N2 C
proof −
  have is-least-false-clause N2 C if
    subsets-agree: {|x |∈| N1. x ⪯c C|} = {|x |∈| N2. x ⪯c C|} and
    C-least: is-least-false-clause N1 C for N1 N2 C
    unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
  proof (intro conjI ballI impI)
    have C |∈| N1
      using C-least
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
      by argo
    thus C |∈| N2
      using subsets-agree by auto
  next
    have ¬ ord-res-Interp (fset N1) C ⊨ C
      using C-least
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
      by argo
    moreover have ord-res-Interp (fset N1) C = ord-res-Interp (fset N2) C
      using subsets-agree by (auto intro!: ord-res.Interp-swap-clause-set)
    ultimately show ¬ ord-res-Interp (fset N2) C ⊨ C
```

```
    by argo
  next
    fix y assume y |∈| N2 and y ≠ C
    show ¬ ord-res-Interp (fset N2) y ⊨ y ⟹ C ≺c y
    proof (erule contrapos-np)
      assume ¬ C ≺c y
      hence y ⪯c C
        by order
      hence y |∈| N1
        using ‹y |∈| N2› using subsets-agree by auto
      hence ¬ ord-res-Interp (fset N1) y ⊨ y ⟶ C ≺c y
        using ‹is-least-false-clause N1 C› ‹y ≠ C›
        unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
        by metis
      moreover have ord-res-Interp (fset N1) y = ord-res-Interp (fset N2) y
      proof (rule ord-res.Interp-swap-clause-set)
        show {D. D |∈| N1 ∧ (≺c)== D y} = {D. D |∈| N2 ∧ (≺c)== D y}
          using subsets-agree ‹y ⪯c C› by fastforce
      qed
      ultimately show ord-res-Interp (fset N2) y ⊨ y
        using ‹y ⪯c C› by auto
    qed
  qed
  thus ?thesis
    using assms by metis
qed

lemma Uniq-is-least-false-clause: ∃≤1 C. is-least-false-clause N C
proof (rule Uniq-I)
  show ⋀x y z. is-least-false-clause x y ⟹ is-least-false-clause x z ⟹ y = z
    unfolding is-least-false-clause-def
    by (meson Uniq-D linorder-cls.Uniq-is-least-in-fset)
qed

lemma mempty-lesseq-cls[simp]: {#} ⪯c C for C
  by (cases C) (simp-all add: strict-subset-implies-multp)

lemma is-least-false-clause-mempty: {#} |∈| N ⟹ is-least-false-clause N {#}
  using is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff mempty-lesseq-cls
  by fastforce

lemma production-union-unproductive-strong:
  assumes
    fin: finite N1 finite N2 and
    N2-unproductive: ∀ x ∈ N2 − N1. ord-res.production (N1 ∪ N2) x = {} and
    C-in: C ∈ N1
  shows ord-res.production (N1 ∪ N2) C = ord-res.production N1 C
  using ord-res.wfP-less-cls C-in
proof (induction C rule: wfp-induct-rule)
```

```
  case (less C)
  hence C-in-iff: C ∈ N1 ∪ N2 ⟷ C ∈ N1
    by simp

  have interp-eq: ord-res.interp (N1 ∪ N2) C = ord-res.interp N1 C
  proof −
    have ord-res.interp (N1 ∪ N2) C = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈
N1 ∪ N2. D ≺c C})
      unfolding ord-res.interp-def ..
    also have ... = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈ N1. D ≺c C} ∪
    ord-res.production (N1 ∪ N2) ‘ {D ∈ N2 − N1. D ≺c C})
      by auto
    also have ... = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈ N1. D ≺c C})
      using N2-unproductive by simp
    also have ... = ⋃ (ord-res.production N1 ‘ {D ∈ N1. D ≺c C})
      using less.IH by simp
    also have ... = ord-res.interp N1 C
      unfolding ord-res.interp-def ..
    finally show ord-res.interp (N1 ∪ N2) C = ord-res.interp N1 C .
  qed

  show ?case
    unfolding ord-res.production-unfold C-in-iff interp-eq by argo
qed

lemma production-union-unproductive:
  assumes
    fin: finite N1 finite N2 and
    N2-unproductive: ∀ x ∈ N2. ord-res.production (N1 ∪ N2) x = {} and
    C-in: C ∈ N1
  shows ord-res.production (N1 ∪ N2) C = ord-res.production N1 C
  using production-union-unproductive-strong assms by simp

lemma interp-union-unproductive:
  assumes
    fin: finite N1 finite N2 and
    N2-unproductive: ∀ x ∈ N2. ord-res.production (N1 ∪ N2) x = {}
  shows ord-res.interp (N1 ∪ N2) = ord-res.interp N1
proof (rule ext)
  fix C
  have ord-res.interp (N1 ∪ N2) C = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈
N1 ∪ N2. D ≺c C})
    unfolding ord-res.interp-def ..
  also have ... = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈ N1. D ≺c C} ∪
    ord-res.production (N1 ∪ N2) ‘ {D ∈ N2. D ≺c C})
    by auto
  also have ... = ⋃ (ord-res.production (N1 ∪ N2) ‘ {D ∈ N1. D ≺c C})
    using N2-unproductive by simp
  also have ... = ⋃ (ord-res.production N1 ‘ {D ∈ N1. D ≺c C})
```

using *production-union-unproductive*[*OF fin N2-unproductive*] **by** *simp*
also have ... = *ord-res.interp N1 C*
unfolding *ord-res.interp-def* **..**
finally show *ord-res.interp* (*N1* ∪ *N2*) *C* = *ord-res.interp N1 C* **.**
qed

lemma *Interp-union-unproductive*:
assumes
fin: *finite N1 finite N2* **and**
N2-unproductive: ∀ *x* ∈ *N2*. *ord-res.production* (*N1* ∪ *N2*) *x* = {}
shows *ord-res-Interp* (*N1* ∪ *N2*) *C* = *ord-res-Interp N1 C*
unfolding *interp-union-unproductive*[*OF assms*]
using *production-union-unproductive*[*OF assms*]
using *N2-unproductive*[*rule-format*]
by (*metis* (*no-types*, *lifting*) *Un-iff empty-Collect-eq ord-res.production-unfold*)

lemma *Interp-insert-unproductive*:
assumes
fin: *finite N1* **and**
x-unproductive: *ord-res.production* (*insert x N1*) *x* = {}
shows *ord-res-Interp* (*insert x N1*) *C* = *ord-res-Interp N1 C*
using *assms Interp-union-unproductive*
by (*metis Un-commute finite.emptyI finite.insertI insert-is-Un singletonD*)

lemma *extended-partial-model-entails-iff-partial-model-entails*:
assumes
fin: *finite N finite N′* **and**
irrelevant: ∀ *D* ∈ *N′*. ∃ *E* ∈ *N*. *E* ⊂# *D* ∧ *set-mset D* = *set-mset E* **and**
C-in: *C* ∈ *N*
shows *ord-res-Interp* (*N* ∪ *N′*) *C* ⊨ *C* ⟷ *ord-res-Interp N C* ⊨ *C*
using *ord-res.interp-add-irrelevant-clauses-to-set*[*OF fin C-in irrelevant*]
using *ord-res.production-add-irrelevant-clauses-to-set*[*OF fin C-in irrelevant*]
by *metis*

lemma *nex-strictly-maximal-pos-lit-if-factorizable*:
assumes *ord-res.ground-factoring C C′*
shows ∄ *L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*
by (*metis Uniq-D add-mset-remove-trivial assms linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.dual-order.order-iff-strict linorder-lit.is-greatest-in-mset-iff
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.not-less
ord-res.ground-factoring.cases union-single-eq-member)

lemma *unproductive-if-nex-strictly-maximal-pos-lit*:
assumes ∄ *L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*
shows *ord-res.production N C* = {}
using *assms* **by** (*simp add*: *ord-res.production-unfold*)

lemma *ball-unproductive-if-nex-strictly-maximal-pos-lit*:
assumes ∀ *C* ∈ *N′*. ∄ *L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*

shows $\forall\, C \in N'.$ *ord-res.production* $(N \cup N')\ C = \{\}$
using *assms unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*

lemma *is-least-false-clause-finsert-cancel*:
assumes
C-unproductive: *ord-res.production* (*fset* (*finsert* $C\ N$)) $C = \{\}$ **and**
C-entailed-by-smaller: $\exists\, D \mid\in\mid N.\ D \prec_c C \wedge \{D\} \Vdash e \{C\}$
shows *is-least-false-clause* (*finsert* $C\ N$) = *is-least-false-clause* N
proof (*intro ext iffI*)
fix E
assume *E-least*: *is-least-false-clause* (*finsert* $C\ N$) E
hence
E-in: $E \mid\in\mid$ *finsert* $C\ N$ **and**
E-false: $\neg$ *ord-res-Interp* (*fset* (*finsert* $C\ N$)) $E \models E$ **and**
E-least: $(\forall\, y \mid\in\mid$ *finsert* $C\ N.\ y \neq E \longrightarrow \neg$ *ord-res-Interp* (*fset* (*finsert* $C\ N$)) $y \models y \longrightarrow E \prec_c y$)
unfolding *atomize-conj is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *metis*

obtain D **where**
$D \mid\in\mid N$ **and** $D \prec_c C$ **and** $\{D\} \Vdash e \{C\}$
using *C-entailed-by-smaller* **by** *metis*

show *is-least-false-clause* $N\ E$
proof (*cases* $C = E$)
case *True*

have $E \prec_c D$
proof (*rule E-least*[*rule-format*])
show $D \mid\in\mid$ *finsert* $C\ N$
using ‹$D \mid\in\mid N$› **by** *simp*
next
show $D \neq E$
using ‹$D \prec_c C$› ‹$C = E$› **by** *order*
next
show $\neg$ *ord-res-Interp* (*fset* (*finsert* $C\ N$)) $D \models D$
using *E-false*
proof (*rule contrapos-nn*)
assume *ord-res-Interp* (*fset* (*finsert* $C\ N$)) $D \models D$
thus *ord-res-Interp* (*fset* (*finsert* $C\ N$)) $E \models E$
using ‹$D \prec_c C$› ‹$C = E$› ‹$\{D\} \Vdash e \{C\}$› *ord-res.entailed-clause-stays-entailed*
by *auto*
qed
qed
hence *False*
using ‹$D \prec_c C$› ‹$C = E$› **by** *order*
thus *?thesis* **..**
next
case *False*

```
    show ?thesis
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      show E |∈| N
        using E-in ‹C ≠ E› by simp
    next
      have ord-res-Interp (fset (finsert C N)) E = ord-res-Interp (fset N) E
        using C-unproductive Interp-insert-unproductive by simp
      thus ¬ ord-res-Interp (fset N) E ⊨ E
        using E-false by argo
    next
     show ⋀y. y |∈| N ⟹ y ≠ E ⟹ ¬ ord-res-Interp (fset N) y ⊨ y ⟹ E ≺c
y
        using E-least C-unproductive Interp-insert-unproductive by auto
    qed
  qed
next
  fix E
  assume is-least-false-clause N E
  hence
    E-in: E |∈| N and
    E-false: ¬ ord-res-Interp (fset N) E ⊨ E and
    E-least: (∀ y |∈| N. y ≠ E ⟶ ¬ ord-res-Interp (fset N) y ⊨ y ⟶ E ≺c y)
    unfolding atomize-conj is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    by metis

  show is-least-false-clause (finsert C N) E
    unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
  proof (intro conjI ballI impI)
    show E |∈| finsert C N
      using E-in by simp
  next
    show ¬ ord-res-Interp (fset (finsert C N)) E ⊨ E
      using E-least E-false C-unproductive Interp-insert-unproductive by simp
  next
    fix y
   assume y |∈| finsert C N and y ≠ E and ¬ ord-res-Interp (fset (finsert C N))
y ⊨ y
    show E ≺c y
    proof (cases y = C)
      case True
      thus ?thesis
      using E-least ‹¬ ord-res-Interp (fset (finsert C N)) y ⊨ y›
    by (metis (no-types, lifting) C-entailed-by-smaller C-unproductive Interp-insert-unproductive
        finite-fset fset-simps(2) linorder-cls.dual-order.strict-trans
        ord-res.entailed-clause-stays-entailed true-clss-singleton)
    next
      case False
      thus ?thesis
```

```
      using E-least ‹y |∈| finsert C N› ‹y ≠ E› ‹¬ ord-res-Interp (fset (finsert C
N)) y ⊨ y›
      using C-unproductive Interp-insert-unproductive by auto
    qed
  qed
qed

lemma is-least-false-clause-funion-cancel-right-strong:
  assumes
    ∀ C |∈| N2 − N1. ∀ U. ord-res.production U C = {} and
    ∀ C |∈| N2 − N1. ∃ D |∈| N1. D ≺c C ∧ {D} ⊨e {C}
  shows is-least-false-clause (N1 |∪| N2) = is-least-false-clause N1
  using assms
proof (induction N2)
  case empty
  thus ?case
    by simp
next
  case (insert C N2)

  have IH: is-least-false-clause (N1 |∪| N2) = is-least-false-clause N1
  proof (rule insert.IH)
    show ∀ C|∈|N2 |−| N1. ∀ U. ord-res.production U C = {}
      using insert.prems(1) by auto
  next
    show ∀ C|∈|N2 |−| N1. ∃ D|∈|N1. D ≺c C ∧ {D} ⊨e {C}
      using insert.prems(2) by auto
  qed

  show ?case
  proof (cases C |∈| N1)
    case True
    hence is-least-false-clause (N1 |∪| finsert C N2) = is-least-false-clause (N1 |∪|
N2)
      by (simp add: finsert-absorb)
    also have ... = is-least-false-clause N1
      using IH .
    finally show ?thesis .
  next
    case False
    then show ?thesis
      using is-least-false-clause-finsert-cancel IH
        by (metis finsertCI fminusI funionI1 funion-finsert-right insert.prems(1)
insert.prems(2))
  qed
qed

lemma is-least-false-clause-funion-cancel-right:
  assumes
```

$\forall\, C \mid\in\mid N2.\ \forall\, U.\ \mathit{ord\text{-}res.production}\ U\ C = \{\}$ **and**
$\forall\, C \mid\in\mid N2.\ \exists\, D \mid\in\mid N1.\ D \prec_c C \wedge \{D\} \mid\!\models\! e\ \{C\}$
shows *is-least-false-clause* (*N1* $\mid\cup\mid$ *N2*) = *is-least-false-clause N1*
using *assms is-least-false-clause-funion-cancel-right-strong* **by** *simp*

definition *ex-false-clause* **where**
ex-false-clause N = ($\exists\, C \in N.\ \neg$ *ord-res.interp N C* $\cup$ *ord-res.production N C* $\models$ *C*)

lemma *obtains-least-false-clause-if-ex-false-clause*:
assumes *ex-false-clause* (*fset N*)
obtains *C* **where** *is-least-false-clause N C*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *bot-fset.rep-eq emptyE ex-false-clause-def ffmember-filter*
linorder-cls.ex-least-in-fset is-least-false-clause-def)

lemma *ex-false-clause-if-least-false-clause*:
assumes *is-least-false-clause N C*
shows *ex-false-clause* (*fset N*)
using *assms*
by (*metis* (*no-types*, *lifting*) *ex-false-clause-def is-least-false-clause-def*
linorder-cls.is-least-in-fset-ffilterD(*1*) *linorder-cls.is-least-in-fset-ffilterD*(*2*))

lemma *ex-false-clause-iff*: *ex-false-clause* (*fset N*) $\longleftrightarrow$ ($\exists\, C.$ *is-least-false-clause N C*)
using *obtains-least-false-clause-if-ex-false-clause ex-false-clause-if-least-false-clause*
by *metis*

definition *ord-res-model* **where**
ord-res-model N = ($\bigcup D \in N.$ *ord-res.production N D*)

lemma *ord-res-model-eq-interp-union-production-of-greatest-clause*:
assumes *C-greatest*: *linorder-cls.is-greatest-in-set N C*
shows *ord-res-model N* = *ord-res.interp N C* $\cup$ *ord-res.production N C*
proof −
have *ord-res-model N* = ($\bigcup D \in N.$ *ord-res.production N D*)
unfolding *ord-res-model-def* **..**
also have ... = ($\bigcup D \in \{D \in N.\ D \preceq_c C\}.$ *ord-res.production N D*)
using *C-greatest linorder-cls.is-greatest-in-set-iff* **by** *auto*
also have ... = ($\bigcup D \in \{D \in N.\ D \prec_c C\} \cup \{C\}.$ *ord-res.production N D*)
using *C-greatest linorder-cls.is-greatest-in-set-iff* **by** *auto*
also have ... = ($\bigcup D \in \{D \in N.\ D \prec_c C\}.$ *ord-res.production N D*) $\cup$ *ord-res.production N C*
by *auto*
also have ... = *ord-res.interp N C* $\cup$ *ord-res.production N C*
unfolding *ord-res.interp-def* **..**
finally show *?thesis* **.**
qed

lemma *ord-res-model-entails-clauses-if-nex-false-clause*:
assumes *finite N* **and** $N \neq \{\}$ **and** $\neg$ *ex-false-clause N*
shows *ord-res-model N* $\models s$ *N*
unfolding *true-clss-def*
proof (*intro ballI*)
from ‹$\neg$ *ex-false-clause N*› **have** *ball-true*:
$\forall\, C \in N.$ *ord-res.interp N C* $\cup$ *ord-res.production N C* $\models C$
by (*simp add*: *ex-false-clause-def*)

from ‹*finite N*› ‹$N \neq \{\}$›
obtain *D* **where**
D-greatest: *linorder-cls.is-greatest-in-set N D*
using *linorder-cls.ex-greatest-in-set* **by** *metis*

fix *C* **assume** $C \in N$
hence *ord-res.interp N C* $\cup$ *ord-res.production N C* $\models C$
using *ball-true* **by** *metis*

moreover have $C \preceq_c D$
using ‹$C \in N$› *D-greatest*[*unfolded linorder-cls.is-greatest-in-set-iff*] **by** *auto*

ultimately have *ord-res.interp N D* $\cup$ *ord-res.production N D* $\models C$
using *ord-res.entailed-clause-stays-entailed* **by** *auto*

thus *ord-res-model N* $\models C$
using *ord-res-model-eq-interp-union-production-of-greatest-clause*[*OF D-greatest*]
by *argo*
qed

lemma *pos-lit-not-greatest-if-maximal-in-least-false-clause*:
assumes
C-least: *linorder-cls.is-least-in-fset* $\{|C \mid\in|\ N.\ \neg$ *ord-res-Interp* (*fset N*) *C* $\models$
$C|\}$ *C* **and**
C-max-lit: *ord-res.is-maximal-lit* (*Pos A*) *C*
shows $\neg$ *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
proof −
from *C-max-lit* **obtain** C' **where** *C-def*: $C =$ *add-mset* (*Pos A*) C'
by (*meson linorder-lit.is-maximal-in-mset-iff mset-add*)

from *C-least* **have**
C-in: *C* $\mid\in\mid$ *N* **and**
C-false: $\neg$ *ord-res-Interp* (*fset N*) *C* $\models$ *C* **and**
C-lt: $\forall\, y \mid\in\mid N.\ y \neq C \longrightarrow \neg$ *ord-res-Interp* (*fset N*) *y* $\models y \longrightarrow C \prec_c y$
unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *auto*

have $\nexists\, A.\ A \in$ *ord-res.production* (*fset N*) *C*
proof (*rule notI*, *elim exE*)
fix *A* **assume** *A-in*: $A \in$ *ord-res.production* (*fset N*) *C*
have *Pos A* $\in\#$ *C*

```
    using A-in by (auto elim: ord-res.mem-productionE)
   moreover have A ∈ ord-res-Interp (fset N) C
    using A-in C-in by blast
   ultimately show False
    using C-false by auto
  qed
  hence C-unproductive: ord-res.production (fset N) C = {}
   using ord-res.production-eq-empty-or-singleton[of fset N C] by simp

  have {D ∈ fset N. D ⪯c C} = {D ∈ fset N. D ≺c C} ∪ {D ∈ fset N. D = C}
   by fastforce
  with C-unproductive have
   ord-res-Interp (fset N) C = ord-res.interp (fset N) C
   by simp
  with C-false have C-false': ¬ ord-res.interp (fset N) C ⊨ C
   by simp

  from C-unproductive have A ∉ ord-res.production (fset N) C
   by simp
  thus ?thesis
  proof (rule contrapos-nn)
   assume ord-res.is-strictly-maximal-lit (Pos A) C
   then show A ∈ ord-res.production (fset N) C
    unfolding ord-res.production-unfold[of fset N C] mem-Collect-eq
    using C-in C-def C-false'
    by metis
  qed
qed

lemma ex-ground-factoringI:
  assumes
   C-max-lit: ord-res.is-maximal-lit (Pos A) C and
   C-not-max-lit: ¬ ord-res.is-strictly-maximal-lit (Pos A) C
  shows ∃ C'. ord-res.ground-factoring C C'
proof −
  from C-max-lit C-not-max-lit have count C (Pos A) ≥ 2
    using linorder-lit.count-ge-2-if-maximal-in-mset-and-not-greatest-in-mset by
metis
  then obtain C' where C-def: C = add-mset (Pos A) (add-mset (Pos A) C')
   by (metis two-le-countE)

  show ?thesis
  proof (rule exI)
   show ord-res.ground-factoring C (add-mset (Pos A) C')
    using ord-res.ground-factoringI[of C A C']
    using C-def C-max-lit by metis
  qed
qed
```

lemma *true-cls-if-true-lit-in*: $L \in\# C \Longrightarrow I \mathrel{\|\models l} L \Longrightarrow I \mathrel{\|\models} C$
by *auto*

lemma *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*:
assumes
C-least-false: *is-least-false-clause N C* **and**
Neg-A-max: *ord-res.is-maximal-lit* (*Neg A*) *C*
shows *fBex N* (λD. $D \prec_c C \wedge$ *ord-res.is-strictly-maximal-lit* (*Pos A*) $D \wedge$
ord-res.production (*fset N*) $D = \{A\}$)
proof −
from *C-least-false* **have**
C-in: $C \mathrel{|\in|} N$ **and**
C-false: $\neg$ *ord-res-Interp* (*fset N*) $C \mathrel{\|\models} C$ **and**
C-min: $\forall y \mathrel{|\in|} N.\ y \neq C \longrightarrow \neg$ *ord-res-Interp* (*fset N*) $y \mathrel{\|\models} y \longrightarrow C \prec_c y$
unfolding *atomize-conj is-least-false-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *argo*

have $\nexists A.\ A \in$ *ord-res.production* (*fset N*) *C*
proof (*rule notI*, *elim exE*)
fix *A* **assume** *A-in*: $A \in$ *ord-res.production* (*fset N*) *C*
have *Pos A* $\in\#$ *C*
using *A-in* **by** (*auto elim*: *ord-res.mem-productionE*)
moreover have $A \in$ *ord-res-Interp* (*fset N*) *C*
using *A-in C-in* **by** *blast*
ultimately show *False*
using *C-false* **by** *auto*
qed
hence *C-unproductive*: *ord-res.production* (*fset N*) $C = \{\}$
using *ord-res.production-eq-empty-or-singleton*[*of fset N C*] **by** *simp*

from *Neg-A-max* **have** *Neg A* $\in\#$ *C*
by (*simp add*: *linorder-lit.is-maximal-in-mset-iff*)

from *C-false* **have** $\neg$ *ord-res-Interp* (*fset N*) *C* $\mathrel{\|\models l}$ *Neg A*
using *true-cls-if-true-lit-in*[*OF* ‹*Neg A* $\in\#$ *C*›]
by *meson*
hence $A \in$ *ord-res-Interp* (*fset N*) *C*
by *simp*
with *C-unproductive* **have** $A \in$ *ord-res.interp* (*fset N*) *C*
by *blast*
then obtain *D* **where**
D-in: $D \mathrel{|\in|} N$ **and** *D-lt-C*: $D \prec_c C$ **and** *D-productive*: $A \in$ *ord-res.production* (*fset N*) *D*
unfolding *ord-res.interp-def* **by** *auto*

from *D-productive* **have** *ord-res.is-strictly-maximal-lit* (*Pos A*) *D*
using *ord-res.mem-productionE* **by** *metis*

moreover have *ord-res.production* (*fset N*) *D* = {*A*}
using *D-productive ord-res.production-eq-empty-or-singleton* **by** *fastforce*

ultimately show *?thesis*
using *D-in D-lt-C* **by** *metis*
qed

lemma *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit'*:
assumes
C-least-false: *is-least-false-clause N C* **and**
L-max: *ord-res.is-maximal-lit L C* **and**
L-neg: *is-neg L*
shows *fBex N* (λ*D*. *D* ≺$_c$ *C* ∧ *ord-res.is-strictly-maximal-lit* (− *L*) *D* ∧
ord-res.production (*fset N*) *D* = {*atm-of L*})
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*[*OF C-least-false*]
by (*simp add*: *L-max L-neg uminus-literal-def*)

lemma *ex-ground-resolutionI*:
assumes
C-max-lit: *ord-res.is-maximal-lit* (*Neg A*) *C* **and**
D-lt: *D* ≺$_c$ *C* **and**
D-max-lit: *ord-res.is-strictly-maximal-lit* (*Pos A*) *D*
shows ∃ *CD*. *ord-res.ground-resolution C D CD*
proof −
from *C-max-lit* **obtain** *C'* **where** *C-def*: *C* = *add-mset* (*Neg A*) *C'*
by (*meson linorder-lit.is-maximal-in-mset-iff mset-add*)

from *D-max-lit* **obtain** *D'* **where** *D-def*: *D* = *add-mset* (*Pos A*) *D'*
by (*meson linorder-lit.is-greatest-in-mset-iff mset-add*)

show *?thesis*
proof (*rule exI*)
show *ord-res.ground-resolution C D* (*C'* + *D'*)
using *ord-res.ground-resolutionI*[*of C A C' D D'*]
using *C-def D-def D-lt C-max-lit D-max-lit* **by** *metis*
qed
qed

lemma
fixes *N N'*
assumes
fin: *finite N finite N'* **and**
irrelevant: ∀ *D* ∈ *N'*. ∃ *E* ∈ *N*. *E* ⊂# *D* ∧ *set-mset D* = *set-mset E* **and**
C-in: *C* ∈ *N* **and**
C-not-entailed: ¬ *ord-res.interp N C* ∪ *ord-res.production N C* ⊨ *C*
shows ¬ *ord-res.interp* (*N* ∪ *N'*) *C* ∪ *ord-res.production* (*N* ∪ *N'*) *C* ⊨ *C*
using *C-not-entailed*
proof (*rule contrapos-nn*)

assume *ord-res.interp* $(N \cup N')$ $C \cup$ *ord-res.production* $(N \cup N')$ $C \models C$
then show *ord-res.interp* N $C \cup$ *ord-res.production* N $C \models C$
using *ord-res.interp-add-irrelevant-clauses-to-set*[*OF fin C-in irrelevant*]
using *ord-res.production-add-irrelevant-clauses-to-set*[*OF fin C-in irrelevant*]
by *metis*
qed

lemma *trail-consistent-if-sorted-wrt-atoms*:
assumes *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))$ Γ
shows *trail-consistent* Γ
using *assms*
proof (*induction* Γ)
case *Nil*
show *?case*
by *simp*
next
case (*Cons Ln* Γ)

obtain L *opt* **where**
$Ln = (L,\ opt)$
by *fastforce*

show *?case*
unfolding ‹$Ln = (L,\ opt)$›
proof (*rule trail-consistent.Cons*)
have $\forall\, x{\in}set\ \Gamma.$ *atm-of* $(fst\ x) \prec_t$ *atm-of* $(fst\ Ln)$
using *Cons.prems* **by** *simp*

hence $\forall\, x{\in}set\ \Gamma.$ *atm-of* $(fst\ x) \neq$ *atm-of* L
unfolding ‹$Ln = (L,\ opt)$› **by** *fastforce*

thus $\neg$ *trail-defined-lit* Γ L
unfolding *trail-defined-lit-def* **by** *fastforce*
next
show *trail-consistent* Γ
using *Cons* **by** *simp*
qed
qed

lemma *mono-atms-lt*: *monotone-on* $(set\ \Gamma)$ $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))$ $(\lambda x\ y.\ y \leq x)$
$(\lambda x.$ *atm-of* $K \preceq_t$ *atm-of* $(fst\ x))$ **for** K
proof (*intro monotone-onI*, *unfold le-bool-def*, *intro impI*)
fix $x\ y$
assume *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x)$ **and** *atm-of* $K \preceq_t$ *atm-of* $(fst\ y)$
thus *atm-of* $K \preceq_t$ *atm-of* $(fst\ x)$
by *order*
qed

```
lemma in-trail-atms-dropWhileI:
  assumes
    sorted-wrt R Γ and
    monotone-on (set Γ) R (≥) (λx. P (atm-of (fst x))) and
    ¬ P A and
    A |∈| trail-atms Γ
  shows A |∈| trail-atms (dropWhile (λLn. P (atm-of (fst Ln))) Γ)
  using assms(1,2,4)
proof (induction Γ)
  case Nil
  thus ?case
    by simp
next
  case (Cons Ln Γ)
  show ?case
  proof (cases P (atm-of (fst Ln)))
    case True

    have A |∈| trail-atms (dropWhile (λLn. P (atm-of (fst Ln))) Γ)
    proof (rule Cons.IH)
      show sorted-wrt R Γ
        using Cons.prems(1) by simp
    next
      show monotone-on (set Γ) R (λx y. y ≤ x) (λx. P (atm-of (fst x)))
        using Cons.prems(2) by (meson monotone-on-subset set-subset-Cons)
    next
      have ¬ P A
        using assms by metis
      hence A ≠ atm-of (fst Ln)
        using True by metis
      moreover have A |∈| trail-atms (Ln # Γ)
        using Cons.prems(3) by metis
      ultimately show A |∈| trail-atms Γ
        by (simp add: trail-defined-lit-def)
    qed

    thus ?thesis
      using True by simp
  next
    case False
    thus ?thesis
      using Cons.prems(3) by simp
  qed
qed

lemma trail-defined-lit-dropWhileI:
  assumes
    sorted-wrt R Γ and
```

monotone-on (*set* Γ) *R* (≥) (λ*x*. *P* (*fst x*)) **and**
¬ *P L* ∧ ¬ *P* (− *L*) **and**
trail-defined-lit Γ *L*
shows *trail-defined-lit* (*dropWhile* (λ*Ln*. *P* (*fst Ln*)) Γ) *L*
using *assms in-trail-atms-dropWhileI*
by (*smt* (*verit*) *imageE image-eqI mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*
trail-defined-lit-def trail-defined-lit-iff-trail-defined-atm)

lemma *trail-defined-cls-dropWhileI*:
assumes
sorted-wrt R Γ **and**
monotone-on (*set* Γ) *R* (≥) (λ*x*. *P* (*fst x*)) **and**
∀ *L* ∈# *C*. ¬ *P L* ∧ ¬ *P* (− *L*) **and**
trail-defined-cls Γ *C*
shows *trail-defined-cls* (*dropWhile* (λ*Ln*. *P* (*fst Ln*)) Γ) *C*
using *assms trail-defined-lit-dropWhileI*
by (*metis trail-defined-cls-def*)

lemma *nbex-less-than-least-in-fset*: ¬ (∃ *w* |∈| *X*. *w* ≺$_c$ *x*)
if *linorder-cls.is-least-in-fset X x* **for** *X x*
using *that* **unfolding** *linorder-cls.is-least-in-fset-iff* **by** *auto*

lemma *clause-le-if-lt-least-greater*:
fixes *N* U_{er} $\mathcal{F}$ *C D*
defines
$\mathcal{C}$ ≡ *The-optional* (*linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) *D*) *N*))
assumes
C-lt: ⋀*E*. $\mathcal{C}$ = *Some E* ⟹ *C* ≺$_c$ *E* **and**
C-in: *C* |∈| *N*
shows *C* ⪯$_c$ *D*
proof (*cases* $\mathcal{C}$)
case *None*

hence ¬ (∃ *E*. *linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) *D*) *N*) *E*)
using $\mathcal{C}$*-def*
by (*metis None-eq-The-optionalD Uniq-D linorder-cls.Uniq-is-least-in-fset*)

hence ¬ (∃ *E* |∈| *N*. *D* ≺$_c$ *E*)
by (*metis femptyE ffmember-filter linorder-cls.ex1-least-in-fset*)

thus *?thesis*
using *C-in linorder-cls.less-linear* **by** *blast*
next
case (*Some E*)

hence *linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) *D*) *N*) *E*
using $\mathcal{C}$*-def* **by** (*metis Some-eq-The-optionalD*)

hence *C* ≺$_c$ *D* ∨ *C* = *D*

by (*metis C-in C-lt Some ffmember-filter linorder-cls.neqE nbex-less-than-least-in-fset*)

thus *?thesis*
by *simp*
qed

end

10 Lemmas about going between ground and first-order terms

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

lemma *ex1-gterm-of-term*:
fixes $t :: 'f$ *gterm*
shows $\exists!(t' :: ('f, 'v)\ term).\ ground\ t' \wedge t = gterm\text{-}of\text{-}term\ t'$
proof (*rule ex1I*)
show *ground* (*term-of-gterm t*) $\wedge$ t = *gterm-of-term* (*term-of-gterm t*)
by *simp*
next
fix $t' :: ('f, 'v)\ term$
show *ground* $t' \wedge t$ = *gterm-of-term* $t' \Longrightarrow t'$ = *term-of-gterm t*
by (*induction* t') (*simp-all add*: *map-idI*)
qed

lemma *binj-betw-gterm-of-term*: *bij-betw gterm-of-term* {*t. ground t*} *UNIV*
unfolding *bij-betw-def*
proof (*rule conjI*)
show *inj-on gterm-of-term* {*t. ground t*}
by (*metis gterm-of-term-inj mem-Collect-eq*)
next
show *gterm-of-term* ‘ {*t. ground t*} = *UNIV*
proof (*rule Set.subset-antisym*)
show *gterm-of-term* ‘ {*t. Term-Context.ground t*} $\subseteq$ *UNIV*
by *simp*
next
show *UNIV* $\subseteq$ *gterm-of-term* ‘ {*t. Term-Context.ground t*}
by (*metis* (*mono-tags*, *opaque-lifting*) *ground-term-of-gterm image-iff mem-Collect-eq subsetI*
term-of-gterm-inv)
qed
qed

end

11 SCL(FOL) for first-order terms

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

definition *less-B* **where**
less-B x y $\longleftrightarrow$ *ground x* $\wedge$ *ground y* $\wedge$ *gterm-of-term x* $\prec_t$ *gterm-of-term y*

sublocale *order-less-B*: *order less-B*$^{==}$ *less-B*
by *unfold-locales* (*auto simp add*: *less-B-def*)

sublocale *scl-fol*: *scl-fol-calculus renaming-vars less-B*
proof *unfold-locales*
fix β :: ($'f$, $'v$) *term*

have *Collect-gterms-eq*: $\bigwedge P$. {*y*. *P y*} = *gterm-of-term* ' {*t*. *ground t* $\wedge$ *P* (*gterm-of-term t*)}
using *Collect-eq-image-filter-Collect-if-bij-betw*[*OF binj-betw-gterm-of-term subset-UNIV*]
by *auto*

have {t_G. $t_G \prec_t$ *gterm-of-term* β} = *gterm-of-term* ' {*x*. *ground x* $\wedge$ *gterm-of-term x* $\prec_t$ *gterm-of-term* β}
using *Collect-gterms-eq*[*of* λt_G. $t_G \prec_t$ *gterm-of-term* β] .
hence *finite* (*gterm-of-term* ' {*x*. *ground x* $\wedge$ *gterm-of-term x* $\prec_t$ *gterm-of-term* β})
using *finite-less-trm*[*of gterm-of-term* β] **by** *metis*
moreover have *inj-on gterm-of-term* {*x*. *ground x* $\wedge$ *gterm-of-term x* $\prec_t$ *gterm-of-term* β}
by (*rule gterm-of-term-inj*) *simp*
ultimately have *finite* {*x*. *ground x* $\wedge$ *gterm-of-term x* $\prec_t$ *gterm-of-term* β}
using *finite-imageD* **by** *blast*
thus *finite* {*x*. *less-B x* β}
unfolding *less-B-def*
using *not-finite-existsD* **by** *force*
qed

end

lemma *trail-atms-eq-trail-atms-if-same-lits*:
assumes *list-all2* ($\lambda x\ y$. *fst x* = *fst y*) Γ_9 Γ_{10}
shows *trail-atms* Γ_9 = *trail-atms* Γ_{10}
using *assms*
proof (*induction* Γ_9 Γ_{10} *rule*: *list.rel-induct*)
case *Nil*
show *?case*
by (*simp add*: *trail-atms-def*)
next
case (*Cons* Ln_9 $\Gamma_9{}'$ Ln_{10} $\Gamma_{10}{}'$)
thus *?case*
by (*simp add*: *trail-atms-def*)
qed

lemma *trail-false-lit-eq-trail-false-lit-if-same-lits*:
  assumes *list-all2* (*λx y. fst x = fst y*) Γ_9 Γ_{10}
  shows *trail-false-lit* Γ_9 = *trail-false-lit* Γ_{10}
  using *assms*
proof (*induction* Γ_9 Γ_{10} *rule*: *list.rel-induct*)
  case *Nil*
  show *?case*
    by (*simp add*: *trail-false-lit-def*)
next
  case (*Cons* Ln_9 $\Gamma_9{'}$ Ln_{10} $\Gamma_{10}{'}$)
  thus *?case*
    unfolding *trail-false-lit-def*
    unfolding *list.set image-insert*
    by (*metis insert-iff*)
qed

lemma *trail-false-cls-eq-trail-false-cls-if-same-lits*:
  assumes *list-all2* (*λx y. fst x = fst y*) Γ_9 Γ_{10}
  shows *trail-false-cls* Γ_9 = *trail-false-cls* Γ_{10}
  unfolding *trail-false-cls-def*
  unfolding *trail-false-lit-eq-trail-false-lit-if-same-lits*[*OF assms*] **..**

lemma *trail-defined-lit-eq-trail-defined-lit-if-same-lits*:
  assumes *list-all2* (*λx y. fst x = fst y*) Γ_9 Γ_{10}
  shows *trail-defined-lit* Γ_9 = *trail-defined-lit* Γ_{10}
  using *assms*
proof (*induction* Γ_9 Γ_{10} *rule*: *list.rel-induct*)
  case *Nil*
  show *?case*
    by (*simp add*: *trail-defined-lit-def*)
next
  case (*Cons* Ln_9 $\Gamma_9{'}$ Ln_{10} $\Gamma_{10}{'}$)
  thus *?case*
    unfolding *trail-defined-lit-def*
    unfolding *list.set image-insert*
    by (*metis* (*no-types*, *opaque-lifting*) *insert-iff*)
qed

lemma *trail-defined-cls-eq-trail-defined-cls-if-same-lits*:
  assumes *list-all2* (*λx y. fst x = fst y*) Γ_9 Γ_{10}
  shows *trail-defined-cls* Γ_9 = *trail-defined-cls* Γ_{10}
  unfolding *trail-defined-cls-def*
  unfolding *trail-defined-lit-eq-trail-defined-lit-if-same-lits*[*OF assms*] **..**

end
theory *ORD-RES*
  imports *Background*
begin

12 ORD-RES

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

lemma *ex-false-clause N* $\longleftrightarrow$ $\neg$ ($\forall$ *C* $\in$ *N*. *ord-res-Interp N C* $\models$ *C*)
unfolding *ex-false-clause-def* **by** *metis*

lemma $\neg$ *ex-false-clause N* $\longleftrightarrow$ ($\forall$ *C* $\in$ *N*. *ord-res-Interp N C* $\models$ *C*)
unfolding *ex-false-clause-def* **by** *metis*

definition *ord-res-final* **where**
ord-res-final N $\longleftrightarrow$ {#} |$\in$| *N* $\vee$ $\neg$ *ex-false-clause* (*fset N*)

inductive *ord-res* **where**
factoring: {#} |$\notin$| *N* $\Longrightarrow$ *ex-false-clause* (*fset N*) $\Longrightarrow$
— Maybe write $\neg$ *ord-res-final N* instead?
P |$\in$| *N* $\Longrightarrow$ *ord-res.ground-factoring P C* $\Longrightarrow$
N' = *finsert C N* $\Longrightarrow$
ord-res N N' |

resolution: {#} |$\notin$| *N* $\Longrightarrow$ *ex-false-clause* (*fset N*) $\Longrightarrow$
P1 |$\in$| *N* $\Longrightarrow$ *P2* |$\in$| *N* $\Longrightarrow$ *ord-res.ground-resolution P1 P2 C* $\Longrightarrow$
N' = *finsert C N* $\Longrightarrow$
ord-res N N'

inductive *ord-res-load* **where**
N $\neq$ {||} $\Longrightarrow$ *ord-res-load N N*

sublocale *ord-res-semantics*: *semantics* **where**
step = *ord-res* **and**
final = *ord-res-final*
proof *unfold-locales*
fix *N* :: *'f gterm clause fset*
assume *ord-res-final N*
thus *finished ord-res N*
unfolding *ord-res-final-def*
proof (*elim disjE*)
show {#} |$\in$| *N* $\Longrightarrow$ *finished ord-res N*
by (*simp add*: *finished-def ord-res.simps*)
next
show $\neg$ *ex-false-clause* (*fset N*) $\Longrightarrow$ *finished ord-res N*
by (*simp add*: *finished-def ord-res.simps*)
qed
qed

sublocale *ord-res-language*: *language* **where**
step = *ord-res* **and**
final = *ord-res-final* **and**
load = *ord-res-load*

by *unfold-locales*

end

end
theory *ORD-RES-1*
imports *ORD-RES*
begin

13 ORD-RES-1 (deterministic)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-1* **where**
factoring:
is-least-false-clause N C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-pos L $\Longrightarrow$
ord-res.ground-factoring C C' $\Longrightarrow$
$N' = finsert\ C'\ N \Longrightarrow$
ord-res-1 N N' |

resolution:
is-least-false-clause N C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-neg L $\Longrightarrow$
$D\ |\in|\ N \Longrightarrow$
$D \prec_c C \Longrightarrow$
ord-res.production (*fset N*) $D = \{atm\text{-}of\ L\} \Longrightarrow$
ord-res.ground-resolution C D CD $\Longrightarrow$
$N' = finsert\ CD\ N \Longrightarrow$
ord-res-1 N N'

lemma
assumes *ord-res.ground-resolution C D CD*
shows $D \prec_c C$
using *assms*
by (*auto simp add*: *ord-res.ground-resolution.simps*)

lemma *right-unique-ord-res-1*: *right-unique ord-res-1*
proof (*rule right-uniqueI*)
fix *N N' N''* :: *'f gterm clause fset*
assume *step1*: *ord-res-1 N N'* **and** *step2*: *ord-res-1 N N''*
from *step1* **show** $N' = N''$
proof (*cases N N' rule*: *ord-res-1.cases*)
case *hyps1*: (*factoring C1 L1 C1'*)
from *step2* **show** *?thesis*
proof (*cases N N'' rule*: *ord-res-1.cases*)
case *hyps2*: (*factoring C2 L2 C2'*)

```
    from hyps1 hyps2 have C1 = C2
      by (meson Uniq-D Uniq-is-least-false-clause)
    with hyps1 hyps2 have C1' = C2'
      by (metis (no-types, lifting) Uniq-D ord-res.unique-ground-factoring)
    with hyps1 hyps2 show ?thesis
      by argo
  next
    case hyps2: (resolution C2 L2 D2 CD2)
    from hyps1 hyps2 have C1 = C2
      by (meson Uniq-D Uniq-is-least-false-clause)
    with hyps1 hyps2 have L1 = L2
      by (meson Uniq-D linorder-lit.Uniq-is-maximal-in-mset)
    with hyps1 hyps2 have False
      by metis
    thus ?thesis ..
  qed
next
  case hyps1: (resolution C1 L1 D1 CD1)
  from step2 show ?thesis
  proof (cases N N'' rule: ord-res-1.cases)
    case hyps2: (factoring C2 L2 C2')
    from hyps1 hyps2 have C1 = C2
      by (meson Uniq-D Uniq-is-least-false-clause)
    with hyps1 hyps2 have L1 = L2
      by (meson Uniq-D linorder-lit.Uniq-is-maximal-in-mset)
    with hyps1 hyps2 have False
      by metis
    thus ?thesis ..
  next
    case hyps2: (resolution C2 L2 D2 CD2)
    from hyps1 hyps2 have C1 = C2
      by (meson Uniq-D Uniq-is-least-false-clause)
    with hyps1 hyps2 have L1 = L2
      by (meson Uniq-D linorder-lit.Uniq-is-maximal-in-mset)
    with hyps1 hyps2 have D1 = D2
      by (metis (mono-tags) Uniq-D ord-res.Uniq-production-eq-singleton)
    with hyps1 hyps2 have CD1 = CD2
    by (metis (no-types, lifting) Uniq-D ‹C1 = C2› ord-res.unique-ground-resolution)
    with hyps1 hyps2 show ?thesis
      by argo
  qed
qed
qed

definition ord-res-1-final where
  ord-res-1-final N ⟷ ord-res-final N

inductive ord-res-1-load where
  N ≠ {||} ⟹ ord-res-1-load N N
```

```
sublocale ord-res-1-semantics: semantics where
  step = ord-res-1 and
  final = ord-res-1-final
proof unfold-locales
  fix N :: 'f gterm clause fset
  assume ord-res-1-final N
  thus finished ord-res-1 N
    unfolding ord-res-1-final-def ord-res-final-def
  proof (elim disjE)
    assume {#} |∈| N
    have False if ord-res-1 N N' for N'
      using that
    proof (cases N N' rule: ord-res-1.cases)
      case hyps: (factoring C L C')
      from hyps ‹{#} |∈| N› have C = {#}
        unfolding is-least-false-clause-def
      by (metis (no-types, lifting) emptyE ffmember-filter linorder-cls.dual-order.strict-trans
          linorder-cls.is-least-in-fset-iff linorder-cls.less-irrefl
          ord-res.multp-if-all-left-smaller set-mset-empty true-cls-empty)
      moreover from hyps have L ∈# C
        using linorder-lit.is-maximal-in-mset-iff by blast
      ultimately show False
        by simp
    next
      case hyps: (resolution C L D CD)
      from hyps ‹{#} |∈| N› have C = {#}
        unfolding is-least-false-clause-def
      by (metis (no-types, lifting) emptyE ffmember-filter linorder-cls.dual-order.strict-trans
          linorder-cls.is-least-in-fset-iff linorder-cls.less-irrefl
          ord-res.multp-if-all-left-smaller set-mset-empty true-cls-empty)
      moreover from hyps have L ∈# C
        using linorder-lit.is-maximal-in-mset-iff by blast
      ultimately show False
        by simp
    qed
    thus finished ord-res-1 N
      unfolding finished-def by metis
  next
    assume ¬ ex-false-clause (fset N)
    have False if ord-res-1 N N' for N'
      using that
    proof (cases N N' rule: ord-res-1.cases)
      case (factoring C L C')
      with ‹¬ ex-false-clause (fset N)› show False
        unfolding ex-false-clause-def is-least-false-clause-def
        using linorder-cls.is-least-in-fset-iff by force
    next
      case (resolution C L D CD)
```

```
      with ‹¬ ex-false-clause (fset N)› show False
        unfolding ex-false-clause-def is-least-false-clause-def
        using linorder-cls.is-least-in-fset-iff by force
    qed
    thus finished ord-res-1 N
      unfolding finished-def by metis
  qed
qed

sublocale ord-res-1-language: language where
  step = ord-res-1 and
  final = ord-res-1-final and
  load = ord-res-1-load
  by unfold-locales

lemma ex-ord-res-1-if-not-final:
  assumes ¬ ord-res-1-final N
  shows ∃ N'. ord-res-1 N N'
proof −
  from assms have {#} |∉| N and ex-false-clause (fset N)
    unfolding ord-res-1-final-def ord-res-final-def by argo+

  obtain C where C-least-false: is-least-false-clause N C
    using ‹ex-false-clause (fset N)› obtains-least-false-clause-if-ex-false-clause by
metis

  hence C ≠ {#}
    using ‹{#} |∉| N›
    unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    by metis

  then obtain L where C-max: linorder-lit.is-maximal-in-mset C L
    using linorder-lit.ex-maximal-in-mset by metis

  show ?thesis
  proof (cases L)
    case (Pos A)

    hence ¬ linorder-lit.is-greatest-in-mset C L
      using pos-lit-not-greatest-if-maximal-in-least-false-clause
      using C-least-false is-least-false-clause-def by blast

    then obtain C' where ord-res.ground-factoring C C'
      using ex-ground-factoringI C-max Pos by metis

    thus ?thesis
      using ord-res-1.factoring
     by (metis ‹is-least-false-clause N C› is-pos-def ord-res.ground-factoring.cases)
  next
```

case (*Neg A*)
then obtain *D* **where**
D |∈| *N* **and**
D ≺$_c$ *C* **and**
ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**
ord-res.production (*fset N*) *D* = {*A*}
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
using *C-least-false C-max* **by** *metis*

moreover then obtain *CD* **where**
ord-res.ground-resolution C D CD
using *ex-ground-resolutionI C-max Neg* **by** *metis*

ultimately show *?thesis*
using *ord-res-1.resolution*[*of N C L D CD finsert CD N*]
using *C-least-false C-max Neg* **by** *auto*
qed
qed

corollary *ord-res-1-safe*: *ord-res-1-final N* ∨ (∃ *N′*. *ord-res-1 N N′*)
using *ex-ord-res-1-if-not-final* **by** *metis*

end

end
theory *Exhaustive-Factorization*
imports *Background*
begin

14 Function for full factorization

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

definition *efac* :: *'f gterm clause* ⇒ *'f gterm clause* **where**
efac C = (*THE C′*. *ord-res.ground-factoring*** *C C′* ∧ (∄ *C″*. *ord-res.ground-factoring C′ C″*))

The function *efac* performs exhaustive factorization of its input clause.

lemma *ex1-efac-eq-factoring-chain*:
∃!*C′*. *efac C* = *C′* ∧ *ord-res.ground-factoring*** *C C′* ∧ (∄ *C″*. *ord-res.ground-factoring C′ C″*)
proof −
have *right-unique* (λ*x y*. *ord-res.ground-factoring*** *x y* ∧ (∄ *z*. *ord-res.ground-factoring y z*))
using *ord-res.unique-ground-factoring right-unique-terminating-rtranclp right-unique-iff*
by *blast*

moreover obtain *C′* **where** *ord-res.ground-factoring*** *C C′* ∧ (∄ *C″*. *ord-res.ground-factoring C′ C″*)

using *ex-terminating-rtranclp*[*OF ord-res.termination-ground-factoring*]
by *metis*

ultimately have *efac C = C'*
by (*simp add*: *efac-def right-unique-def the-equality*)

then show *?thesis*
using ‹*ord-res.ground-factoring*** *C C'* ∧ (∄*C''*. *ord-res.ground-factoring C' C''*)› **by** *blast*
qed

lemma *efac-eq-disj*:
efac C = C ∨ (∃!*C'*. *efac C = C'* ∧ *ord-res.ground-factoring*** *C C'* ∧ (∄*C''*. *ord-res.ground-factoring C' C''*))
using *ex1-efac-eq-factoring-chain*
by (*metis is-pos-def*)

lemma *member-mset-if-count-eq-Suc*: *count X x = Suc n* ⟹ *x* ∈# *X*
by (*simp add*: *count-inI*)

lemmas *member-fsetE = mset-add*

lemma *ord-res-ground-factoring-iff*: *ord-res.ground-factoring C C'* ⟷
(∃*A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ (∃*n*. *count C* (*Pos A*) = *Suc* (*Suc n*) ∧ *C' = C* − {#*Pos A*#}))
proof (*rule iffI*)
assume *ord-res.ground-factoring C C'*
thus ∃*A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ (∃*n*. *count C* (*Pos A*) = *Suc* (*Suc n*) ∧ *C' = C* − {#*Pos A*#})
proof (*cases C C' rule*: *ord-res.ground-factoring.cases*)
case (*ground-factoringI A P'*)
show *?thesis*
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit* (*Pos A*) *C*
using ‹*ord-res.is-maximal-lit* (*Pos A*) *C*› **.**
next
show *count C* (*Pos A*) = *Suc* (*Suc* (*count P'* (*Pos A*)))
unfolding ‹*C = add-mset* (*Pos A*) (*add-mset* (*Pos A*) *P'*)› **by** *simp*
next
show *C' = remove1-mset* (*Pos A*) *C*
unfolding ‹*C = add-mset* (*Pos A*) (*add-mset* (*Pos A*) *P'*)› ‹*C' = add-mset* (*Pos A*) *P'*› **by** *simp*
qed
qed
next
assume ∃*A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧
(∃*n*. *count C* (*Pos A*) = *Suc* (*Suc n*) ∧ *C' = C* − {#*Pos A*#})
then obtain *A n* **where**
ord-res.is-maximal-lit (*Pos A*) *C* **and**

```
    count C (Pos A) = Suc (Suc n) and
    C' = C - {#Pos A#}
    by metis

  have Pos A ∈# C
    using ‹count C (Pos A) = Suc (Suc n)› member-mset-if-count-eq-Suc by metis
  then obtain C'' where C = add-mset (Pos A) C''
    by (auto elim: member-fsetE)
  with ‹count C (Pos A) = Suc (Suc n)› have count C'' (Pos A) = Suc n
    by simp
  hence Pos A ∈# C''
    using member-mset-if-count-eq-Suc by metis
  then obtain C''' where C'' = add-mset (Pos A) C'''
    by (auto elim: member-fsetE)

  show ord-res.ground-factoring C C'
  proof (rule ord-res.ground-factoringI)
    show C = add-mset (Pos A) (add-mset (Pos A) C''')
      using ‹C = add-mset (Pos A) C''› ‹C'' = add-mset (Pos A) C'''› by metis
  next
    show ord-res.is-maximal-lit (Pos A) C
      using ‹ord-res.is-maximal-lit (Pos A) C› .
  next
    show C' = add-mset (Pos A) C'''
      using ‹C' = C - {#Pos A#}› ‹C = add-mset (Pos A) C''› ‹C'' = add-mset
(Pos A) C'''› by simp
  qed simp-all
qed

lemma tranclp-ord-res-ground-factoring-iff:
  ord-res.ground-factoring^++ C C' ∧ (∄C''. ord-res.ground-factoring C' C'') ⟷
  (∃A. ord-res.is-maximal-lit (Pos A) C ∧ (∃n. count C (Pos A) = Suc (Suc n) ∧
    C' = C - replicate-mset (Suc n) (Pos A)))
proof (intro iffI; elim exE conjE)
  assume ord-res.ground-factoring^++ C C' and (∄C''. ord-res.ground-factoring
C' C'')
  then show ∃A. ord-res.is-maximal-lit (Pos A) C ∧ (∃n. count C (Pos A) =
Suc (Suc n) ∧
    C' = C - replicate-mset (Suc n) (Pos A))
  proof (induction C rule: converse-tranclp-induct)
    case (base C)
    from base.hyps obtain A n where
      ord-res.is-maximal-lit (Pos A) C and
      count C (Pos A) = Suc (Suc n) and
      C' = remove1-mset (Pos A) C
      unfolding ord-res-ground-factoring-iff by auto

    moreover have n = 0
    proof (rule ccontr)
```

```
    assume n ≠ 0
    then obtain C'' where C' = add-mset (Pos A) (add-mset (Pos A) C'')
      by (metis (no-types, lifting) Zero-not-Suc calculation(2,3) count-add-mset
count-inI
          diff-Suc-1 insert-DiffM)
    hence ord-res.ground-factoring C' (add-mset (Pos A) C'')
      using ord-res.ground-factoringI
    by (metis calculation(1,3) linorder-lit.is-maximal-in-mset-iff more-than-one-mset-mset-diff
          union-single-eq-member)
    with base.prems show False
      by metis
  qed

  ultimately show ?case
    by (metis replicate-mset-0 replicate-mset-Suc)
 next
  case (step C C'')
  from step.IH step.prems obtain A n where
    ord-res.is-maximal-lit (Pos A) C'' and
    count C'' (Pos A) = Suc (Suc n) and
    C' = C'' − replicate-mset (Suc n) (Pos A)
    by metis

  from step.hyps(1) obtain A' m where
    ord-res.is-maximal-lit (Pos A') C and
    count C (Pos A') = Suc (Suc m) and
    C'' = remove1-mset (Pos A') C
    unfolding ord-res-ground-factoring-iff by metis

  have A' = A
     using ‹ord-res.is-maximal-lit (Pos A) C''› ‹ord-res.is-maximal-lit (Pos A')
C›
    by (metis ‹C'' = remove1-mset (Pos A') C› ‹count C (Pos A') = Suc (Suc
m)›
        add-mset-remove-trivial-eq count-add-mset count-greater-zero-iff diff-Suc-1
       linorder-lit.antisym-conv3 linorder-lit.is-maximal-in-mset-iff literal.inject(1)
        zero-less-Suc)

  have m = Suc n
   using ‹count C'' (Pos A) = Suc (Suc n)› ‹count C (Pos A') = Suc (Suc m)›
   unfolding ‹C'' = remove1-mset (Pos A') C› ‹A' = A›
   by simp

  show ?case
  proof (intro exI conjI)
    show ord-res.is-maximal-lit (Pos A) C
      using ‹ord-res.is-maximal-lit (Pos A') C› ‹A' = A› by metis
  next
    show count C (Pos A) = Suc (Suc m)
```

```
      using ‹count C (Pos A') = Suc (Suc m)› ‹A' = A› by metis
    next
      show C' = C − replicate-mset (Suc m) (Pos A)
       unfolding ‹C' = C'' − replicate-mset (Suc n) (Pos A)› ‹C'' = remove1-mset
(Pos A') C›
         ‹A' = A› ‹m = Suc n›
       by simp
    qed
  qed
next
  fix A n assume ord-res.is-maximal-lit (Pos A) C
  thus count C (Pos A) = Suc (Suc n) ⟹ C' = C − replicate-mset (Suc n) (Pos
A) ⟹
    ord-res.ground-factoring⁺⁺ C C' ∧ (∄ C''. ord-res.ground-factoring C' C'')
  proof (induction n arbitrary: C)
    case 0
    hence (ord-res.is-maximal-lit (Pos A) C ∧
        (count C (Pos A) = Suc (Suc 0) ∧
            C' = remove1-mset (Pos A) C))
      by (metis replicate-mset-0 replicate-mset-Suc)
    hence ord-res.ground-factoring C C' ∧ (∄ a. ord-res.ground-factoring C' a)
      unfolding ord-res-ground-factoring-iff
      by (metis Zero-not-Suc add-mset-remove-trivial-eq count-add-mset count-inI
          linorder-lit.antisym-conv3 linorder-lit.is-maximal-in-mset-iff nat.inject)
    thus ?case
      by blast
  next
    case (Suc n)
   have ord-res.ground-factoring⁺⁺ (C − {#Pos A#}) C' ∧ (∄ a. ord-res.ground-factoring
C' a)
    proof (rule Suc.IH)
      show count (remove1-mset (Pos A) C) (Pos A) = Suc (Suc n)
        using Suc.prems by simp
    next
      show C' = remove1-mset (Pos A) C − replicate-mset (Suc n) (Pos A)
        using Suc.prems by simp
    next
      show ord-res.is-maximal-lit (Pos A) (remove1-mset (Pos A) C)
        using Suc.prems
      by (smt (verit, ccfv-SIG) Zero-not-Suc add-diff-cancel-left' add-mset-remove-trivial-eq
          count-add-mset count-inI linorder-lit.is-maximal-in-mset-iff plus-1-eq-Suc)
    qed

    moreover have ord-res.ground-factoring C (C − {#Pos A#})
      unfolding ord-res-ground-factoring-iff
    proof (intro exI conjI)
      show ord-res.is-maximal-lit (Pos A) C
        using Suc.prems by metis
    next
```

show *count C* (*Pos A*) = *Suc* (*Suc* (*Suc n*))
using *Suc.prems* **by** *metis*
next
show *remove1-mset* (*Pos A*) *C* = *remove1-mset* (*Pos A*) *C* **..**
qed

ultimately show *?case*
by *auto*
qed
qed

lemma *minus-mset-replicate-mset-eq-add-mset-filter-mset*:
assumes *count X x* = *Suc n*
shows *X* − *replicate-mset n x* = *add-mset x* {#*y* ∈# *X*. *y* ≠ *x*#}
using *assms*
by (*metis add-diff-cancel-left' add-mset-diff-bothsides filter-mset-eq filter-mset-neq multiset-partition replicate-mset-Suc union-mset-add-mset-right*)

lemma *minus-mset-replicate-mset-eq-add-mset-add-mset-filter-mset*:
assumes *count X x* = *Suc* (*Suc n*)
shows *X* − *replicate-mset n x* = *add-mset x* (*add-mset x* {#*y* ∈# *X*. *y* ≠ *x*#})
using *assms*
by (*metis add-diff-cancel-left' add-mset-diff-bothsides filter-mset-eq filter-mset-neq multiset-partition replicate-mset-Suc union-mset-add-mset-right*)

lemma *rtrancl-ground-factoring-iff*:
shows *ord-res.ground-factoring*** *C C'* ∧ (∄ *C''*. *ord-res.ground-factoring C' C''*)
⟷
((∄ *A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ *count C* (*Pos A*) ≥ *2*) ∧ *C* = *C'* ∨
(∃ *A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ *C'* = *add-mset* (*Pos A*) {#*L* ∈# *C*.
L ≠ *Pos A*#}))
proof (*intro iffI*; *elim exE conjE disjE*)
show *ord-res.ground-factoring*** *C C'* ⟹ ∄ *C''*. *ord-res.ground-factoring C' C''*
⟹
(∄ *A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ *2* ≤ *count C* (*Pos A*)) ∧ *C* = *C'* ∨
(∃ *A*. *ord-res.is-maximal-lit* (*Pos A*) *C* ∧ *C'* = *add-mset* (*Pos A*) {#*L* ∈# *C*.
L ≠ *Pos A*#})
proof (*induction C rule*: *converse-rtranclp-induct*)
case *base*
thus *?case*
by (*metis add-2-eq-Suc le-Suc-ex ord-res-ground-factoring-iff*)
next
case (*step y z*)
hence *ord-res.ground-factoring*$^{++}$ *y C'* ∧ (∄ *x*. *ord-res.ground-factoring C' x*)
by *simp*
thus *?case*
unfolding *tranclp-ord-res-ground-factoring-iff*
by (*metis minus-mset-replicate-mset-eq-add-mset-filter-mset*)
qed

```
next
  assume ∄A. ord-res.is-maximal-lit (Pos A) C ∧ 2 ≤ count C (Pos A) and C
= C'
  thus ord-res.ground-factoring** C C' ∧ (∄C''. ord-res.ground-factoring C' C'')
    by (metis One-nat-def Suc-1 Suc-le-eq Suc-le-mono ord-res-ground-factoring-iff
        rtranclp.rtrancl-refl zero-less-Suc)
next
  fix A assume ord-res.is-maximal-lit (Pos A) C
  then obtain n where count C (Pos A) = Suc n
    by (meson in-countE linorder-lit.is-maximal-in-mset-iff)
  with ‹ord-res.is-maximal-lit (Pos A) C› show C' = add-mset (Pos A) {#L ∈#
C. L ≠ Pos A#} ⟹
    ord-res.ground-factoring** C C' ∧ (∄C''. ord-res.ground-factoring C' C'')
  proof (induction n arbitrary: C)
    case 0

    have (∄a. ord-res.ground-factoring C a)
    proof (intro notI, elim exE)
      fix D assume ord-res.ground-factoring C D
      thus False
      proof (cases rule: ord-res.ground-factoring.cases)
        case (ground-factoringI A' P')
        hence A' = A
          using ‹ord-res.is-maximal-lit (Pos A) C›
          using linorder-lit.Uniq-is-maximal-in-mset
          by (metis Uniq-D literal.inject(1))
        thus False
          using ‹count C (Pos A) = Suc 0› ‹C = add-mset (Pos A') (add-mset
(Pos A') P')› by simp
      qed
    qed
    thus ?case
      by (metis 0.prems(1) 0.prems(3) diff-zero
          minus-mset-replicate-mset-eq-add-mset-filter-mset replicate-mset-0 rtran-
clp.rtrancl-refl)
  next
    case (Suc x)
    then show ?case
    by (metis minus-mset-replicate-mset-eq-add-mset-filter-mset tranclp-into-rtranclp
        tranclp-ord-res-ground-factoring-iff)
  qed
qed

lemma efac-spec: efac C = C ∨
  (∃A. ord-res.is-maximal-lit (Pos A) C ∧ efac C = add-mset (Pos A) {#L ∈#
C. L ≠ Pos A#})
  using efac-eq-disj[of C]
proof (elim disjE)
  assume efac C = C
```

```
  thus efac C = C ∨
    (∃ A. ord-res.is-maximal-lit (Pos A) C ∧ efac C = add-mset (Pos A) {#L ∈#
C. L ≠ Pos A#})
    by metis
next
  assume ∃!C'. efac C = C' ∧ ord-res.ground-factoring** C C' ∧
    (∄ C''. ord-res.ground-factoring C' C'')
  then obtain C' where
    efac C = C' and
    ord-res.ground-factoring** C C' ∧ (∄ C''. ord-res.ground-factoring C' C'')
    by metis
  thus efac C = C ∨
    (∃ A. ord-res.is-maximal-lit (Pos A) C ∧ efac C = add-mset (Pos A) {#L ∈#
C. L ≠ Pos A#})
    unfolding rtrancl-ground-factoring-iff
    by metis
qed

lemma efac-spec-if-pos-lit-is-maximal:
  assumes L-pos: is-pos L and L-max: ord-res.is-maximal-lit L C
  shows efac C = add-mset L {#K ∈# C. K ≠ L#}
proof −
  from assms obtain C' where
    efac C = C' and
    ord-res.ground-factoring** C C' ∧ (∄ C''. ord-res.ground-factoring C' C'')
    using ex1-efac-eq-factoring-chain by metis
  thus ?thesis
    unfolding rtrancl-ground-factoring-iff
  proof (elim disjE conjE)
    assume hyps: ∄ A. ord-res.is-maximal-lit (Pos A) C ∧ 2 ≤ count C (Pos A)
C = C'
    with assms have count C L = 1
      by (metis One-nat-def in-countE is-pos-def le-less-linear less-2-cases-iff
          linorder-lit.is-maximal-in-mset-iff nat-less-le zero-less-Suc)
    hence C = add-mset L {#K ∈# C. K ≠ L#}
    by (metis One-nat-def diff-zero minus-mset-replicate-mset-eq-add-mset-filter-mset
          replicate-mset-0)
    thus efac C = add-mset L {#K ∈# C. K ≠ L#}
      using ‹efac C = C'› ‹C = C'› by argo
  next
    assume ∃ A. ord-res.is-maximal-lit (Pos A) C ∧ C' = add-mset (Pos A) {#L
∈# C. L ≠ Pos A#}
    thus efac C = add-mset L {#K ∈# C. K ≠ L#}
      by (metis L-max Uniq-D ‹efac C = C'› linorder-lit.Uniq-is-maximal-in-mset)
  qed
qed

lemma efac-mempty[simp]: efac {#} = {#}
  by (metis empty-iff linorder-lit.is-maximal-in-mset-iff set-mset-empty efac-spec)
```

lemma *set-mset-efac*[*simp*]: *set-mset* (*efac C*) = *set-mset C*
using *efac-spec*[*of C*]
proof (*elim disjE exE conjE*)
show *efac C* = *C* ⟹ *set-mset* (*efac C*) = *set-mset C*
by *simp*
next
fix *A*
assume *ord-res.is-maximal-lit* (*Pos A*) *C*
hence *Pos A* ∈# *C*
by (*simp add*: *linorder-lit.is-maximal-in-mset-iff*)

assume *efac-C-eq*: *efac C* = *add-mset* (*Pos A*) {#*L* ∈# *C*. *L* ≠ *Pos A*#}
show *set-mset* (*efac C*) = *set-mset C*
proof (*intro Set.subset-antisym Set.subsetI*)
fix *L* **assume** *L* ∈# *efac C*
then show *L* ∈# *C*
unfolding *efac-C-eq*
using ‹*Pos A* ∈# *C*› **by** *auto*
next
fix *L* **assume** *L* ∈# *C*
then show *L* ∈# *efac C*
unfolding *efac-C-eq*
by *simp*
qed
qed

lemma *efac-subset*: *efac C* ⊆# *C*
using *efac-spec*[*of C*]
proof (*elim disjE exE conjE*)
show *efac C* = *C* ⟹ *efac C* ⊆# *C*
by *simp*
next
fix *A*
assume *ord-res.is-maximal-lit* (*Pos A*) *C* **and**
efac-C-eq: *efac C* = *add-mset* (*Pos A*) {#*L* ∈# *C*. *L* ≠ *Pos A*#}
then show *efac C* ⊆# *C*
by (*smt* (*verit*, *ccfv-SIG*) *filter-mset-add-mset insert-DiffM insert-subset-eq-iff*
linorder-lit.is-maximal-in-mset-iff multiset-filter-subset)
qed

lemma *true-cls-efac-iff*[*simp*]:
fixes *I* :: *'f gterm set* **and** *C* :: *'f gclause*
shows *I* ⊨ *efac C* ⟷ *I* ⊨ *C*
by (*metis set-mset-efac true-cls-iff-set-mset-eq*)

lemma *obtains-positive-greatest-lit-if-efac-not-ident*:
assumes *efac C* ≠ *C*
obtains *L* **where** *is-pos L* **and** *linorder-lit.is-greatest-in-mset* (*efac C*) *L*

proof −
from ‹*efac C* ≠ *C*› **obtain** *A* **where**
Pos-A-maximal: *linorder-lit.is-maximal-in-mset C* (*Pos A*) **and**
efac-C-eq: *efac C* = *add-mset* (*Pos A*) {#*L* ∈# *C*. *L* ≠ *Pos A*#}
using *efac-spec* **by** *metis*

assume *hyp*: ⋀*L*. *is-pos L* ⟹ *linorder-lit.is-greatest-in-mset* (*efac C*) *L* ⟹ *thesis*
show *thesis*
proof (*rule hyp*)
show *is-pos* (*Pos A*)
by *simp*
next
show *linorder-lit.is-greatest-in-mset*(*efac C*) (*Pos A*)
unfolding *efac-C-eq linorder-lit.is-greatest-in-mset-iff*
using *Pos-A-maximal*[*unfolded linorder-lit.is-maximal-in-mset-iff*]
by *auto*
qed
qed

lemma *mempty-in-image-efac-iff*[*simp*]: {#} ∈ *efac* ‘ *N* ⟷ {#} ∈ *N*
by (*metis empty-iff imageE image-eqI linorder-lit.is-maximal-in-mset-iff set-mset-empty set-mset-efac efac-spec*)

lemma *greatest-literal-in-efacI*:
assumes *is-pos L* **and** *C-max-lit*: *linorder-lit.is-maximal-in-mset C L*
shows *linorder-lit.is-greatest-in-mset* (*efac C*) *L*
unfolding *efac-spec-if-pos-lit-is-maximal*[*OF assms*] *linorder-lit.is-greatest-in-mset-iff*
proof (*intro conjI ballI*)
show *L* ∈# *add-mset L* {#*K* ∈# *C*. *K* ≠ *L*#}
by *simp*
next
fix *y* :: ′*f gterm literal*
assume *y* ∈# *remove1-mset L* (*add-mset L* {#*K* ∈# *C*. *K* ≠ *L*#})
then show *y* ≺*l* *L*
using *C-max-lit*[*unfolded linorder-lit.is-maximal-in-mset-iff*]
by *auto*
qed

lemma *linorder-lit.is-maximal-in-mset* (*efac C*) *L* ⟷ *linorder-lit.is-maximal-in-mset C L*
by (*simp add*: *linorder-lit.is-maximal-in-mset-iff*)

lemma
assumes *is-pos L*
shows *linorder-lit.is-greatest-in-mset* (*efac C*) *L* ⟷ *linorder-lit.is-maximal-in-mset C L*
by (*metis* (*no-types*, *opaque-lifting*) *Uniq-D assms efac-spec greatest-literal-in-efacI linorder-lit.Uniq-is-greatest-in-mset linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*

literal.disc(*1*))

lemma *factorizable-if-neq-efac*:
assumes $C \neq efac\ C$
shows $\exists\, C'.$ *ord-res.ground-factoring* $C\ C'$
using *assms*
by (*metis converse-rtranclpE ex1-efac-eq-factoring-chain*)

lemma *nex-strictly-maximal-pos-lit-if-neq-efac*:
assumes $C \neq efac\ C$
shows $\nexists\, L.$ *is-pos* $L \wedge$ *ord-res.is-strictly-maximal-lit* $L\ C$
using *assms factorizable-if-neq-efac nex-strictly-maximal-pos-lit-if-factorizable* **by** *metis*

lemma *efac-properties-if-not-ident*:
assumes $efac\ C \neq C$
shows $efac\ C \prec_c C$ **and** $\{efac\ C\} \Vdash e\ \{C\}$
proof −
have $efac\ C \subseteq\# C$
using *efac-subset* **.**
hence $efac\ C \preceq_c C$
using *subset-implies-reflclp-multp* **by** *blast*
thus $efac\ C \prec_c C$
using ‹$efac\ C \neq C$› **by** *order*

show $\{efac\ C\} \Vdash e\ \{C\}$
using ‹$efac\ C \subseteq\# C$› *true-clss-subclause* **by** *metis*
qed

end

end
theory *ORD-RES-2*
imports
ORD-RES
Exhaustive-Factorization
begin

15 ORD-RES-2 (full factorization)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-2* **where**
factoring:
is-least-false-clause $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})\ C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C\ L \Longrightarrow$
is-pos $L \Longrightarrow$
$U_{ef}{}' = finsert\ (efac\ C)\ U_{ef} \Longrightarrow$
ord-res-2 $N\ (U_r,\ U_{ef})\ (U_r,\ U_{ef}{}')\ \mid$

resolution:
is-least-false-clause $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$ $C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
is-neg $L \Longrightarrow$
$D \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef} \Longrightarrow$
$D \prec_c C \Longrightarrow$
ord-res.production (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$) $D = \{$*atm-of* $L\} \Longrightarrow$
ord-res.ground-resolution C D $CD \Longrightarrow$
$U_r' =$ *finsert* CD $U_r \Longrightarrow$
ord-res-2 N $(U_r,\ U_{ef})$ $(U_r',\ U_{ef})$

inductive *ord-res-2-step* **where**
ord-res-2 N S $S' \Longrightarrow$ *ord-res-2-step* $(N,\ S)$ $(N,\ S')$

inductive *ord-res-2-final* **where**
ord-res-final $(N \mid\cup\mid U_r \mid\cup\mid U_{ef}) \Longrightarrow$ *ord-res-2-final* $(N,\ (U_r,\ U_{ef}))$

inductive *ord-res-2-load* **where**
$N \neq \{||\} \Longrightarrow$ *ord-res-2-load* N $(N,\ (\{||\},\ \{||\}))$

sublocale *ord-res-2-semantics*: *semantics* **where**
step = *ord-res-2-step* **and**
final = *ord-res-2-final*
proof *unfold-locales*
fix S :: *'f gterm clause fset* × *'f gterm clause fset* × *'f gterm clause fset*

obtain N U_r U_{ef} :: *'f gterm clause fset* **where**
S-def: $S = (N,\ (U_r,\ U_{ef}))$
by (*metis prod.exhaust*)

assume *ord-res-2-final* S
hence $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef} \vee \neg$ *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
by (*simp add*: *S-def ord-res-2-final.simps ord-res-final-def*)
thus *finished ord-res-2-step* S
proof (*elim disjE*)
assume $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
have *False* **if** *ord-res-2* N $(U_r,\ U_{ef})$ x **for** x
using *that*[*unfolded S-def*]
proof (*cases* N $(U_r,\ U_{ef})$ x *rule*: *ord-res-2.cases*)
case *hyps*: (*factoring* C L U_{ef}')
from *hyps* **have** $C = \{\#\}$
using *is-least-false-clause-mempty*[*OF* ‹$\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$›]
by (*metis Uniq-D Uniq-is-least-false-clause*)
moreover from *hyps* **have** $L \in\# C$
using *linorder-lit.is-maximal-in-mset-iff* **by** *blast*
ultimately show *False*
by *simp*
next

```
      case hyps: (resolution C L D CD U_ef')
      from hyps ‹{#} |∈| N |∪| U_r |∪| U_ef›› have C = {#}
        using is-least-false-clause-mempty[OF ‹{#} |∈| N |∪| U_r |∪| U_ef›]
        by (metis Uniq-D Uniq-is-least-false-clause)
      moreover from hyps have L ∈# C
        using linorder-lit.is-maximal-in-mset-iff by blast
      ultimately show False
        by simp
    qed
    thus finished ord-res-2-step S
      unfolding finished-def ord-res-2-step.simps S-def
      by (metis prod.inject)
  next
    assume no-false-cls: ¬ ex-false-clause (fset (N |∪| U_r |∪| U_ef))
    have False if ord-res-2 N (U_r, U_ef) x for x
      using that[unfolded S-def]
    proof (cases N (U_r, U_ef) x rule: ord-res-2.cases)
      case hyps: (factoring C L U_ef')
      thus False
        using no-false-cls[unfolded ex-false-clause-def]
        using is-least-false-clause-def linorder-cls.is-least-in-fset-iff by auto
    next
      case hyps: (resolution C L D CD U_ef')
      thus False
        using no-false-cls[unfolded ex-false-clause-def]
        using is-least-false-clause-def linorder-cls.is-least-in-fset-iff by auto
    qed
    thus finished ord-res-2-step S
      unfolding finished-def ord-res-2-step.simps S-def
      by (metis prod.inject)
  qed
qed

sublocale ord-res-2-language: language where
  step = ord-res-2-step and
  final = ord-res-2-final and
  load = ord-res-2-load
  by unfold-locales

lemma is-least-in-fset-with-irrelevant-clauses-if-is-least-in-fset:
  assumes
    irrelevant: ∀ D |∈| N'. ∃ E |∈| N. E ⊂# D ∧ set-mset D = set-mset E and
    C-least: linorder-cls.is-least-in-fset {|C |∈| N. ¬ ord-res-Interp (fset N) C ⊫
C|} C
  shows linorder-cls.is-least-in-fset {|C |∈| N |∪| N'. ¬ ord-res-Interp (fset (N |∪|
N')) C ⊫ C|} C
proof −
  have
    C-in: C |∈| N and
```

```
  C-not-entailed: ¬ ord-res-Interp (fset N) C ⊨ C and
  C-lt: ∀ x |∈| N. x ≠ C ⟶ ¬ ord-res-Interp (fset N) x ⊨ x ⟶ C ≺c x
  using C-least linorder-cls.is-least-in-ffilter-iff by simp-all

have C |∈| N |∪| N'
  using C-in by simp

moreover have ¬ ord-res-Interp (fset (N |∪| N')) C ⊨ C
  using extended-partial-model-entails-iff-partial-model-entails[
      of fset N fset N', OF finite-fset finite-fset irrelevant]
  using C-in C-not-entailed
  by simp

moreover have C ≺c x
  if
    x-in: x |∈| N |∪| N' and
    x-neq: x ≠ C and
    x-not-entailed: ¬ ord-res-Interp (fset (N |∪| N')) x ⊨ x
  for x
proof −

  from x-in have x |∈| N ∨ x |∈| N'
    by simp
  thus C ≺c x
  proof (elim disjE)
    assume x-in: x |∈| N

    moreover have ¬ ord-res-Interp (fset N) x ⊨ x
      using extended-partial-model-entails-iff-partial-model-entails[
          of fset N fset N', OF finite-fset finite-fset irrelevant x-in]
      using x-not-entailed by simp

    ultimately show C ≺c x
      using C-lt[rule-format, of x] x-neq by argo
  next
    assume x |∈| N'
    then obtain x' where x' |∈| N and x' ⊂# x set-mset x' = set-mset x
      using irrelevant by metis

    have x' ≺c x
      using ‹x' ⊂# x› by (metis strict-subset-implies-multp)

    moreover have C ⪯c x'
    proof (cases x' = C)
      case True
      thus ?thesis
        by order
    next
      case False
```

```
      have C ≺c x'
      proof (rule C-lt[rule-format])
        show x' |∈| N
          using ‹x' |∈| N› .
      next
        show x' ≠ C
          using False .
      next
        have ¬ ord-res-Interp (fset (N |∪| N')) x ⊨ x'
          using x-not-entailed ‹set-mset x' = set-mset x›
          by (metis true-cls-def)
        hence ¬ ord-res-Interp (fset (N |∪| N')) x' ⊨ x'
          by (metis ‹x' ≺c x› ord-res.entailed-clause-stays-entailed)
        thus ¬ ord-res-Interp (fset N) x' ⊨ x'
          using extended-partial-model-entails-iff-partial-model-entails[
              of fset N fset N' x', OF finite-fset finite-fset irrelevant]
          using ‹x' |∈| N› by simp
      qed
      thus ?thesis
        by order
    qed

    ultimately show C ≺c x
      by order
  qed
qed

ultimately show linorder-cls.is-least-in-fset {|C |∈| N |∪| N'.
  ¬ ord-res-Interp (fset (N |∪| N')) C ⊨ C|} C
  using C-in C-not-entailed
  unfolding linorder-cls.is-least-in-ffilter-iff by metis
qed

primrec fset-upto :: nat ⇒ nat ⇒ nat fset where
  fset-upto i 0 = (if i = 0 then {|0|} else {||}) |
  fset-upto i (Suc n) = (if i ≤ Suc n then finsert (Suc n) (fset-upto i n) else {||})

lemma
  fset-upto 0 0 = {|0|}
  fset-upto 0 1 = {|0, 1|}
  fset-upto 0 2 = {|0, 1, 2|}
  fset-upto 0 3 = {|0, 1, 2, 3|}
  fset-upto 1 3 = {|1, 2, 3|}
  fset-upto 2 3 = {|2, 3|}
  fset-upto 3 3 = {|3|}
  fset-upto 4 3 = {||}
  unfolding numeral-2-eq-2 numeral-3-eq-3
  by auto
```

```
lemma i ≤ 1 + j ⟹ List.upto i (1 + j) = List.upto i j @ [1 + j]
  using upto-rec2 by simp

lemma fset-of-append-singleton: fset-of-list (xs @ [x]) = finsert x (fset-of-list xs)
  by simp

lemma fset-of-list (List.upto (int i) (int j)) = int |‘| fset-upto i j
proof (induction j)
  case 0
  show ?case
    by simp
next
  case (Suc j)
  show ?case
  proof (cases i ≤ Suc j)
    case True
    hence AAA: int i ≤ 1 + int j
      by presburger

    from True show ?thesis
      apply simp
      unfolding Suc.IH[symmetric]
      unfolding upto-rec2[OF AAA] fset-of-append-singleton
      by simp
  next
    case False
    thus ?thesis
      by simp
  qed
qed

lemma fset-fset-upto[simp]: fset (fset-upto m n) = {m..n}
  apply (induction n)
  apply simp
  apply simp
  using atLeastAtMostSuc-conv by presburger

lemma minus-mset-replicate-strict-subset-minus-msetI:
  assumes m < n and n < count C L
  shows C − replicate-mset n L ⊂# C − replicate-mset m L
proof −
  from ‹m < n› obtain k1 where n-def: n = m + Suc k1
    using less-natE by auto

  with ‹n < count C L› obtain k2 where
    count-eq: count C L = m + Suc k1 + Suc k2
    by (metis add.commute add-Suc group-cancel.add1 less-natE)
```

```
  define C0 where
    C0 = {#K ∈# C. K ≠ L#}

  have C-eq: C = C0 + replicate-mset m L + replicate-mset (Suc k1) L + repli-
cate-mset (Suc k2) L
    using C0-def count-eq
   by (metis (mono-tags, lifting) filter-mset-eq group-cancel.add1 replicate-mset-plus
       union-filter-mset-complement)

  have C − replicate-mset n L = C0 + replicate-mset (Suc k2) L
    unfolding C-eq n-def
    by (simp add: replicate-mset-plus)
  also have ... ⊂# C0 + replicate-mset (Suc k1) L + replicate-mset (Suc k2) L
    by simp
  also have ... = C − replicate-mset m L
    unfolding C-eq
    by (simp add: replicate-mset-plus)
  finally show ?thesis .
qed

lemma factoring-all-is-between-efac-and-original-clause:
  fixes z
  assumes
    is-pos L and ord-res.is-maximal-lit L C and count C L = Suc (Suc n)
    m' ≤ n and
    z-in: z |∈| (λi. C − replicate-mset i L) |‘| fset-upto 0 m'
  shows efac C ⊂# z and z ⊆# C
proof −
  from z-in obtain i where
    i ≤ m' and
    z-def: z = C − replicate-mset i L
    by auto

  have i ≤ n
    using ‹i ≤ m'› ‹m' ≤ n› by presburger
  hence i < count C L
    using ‹count C L = Suc (Suc n)› by presburger
  thus z ⊆# C
    unfolding z-def by simp

  show efac C ⊂# z
  proof −
    have efac C = add-mset L {#K ∈# C. K ≠ L#}
      using efac-spec-if-pos-lit-is-maximal[OF ‹is-pos L› ‹ord-res.is-maximal-lit L
C›] .
    also have ... ⊂# add-mset L (add-mset L {#K ∈# C. K ≠ L#})
      by simp
    also have ... = C − replicate-mset n L
      using minus-mset-replicate-mset-eq-add-mset-add-mset-filter-mset[
```

```
      OF ‹count C L = Suc (Suc n)›] ..
  also have ... ⊆# C − replicate-mset i L
    using ‹i ≤ n› by (simp add: subseteq-mset-def)
  also have ... = z
    using z-def ..
  finally show ?thesis .
 qed
qed

lemma
 assumes
  linorder-cls.is-least-in-fset {|x |∈| N1. P N1 x|} x and
  linorder-cls.is-least-in-fset N2 y and
  ∀ z |∈| N2. z ⪯c x and
  P (N1 |∪| N2) y and
  ∀ z |∈| N1. z ≺c x ⟶ ¬ P (N1 |∪| N2) z
 shows linorder-cls.is-least-in-fset {|x |∈| N1 |∪| N2. P (N1 |∪| N2) x|} y
proof −
 show ?thesis
  unfolding linorder-cls.is-least-in-ffilter-iff
 proof (intro conjI ballI impI)
  from assms(2) show y |∈| N1 |∪| N2
   unfolding linorder-cls.is-least-in-fset-iff by simp
 next
  from assms(4) show P (N1 |∪| N2) y
   by argo
 next
  fix z
  assume z-in: z |∈| N1 |∪| N2 and z ≠ y and P (N1 |∪| N2) z
  show y ≺c z
   using z-in[unfolded funion-iff]
  proof (elim disjE)
   from assms(2,3,5) show z |∈| N1 ⟹ y ≺c z
   by (metis ‹P (N1 |∪| N2) z› ‹z ≠ y› linorder-cls.dual-order.not-eq-order-implies-strict
       linorder-cls.is-least-in-fset-iff linorder-cls.less-linear
       linorder-cls.order.strict-trans)
  next
   from assms(2) show z |∈| N2 ⟹ y ≺c z
    using ‹z ≠ y› linorder-cls.is-least-in-fset-iff by blast
  qed
 qed
qed

lemma ground-factoring-preserves-efac:
 assumes ord-res.ground-factoring P C
 shows efac P = efac C
 using assms
 by (smt (verit, ccfv-threshold) filter-mset-add-mset is-pos-def ord-res.ground-factoring.cases
   ord-res.ground-factoring-preserves-maximal-literal efac-spec-if-pos-lit-is-maximal)
```

lemma *ground-factorings-preserves-efac*:
assumes *ord-res.ground-factoring*** *P C*
shows *efac P* = *efac C*
using *assms*
by (*induction P rule*: *converse-rtranclp-induct*)
(*simp-all add*: *ground-factoring-preserves-efac*)

lemma *ex-ord-res-2-if-not-final*:
assumes ¬ *ord-res-2-final S*
shows ∃ *S'*. *ord-res-2-step S S'*
proof −
from *assms* **obtain** *N* U_r U_{ef} **where**
S-def: *S* = (*N*, (U_r, U_{ef})) **and**
{#} |∉| *N* |∪| U_r |∪| U_{ef} **and**
ex-false-clause (*fset* (*N* |∪| U_r |∪| U_{ef}))
by (*metis ord-res-2-final.intros ord-res-final-def surj-pair*)

obtain *C* **where** *C-least-false*: *is-least-false-clause* (*N* |∪| U_r |∪| U_{ef}) *C*
using ‹*ex-false-clause* (*fset* (*N* |∪| U_r |∪| U_{ef}))› *obtains-least-false-clause-if-ex-false-clause*
by *metis*

hence *C* ≠ {#}
using ‹{#} |∉| *N* |∪| U_r |∪| U_{ef}›
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *metis*

then obtain *L* **where** *C-max*: *linorder-lit.is-maximal-in-mset C L*
using *linorder-lit.ex-maximal-in-mset* **by** *metis*

show *?thesis*
proof (*cases L*)
case (*Pos A*)
thus *?thesis*
using *ord-res-2.factoring*[*OF C-least-false C-max*] *S-def is-pos-def*
by (*metis ord-res-2-step.intros*)
next
case (*Neg A*)
then obtain *D* **where**
D |∈| *N* |∪| U_r |∪| U_{ef} **and**
D ≺$_c$ *C* **and**
ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**
ord-res.production (*fset* (*N* |∪| U_r |∪| U_{ef})) *D* = {*A*}
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
using *C-least-false C-max* **by** *metis*

moreover then obtain *CD* **where**
ord-res.ground-resolution C D CD

using *ex-ground-resolutionI C-max Neg* **by** *metis*

ultimately show *?thesis*
using *ord-res-2.resolution*[*OF C-least-false C-max*]
by (*metis Neg S-def literal.disc*(*2*) *literal.sel*(*2*) *ord-res-2-step.intros*)
qed
qed

corollary *ord-res-2-step-safe*: *ord-res-2-final S* ∨ (∃ *S'*. *ord-res-2-step S S'*)
using *ex-ord-res-2-if-not-final* **by** *metis*

lemma *is-least-false-clause-if-is-least-false-clause-in-union-unproductive*:
assumes
N2-unproductive: ∀ *C* |∈| *N2*. *ord-res.production* (*fset* (*N1* |∪| *N2*)) *C* = {}
and
C-in: *C* |∈| *N1* **and**
C-least-false: *is-least-false-clause* (*N1* |∪| *N2*) *C*
shows *is-least-false-clause N1 C*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *C* |∈| *N1*
using *C-in* **.**
next
have ¬ *ord-res-Interp* (*fset* (*N1* |∪| *N2*)) *C* ⊨ *C*
using *C-least-false*[*unfolded is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*]
by *argo*
thus ¬ *ord-res.interp* (*fset N1*) *C* ∪ *ord-res.production* (*fset N1*) *C* ⊨ *C*
unfolding *Interp-union-unproductive*[*of fset N1 fset N2*, *folded union-fset*,
OF finite-fset finite-fset N2-unproductive] **.**
next
fix *D*
have ∀ *D* |∈| *N1* |∪| *N2*. *D* ≠ *C* ⟶ ¬ *ord-res-Interp* (*fset* (*N1* |∪| *N2*)) *D* ⊨ *D* ⟶ *C* ≺$_c$ *D*
using *C-least-false*[*unfolded is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*]
by *argo*

moreover assume *D* |∈| *N1* **and** *D* ≠ *C* **and** ¬ *ord-res-Interp* (*fset N1*) *D* ⊨ *D*

ultimately show *C* ≺$_c$ *D*
using *Interp-union-unproductive*[*of fset N1 fset N2*, *folded union-fset*,
OF finite-fset finite-fset N2-unproductive]
by *simp*
qed

lemma *ground-factoring-replicate-max-pos-lit*:
ord-res.ground-factoring
(*C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*))
(*C*$_0$ + *replicate-mset* (*Suc n*) (*Pos A*))

if *ord-res.is-maximal-lit* (*Pos A*) (*C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*))
for *A C*$_0$ *n*
proof (*rule ord-res.ground-factoringI*)
show *C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*) =
add-mset (*Pos A*) (*add-mset* (*Pos A*) (*C*$_0$ + *replicate-mset n* (*Pos A*)))
by *simp*
next
show *ord-res.is-maximal-lit* (*Pos A*) (*C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*))
using *that* **.**
next
show *C*$_0$ + *replicate-mset* (*Suc n*) (*Pos A*) =
add-mset (*Pos A*) (*C*$_0$ + *replicate-mset n* (*Pos A*))
by *simp*
qed *simp*

lemma *ground-factorings-replicate-max-pos-lit*:
assumes
ord-res.is-maximal-lit (*Pos A*) (*C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*))
shows *m* ≤ *Suc n* ⟹ (*ord-res.ground-factoring* ^^ *m*)
(*C*$_0$ + *replicate-mset* (*Suc* (*Suc n*)) (*Pos A*))
(*C*$_0$ + *replicate-mset* (*Suc* (*Suc n* − *m*)) (*Pos A*))
proof (*induction m*)
case *0*
show *?case*
by *simp*
next
case (*Suc m*′)
then show *?case*
apply (*cases m*′)
using *assms ground-factoring-replicate-max-pos-lit* **apply** *auto*[*1*]
by (*metis* (*no-types*, *lifting*) *Suc-diff-le Suc-leD assms diff-Suc-Suc*
ground-factoring-replicate-max-pos-lit ord-res.ground-factorings-preserves-maximal-literal
relpowp-Suc-I relpowp-imp-rtranclp)
qed

lemma *ord-res-Interp-entails-if-greatest-lit-is-pos*:
assumes *C-in*: *C* ∈ *N* **and** *L-greatest*: *linorder-lit.is-greatest-in-mset C L* **and**
L-pos: *is-pos L*
shows *ord-res-Interp N C* ⊨ *C*
proof (*cases ord-res.interp N C* ⊨ *C*)
case *True*
hence *ord-res.production N C* = {}
by (*simp add*: *ord-res.production-unfold*)
with *True* **show** *?thesis*
by *simp*
next
case *False*

```
  from L-pos obtain A where L-def: L = Pos A
    by (cases L) simp-all

  from L-greatest obtain C' where C-def: C = add-mset L C'
    unfolding linorder-lit.is-greatest-in-mset-iff
    by (metis insert-DiffM)

  with C-in L-greatest have A ∈ ord-res.production N C
    unfolding L-def ord-res.production-unfold
    using False
    by (simp add: linorder-lit.is-greatest-in-mset-iff multi-member-split)
  thus ?thesis
    by (simp add: true-cls-def C-def L-def)
qed

lemma right-unique-ord-res-2: right-unique (ord-res-2 N)
proof (rule right-uniqueI)
  fix s s' s'' :: 'f gterm clause fset × 'f gterm clause fset
  assume step1: ord-res-2 N s s' and step2: ord-res-2 N s s''
  show s' = s''
    using step1
  proof (cases N s s' rule: ord-res-2.cases)
    case hyps1: (factoring U_r1 U_ef1 C1 L1 U_ef'1)
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-2.cases)
      case (factoring U_r2 U_ef2 C2 L2 U_ef'2)
      with hyps1 show ?thesis
        by (metis Uniq-D Uniq-is-least-false-clause prod.inject)
    next
      case (resolution U_r U_ef C L D CD U_r')
      with hyps1 have False
      by (metis Pair-inject Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
            the1-equality')
      thus ?thesis ..
    qed
  next
    case hyps1: (resolution U_r1 U_ef1 C1 L1 D1 CD1 U_r'1)
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-2.cases)
      case (factoring U_r2 U_ef2 C2 L2 U_ef'2)
      with hyps1 have False
      by (metis Pair-inject Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
            the1-equality')
      thus ?thesis ..
    next
      case (resolution U_r U_ef C L D CD U_r')
      with hyps1 show ?thesis
```

by (*metis* (*mono-tags*, *lifting*) *Uniq-is-least-false-clause*
linorder-lit.Uniq-is-maximal-in-mset ord-res.Uniq-production-eq-singleton
ord-res.unique-ground-resolution prod.inject the1-equality')
qed
qed
qed

lemma *right-unique-ord-res-2-step*: *right-unique ord-res-2-step*
proof (*rule right-uniqueI*)
fix *x y z*
show *ord-res-2-step x y* $\Longrightarrow$ *ord-res-2-step x z* $\Longrightarrow$ $y = z$
apply (*cases x*; *cases y*; *cases z*)
apply (*simp add*: *ord-res-2-step.simps*)
using *right-unique-ord-res-2*[*THEN right-uniqueD*]
by *blast*
qed

end

end
theory *Exhaustive-Resolution*
imports *Background*
begin

16 Function for full resolution

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

definition *ground-resolution* **where**
ground-resolution D C CD = *ord-res.ground-resolution C D CD*

lemma *Uniq-ground-resolution*: $\exists_{\leq 1} DC$. *ground-resolution D C DC*
by (*simp add*: *ground-resolution-def ord-res.unique-ground-resolution*)

lemma *ground-resolution-terminates*: *wfP* (*ground-resolution D*)$^{-1-1}$
proof (*rule wfp-if-convertible-to-wfp*)
show *wfP* ($\prec_c$)
using *ord-res.wfP-less-cls* **.**
next
show $\bigwedge x\ y$. (*ground-resolution D*)$^{-1-1}$ *x y* $\Longrightarrow$ $x \prec_c y$
unfolding *ground-resolution-def conversep-iff*
using *ord-res.ground-resolution-smaller-conclusion* **by** *metis*
qed

lemma *not-ground-resolution-mempty-left*: $\neg$ *ground-resolution* {#} *C x*
by (*auto simp*: *ground-resolution-def elim*: *ord-res.ground-resolution.cases*)

lemma *not-ground-resolution-mempty-right*: $\neg$ *ground-resolution C* {#} *x*
by (*auto simp*: *ground-resolution-def elim*: *ord-res.ground-resolution.cases*)

lemma *not-tranclp-ground-resolution-mempty-left*: $\neg$ (*ground-resolution* {#})$^{++}$ *C x*
by (*metis not-ground-resolution-mempty-left tranclpD*)

lemma *not-tranclp-ground-resolution-mempty-right*: $\neg$ (*ground-resolution C*)$^{++}$ {#} *x*
by (*metis not-ground-resolution-mempty-right tranclpD*)

lemma *left-premise-lt-right-premise-if-ground-resolution*:
ground-resolution D C DC $\Longrightarrow$ *D* $\prec_c$ *C*
by (*auto simp*: *ground-resolution-def elim*: *ord-res.ground-resolution.cases*)

lemma *left-premise-lt-right-premise-if-tranclp-ground-resolution*:
(*ground-resolution D*)$^{++}$ *C DC* $\Longrightarrow$ *D* $\prec_c$ *C*
by (*induction DC rule*: *tranclp-induct*)
(*auto simp add*: *left-premise-lt-right-premise-if-ground-resolution*)

lemma *resolvent-lt-right-premise-if-ground-resolution*:
ground-resolution D C DC $\Longrightarrow$ *DC* $\prec_c$ *C*
by (*simp add*: *ground-resolution-def ord-res.ground-resolution-smaller-conclusion*)

lemma *resolvent-lt-right-premise-if-tranclp-ground-resolution*:
(*ground-resolution D*)$^{++}$ *C DC* $\Longrightarrow$ *DC* $\prec_c$ *C*
proof (*induction DC rule*: *tranclp-induct*)
case (*base y*)
thus *?case*
by (*simp add*: *resolvent-lt-right-premise-if-ground-resolution*)
next
case (*step y z*)
have *z* $\prec_c$ *y*
using *step.hyps resolvent-lt-right-premise-if-ground-resolution* **by** *metis*
thus *?case*
using *step.IH* **by** *order*
qed

Exhaustive resolution

definition *eres* **where**
eres D C = (*THE DC*. *full-run* (*ground-resolution D*) *C DC*)

The function *eres* performs exhaustive resolution between its two input clauses. The first clause is repeatedly used, while the second clause is only use to start the resolution chain.

lemma *eres-ident-iff*: *eres D C* = *C* $\longleftrightarrow$ ($\nexists$ *DC*. *ground-resolution D C DC*)
proof (*rule iffI*)
assume *eres D C* = *C*
thus $\nexists$ *DC*. *ground-resolution D C DC*
unfolding *eres-def*
by (*metis Uniq-full-run Uniq-ground-resolution full-run-def ground-resolution-terminates*

```
      ex1-full-run the1-equality')
next
  assume stuck: ∄DC. ground-resolution D C DC
  have (ground-resolution D)** C C
    by auto

  with stuck have full-run (ground-resolution D) C C
    unfolding full-run-def by argo

  moreover have Uniq: ∃≤1 y. full-run (ground-resolution D) C y
    by (metis Uniq-ground-resolution Uniq-full-run)

  ultimately show eres D C = C
    unfolding eres-def by (metis the1-equality')
qed

lemma
  assumes
    step1: ground-resolution D C DC and
    stuck: ∄DDC. ground-resolution D DC DDC
  shows eres D C = DC
proof −
  from step1 have (ground-resolution D)** C DC
    by auto

  with stuck have full-run (ground-resolution D) C DC
    unfolding full-run-def by argo

  moreover have Uniq: ∃≤1 y. full-run (ground-resolution D) C y
    by (metis Uniq-ground-resolution Uniq-full-run)

  ultimately show ?thesis
    unfolding eres-def by (metis the1-equality')
qed

lemma
  assumes
    step1: ground-resolution D C DC and
    step2: ground-resolution D DC DDC and
    stuck: ∄DDDC. ground-resolution D DDC DDDC
  shows eres D C = DDC
proof −
  from step1 have (ground-resolution D)** C DC
    by auto

  with step2 have (ground-resolution D)** C DDC
    by (metis rtranclp.simps)

  with stuck have full-run (ground-resolution D) C DDC
```

```
    unfolding full-run-def by argo

  moreover have Uniq: ∃≤1 y. full-run (ground-resolution D) C y
    by (metis Uniq-ground-resolution Uniq-full-run)

  ultimately show ?thesis
    unfolding eres-def by (metis the1-equality')
qed

lemma
  assumes
    step1: ground-resolution D C DC and
    step2: ground-resolution D DC DDC and
    step3: ground-resolution D DDC DDDC and
    stuck: ∄DDDDC. ground-resolution D DDDC DDDDC
  shows eres D C = DDDC
proof −
  from step1 have (ground-resolution D)** C DC
    by auto

  with step2 have (ground-resolution D)** C DDC
    by (metis rtranclp.simps)

  with step3 have (ground-resolution D)** C DDDC
    by (metis rtranclp.simps)

  with stuck have full-run (ground-resolution D) C DDDC
    unfolding full-run-def by argo

  moreover have Uniq: ∃≤1 y. full-run (ground-resolution D) C y
    by (metis Uniq-ground-resolution Uniq-full-run)

  ultimately show ?thesis
    unfolding eres-def by (metis the1-equality')
qed

lemma eres-mempty-left[simp]: eres {#} C = C
  unfolding eres-def
 by (metis Uniq-full-run Uniq-ground-resolution full-run-def not-ground-resolution-mempty-left
      rtranclp.rtrancl-refl the1-equality')

lemma eres-mempty-right[simp]: eres C {#} = {#}
  unfolding eres-def
 by (metis Uniq-full-run Uniq-ground-resolution full-run-def not-ground-resolution-mempty-right
      rtranclp.rtrancl-refl the1-equality')

lemma ex1-eres-eq-full-run-ground-resolution: ∃!DC. eres D C = DC ∧ full-run
(ground-resolution D) C DC
  using ex1-full-run[of ground-resolution D C]
```

by (*metis Uniq-ground-resolution eres-def ground-resolution-terminates theI'*)

lemma *eres-le*: *eres D C* $\preceq_c$ *C*
proof −
have *full-run* (*ground-resolution D*) *C* (*eres D C*)
using *ex1-eres-eq-full-run-ground-resolution* **by** *metis*
thus *?thesis*
proof (*rule full-run-preserves-invariant*)
show *C* $\preceq_c$ *C*
by *simp*
next
show $\bigwedge x\ y.$ *ground-resolution D x y* $\Longrightarrow$ *x* $\preceq_c$ *C* $\Longrightarrow$ *y* $\preceq_c$ *C*
unfolding *ground-resolution-def*
using *ord-res.ground-resolution-smaller-conclusion* **by** *fastforce*
qed
qed

lemma *clause-lt-clause-if-max-lit-comp*:
assumes *E-max-lit*: *linorder-lit.is-maximal-in-mset E L* **and** *L-neg*: *is-neg L* **and**
D-max-lit: *linorder-lit.is-maximal-in-mset D* (− *L*)
shows *D* $\prec_c$ *E*
proof −
have − *L* $\prec_l$ *L*
using *L-neg* **by** (*cases L*) *simp-all*

thus *?thesis*
using *D-max-lit E-max-lit*
by (*metis linorder-lit.multp-if-maximal-less-that-maximal*)
qed

lemma *eres-lt-if*:
assumes *E-max-lit*: *ord-res.is-maximal-lit L E* **and** *L-neg*: *is-neg L* **and**
D-max-lit: *linorder-lit.is-greatest-in-mset D* (− *L*)
shows *eres D E* $\prec_c$ *E*
proof −
have *eres D E* $\neq$ *E*
unfolding *eres-ident-iff not-not ground-resolution-def*
proof (*rule ex-ground-resolutionI*)
show *ord-res.is-maximal-lit* (*Neg* (*atm-of L*)) *E*
using *E-max-lit L-neg* **by** (*cases L*) *simp-all*
next
show *D* $\prec_c$ *E*
using *E-max-lit D-max-lit L-neg*
by (*metis clause-lt-clause-if-max-lit-comp*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
next
show *ord-res.is-strictly-maximal-lit* (*Pos* (*atm-of L*)) *D*
using *D-max-lit* ‹*is-neg L*›
by (*cases L*) *simp-all*

qed

thus *eres D E* $\prec_c$ *E*
using *eres-le*[*of D E*] **by** *order*
qed

lemma *eres-eq-after-ground-resolution*:
assumes *ground-resolution D C DC*
shows *eres D C* = *eres D DC*
using *assms*
by (*metis* (*no-types*, *opaque-lifting*) *Uniq-def Uniq-full-run Uniq-ground-resolution converse-rtranclpE ex1-eres-eq-full-run-ground-resolution full-run-def*)

lemma *eres-eq-after-rtranclp-ground-resolution*:
assumes (*ground-resolution D*)** *C DC*
shows *eres D C* = *eres D DC*
using *assms*
by (*induction DC rule*: *rtranclp-induct*) (*simp-all add*: *eres-eq-after-ground-resolution*)

lemma *eres-eq-after-tranclp-ground-resolution*:
assumes (*ground-resolution D*)$^{++}$ *C DC*
shows *eres D C* = *eres D DC*
using *assms*
by (*induction DC rule*: *tranclp-induct*) (*simp-all add*: *eres-eq-after-ground-resolution*)

lemma *resolvable-if-neq-eres*:
assumes *C* ≠ *eres D C*
shows ∃!*DC*. *ground-resolution D C DC*
using *assms ex1-eres-eq-full-run-ground-resolution*
by (*metis* (*no-types*, *opaque-lifting*) *Uniq-def Uniq-full-run Uniq-ground-resolution full-run-def*
rtranclp.rtrancl-refl)

lemma *nex-maximal-pos-lit-if-resolvable*:
assumes *ground-resolution D C DC*
shows ∄*L*. *is-pos L* ∧ *ord-res.is-maximal-lit L C*
using *assms* **unfolding** *ground-resolution-def*
by (*metis Uniq-D empty-iff is-pos-def linorder-lit.Uniq-is-maximal-in-mset literal.simps*(*4*) *ord-res.ground-resolution.cases set-mset-empty*)

corollary *nex-strictly-maximal-pos-lit-if-resolvable*:
assumes *ground-resolution D C DC*
shows ∄*L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*
using *assms nex-maximal-pos-lit-if-resolvable* **by** *blast*

corollary *nex-maximal-pos-lit-if-neq-eres*:
assumes *C* ≠ *eres D C*
shows ∄*L*. *is-pos L* ∧ *ord-res.is-maximal-lit L C*
using *assms resolvable-if-neq-eres nex-maximal-pos-lit-if-resolvable* **by** *metis*

corollary *nex-strictly-maximal-pos-lit-if-neq-eres*:
assumes $C \neq eres\ D\ C$
shows $\nexists L.\ is\text{-}pos\ L \wedge ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit\ L\ C$
using *assms resolvable-if-neq-eres nex-strictly-maximal-pos-lit-if-resolvable* **by** *metis*

lemma *ground-resolutionD*:
assumes *ground-resolution D C DC*
shows $\exists m\ A\ D'\ C'.$
$linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ D\ (Pos\ A) \wedge$
$linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ (Neg\ A) \wedge$
$D = add\text{-}mset\ (Pos\ A)\ D' \wedge$
$C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C' \wedge Neg\ A \notin\# C' \wedge$
$DC = D' + replicate\text{-}mset\ m\ (Neg\ A) + C'$
using *assms*
unfolding *ground-resolution-def*
proof (*cases C D DC rule*: *ord-res.ground-resolution.cases*)
case (*ground-resolutionI A C' D'*)

then obtain m **where** $count\ C\ (Neg\ A) = Suc\ m$
by *simp*

define C'' **where**
$C'' = \{\#L \in\# C.\ L \neq Neg\ A\#\}$

have $C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C''$
using ‹$count\ C\ (Neg\ A) = Suc\ m$› C''*-def*
by (*metis filter-eq-replicate-mset union-filter-mset-complement*)

show *?thesis*
proof (*intro exI conjI*)
show $linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ D\ (Pos\ A)$
using ‹$linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ D\ (Pos\ A)$› **.**
next
show $linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ (Neg\ A)$
using *ground-resolutionI* **by** *simp*
next
show $D = add\text{-}mset\ (Pos\ A)\ D'$
using ‹$D = add\text{-}mset\ (Pos\ A)\ D'$› **.**
next
show $C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C''$
using ‹$C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C''$› **.**
next
show $Neg\ A \notin\# C''$
by (*simp add*: C''*-def*)
next
show $DC = D' + replicate\text{-}mset\ m\ (Neg\ A) + C''$
using ‹$DC = C' + D'$› ‹$C = add\text{-}mset\ (Neg\ A)\ C'$› ‹$C = replicate\text{-}mset\ (Suc$

```
m) (Neg A) + C''›
      by simp
  qed
qed

lemma relpowp-ground-resolutionD:
  assumes n ≠ 0 and (ground-resolution D ^^ n) C DnC
  shows ∃ m A D' C'. Suc m ≥ n ∧
    linorder-lit.is-greatest-in-mset D (Pos A) ∧
    linorder-lit.is-maximal-in-mset C (Neg A) ∧
    D = add-mset (Pos A) D' ∧
    C = replicate-mset (Suc m) (Neg A) + C' ∧ Neg A ∉# C' ∧
    DnC = repeat-mset n D' + replicate-mset (Suc m − n) (Neg A) + C'
  using assms
proof (induction n arbitrary: C)
  case 0
  hence False
    by simp
  thus ?case ..
next
  case (Suc n')
  then obtain DC where
    ground-resolution D C DC and (ground-resolution D ^^ n') DC DnC
    by (metis relpowp-Suc-E2)

  then obtain m A D' C' where
    linorder-lit.is-greatest-in-mset D (Pos A) and
    linorder-lit.is-maximal-in-mset C (Neg A)
    D = add-mset (Pos A) D' and
    C = replicate-mset (Suc m) (Neg A) + C' and
    Neg A ∉# C' and
    DC = D' + replicate-mset m (Neg A) + C'
    using ‹ground-resolution D C DC›[THEN ground-resolutionD] by metis

  have Neg A ∉# D'
    using ‹linorder-lit.is-greatest-in-mset D (Pos A)›
    unfolding ‹D = add-mset (Pos A) D'›
    unfolding linorder-lit.is-greatest-in-mset-iff
    by auto

  show ?case
  proof (cases n')
    case 0
    hence DnC = DC
      using ‹(ground-resolution D ^^ n') DC DnC› by simp

    show ?thesis
      unfolding 0 ‹DnC = DC›
      unfolding repeat-mset-Suc repeat-mset-0 empty-neutral
```

```
    unfolding diff-Suc-Suc minus-nat.diff-0
    using ‹C = replicate-mset (Suc m) (Neg A) + C'› ‹D = add-mset (Pos A) D'›
      ‹DC = D' + replicate-mset m (Neg A) + C'› ‹Neg A ∉# C'›
      ‹linorder-lit.is-greatest-in-mset D (Pos A)› ‹linorder-lit.is-maximal-in-mset C (Neg A)›
    using linorder-lit.is-greatest-in-mset-iff
    by blast
 next
  case (Suc n'')
  hence n' ≠ 0
    by presburger
  then obtain m' A' D'' DC' where n' ≤ Suc m' and
    ord-res.is-strictly-maximal-lit (Pos A') D and
    ord-res.is-maximal-lit (Neg A') DC and
    D = add-mset (Pos A') D'' and
    DC = replicate-mset (Suc m') (Neg A') + DC' and
    Neg A' ∉# DC' and
    DnC = repeat-mset n' D'' + replicate-mset (Suc m' − n') (Neg A') + DC'
    using Suc.IH[OF - ‹(ground-resolution D ^^ n') DC DnC›]
    by metis

  have A' = A
  using ‹ord-res.is-strictly-maximal-lit (Pos A') D› ‹ord-res.is-strictly-maximal-lit (Pos A) D›
    by (meson Uniq-D linorder-lit.Uniq-is-maximal-in-mset
      linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.inject(1))

  hence D'' = D'
    using ‹D = add-mset (Pos A') D''› ‹D = add-mset (Pos A) D'› by auto

  have m = Suc m'
    using ‹DC = D' + replicate-mset m (Neg A) + C'› ‹DC = replicate-mset (Suc m') (Neg A') + DC'›
      ‹Neg A ∉# D'› ‹Neg A ∉# C'› ‹Neg A' ∉# DC'›
    unfolding ‹A' = A›
  by (metis add-0 count-eq-zero-iff count-replicate-mset count-union union-commute)

  hence DC' = D' + C'
    using ‹DC = D' + replicate-mset m (Neg A) + C'› ‹DC = replicate-mset (Suc m') (Neg A') + DC'›
    by (simp add: ‹A' = A›)

  show ?thesis
  proof (intro exI conjI)
    show Suc n' ≤ Suc (Suc m')
      using ‹n' ≤ Suc m'› by presburger
  next
    show ord-res.is-strictly-maximal-lit (Pos A) D
```

```
      using ‹ord-res.is-strictly-maximal-lit (Pos A) D› .
    next
      show ord-res.is-maximal-lit (Neg A) C
        using ‹ord-res.is-maximal-lit (Neg A) C› by metis
    next
      show D = add-mset (Pos A) D'
        using ‹D = add-mset (Pos A) D'› .
    next
      show C = replicate-mset (Suc (Suc m')) (Neg A) + C'
        using ‹C = replicate-mset (Suc m) (Neg A) + C'› ‹m = Suc m'› by argo
    next
      show Neg A ∉# C'
        using ‹Neg A ∉# C'› .
    next
      show DnC = repeat-mset (Suc n') D' + replicate-mset (Suc (Suc m') - Suc
n') (Neg A) + C'
        using ‹DnC = repeat-mset n' D'' + replicate-mset (Suc m' - n') (Neg A')
+ DC'›
        unfolding ‹A' = A› ‹D'' = D'› diff-Suc-Suc ‹DC' = D' + C'›
        by simp
    qed
  qed
qed


lemma tranclp-ground-resolutionD:
  assumes (ground-resolution D)^++ C DnC
  shows ∃ n m A D' C'. Suc m ≥ Suc n ∧
    linorder-lit.is-greatest-in-mset D (Pos A) ∧
    linorder-lit.is-maximal-in-mset C (Neg A) ∧
    D = add-mset (Pos A) D' ∧
    C = replicate-mset (Suc m) (Neg A) + C' ∧ Neg A ∉# C' ∧
    DnC = repeat-mset (Suc n) D' + replicate-mset (Suc m - Suc n) (Neg A) +
C'
proof -
  from assms obtain n :: nat where
    (ground-resolution D ^^ Suc n) C DnC
    by (metis Suc-pred tranclp-power)
  thus ?thesis
    using assms relpowp-ground-resolutionD
    by (meson nat.discI)
qed

lemma eres-not-identD:
  assumes eres D C ≠ C
  shows ∃ m A D' C'.
    linorder-lit.is-greatest-in-mset D (Pos A) ∧
    linorder-lit.is-maximal-in-mset C (Neg A) ∧
    D = add-mset (Pos A) D' ∧
```

```
    C = replicate-mset (Suc m) (Neg A) + C' ∧ Neg A ∉# C' ∧
    eres D C = repeat-mset (Suc m) D' + C'
proof −
  have ⋀n. Suc n ≠ 0
    by presburger

  obtain n where
    steps: (ground-resolution D ^^ Suc n) C (eres D C) and
    stuck: ∄x. ground-resolution D (eres D C) x
    using ‹eres D C ≠ C› ex1-eres-eq-full-run-ground-resolution
    by (metis full-run-def gr0-conv-Suc rtranclpD tranclp-power)

  obtain m A D' C' where
    Suc n ≤ Suc m and
    D-max-lit: ord-res.is-strictly-maximal-lit (Pos A) D and
    C-max-lit: ord-res.is-maximal-lit (Neg A) C and
    D-eq: D = add-mset (Pos A) D' and
    C-eq: C = replicate-mset (Suc m) (Neg A) + C' and
    Neg A ∉# C' and
    eres-eq: eres D C = repeat-mset (Suc n) D' + replicate-mset (Suc m − Suc n)
(Neg A) + C'
    using relpowp-ground-resolutionD[of Suc n, OF ‹Suc n ≠ 0› steps] by metis

  from stuck have count (eres D C) (Neg A) = 0
  proof (rule contrapos-np)
    assume count (eres D C) (Neg A) ≠ 0
    then obtain ERES' where eres D C = add-mset (Neg A) ERES'
      by (meson count-eq-zero-iff mset-add)

    moreover have ord-res.is-maximal-lit (Neg A) (eres D C)
      unfolding linorder-lit.is-maximal-in-mset-iff
    proof (intro conjI ballI impI)
      show Neg A ∈# eres D C
        unfolding ‹eres D C = add-mset (Neg A) ERES'› by simp
    next
      fix L
      assume L ∈# eres D C and L ≠ Neg A
      hence L ∈# repeat-mset (Suc n) D' ∨ L ∈# C'
        unfolding eres-eq
        by (metis Zero-not-Suc count-replicate-mset in-countE union-iff)
      thus ¬ Neg A ≺l L
      proof (elim disjE)
        assume L ∈# repeat-mset (Suc n) D'
        hence L ∈# D'
          using member-mset-repeat-msetD by metis
        hence L ≺l Pos A
          using D-max-lit
          unfolding D-eq linorder-lit.is-greatest-in-mset-iff by simp
        also have Pos A ≺l Neg A
```

```
        by simp
      finally show ?thesis
        by order
    next
      assume L ∈# C'
      thus ?thesis
        using C-eq ‹L ≠ Neg A› C-max-lit linorder-lit.is-maximal-in-mset-iff by
auto
    qed
  qed

  moreover have D ≺c eres D C
    using D-max-lit
    using ‹ord-res.is-maximal-lit (Neg A) (eres D C)›
    using linorder-lit.multpHO-if-maximal-less-that-maximal[of D Pos A eres D
C Neg A, simplified]
    using multpDM-imp-multp multpHO-imp-multpDM by blast

  ultimately show ∃x. ground-resolution D (eres D C) x
    unfolding ground-resolution-def
    using D-eq D-max-lit
    using ord-res.ground-resolutionI[of eres D C A ERES' D D' ERES' + D']
    by metis
qed

hence m = n
  using ‹eres D C = repeat-mset (Suc n) D' + replicate-mset (Suc m − Suc n)
(Neg A) + C'›
  using ‹Suc n ≤ Suc m› by auto

show ?thesis
proof (intro exI conjI)
  show ord-res.is-strictly-maximal-lit (Pos A) D
    using D-max-lit .
next
  show ord-res.is-maximal-lit (Neg A) C
    using C-max-lit .
next
  show D = add-mset (Pos A) D'
    using D-eq .
next
  show C = replicate-mset (Suc m) (Neg A) + C'
    using C-eq .
next
  show Neg A ∉# C'
    using ‹Neg A ∉# C'› .
next
  show eres D C = repeat-mset (Suc m) D' + C'
    using eres-eq unfolding ‹m = n› by simp
```

```
  qed
qed

lemma lit-in-one-of-resolvents-if-in-eres:
  fixes L :: 'f gterm literal and C D :: 'f gclause
  assumes L ∈# eres C D
  shows L ∈# C ∨ L ∈# D
proof (cases eres C D = D)
  assume eres C D = D
  thus L ∈# C ∨ L ∈# D
    using ‹L ∈# eres C D› by argo
next
  assume eres C D ≠ D
  thus L ∈# C ∨ L ∈# D
    using ‹L ∈# eres C D›
   by (metis eres-not-identD member-mset-repeat-msetD repeat-mset-distrib-add-mset
union-iff)
qed

lemma strong-lit-in-one-of-resolvents-if-in-eres:
  fixes L :: 'f gterm literal and C D :: 'f gclause
  assumes
    D-max-lit: linorder-lit.is-maximal-in-mset D L and
    K-in: K ∈# eres C D
  shows K ∈# C ∧ K ≠ −L ∨ K ∈# D
proof (cases eres C D = D)
  assume eres C D = D
  thus K ∈# C ∧ K ≠ −L ∨ K ∈# D
    using K-in by argo
next
  assume eres C D ≠ D
  then obtain m :: nat and A :: 'f gterm and C' D' :: 'f gterm literal multiset
where
    C-max-lit: ord-res.is-strictly-maximal-lit (Pos A) C and
    D-max-lit': ord-res.is-maximal-lit (Neg A) D and
    C-eq: C = add-mset (Pos A) C' and
    D-eq: D = replicate-mset (Suc m) (Neg A) + D' and
    Neg A ∉# D' and
    eres C D = repeat-mset (Suc m) C' + D'
    using eres-not-identD by metis

  have L-eq: L = Neg A
   using D-max-lit D-max-lit' by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset)

  have K ∈# repeat-mset (Suc m) C' + D'
    using K-in unfolding ‹eres C D = repeat-mset (Suc m) C' + D'› .

  hence K ∈# repeat-mset (Suc m) C' ∨ K ∈# D'
    unfolding Multiset.union-iff .
```

```
  hence K ∈# C' ∨ K ∈# D'
    unfolding member-mset-repeat-mset-Suc .

  thus K ∈# C ∧ K ≠ −L ∨ K ∈# D
  proof (elim disjE)
    assume K ∈# C'

    hence K ∈# C ∧ K ≠ − L
      using C-max-lit
      unfolding C-eq L-eq linorder-lit.is-greatest-in-mset-iff by auto

    thus K ∈# C ∧ K ≠ − L ∨ K ∈# D ..
  next
    assume K ∈# D'

    hence K ∈# D
      unfolding D-eq by simp

    thus K ∈# C ∧ K ≠ − L ∨ K ∈# D ..
  qed
qed

lemma stronger-lit-in-one-of-resolvents-if-in-eres:
  fixes K L :: 'f gterm literal and C D :: 'f gclause
  assumes eres C D ≠ D and
    D-max-lit: linorder-lit.is-maximal-in-mset D L and
    K-in-eres: K ∈# eres C D
  shows K ∈# C ∧ K ≠ − L ∨ K ∈# D ∧ K ≠ L
proof −
  obtain m :: nat and A :: 'f gterm and C' D' :: 'f gterm literal multiset where
    C-max-lit: ord-res.is-strictly-maximal-lit (Pos A) C and
    C-def: C = add-mset (Pos A) C' and
    D = replicate-mset (Suc m) (Neg A) + D' and
    Neg A ∉# D' and
    eres C D = repeat-mset (Suc m) C' + D'
    using ‹eres C D ≠ D›[THEN eres-not-identD] by metis

  have L = Neg A
    using assms(1) D-max-lit C-max-lit
    by (metis ground-resolutionD linorder-lit.Uniq-is-greatest-in-mset
      linorder-lit.Uniq-is-maximal-in-mset resolvable-if-neq-eres the1-equality' umi-
nus-Pos)

  have K ∈# repeat-mset (Suc m) C' + D'
    using K-in-eres unfolding ‹eres C D = repeat-mset (Suc m) C' + D'› .

  hence K ∈# repeat-mset (Suc m) C' ∨ K ∈# D'
    unfolding Multiset.union-iff .
```

```
  hence K ∈# C' ∨ K ∈# D'
    unfolding member-mset-repeat-mset-Suc .

  thus K ∈# C ∧ K ≠ - L ∨ K ∈# D ∧ K ≠ L
  proof (elim disjE)
    assume K ∈# C'

    hence K ∈# C ∧ K ≠ - L
      using C-max-lit[unfolded linorder-lit.is-greatest-in-mset-iff]
      unfolding ‹C = add-mset (Pos A) C'› ‹L = Neg A›
      by auto

    thus ?thesis
      by argo
  next
    assume K ∈# D'

    hence K ∈# D ∧ K ≠ L
      unfolding ‹D = replicate-mset (Suc m) (Neg A) + D'› ‹L = Neg A›
      using ‹Neg A ∉# D'›
      by auto

    thus ?thesis
      by argo
  qed
qed

lemma lit-in-eres-lt-greatest-lit-in-grestest-resolvant:
  fixes K L :: 'f gterm literal and C D :: 'f gclause
  assumes eres C D ≠ D and
    D-max-lit: linorder-lit.is-maximal-in-mset D L and
    - L ∉# D and
    K-in-eres: K ∈# eres C D
  shows atm-of K ≺t atm-of L
proof -
  obtain m :: nat and A :: 'f gterm and C' D' :: 'f gterm literal multiset where
    C-max-lit: ord-res.is-strictly-maximal-lit (Pos A) C and
    C-def: C = add-mset (Pos A) C' and
    D = replicate-mset (Suc m) (Neg A) + D' and
    Neg A ∉# D' and
    eres C D = repeat-mset (Suc m) C' + D'
    using ‹eres C D ≠ D›[THEN eres-not-identD] by metis

  have L = Neg A
    using assms(1) D-max-lit C-max-lit
    by (metis ground-resolutionD linorder-lit.Uniq-is-greatest-in-mset
      linorder-lit.Uniq-is-maximal-in-mset resolvable-if-neq-eres the1-equality' umi-
nus-Pos)
```

```
  have K ∈# repeat-mset (Suc m) C' + D'
    using K-in-eres unfolding ‹eres C D = repeat-mset (Suc m) C' + D'› .

  hence K ∈# repeat-mset (Suc m) C' ∨ K ∈# D'
    unfolding Multiset.union-iff .

  hence K ∈# C' ∨ K ∈# D'
    unfolding member-mset-repeat-mset-Suc .

  thus atm-of K ≺t atm-of L
  proof (elim disjE)
    assume K ∈# C'
    hence K ≺l Pos A
      using C-max-lit C-def ‹L = Neg A›
      unfolding linorder-lit.is-greatest-in-mset-iff
      by simp
    thus atm-of K ≺t atm-of L
      unfolding ‹L = Neg A› literal.sel
      by (cases K) simp-all
  next
    assume K ∈# D'
    hence K ≺l Neg A
      by (metis D-max-lit ‹D = replicate-mset (Suc m) (Neg A) + D'› ‹L = Neg
A› ‹Neg A ∉# D'›
          linorder-lit.is-maximal-in-mset-iff linorder-lit.neqE union-iff)

    moreover have K ≠ Pos A
      using ‹- L ∉# D›
      using ‹D = replicate-mset (Suc m) (Neg A) + D'› ‹K ∈# D'› ‹L = Neg A›
by fastforce

    ultimately have K ≺l Pos A
      by (metis linorder-lit.less-asym linorder-lit.less-linear literal.exhaust
          ord-res.less-lit-simps(1) ord-res.less-lit-simps(3) ord-res.less-lit-simps(4))

    thus atm-of K ≺t atm-of L
      unfolding ‹L = Neg A› literal.sel
      by (cases K) simp-all
  qed
qed

lemma eres-entails-resolvent:
  fixes C D :: 'f gterm clause
  assumes (ground-resolution C)++ D0 D
  shows {eres C D0} ⊨e {D}
  unfolding true-clss-singleton
proof (intro allI impI)
  have eres C D0 = eres C D
```

using *assms eres-eq-after-tranclp-ground-resolution* **by** *metis*

obtain *n m* :: *nat* **and** *A* :: *'f gterm* **and** *C' D₀'* :: *'f gterm clause* **where**
Suc n ≤ *Suc m* **and**
ord-res.is-strictly-maximal-lit (*Pos A*) *C* **and**
ord-res.is-maximal-lit (*Neg A*) *D₀* **and**
C = *add-mset* (*Pos A*) *C'* **and**
D₀ = *replicate-mset* (*Suc m*) (*Neg A*) + *D₀'* **and**
Neg A ∉# *D₀'* **and**
D = *repeat-mset* (*Suc n*) *C'* + *replicate-mset* (*Suc m* − *Suc n*) (*Neg A*) + *D₀'*
using ‹(*ground-resolution C*)⁺⁺ *D₀ D*›[*THEN tranclp-ground-resolutionD*] **by** *metis*

fix *I* :: *'f gterm set*
assume *I* ⊨ *eres C D₀*
show *I* ⊨ *D*
proof (*cases eres C D₀* = *D*)
case *True*
thus *?thesis*
using ‹*I* ⊨ *eres C D₀*› **by** *argo*
next
case *False*
then obtain *k* :: *nat* **and** *D'* :: *'f gterm clause* **where**
ord-res.is-strictly-maximal-lit (*Pos A*) *C* **and**
C = *add-mset* (*Pos A*) *C'* **and**
D = *replicate-mset* (*Suc k*) (*Neg A*) + *D'* **and**
Neg A ∉# *D'* **and**
eres C D = *repeat-mset* (*Suc k*) *C'* + *D'*
unfolding ‹*eres C D₀* = *eres C D*›
using *eres-not-identD*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos A*) *C*› ‹*C* = *add-mset* (*Pos A*) *C'*›
by (*metis Uniq-D add-mset-remove-trivial linorder-lit.Uniq-is-greatest-in-mset literal.sel*(*1*))

have *I* ⊨ *repeat-mset* (*Suc k*) *C'* + *D'*
using ‹*I* ⊨ *eres C D₀*›
unfolding ‹*eres C D₀* = *eres C D*› ‹*eres C D* = *repeat-mset* (*Suc k*) *C'* + *D'*› .

hence *I* ⊨ *D'* ∨ *I* ⊨ *repeat-mset* (*Suc k*) *C'*
by *auto*

thus *I* ⊨ *D*
proof (*elim disjE*)
assume *I* ⊨ *D'*
thus *I* ⊨ *D*
unfolding ‹*D* = *replicate-mset* (*Suc k*) (*Neg A*) + *D'*›
by *simp*
next

```
      assume I ⊨ repeat-mset (Suc k) C'
      thus I ⊨ D
        using ‹D = replicate-mset (Suc k) (Neg A) + D'›
        using ‹D = repeat-mset (Suc n) C' + replicate-mset (Suc m − Suc n) (Neg
A) + D₀'›
      by (metis member-mset-repeat-msetD repeat-mset-Suc true-cls-def true-cls-union)
    qed
  qed
qed



lemma clause-true-if-resolved-true:
  assumes
    (ground-resolution D)^++ C DC and
    D-productive: ord-res.production N D ≠ {} and
    C-true: ord-res-Interp N DC ⊨ DC
  shows ord-res-Interp N C ⊨ C
proof −
  obtain n where
    steps: (ground-resolution D ^^ Suc n) C DC
    using ‹(ground-resolution D)^++ C DC›
    by (metis less-not-refl not0-implies-Suc tranclp-power)

  obtain m A D' C' where
    n ≤ m and
    ord-res.is-strictly-maximal-lit (Pos A) D and
    ord-res.is-maximal-lit (Neg A) C and
    D = add-mset (Pos A) D' and
    C = replicate-mset (Suc m) (Neg A) + C' and
    Neg A ∉# C' and
    DC = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A) + C'
    using relpowp-ground-resolutionD[OF Suc-not-Zero steps]
    by (metis diff-Suc-Suc Suc-le-mono)

  have Neg A ∉# D'
    by (metis ‹D = add-mset (Pos A) D'› ‹ord-res.is-strictly-maximal-lit (Pos A)
D›
      ord-res.less-lit-simps(4) linorder-lit.is-greatest-in-mset-iff linorder-trm.eq-refl
      linorder-trm.leD remove1-mset-add-mset-If)

  have DC ≺c C
  proof (cases m = n)
    case True
    show ?thesis
    proof (intro one-step-implies-multp[of - - - {#}, simplified] ballI)
      show C ≠ {#}
        by (simp add: ‹C = replicate-mset (Suc m) (Neg A) + C'›)
    next
```

fix L
assume $L \in\# DC$
hence $L \in\# D' \vee L \in\# C'$
unfolding ‹$DC = repeat\text{-}mset\ (Suc\ n)\ D' + replicate\text{-}mset\ (m - n)\ (Neg\ A) + C'$› ‹$m = n$›
using *member-mset-repeat-msetD* **by** *fastforce*
hence $L \prec_l Neg\ A$
using ‹*ord-res.is-strictly-maximal-lit* $(Pos\ A)\ D$› ‹*ord-res.is-maximal-lit* $(Neg\ A)\ C$›
unfolding ‹$D = add\text{-}mset\ (Pos\ A)\ D'$› ‹$C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C'$›
unfolding *linorder-lit.is-maximal-in-mset-iff linorder-lit.is-greatest-in-mset-iff*
by (*metis* ‹*Neg A* $\notin\#$ C'› *add-mset-remove-trivial ord-res.less-lit-simps*(*4*) *linorder-lit.antisym-conv3 linorder-lit.dual-order.strict-trans linorder-trm.dual-order.asym union-iff*)

moreover have $Neg\ A \in\# C$
by (*simp add*: ‹$C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C'$›)

ultimately show $\exists K \in\# C.\ L \prec_l K$
by *metis*
qed
next
case *False*
hence $n < m$
using ‹$n \leq m$› **by** *presburger*

have $multp_{HO}\ (\prec_l)\ DC\ C$
proof (*rule linorder-lit.multp$_{HO}$-if-same-maximal-and-count-lt*)
show *ord-res.is-maximal-lit* $(Neg\ A)\ DC$
unfolding *linorder-lit.is-maximal-in-mset-iff*
proof (*intro conjI ballI impI*)
show $Neg\ A \in\# DC$
unfolding ‹$DC = repeat\text{-}mset\ (Suc\ n)\ D' + replicate\text{-}mset\ (m - n)\ (Neg\ A) + C'$›
using ‹$n < m$› **by** *simp*
next
fix L
assume $L \in\# DC$ **and** $L \neq Neg\ A$
hence $L \in\# D' \vee L \in\# C'$
unfolding ‹$DC = repeat\text{-}mset\ (Suc\ n)\ D' + replicate\text{-}mset\ (m - n)\ (Neg\ A) + C'$›
by (*metis in-replicate-mset member-mset-repeat-msetD union-iff*)
thus $\neg\ Neg\ A \prec_l L$
using ‹*ord-res.is-strictly-maximal-lit* $(Pos\ A)\ D$› ‹*ord-res.is-maximal-lit* $(Neg\ A)\ C$›
unfolding ‹$D = add\text{-}mset\ (Pos\ A)\ D'$› ‹$C = replicate\text{-}mset\ (Suc\ m)\ (Neg\ A) + C'$›
unfolding *linorder-lit.is-maximal-in-mset-iff linorder-lit.is-greatest-in-mset-iff*

```
      by (metis ‹L ≠ Neg A› add-mset-diff-bothsides diff-zero
          linorder-lit.dual-order.strict-trans linorder-trm.less-irrefl
          ord-res.less-lit-simps(4) union-iff)
    qed
  next
    show ord-res.is-maximal-lit (Neg A) C
      using ‹ord-res.is-maximal-lit (Neg A) C› .
  next
    have count DC (Neg A) = count (repeat-mset (Suc n) D') (Neg A) +
    count (replicate-mset (m − n) (Neg A)) (Neg A) + count C' (Neg A)
      unfolding ‹DC = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg
A) + C'› by simp
    also have ... = count D' (Neg A) * Suc n + count {#Neg A#} (Neg A) *
(m − n) + count C' (Neg A)
      by simp
    also have ... = 0 * Suc n + 1 * (m − n) + 0
      by (simp add: ‹Neg A ∉# C'› ‹Neg A ∉# D'› count-eq-zero-iff)
    also have ... = m − n
      by presburger
    also have ... < Suc m
      by presburger
    also have ... = 1 * Suc m + 0
      by presburger
    also have ... = count {#Neg A#} (Neg A) * Suc m + count C' (Neg A)
      by (simp add: ‹Neg A ∉# C'› count-eq-zero-iff)
    also have ... = count (replicate-mset (Suc m) (Neg A)) (Neg A) + count C'
(Neg A)
      by simp
    also have ... = count C (Neg A)
      unfolding ‹C = replicate-mset (Suc m) (Neg A) + C'› by simp
    finally show count DC (Neg A) < count C (Neg A) .
  qed
  thus ?thesis
    by (simp add: multp_DM-imp-multp multp_HO-imp-multp_DM)
qed

with C-true have ord-res-Interp N C ⊨ DC
  using ord-res.entailed-clause-stays-entailed by metis

thus ord-res-Interp N C ⊨ C
  unfolding true-cls-def
proof (elim bexE)
  fix L
  assume
    L-in: L ∈# DC and
    L-true: ord-res-Interp N C ⊨l L

  from L-in have L ∈# D' ∨ L = Neg A ∨ L ∈# C'
    unfolding ‹DC = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A)
```

$+ C'\rangle$
by (*metis in-replicate-mset member-mset-repeat-msetD union-iff*)

moreover have $L \notin\# D'$
proof (*rule notI*)
assume $L \in\# D'$

moreover have $\neg$ *ord-res.interp N* (*add-mset* (*Pos A*) D') $\models$ *add-mset* (*Pos A*) D'
using *D-productive*[*unfolded* $\langle D$ = *add-mset* (*Pos A*) $D'\rangle$]
unfolding *ord-res.production-unfold*
by *fast*

ultimately have $\neg$ *ord-res.interp N* (*add-mset* (*Pos A*) D') $\models l$ L
by *auto*

have $L \prec_l$ *Pos A*
using $\langle D$ = *add-mset* (*Pos A*) $D'\rangle$ $\langle L \in\# D'\rangle$ $\langle$*ord-res.is-strictly-maximal-lit* (*Pos A*) $D\rangle$
linorder-lit.is-greatest-in-mset-iff **by** *fastforce*

have $\neg$ *ord-res-Interp N C* $\models l$ L
proof (*cases L*)
case (*Pos B*)
hence $B \notin$ *ord-res.interp N* (*add-mset* (*Pos A*) D')
using $\langle\neg$ *ord-res.interp N* (*add-mset* (*Pos A*) D') $\models l$ $L\rangle$ **by** *simp*

moreover have *add-mset* (*Pos A*) $D' \prec_c C$
by (*metis* $\langle D$ = *add-mset* (*Pos A*) $D'\rangle$ $\langle\bigwedge$*thesis*. ($\bigwedge m$ *A* D' C'. $[\![n \leq m$; *ord-res.is-strictly-maximal-lit* (*Pos A*) *D*; *ord-res.is-maximal-lit* (*Neg A*) *C*; D = *add-mset* (*Pos A*) D'; C = *replicate-mset* (*Suc m*) (*Neg A*) + C'; *Neg A* $\notin\#$ C'; DC = *repeat-mset* (*Suc n*) D' + *replicate-mset* ($m - n$) (*Neg A*) + $C']\!] \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*$\rangle$ *linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.multp*$_{HO}$*-if-maximal-less-that-maximal multp*$_{DM}$*-imp-multp multp*$_{HO}$*-imp-multp*$_{DM}$ *ord-res.less-lit-simps*(*2*) *reflclp-iff*)

ultimately have $B \notin$ *ord-res.interp N C*
using $\langle L \prec_l$ *Pos A*$\rangle$[*unfolded Pos*, *simplified*]
using *ord-res.interp-fixed-for-smaller-literals*
by (*metis* $\langle D$ = *add-mset* (*Pos A*) $D'\rangle$ $\langle$*ord-res.is-strictly-maximal-lit* (*Pos A*) *D*$\rangle$
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.sel(*1*))

moreover have $B \notin$ *ord-res.production N C*
by (*metis Uniq-D* $\langle$*ord-res.is-maximal-lit* (*Neg A*) *C*$\rangle$ *ground-ordered-resolution-calculus.mem-production linorder-lit.Uniq-is-maximal-in-mset linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.simps*(*4*) *ord-res.ground-ordered-resolution-calculus-axioms*)

ultimately show *?thesis*

```
      unfolding Pos by simp
    next
      case (Neg B)
      hence B ∈ ord-res.interp N (add-mset (Pos A) D')
        using ‹¬ ord-res.interp N (add-mset (Pos A) D') ⊨l L› by simp

      moreover have add-mset (Pos A) D' ≺c C
          by (metis ‹D = add-mset (Pos A) D'› ‹⋀thesis. (⋀m A D' C'. ⟦n
≤ m; ord-res.is-strictly-maximal-lit (Pos A) D; ord-res.is-maximal-lit (Neg A) C;
D = add-mset (Pos A) D'; C = replicate-mset (Suc m) (Neg A) + C'; Neg
A ∉# C'; DC = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A)
+ C'⟧ ⟹ thesis) ⟹ thesis› linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.multp_HO-if-maximal-less-that-maximal multp_DM-imp-multp multp_HO-imp-multp_DM
ord-res.less-lit-simps(2) reflclp-iff)

      ultimately have B ∈ ord-res.interp N C
        by (metis Un-iff ord-res.not-interp-to-Interp-imp-le linorder-cls.leD)

      then show ?thesis
        unfolding Neg
        by simp
    qed

    with L-true show False
      by contradiction
  qed

  ultimately have L ∈# C
    unfolding ‹C = replicate-mset (Suc m) (Neg A) + C'› by simp

  with L-true show ∃ L ∈# C. ord-res-Interp N C ⊨l L
    by metis
 qed
qed

lemma clause-true-if-eres-true:
 assumes
  (ground-resolution D1)^++ D2 C and
  C ≠ eres D1 C and
  eres-C-true: ord-res-Interp N (eres D1 C) ⊨ eres D1 C
 shows ord-res-Interp N C ⊨ C
proof −
 obtain n where
  steps: (ground-resolution D1 ^^ Suc n) D2 C
  using ‹(ground-resolution D1)^++ D2 C›
  by (metis less-not-refl not0-implies-Suc tranclp-power)

 obtain m A D' C' where
  n ≤ m and
```

```
    ord-res.is-strictly-maximal-lit (Pos A) D1 and
    ord-res.is-maximal-lit (Neg A) D2 and
    D1 = add-mset (Pos A) D' and
    D2 = replicate-mset (Suc m) (Neg A) + C' and
    Neg A ∉# C' and
    C = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A) + C'
    using relpowp-ground-resolutionD[OF Suc-not-Zero steps]
    by (metis diff-Suc-Suc Suc-le-mono)

  have Neg A ∉# D'
    by (metis ‹D1 = add-mset (Pos A) D'› ‹ord-res.is-strictly-maximal-lit (Pos A)
D1›
        ord-res.less-lit-simps(4) linorder-lit.is-greatest-in-mset-iff linorder-trm.eq-refl
        linorder-trm.leD remove1-mset-add-mset-If)

  obtain m' C'' where
    C = replicate-mset (Suc m') (Neg A) + C'' and
    Neg A ∉# C'' and
    eres D1 C = repeat-mset (Suc m') D' + C''
    using ‹C ≠ eres D1 C› eres-not-identD
   using ‹ord-res.is-strictly-maximal-lit (Pos A) D1› linorder-lit.Uniq-is-greatest-in-mset
    using ‹D1 = add-mset (Pos A) D'›
    by (metis Uniq-D add-mset-remove-trivial literal.inject(1))

  have m − n = Suc m'
  proof −
    have count C (Neg A) = count (repeat-mset (Suc n) D') (Neg A) +
            count (replicate-mset (m − n) (Neg A)) (Neg A) + count C' (Neg A)
      using ‹C = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A) + C'›
by simp
    also have ... = count D' (Neg A) * Suc n + count {#Neg A#} (Neg A) * (m
− n) +
            count C' (Neg A)
      by simp
    also have ... = 0 * Suc n + 1 * (m − n) + 0
      using ‹Neg A ∉# D'› ‹Neg A ∉# C'› by (simp add: count-eq-zero-iff)
    also have ... = m − n
      by presburger
    finally have count C (Neg A) = m − n .

    have count C (Neg A) = count (replicate-mset (Suc m') (Neg A)) (Neg A) +
            count C'' (Neg A)
      using ‹C = replicate-mset (Suc m') (Neg A) + C''› by simp
    also have ... = count {#Neg A#} (Neg A) * Suc m' + count C'' (Neg A)
      by simp
    also have ... = 1 * Suc m' + 0
      using ‹Neg A ∉# C''› by (simp add: count-eq-zero-iff)
    also have ... = Suc m'
      by presburger
```

finally have *count C (Neg A) = Suc m'* **.**

show *?thesis*
using ‹*count C (Neg A) = m − n*› ‹*count C (Neg A) = Suc m'*› **by** *argo*
qed

hence *C'' = repeat-mset (Suc n) D' + C'*
using ‹*C = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A) + C'*›
‹*C = replicate-mset (Suc m') (Neg A) + C''*›
by *simp*

hence *eres-D1-C-eq*: *eres D1 C = repeat-mset (Suc m' + Suc n) D' + C'*
using ‹*eres D1 C = repeat-mset (Suc m') D' + C''*› **by** *simp*

have *ord-res-Interp N (eres D1 C)* ⊨ *eres D1 C*
using *eres-C-true* **.**

moreover have *eres D1 C* ≺$_c$ *C*
using *eres-le*[*of D1 C*] ‹*C* ≠ *eres D1 C*› **by** *order*

ultimately have *ord-res-Interp N C* ⊨ *eres D1 C*
using *ord-res.entailed-clause-stays-entailed* **by** *metis*

thus *ord-res-Interp N C* ⊨ *C*
unfolding *true-cls-def*
proof (*elim bexE*)
fix *L*
assume
L-in: *L* ∈# *eres D1 C* **and**
L-true: *ord-res-Interp N C* ⊨*l L*

from *L-in* **have** *L* ∈# *D'* ∨ *L* ∈# *C'*
unfolding *eres-D1-C-eq*
using *member-mset-repeat-msetD* **by** *fastforce*
hence *L* ∈# *C*
by (*auto simp*: ‹*C = repeat-mset (Suc n) D' + replicate-mset (m − n) (Neg A) + C'*›)
with *L-true* **show** ∃ *L* ∈# *C*. *ord-res-Interp N C* ⊨*l L*
by *metis*
qed
qed

end

end
theory *ORD-RES-3*
imports
ORD-RES
Exhaustive-Factorization

Exhaustive-Resolution
begin

17 ORD-RES-3 (full resolve)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-3* **where**
factoring:
is-least-false-clause $(N \mid\cup\mid U_{er} \mid\cup\mid U_{ef})\ C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C\ L \Longrightarrow$
is-pos $L \Longrightarrow$
$U_{ef}{}' =$ *finsert* $(efac\ C)\ U_{ef} \Longrightarrow$
ord-res-3 $N\ (U_{er},\ U_{ef})\ (U_{er},\ U_{ef}{}')$ |

resolution:
is-least-false-clause $(N \mid\cup\mid U_{er} \mid\cup\mid U_{ef})\ C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C\ L \Longrightarrow$
is-neg $L \Longrightarrow$
$D \mid\in\mid N \mid\cup\mid U_{er} \mid\cup\mid U_{ef} \Longrightarrow$
$D \prec_c C \Longrightarrow$
ord-res.production $(fset\ (N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}))\ D = \{atm\text{-}of\ L\} \Longrightarrow$
$U_{er}{}' =$ *finsert* $(eres\ D\ C)\ U_{er} \Longrightarrow$
ord-res-3 $N\ (U_{er},\ U_{ef})\ (U_{er}{}',\ U_{ef})$

inductive *ord-res-3-step* **where**
ord-res-3 $N\ s\ s' \Longrightarrow$ *ord-res-3-step* $(N,\ s)\ (N,\ s')$

inductive *ord-res-3-final* **where**
ord-res-final $(N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef}) \Longrightarrow$ *ord-res-3-final* $(N,\ (U_{rr},\ U_{ef}))$

inductive *ord-res-3-load* **where**
$N \neq \{||\} \Longrightarrow$ *ord-res-3-load* $N\ (N,\ (\{||\},\ \{||\}))$

sublocale *ord-res-3-semantics*: *semantics* **where**
step = *ord-res-3-step* **and**
final = *ord-res-3-final*
proof *unfold-locales*
fix *S3* :: *'f gterm clause fset* × *'f gterm clause fset* × *'f gterm clause fset*

obtain $N\ U_{rr}\ U_{ef}$ :: *'f gterm clause fset* **where**
S3-def: $S3 = (N,\ (U_{rr},\ U_{ef}))$
by (*metis prod.exhaust*)

assume *ord-res-3-final S3*
hence $\{\#\} \mid\in\mid N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef} \vee \neg$ *ex-false-clause* $(fset\ (N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef}))$
by (*simp add*: *S3-def ord-res-3-final.simps ord-res-final-def*)
thus *finished ord-res-3-step S3*
proof (*elim disjE*)

assume $\{\#\} \mid\in\mid N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef}$
hence *is-least-false-clause* $(N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef})$ $\{\#\}$
using *is-least-false-clause-mempty* **by** *metis*
hence $\nexists\, C\, L$. *is-least-false-clause* $(N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef})$ $C \wedge$ *linorder-lit.is-maximal-in-mset* C L
by (*metis Uniq-D Uniq-is-least-false-clause bot-fset.rep-eq fBex-fempty linorder-lit.is-maximal-in-mset-iff set-mset-empty*)
hence $\nexists\, x$. *ord-res-3* N $(U_{rr},\ U_{ef})$ x
by (*auto simp*: *S3-def elim*: *ord-res-3.cases*)
thus *?thesis*
by (*simp add*: *finished-def ord-res-3-step.simps S3-def*)
next
assume $\neg$ *ex-false-clause* (*fset* $(N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef})$)
hence $\nexists\, C$. *is-least-false-clause* $(N \mid\cup\mid U_{rr} \mid\cup\mid U_{ef})$ C
unfolding *ex-false-clause-def is-least-false-clause-def*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-fset-ffilterD*(*1*) *linorder-cls.is-least-in-fset-ffilterD*(*2*))
hence $\nexists\, x$. *ord-res-3* N $(U_{rr},\ U_{ef})$ x
by (*auto simp*: *S3-def elim*: *ord-res-3.cases*)
thus *?thesis*
by (*simp add*: *finished-def ord-res-3-step.simps S3-def*)
qed
qed

sublocale *ord-res-3-language*: *language* **where**
step = *ord-res-3-step* **and**
final = *ord-res-3-final* **and**
load = *ord-res-3-load*
by *unfold-locales*

lemma *is-least-false-clause-conv-if-partial-resolution-invariant*:
assumes $\forall\, C \mid\in\mid U_{pr}.\ \exists\, D1 \mid\in\mid N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}.\ \exists\, D2 \mid\in\mid N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}.$
$(\mathit{ground\text{-}resolution}\ D1)^{++}\ D2\ C \wedge C \neq \mathit{eres}\ D1\ D2 \wedge \mathit{eres}\ D1\ D2 \mid\in\mid U_{er}$
shows *is-least-false-clause* $(N \mid\cup\mid U_{pr} \mid\cup\mid U_{er} \mid\cup\mid U_{ef})$ = *is-least-false-clause* $(N \mid\cup\mid U_{er} \mid\cup\mid U_{ef})$
proof −
have *is-least-false-clause* $(N \mid\cup\mid U_{pr} \mid\cup\mid U_{er} \mid\cup\mid U_{ef})$ = *is-least-false-clause* $(N \mid\cup\mid U_{er} \mid\cup\mid U_{ef} \mid\cup\mid U_{pr})$
by (*simp add*: *sup-commute sup-left-commute*)
also have ... = *is-least-false-clause* $(N \mid\cup\mid U_{er} \mid\cup\mid U_{ef})$
proof (*rule is-least-false-clause-funion-cancel-right*)
show $\forall\, C \mid\in\mid U_{pr}.\ \forall\, U.$ *ord-res.production* U $C = \{\}$
proof (*intro ballI*)
fix C
assume $C \mid\in\mid U_{pr}$
hence $\exists\, D \mid\in\mid N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}.\ (\exists\, C'.$ *ground-resolution* D C $C')$
using *assms* **by** (*metis eres-eq-after-tranclp-ground-resolution resolvable-if-neq-eres*)
hence $\nexists\, L$. *is-pos* $L \wedge$ *ord-res.is-strictly-maximal-lit* L C
using *nex-strictly-maximal-pos-lit-if-resolvable* **by** *metis*

```
    thus ∀ U. ord-res.production U C = {}
      using unproductive-if-nex-strictly-maximal-pos-lit by metis
  qed
 next
  show ∀ C |∈| U_pr. ∃ D |∈| N |∪| U_er |∪| U_ef. D ≺_c C ∧ {D} ⟙e {C}
  proof (intro ballI)
    fix C
    assume C |∈| U_pr
    then obtain D1 D2 where
      D1|∈|N |∪| U_er |∪| U_ef and
      D2|∈|N |∪| U_er |∪| U_ef and
      (ground-resolution D1)^++ D2 C and
      C ≠ eres D1 D2 and
      eres D1 D2 |∈| U_er
      using assms by metis

    have eres D1 D2 = eres D1 C
    using ‹(ground-resolution D1)^++ D2 C› eres-eq-after-tranclp-ground-resolution
by metis

    show ∃ D |∈| N |∪| U_er |∪| U_ef. D ≺_c C ∧ {D} ⟙e {C}
    proof (intro bexI conjI)
      have eres D1 C ⪯_c C
        using eres-le .
      thus eres D1 D2 ≺_c C
        using ‹C ≠ eres D1 D2› ‹eres D1 D2 = eres D1 C› by order
    next
      show {eres D1 D2} ⟙e {C}
        using ‹(ground-resolution D1)^++ D2 C› eres-entails-resolvent by metis
    next
      show eres D1 D2 |∈| N |∪| U_er |∪| U_ef
        using ‹eres D1 D2 |∈| U_er› by simp
    qed
  qed
 qed
 finally show ?thesis .
qed

lemma right-unique-ord-res-3: right-unique (ord-res-3 N)
proof (rule right-uniqueI)
 fix s s' s'' :: 'f gterm clause fset × 'f gterm clause fset
 assume step1: ord-res-3 N s s' and step2: ord-res-3 N s s''
 show s' = s''
  using step1
 proof (cases N s s' rule: ord-res-3.cases)
  case hyps1: (factoring U_rr1 U_ef1 C1 L1 U_ef1')
  show ?thesis
   using step2
  proof (cases N s s'' rule: ord-res-3.cases)
```

```
      case (factoring U_rr2 U_ef2 C2 L2 U_ef2')
      with hyps1 show ?thesis
        by (metis Uniq-D Uniq-is-least-false-clause prod.inject)
    next
      case (resolution U_rr2 U_ef2 C2 L2 D2 U_rr2')
      with hyps1 have False
      by (metis Pair-inject Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
          the1-equality')
      thus ?thesis ..
    qed
  next
    case hyps1: (resolution U_rr1 U_ef1 C1 L1 D1 U_rr1')
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-3.cases)
      case (factoring U_rr2 U_ef2 C2 L2 U_ef2')
      with hyps1 have False
          by (metis Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
prod.inject the1-equality')
      thus ?thesis ..
    next
      case hyps2: (resolution U_rr2 U_ef2 C2 L2 D2 U_rr2')

      have *: U_rr1 = U_rr2 U_ef1 = U_ef2
        using hyps1 hyps2 by  simp-all

      have **: C1 = C2
        using hyps1 hyps2
        unfolding *
        by (metis Uniq-is-least-false-clause the1-equality')

      have ***: L1 = L2
        using hyps1 hyps2
        unfolding * **
        by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')

      have ****: D1 = D2
        using hyps1 hyps2
        unfolding * ** ***
        by (metis linorder-cls.less-irrefl linorder-cls.linorder-cases
            ord-res.less-trm-iff-less-cls-if-mem-production singletonI)

      show ?thesis
        using hyps1 hyps2
        unfolding * ** *** ****
        by argo
    qed
  qed
qed
```

lemma *right-unique-ord-res-3-step*: *right-unique ord-res-3-step*
proof (*rule right-uniqueI*)
  fix *x y z*
  show *ord-res-3-step x y* $\Longrightarrow$ *ord-res-3-step x z* $\Longrightarrow$ *y* = *z*
    apply (*cases x*; *cases y*; *cases z*)
    apply (*simp add*: *ord-res-3-step.simps*)
    using *right-unique-ord-res-3*[*THEN right-uniqueD*]
    by *blast*
qed

lemma *ex-ord-res-3-if-not-final*:
  assumes ¬ *ord-res-3-final S*
  shows ∃ *S′*. *ord-res-3-step S S′*
proof −
  from *assms* **obtain** *N* U_r U_{ef} **where**
    S-def: *S* = (*N*, (*U*$_r$, *U*$_{ef}$)) **and**
    {#} |∉| *N* |∪| *U*$_r$ |∪| *U*$_{ef}$ **and**
    ex-false-clause (*fset* (*N* |∪| *U*$_r$ |∪| *U*$_{ef}$))
    by (*metis ord-res-3-final.intros ord-res-final-def surj-pair*)

  obtain *C* **where** *C-least-false*: *is-least-false-clause* (*N* |∪| *U*$_r$ |∪| *U*$_{ef}$) *C*
   using ‹*ex-false-clause* (*fset* (*N* |∪| *U*$_r$ |∪| *U*$_{ef}$))› *obtains-least-false-clause-if-ex-false-clause*
    by *metis*

  hence *C* ≠ {#}
    using ‹{#} |∉| *N* |∪| *U*$_r$ |∪| *U*$_{ef}$›
    unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
    by *metis*

  then obtain *L* **where** *C-max*: *linorder-lit.is-maximal-in-mset C L*
    using *linorder-lit.ex-maximal-in-mset* **by** *metis*

  show *?thesis*
  proof (*cases L*)
    case (*Pos A*)
    thus *?thesis*
      using *ord-res-3.factoring*[*OF C-least-false C-max*] *S-def is-pos-def*
      by (*metis ord-res-3-step.intros*)
  next
    case (*Neg A*)
    then obtain *D* **where**
      D |∈| *N* |∪| *U*$_r$ |∪| *U*$_{ef}$ **and**
      D ≺$_c$ *C* **and**
      ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**
      ord-res.production (*fset* (*N* |∪| *U*$_r$ |∪| *U*$_{ef}$)) *D* = {*A*}
      using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
      using *C-least-false C-max* **by** *metis*

moreover then obtain *DC* **where**
full-run (*ground-resolution D*) *C DC*
using *ex-ground-resolutionI C-max Neg*
using *ex1-eres-eq-full-run-ground-resolution* **by** *blast*

ultimately show *?thesis*
using *ord-res-3.resolution*[*OF C-least-false C-max*]
by (*metis Neg S-def literal.disc*(*2*) *literal.sel*(*2*) *ord-res-3-step.intros*)
qed
qed

corollary *ord-res-3-step-safe*: *ord-res-3-final S* ∨ (∃ *S′*. *ord-res-3-step S S′*)
using *ex-ord-res-3-if-not-final* **by** *metis*

end

end
theory *Implicit-Exhaustive-Factorization*
imports
Exhaustive-Factorization
Exhaustive-Resolution
begin

18 Function for implicit full factorization

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

definition *iefac* **where**
iefac $\mathcal{F}$ *C* = (*if C* |∈| $\mathcal{F}$ *then efac C else C*)

lemma *iefac-mempty*[*simp*]:
fixes $\mathcal{F}$ :: *'f gclause fset*
shows *iefac* $\mathcal{F}$ {#} = {#}
by (*metis efac-mempty iefac-def*)

lemma *fset-mset-iefac*[*simp*]:
fixes $\mathcal{F}$ :: *'f gclause fset* **and** *C* :: *'f gclause*
shows *fset-mset* (*iefac* $\mathcal{F}$ *C*) = *fset-mset C*
proof (*cases C* |∈| $\mathcal{F}$)
case *True*
hence *iefac* $\mathcal{F}$ *C* = *efac C*
unfolding *iefac-def* **by** *simp*
thus *?thesis*
by *auto*
next
case *False*
hence *iefac* $\mathcal{F}$ *C* = *C*
unfolding *iefac-def* **by** *simp*
thus *?thesis* **by** *simp*

qed

lemma *atms-of-cls-iefac*[*simp*]:
fixes $\mathcal{F}$ :: *'f gclause fset* **and** C :: *'f gclause*
shows *atms-of-cls* (*iefac* $\mathcal{F}$ C) = *atms-of-cls* C
by (*simp add*: *atms-of-cls-def*)

lemma *iefac-le*:
fixes $\mathcal{F}$ :: *'f gclause fset* **and** C :: *'f gclause*
shows *iefac* $\mathcal{F}$ C $\preceq_c$ C
by (*metis efac-subset iefac-def reflclp-iff subset-implies-reflclp-multp*)

lemma *true-cls-iefac-iff*[*simp*]:
fixes $\mathcal{I}$ :: *'f gterm set* **and** $\mathcal{F}$ :: *'f gclause fset* **and** C :: *'f gclause*
shows $\mathcal{I} \models$ *iefac* $\mathcal{F}$ $C \longleftrightarrow \mathcal{I} \models C$
by (*metis iefac-def true-cls-efac-iff*)

lemma *funion-funion-eq-funion-funion-fimage-iefac-if*:
assumes U_{ef}-*eq*: U_{ef} = *iefac* $\mathcal{F}$ |‘| {|C |∈| N |∪| U_{er}. *iefac* $\mathcal{F}$ $C \neq C$|}
shows N |∪| U_{er} |∪| U_{ef} = N |∪| U_{er} |∪| (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
proof (*intro fsubset-antisym fsubsetI*)
fix C :: *'f gterm clause*
assume C |∈| N |∪| U_{er} |∪| U_{ef}
hence C |∈| N |∪| U_{er} ∨ C |∈| U_{ef}
by *simp*
thus C |∈| N |∪| U_{er} |∪| (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
proof (*elim disjE*)
assume C |∈| N |∪| U_{er}
thus *?thesis*
by *simp*
next
assume C |∈| U_{ef}
hence C |∈| *iefac* $\mathcal{F}$ |‘| {|C |∈| N |∪| U_{er}. *iefac* $\mathcal{F}$ $C \neq C$|}
using U_{ef}-*eq* **by** *argo*
then obtain C_0 :: *'f gterm clause* **where**
C_0 |∈| N |∪| U_{er} **and** *iefac* $\mathcal{F}$ $C_0 \neq C_0$ **and** C = *iefac* $\mathcal{F}$ C_0
by *auto*
thus *?thesis*
by *simp*
qed
next
fix C :: *'f gterm clause*
assume C |∈| N |∪| U_{er} |∪| (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
hence C |∈| N |∪| U_{er} ∨ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
by *simp*
thus C |∈| N |∪| U_{er} |∪| U_{ef}
proof (*elim disjE*)
assume C |∈| N |∪| U_{er}

```
    thus ?thesis
      by simp
  next
    assume C |∈| iefac F |‘| (N |∪| U_er)
    then obtain C_0 :: 'f gterm clause where
      C_0 |∈| N |∪| U_er and C = iefac F C_0
      by auto

    show ?thesis
    proof (cases iefac F C_0 = C_0)
      case True
      hence C = C_0
        using ‹C = iefac F C_0› by argo
      thus ?thesis
        using ‹C_0 |∈| N |∪| U_er› by simp
    next
      case False
      hence C |∈| U_ef
        unfolding U_ef-eq
        using ‹C_0 |∈| N |∪| U_er› ‹C = iefac F C_0› by simp
      thus ?thesis
        by simp
    qed
  qed
qed


lemma clauses-for-iefac-are-unproductive:
  ∀ C |∈| N |−| iefac F |‘| N. ∀ U. ord-res.production U C = {}
proof (intro ballI allI)
  fix C U
  assume C |∈| N |−| iefac F |‘| N
  hence C |∈| N and C |∉| iefac F |‘| N
    by simp-all
  hence iefac F C ≠ C
    by (metis fimage-iff)
  hence efac C ≠ C
    by (metis iefac-def)
  hence ∄ L. is-pos L ∧ ord-res.is-strictly-maximal-lit L C
    using nex-strictly-maximal-pos-lit-if-neq-efac by metis
  thus ord-res.production U C = {}
    using unproductive-if-nex-strictly-maximal-pos-lit by metis
qed

lemma clauses-for-iefac-have-smaller-entailing-clause:
  ∀ C |∈| N |−| iefac F |‘| N. ∃ D |∈| iefac F |‘| N. D ≺_c C ∧ {D} ⊫e {C}
proof (intro ballI allI)
  fix C
  assume C |∈| N |−| iefac F |‘| N
```

```
  hence C |∈| N and C |∉| iefac F |‘| N
    by simp-all
  hence iefac F C ≠ C
    by (metis fimage-iff)
  hence efac C ≠ C
    by (metis iefac-def)

  show ∃ D |∈| iefac F |‘| N. D ≺c C ∧ {D} ⫦e {C}
  proof (intro bexI conjI)
    show efac C ≺c C and {efac C} ⫦e {C}
      using efac-properties-if-not-ident[OF ‹efac C ≠ C›] by simp-all
  next
    show efac C |∈| iefac F |‘| N
      using ‹C |∈| N› ‹iefac F C ≠ C› by (metis fimageI iefac-def)
  qed
qed

lemma is-least-false-clause-with-iefac-conv:
  is-least-false-clause (N |∪| U_er |∪| iefac F |‘| (N |∪| U_er)) =
    is-least-false-clause (iefac F |‘| (N |∪| U_er))
 using is-least-false-clause-funion-cancel-right-strong[OF clauses-for-iefac-are-unproductive
    clauses-for-iefac-have-smaller-entailing-clause]
  by (simp add: sup-commute)

lemma MAGIC4:
  fixes N F F' U_er U_er'
  defines
    N1 ≡ iefac F |‘| (N |∪| U_er) and
    N2 ≡ iefac F' |‘| (N |∪| U_er')
  assumes
    subsets-agree: {|x |∈| N1. x ≺c C|} = {|x |∈| N2. x ≺c C|} and
    is-least-false-clause N1 D and
    is-least-false-clause N2 E and
    C ≺c D
  shows C ⪯c E
proof −
  have ¬ E ≺c C
  proof (rule notI)
    assume E ≺c C

    have is-least-false-clause N2 E ⟷ is-least-false-clause {|x |∈| N2. x ⪯c E|}
E
    proof (rule is-least-false-clause-swap-swap-compatible-fsets)
      show {|x |∈| N2. (≺c)== x E|} = {|x |∈| {|x |∈| N2. (≺c)== x E|}. (≺c)==
x E|}
        using ‹E ≺c C› by force
    qed

    also have ... ⟷ is-least-false-clause {|x |∈| N1. x ⪯c E|} E
```

proof (*rule is-least-false-clause-swap-swap-compatible-fsets*)
show $\{|x \mid\in| \{|x \mid\in| N2.\ (\prec_c)^{==}\ x\ E|\}.\ (\prec_c)^{==}\ x\ E|\} =$
$\{|x \mid\in| \{|x \mid\in| N1.\ (\prec_c)^{==}\ x\ E|\}.\ (\prec_c)^{==}\ x\ E|\}$
using *subsets-agree* ‹$E \prec_c C$› **by** *fastforce*
qed

also have $\ldots \longleftrightarrow$ *is-least-false-clause N1 E*
proof (*rule is-least-false-clause-swap-swap-compatible-fsets*)
show $\{|x \mid\in| \{|x \mid\in| N1.\ (\prec_c)^{==}\ x\ E|\}.\ (\prec_c)^{==}\ x\ E|\} = \{|x \mid\in| N1.\ (\prec_c)^{==}\ x\ E|\}$
using ‹$E \prec_c C$› **by** *fastforce*
qed

finally have *is-least-false-clause N1 E*
using ‹*is-least-false-clause N2 E*› **by** *argo*

hence $D = E$
using ‹*is-least-false-clause N1 D*›
by (*metis Uniq-is-least-false-clause the1-equality'*)

thus *False*
using ‹$E \prec_c C$› ‹$C \prec_c D$› **by** *order*
qed
thus $C \preceq_c E$
by *order*
qed

lemma *atms-of-clss-fimage-iefac*[*simp*]:
atms-of-clss (*iefac* $\mathcal{F}$ |‘| N) = *atms-of-clss* N
proof −
have *atms-of-clss* (*iefac* $\mathcal{F}$ |‘| N) = *ffUnion* (*atms-of-cls* |‘| *iefac* $\mathcal{F}$ |‘| N)
unfolding *atms-of-clss-def* **..**

also have $\ldots$ = *ffUnion* ((*atms-of-cls* ∘ *iefac* $\mathcal{F}$) |‘| N)
by *simp*

also have $\ldots$ = *ffUnion* (*atms-of-cls* |‘| N)
unfolding *comp-def atms-of-cls-iefac* **..**

also have $\ldots$ = *atms-of-clss* N
unfolding *atms-of-clss-def* **..**

finally show *?thesis* **.**
qed

lemma *atm-of-in-atms-of-clssI*:
assumes *L-in*: $L \in\#\ C$ **and** *C-in*: C |∈| *iefac* $\mathcal{F}$ |‘| N
shows *atm-of* L |∈| *atms-of-clss* N
proof −

have *atm-of L* $|\in|$ *atms-of-cls C*
unfolding *atms-of-cls-def*
using *L-in* **by** *simp*

hence *atm-of L* $|\in|$ *atms-of-clss* (*iefac* $\mathcal{F}$ $|`|$ *N*)
unfolding *atms-of-clss-def*
using *C-in* **by** (*metis fmember-ffUnion-iff*)

thus *atm-of L* $|\in|$ *atms-of-clss N*
by *simp*
qed

lemma *clause-almost-almost-definedI*:
fixes Γ *D K*
assumes
D-in: *D* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) **and**
D-max-lit: *ord-res.is-maximal-lit K D* **and**
no-undef-atm: ¬ (∃ *A*$|\in|$*atms-of-clss* (*N* $|\cup|$ U_{er}). *A* $\prec_t$ *atm-of K* ∧ *A* $|\notin|$ *trail-atms* Γ)
shows *trail-defined-cls* Γ {#*L* ∈# *D*. *L* ≠ *K* ∧ *L* ≠ − *K*#}
unfolding *trail-defined-cls-def*
proof (*intro ballI*)
have *K* ∈# *D* **and** *lit-in-D-le-K*: ⋀*L*. *L* ∈# *D* ⟹ *L* $\preceq_l$ *K*
using *D-max-lit*
unfolding *atomize-imp atomize-all atomize-conj linorder-lit.is-maximal-in-mset-iff*
using *linorder-lit.leI* **by** *blast*

fix *L* :: *'f gterm literal*
assume *L* ∈# {#*L* ∈# *D*. *L* ≠ *K* ∧ *L* ≠ − *K*#}

hence *L* ∈# *D* **and** *L* ≠ *K* **and** *L* ≠ − *K*
by *simp-all*

have *atm-of L* $|\in|$ *atms-of-clss* (*N* $|\cup|$ U_{er})
using ‹*L* ∈# *D*› *D-in* **by** (*metis atm-of-in-atms-of-clssI*)

hence *atm-of L* $\prec_t$ *atm-of K* ⟹ *atm-of L* $|\in|$ *trail-atms* Γ
using *no-undef-atm* **by** *metis*

moreover have *atm-of L* $\prec_t$ *atm-of K*
using *lit-in-D-le-K*[*OF* ‹*L* ∈# *D*›] ‹*L* ≠ *K*› ‹*L* ≠ − *K*›
by (*cases L*; *cases K*) *simp-all*

ultimately show *trail-defined-lit* Γ *L*
using *trail-defined-lit-iff-trail-defined-atm* **by** *auto*
qed

lemma *clause-almost-definedI*:
fixes Γ *D K*

assumes
D-in: $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
D-max-lit: *ord-res.is-maximal-lit K D* **and**
no-undef-atm: $\neg (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma)$ **and**
K-defined: *trail-defined-lit* Γ K
shows *trail-defined-cls* Γ $\{\#Ka \in\# D.\ Ka \neq K\#\}$
using *clause-almost-almost-definedI*[*OF D-in D-max-lit no-undef-atm*]
using *K-defined*
unfolding *trail-defined-cls-def trail-defined-lit-def*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq set-mset-filter uminus-lit-swap*)

lemma *eres-not-in-known-clauses-if-trail-false-cls*:
fixes
$\mathcal{F}$ :: *'f gclause fset* **and**
Γ :: (*'f gliteral* $\times$ *'f gclause option*) *list*
assumes
Γ-*consistent*: *trail-consistent* Γ **and**
clauses-lt-E-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$ *trail-true-cls* Γ C **and**
eres D E $\prec_c E$ **and**
trail-false-cls Γ (*eres D E*)
shows *eres D E* $\mid\notin\mid N \mid\cup\mid U_{er}$
proof (*rule notI*)
have *iefac* $\mathcal{F}$ (*eres D E*) $\preceq_c$ *eres D E*
using *iefac-le* **by** *metis*
hence *iefac* $\mathcal{F}$ (*eres D E*) $\prec_c E$
using ‹*eres D E* $\prec_c E$› **by** *order*

moreover assume *eres D E* $\mid\in\mid N \mid\cup\mid U_{er}$

ultimately have *trail-true-cls* Γ (*iefac* $\mathcal{F}$ (*eres D E*))
using *clauses-lt-E-true*[*rule-format*, *of iefac* $\mathcal{F}$ (*eres D E*)]
by (*simp add*: *iefac-def linorder-lit.is-maximal-in-mset-iff*)

hence *trail-true-cls* Γ (*eres D E*)
unfolding *trail-true-cls-def*
by (*metis fset-fset-mset fset-mset-iefac*)

thus *False*
using Γ-*consistent* ‹*trail-false-cls* Γ (*eres D E*)›
by (*metis not-trail-true-cls-and-trail-false-cls*)
qed

lemma *no-undefined-atom-le-max-lit-of-false-clause*:
assumes
Γ-*lower-set*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$)
and
D-in: $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**

D-false: *trail-false-cls* Γ *D* **and**
D-max-lit: *linorder-lit.is-maximal-in-mset D L*
shows ¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* ≼$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ)
proof −
have *trail-false-lit* Γ *L*
using *D-false D-max-lit*
unfolding *trail-false-cls-def linorder-lit.is-maximal-in-mset-iff* **by** *simp*

hence *trail-defined-lit* Γ *L*
unfolding *trail-false-lit-def trail-defined-lit-def* **by** *argo*

hence *atm-of L* |∈| *trail-atms* Γ
unfolding *trail-defined-lit-iff-trail-defined-atm* **.**

thus *?thesis*
using Γ*-lower-set*
using *linorder-trm.not-in-lower-setI* **by** *blast*
qed

lemma *trail-defined-if-no-undef-atom-le-max-lit*:
assumes
C-in: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C K* **and**
no-undef-atom-le-K:
¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* ≼$_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ)
shows *trail-defined-cls* Γ *C*
proof −

have ⋀*x*. *x* ∈# *C* ⟹ *atm-of x* ≼$_t$ *atm-of K*
using *C-in C-max-lit*
unfolding *linorder-lit.is-maximal-in-mset-iff*
by (*metis linorder-trm.le-cases linorder-trm.le-less-linear literal.exhaust-sel*
ord-res.less-lit-simps(*1*) *ord-res.less-lit-simps*(*2*) *ord-res.less-lit-simps*(*3*)
ord-res.less-lit-simps(*4*))

hence ⋀*x*. *x* ∈# *C* ⟹ *trail-defined-lit* Γ *x*
using *C-in no-undef-atom-le-K*
by (*meson atm-of-in-atms-of-clssI trail-defined-lit-iff-trail-defined-atm*)

thus *trail-defined-cls* Γ *C*
unfolding *trail-defined-cls-def*
by *metis*
qed

lemma *no-undef-atom-le-max-lit-if-lt-false-clause*:
assumes
Γ*-lower-set*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
and
D-in: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**

D-false: *trail-false-cls* Γ *D* **and**
D-max-lit: *linorder-lit.is-maximal-in-mset D L* **and**
C-in: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C K* **and**
C-lt: *C* $\prec_c$ *D*
shows ¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* $\preceq_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ)
proof −
have *K* $\preceq_l$ *L*
using *C-lt C-max-lit D-max-lit*
using *linorder-cls.less-imp-not-less linorder-lit.multp-if-maximal-less-that-maximal linorder-lit.nle-le* **by** *blast*

hence *atm-of K* $\preceq_t$ *atm-of L*
by (*cases K*; *cases L*) *simp-all*

thus ¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* $\preceq_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ)
using *no-undefined-atom-le-max-lit-of-false-clause*[*OF assms*(*1*,*2*,*3*,*4*)]
by *fastforce*
qed

lemma *bex-trail-false-cls-simp*:
fixes $\mathcal{F}$ *N* Γ
shows *fBex* (*iefac* $\mathcal{F}$ |‘| *N*) (*trail-false-cls* Γ) ⟷ *fBex N* (*trail-false-cls* Γ)
proof (*rule iffI* ; *elim bexE*)
fix *C* :: *'f gclause*
assume *C-in*: *C* |∈| *iefac* $\mathcal{F}$ |‘| *N* **and** *C-false*: *trail-false-cls* Γ *C*
thus *fBex N* (*trail-false-cls* Γ)
by (*smt* (*verit*, *ccfv-SIG*) *fimage-iff iefac-def set-mset-efac trail-false-cls-def*)
next
fix *C* :: *'f gclause*
assume *C* |∈| *N* **and** *trail-false-cls* Γ *C*
thus *fBex* (*iefac* $\mathcal{F}$ |‘| *N*) (*trail-false-cls* Γ)
by (*metis fimageI iefac-def set-mset-efac trail-false-cls-def*)
qed

end

end
theory *ORD-RES-4*
imports
ORD-RES
Implicit-Exhaustive-Factorization
Exhaustive-Resolution
begin

19 ORD-RES-4 (implicit factorization)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-4* **where**
factoring:
$NN = iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}) \Longrightarrow$
is-least-false-clause NN C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-pos L $\Longrightarrow$
$\mathcal{F}' = finsert\ C\ \mathcal{F} \Longrightarrow$
ord-res-4 $N\ (U_{er}, \mathcal{F})\ (U_{er}, \mathcal{F}')\ |$

resolution:
$NN = iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}) \Longrightarrow$
is-least-false-clause NN C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-neg L $\Longrightarrow$
$D\ |\in|\ NN \Longrightarrow$
$D \prec_c C \Longrightarrow$
ord-res.production $(fset\ NN)\ D = \{atm\text{-}of\ L\} \Longrightarrow$
$U_{er}' = finsert\ (eres\ D\ C)\ U_{er} \Longrightarrow$
ord-res-4 $N\ (U_{er}, \mathcal{F})\ (U_{er}', \mathcal{F})$

inductive *ord-res-4-step* **where**
ord-res-4 $N\ s\ s' \Longrightarrow$ *ord-res-4-step* $(N, s)\ (N, s')$

inductive *ord-res-4-final* **where**
ord-res-final $(iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})) \Longrightarrow$ *ord-res-4-final* $(N, U_{er}, \mathcal{F})$

sublocale *ord-res-4-semantics*: *semantics* **where**
step = *ord-res-4-step* **and**
final = *ord-res-4-final*
proof *unfold-locales*
fix *S4* :: *'f gterm clause fset* × *'f gterm clause fset* × *'f gterm clause fset*

obtain $N\ U_{er}\ \mathcal{F}$ :: *'f gterm clause fset* **where**
S4-def: $S4 = (N, (U_{er}, \mathcal{F}))$
by (*metis prod.exhaust*)

assume *ord-res-4-final S4*
hence $\{\#\}\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}) \vee \neg$ *ex-false-clause* (*fset* (*iefac* $\mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$))
by (*simp add*: *S4-def ord-res-4-final.simps ord-res-final-def*)
thus *finished ord-res-4-step S4*
proof (*elim disjE*)
assume $\{\#\}\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
hence *is-least-false-clause* $(iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\ \{\#\}$
using *is-least-false-clause-mempty* **by** *metis*
hence $\nexists C\ L.$ *is-least-false-clause* $(iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\ C \wedge$ *linorder-lit.is-maximal-in-mset C L*
by (*metis Uniq-D Uniq-is-least-false-clause bot-fset.rep-eq fBex-fempty*
linorder-lit.is-maximal-in-mset-iff set-mset-empty)

```
    hence ∄x. ord-res-4 N (U_er, F) x
      by (auto simp: S4-def elim: ord-res-4.cases)
    thus ?thesis
      by (simp add: S4-def finished-def ord-res-4-step.simps)
  next
    assume ¬ ex-false-clause (fset (iefac F |'| (N |∪| U_er)))
    hence ∄C. is-least-false-clause (iefac F |'| (N |∪| U_er)) C
      unfolding ex-false-clause-def is-least-false-clause-def
      by (metis (no-types, lifting) linorder-cls.is-least-in-fset-ffilterD(1)
          linorder-cls.is-least-in-fset-ffilterD(2))
    hence ∄x. ord-res-4 N (U_er, F) x
      by (auto simp: S4-def elim: ord-res-4.cases)
    thus ?thesis
      by (simp add: S4-def finished-def ord-res-4-step.simps)
  qed
qed

lemma right-unique-ord-res-4: right-unique (ord-res-4 N)
proof (rule right-uniqueI)
  fix s s' s'' :: 'f gterm clause fset × 'f gterm clause fset
  assume step1: ord-res-4 N s s' and step2: ord-res-4 N s s''
  show s' = s''
    using step1
  proof (cases N s s' rule: ord-res-4.cases)
    case hyps1: (factoring NN1 F1 U_er1 C1 L1 F1')
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-4.cases)
      case (factoring NN2 F2 U_er2 C2 L2 F2')
      with hyps1 show ?thesis
        by (metis Uniq-D Uniq-is-least-false-clause prod.inject)
    next
      case (resolution NN2 F2 U_er2 C2 L2 D2 U_er2')
      with hyps1 have False
      by (metis Pair-inject Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
          the1-equality')
      thus ?thesis ..
    qed
  next
    case hyps1: (resolution NN1 F1 U_er1 C1 L1 D1 U_er1')
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-4.cases)
      case (factoring NN F U_er C L F')
      with hyps1 have False
      by (metis Pair-inject Uniq-is-least-false-clause linorder-lit.Uniq-is-maximal-in-mset
          the1-equality')
      thus ?thesis ..
    next
```

```
    case (resolution NN F U_er C L D U_er')
    with hyps1 show ?thesis
      by (metis (mono-tags, lifting) Uniq-D Uniq-is-least-false-clause
        ord-res.less-trm-iff-less-cls-if-mem-production insertI1 linorder-cls.neq-iff
        linorder-lit.Uniq-is-maximal-in-mset prod.inject)
  qed
 qed
qed

lemma right-unique-ord-res-4-step: right-unique ord-res-4-step
proof (rule right-uniqueI)
 fix x y z
 show ord-res-4-step x y ⟹ ord-res-4-step x z ⟹ y = z
  using right-unique-ord-res-4[THEN right-uniqueD]
  by (elim ord-res-4-step.cases) (metis prod.inject)
qed

lemma ex-ord-res-4-if-not-final:
 assumes ¬ ord-res-4-final S
 shows ∃ S'. ord-res-4-step S S'
proof −
 from assms obtain N U_er F where
  S-def: S = (N, (U_er, F)) and
  {#} |∉| iefac F |`| (N |∪| U_er) and
  ex-false-clause (fset (iefac F |`| (N |∪| U_er)))
  by (metis ord-res-4-final.intros ord-res-final-def surj-pair)

 obtain C where C-least-false: is-least-false-clause (iefac F |`| (N |∪| U_er)) C
  using ‹ex-false-clause (fset (iefac F |`| (N |∪| U_er)))›
    obtains-least-false-clause-if-ex-false-clause
  by metis

 hence C ≠ {#}
  using ‹{#} |∉| iefac F |`| (N |∪| U_er)›
  unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
  by metis

 then obtain L where C-max: linorder-lit.is-maximal-in-mset C L
  using linorder-lit.ex-maximal-in-mset by metis

 show ?thesis
 proof (cases L)
  case (Pos A)
  thus ?thesis
   using ord-res-4.factoring[OF refl C-least-false C-max] S-def is-pos-def
   by (metis ord-res-4-step.intros)
 next
  case (Neg A)
  then obtain D where
```

D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
$D \prec_c C$ **and**
ord-res.is-strictly-maximal-lit (*Pos* A) D **and**
ord-res.production (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $D = \{A\}$
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
using *C-least-false C-max* **by** *metis*

thus *?thesis*
using *ord-res-4.resolution*[*OF refl C-least-false C-max*]
by (*metis Neg S-def literal.disc*(2) *literal.sel*(2) *ord-res-4-step.simps*)
qed
qed

corollary *ord-res-4-step-safe*: *ord-res-4-final* $S \vee (\exists S'.$ *ord-res-4-step* S $S')$
using *ex-ord-res-4-if-not-final* **by** *metis*

end

end
theory *ORD-RES-5*
imports
Background
Implicit-Exhaustive-Factorization
Exhaustive-Resolution
begin

20 ORD-RES-5 (explicit model construction)

type-synonym *'f ord-res-5* = *'f gclause fset* × *'f gclause fset* × *'f gclause fset* ×
(*'f gterm* ⇒ *'f gclause option*) × *'f gclause option*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-5* **where**
skip:
(*dom* $\mathcal{M}$) $\models C \Longrightarrow$
$\mathcal{C}' =$ *The-optional* (*linorder-cls.is-least-in-fset* {|D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}).
$C \prec_c D$|}) $\Longrightarrow$
ord-res-5 N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* C) (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}'$) |

production:
¬ (*dom* $\mathcal{M}$) $\models C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
is-pos $L \Longrightarrow$
linorder-lit.is-greatest-in-mset C $L \Longrightarrow$
$\mathcal{M}' = \mathcal{M}$(*atm-of* $L :=$ *Some* C) $\Longrightarrow$
$\mathcal{C}' =$ *The-optional* (*linorder-cls.is-least-in-fset* {|D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}).
$C \prec_c D$|}) $\Longrightarrow$
ord-res-5 N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* C) (U_{er}, $\mathcal{F}$, $\mathcal{M}'$, $\mathcal{C}'$) |

factoring:
$\neg$ (*dom* $\mathcal{M}$) $\models$ C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-pos L $\Longrightarrow$
$\neg$ *linorder-lit.is-greatest-in-mset* C L $\Longrightarrow$
$\mathcal{F}'$ = *finsert* C $\mathcal{F}$ $\Longrightarrow$
$\mathcal{M}'$ = (λ-. *None*) $\Longrightarrow$
$\mathcal{C}'$ = *The-optional* (*linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}'$ |‘| (N |$\cup$| U_{er}))) $\Longrightarrow$
ord-res-5 N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* C) (U_{er}, $\mathcal{F}'$, $\mathcal{M}'$, $\mathcal{C}'$) |

resolution:
$\neg$ (*dom* $\mathcal{M}$) $\models$ C $\Longrightarrow$
linorder-lit.is-maximal-in-mset C L $\Longrightarrow$
is-neg L $\Longrightarrow$
$\mathcal{M}$ (*atm-of* L) = *Some* D $\Longrightarrow$
U_{er}' = *finsert* (*eres* D C) U_{er} $\Longrightarrow$
$\mathcal{M}'$ = (λ-. *None*) $\Longrightarrow$
$\mathcal{C}'$ = *The-optional* (*linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (N |$\cup$| U_{er}'))) $\Longrightarrow$
ord-res-5 N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* C) (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, $\mathcal{C}'$)

inductive *ord-res-5-step* :: *'f ord-res-5* $\Rightarrow$ *'f ord-res-5* $\Rightarrow$ *bool* **where**
ord-res-5 N s s' $\Longrightarrow$ *ord-res-5-step* (N, s) (N, s')

lemma *tranclp-ord-res-5-step-if-tranclp-ord-res-5*:
(*ord-res-5* N)$^{++}$ s s' $\Longrightarrow$ *ord-res-5-step*$^{++}$ (N, s) (N, s')
by (*induction* s' *rule*: *tranclp-induct*)
(*auto intro*: *ord-res-5-step.intros tranclp.intros*)

inductive *ord-res-5-final* :: *'f ord-res-5* $\Rightarrow$ *bool* **where**
model-found:
ord-res-5-final (N, U_{er}, $\mathcal{F}$, $\mathcal{M}$, *None*) |

contradiction-found:
ord-res-5-final (N, U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* {#})

sublocale *ord-res-5-semantics*: *semantics* **where**
step = *ord-res-5-step* **and**
final = *ord-res-5-final*
proof *unfold-locales*
fix *S5* :: *'f gclause fset* $\times$ *'f gclause fset* $\times$ *'f gclause fset* $\times$
(*'f gterm* $\Rightarrow$ *'f gclause option*) $\times$ *'f gclause option*

obtain
N U_{er} $\mathcal{F}$ :: *'f gterm clause fset* **and**
$\mathcal{M}$ :: *'f gterm* $\Rightarrow$ *'f gclause option* **and**
$\mathcal{C}$ :: *'f gclause option* **where**
S5-def: *S5* = (N, (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$))
by (*metis prod.exhaust*)

```
  assume ord-res-5-final S5
  hence 𝒞 = None ∨ 𝒞 = Some {#}
    by (simp add: S5-def ord-res-5-final.simps)
  hence ∄x. ord-res-5 N (U_er, ℱ, ℳ, 𝒞) x
    by (auto simp: linorder-lit.is-maximal-in-mset-iff elim: ord-res-5.cases)
  thus finished ord-res-5-step S5
    by (simp add: S5-def finished-def ord-res-5-step.simps)
qed

lemma right-unique-ord-res-5: right-unique (ord-res-5 N)
proof (rule right-uniqueI)
  fix s s' s''
  assume step1: ord-res-5 N s s' and step2: ord-res-5 N s s''
  show s' = s''
    using step1
  proof (cases N s s' rule: ord-res-5.cases)
    case hyps1: (skip ℳ1 C1 𝒞1' ℱ1 U1_er)
    with step2 show ?thesis
      by (cases N s s'' rule: ord-res-5.cases) simp-all
  next
    case hyps1: (production ℳ1 C1 L1 ℳ1' 𝒞1' ℱ1 U1_er)
    show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-5.cases)
      case (skip ℳ2 C2 𝒞2' ℱ2 U2_er)
      with hyps1 show ?thesis
        by simp
    next
      case hyps2: (production ℳ2 C2 L2 ℳ2' 𝒞2' ℱ2 U2_er)
      have C1 = C2
        using hyps1 hyps2 by simp
      hence L1 = L2
        using hyps1 hyps2
        by (metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D)
      thus ?thesis
        using hyps1 hyps2 by simp
    next
      case hyps2: (factoring ℳ2 C2 L2 ℱ2 ℱ2' ℳ2' 𝒞2' U2_er)
      have C1 = C2
        using hyps1 hyps2 by simp
      hence L1 = L2
        using hyps1 hyps2
        by (metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D)
      thus ?thesis
        using hyps1 hyps2 by simp
    next
      case hyps2: (resolution ℳ2 C2 L2 D2 U2_er' U2_er ℳ2' 𝒞2' ℱ2)
      have C1 = C2
```

```
      using hyps1 hyps2 by simp
    hence L1 = L2
      using hyps1 hyps2
      by (metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D)
    thus ?thesis
      using hyps1 hyps2 by simp
  qed
next
  case hyps1: (factoring M1 C1 L1 F1 F1' M1' C1' U1_er)
  show ?thesis
    using step2
  proof (cases N s s'' rule: ord-res-5.cases)
    case (skip M2 C2 C2' F2 U2_er)
    with hyps1 show ?thesis
      by simp
  next
    case hyps2: (production M2 C2 L2 M2' C2' F2 U2_er)
    have C1 = C2
      using hyps1 hyps2 by simp
    hence L1 = L2
      using hyps1 hyps2
      by (metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D)
    thus ?thesis
      using hyps1 hyps2 by simp
  next
    case hyps2: (factoring M2 C2 L2 F2 F2' M2' C2' U2_er)
    have C1 = C2
      using hyps1 hyps2 by simp
    thus ?thesis
      using hyps1 hyps2 by simp
  next
    case hyps2: (resolution M2 C2 L2 D2 U2_er' U2_er M2' C2' F2)
    have C1 = C2
      using hyps1 hyps2 by simp
    hence L1 = L2
      using hyps1 hyps2
      by (metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D)
    hence False
      using hyps1 hyps2 by argo
    thus ?thesis ..
  qed
next
  case hyps1: (resolution M1 C1 L1 D1 U1_er' U1_er M1' C1' F1)
  show ?thesis
    using step2
  proof (cases N s s'' rule: ord-res-5.cases)
    case (skip M2 C2 C2' F2 U2_er)
    with hyps1 show ?thesis
      by simp
```

next
case *hyps2*: (*production* $\mathcal{M}2$ *C2 L2* $\mathcal{M}2'$ $\mathcal{C}2'$ $\mathcal{F}2$ $U2_{er}$)
have *C1* = *C2*
using *hyps1 hyps2* **by** *simp*
hence *L1* = *L2*
using *hyps1 hyps2*
by (*metis linorder-lit.Uniq-is-maximal-in-mset Uniq-D*)
thus *?thesis*
using *hyps1 hyps2* **by** *simp*
next
case *hyps2*: (*factoring* $\mathcal{M}2$ *C2 L2* $\mathcal{F}2$ $\mathcal{F}2'$ $\mathcal{M}2'$ $\mathcal{C}2'$ $U2_{er}$)
have *C1* = *C2*
using *hyps1 hyps2* **by** *simp*
hence *L1* = *L2*
using *hyps1 hyps2*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
hence *False*
using *hyps1 hyps2* **by** *argo*
thus *?thesis* **..**
next
case *hyps2*: (*resolution* $\mathcal{M}2$ *C2 L2 D2* $U2_{er}'$ $U2_{er}$ $\mathcal{M}2'$ $\mathcal{C}2'$ $\mathcal{F}2$)
have $U1_{er}$ = $U2_{er}$ $\mathcal{F}1$ = $\mathcal{F}2$ $\mathcal{M}1$ = $\mathcal{M}2$ *C1* = *C2*
using *hyps1 hyps2* **by** *simp-all*
hence *L1* = *L2*
using *hyps1 hyps2*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
hence *D1* = *D2*
using ‹$\mathcal{M}1$ (*atm-of L1*) = *Some D1*› ‹$\mathcal{M}2$ (*atm-of L2*) = *Some D2*› ‹$\mathcal{M}1$ = $\mathcal{M}2$›
by *simp*

have $U1_{er}'$ = $U2_{er}'$
using ‹$U1_{er}'$ = *finsert* (*eres D1 C1*) $U1_{er}$› ‹$U2_{er}'$ = *finsert* (*eres D2 C2*) $U2_{er}$›
‹*D1* = *D2*› ‹*C1* = *C2*› ‹$U1_{er}$ = $U2_{er}$›
by *argo*

moreover have $\mathcal{M}1'$ = $\mathcal{M}2'$
using ‹$\mathcal{M}1'$ = (λ-. *None*)› ‹$\mathcal{M}2'$ = (λ-. *None*)›
by *argo*

moreover have $\mathcal{C}1'$ = $\mathcal{C}2'$
using *hyps1 hyps2* ‹$\mathcal{F}1$ = $\mathcal{F}2$› ‹$U1_{er}'$ = $U2_{er}'$› **by** *simp*

ultimately show *?thesis*
using ‹*s'* = ($U1_{er}'$, $\mathcal{F}1$, $\mathcal{M}1'$, $\mathcal{C}1'$)› ‹*s''* = ($U2_{er}'$, $\mathcal{F}2$, $\mathcal{M}2'$, $\mathcal{C}2'$)›
‹$\mathcal{F}1$ = $\mathcal{F}2$›
by *argo*
qed

qed
qed

lemma *right-unique-ord-res-5-step*: *right-unique ord-res-5-step*
proof (*rule right-uniqueI*)
fix *x y z*
show *ord-res-5-step x y* $\Longrightarrow$ *ord-res-5-step x z* $\Longrightarrow$ $y = z$
using *right-unique-ord-res-5*[*THEN right-uniqueD*]
by (*elim ord-res-5-step.cases*) (*metis prod.inject*)
qed

definition *next-clause-in-factorized-clause* **where**
next-clause-in-factorized-clause N s $\longleftrightarrow$
$(\forall\, U_{er}\ \mathcal{F}\ \mathcal{M}\ C.\ s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathit{Some}\ C) \longrightarrow C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))$

lemma *next-clause-in-factorized-clause*:
assumes *step*: *ord-res-5 N s s′*
shows *next-clause-in-factorized-clause N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-5.cases*)
case (*skip* $\mathcal{M}\ C\ \mathcal{C}'\ \mathcal{F}\ U_{er}$)
thus *?thesis*
unfolding *next-clause-in-factorized-clause-def*
by (*metis Pair-inject The-optional-eq-SomeD linorder-cls.is-minimal-in-fset-eq-is-least-in-fset*
linorder-cls.is-minimal-in-fset-ffilter-iff)
next
case (*production* $\mathcal{M}\ C\ L\ \mathcal{M}'\ \mathcal{C}'\ \mathcal{F}\ U_{er}$)
thus *?thesis*
unfolding *next-clause-in-factorized-clause-def*
by (*metis Pair-inject The-optional-eq-SomeD linorder-cls.is-minimal-in-fset-eq-is-least-in-fset*
linorder-cls.is-minimal-in-fset-ffilter-iff)
next
case (*factoring* $\mathcal{M}\ C\ L\ \mathcal{F}'\ \mathcal{F}\ \mathcal{M}'\ \mathcal{C}'\ U_{er}$)
thus *?thesis*
unfolding *next-clause-in-factorized-clause-def*
by (*metis Pair-inject The-optional-eq-SomeD linorder-cls.is-least-in-fset-iff*)
next
case (*resolution* $\mathcal{M}\ C\ L\ D\ U_{er}{}'\ U_{er}\ \mathcal{M}'\ \mathcal{C}'\ \mathcal{F}$)
thus *?thesis*
unfolding *next-clause-in-factorized-clause-def*
by (*metis Pair-inject The-optional-eq-SomeD linorder-cls.is-least-in-fset-iff*)
qed

definition *implicitly-factorized-clauses-subset* **where**
implicitly-factorized-clauses-subset N s $\longleftrightarrow$
$(\forall\, U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}.\ s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow \mathcal{F}\ |\subseteq|\ N\ |\cup|\ U_{er})$

lemma *ord-res-5-preserves-implicitly-factorized-clauses-subset*:
assumes

step: *ord-res-5 N s s′* **and**
invars:
implicitly-factorized-clauses-subset N s **and**
next-clause-in-factorized-clause N s
shows *implicitly-factorized-clauses-subset N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-5.cases*)
case (*skip* $\mathcal{M}$ C $\mathcal{C}'$ $\mathcal{F}$ U_{er})
thus *?thesis*
using *invars*
by (*simp add*: *implicitly-factorized-clauses-subset-def*)
next
case (*production* $\mathcal{M}$ C L $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$ U_{er})
thus *?thesis*
using *invars*
by (*simp add*: *implicitly-factorized-clauses-subset-def*)
next
case (*factoring* $\mathcal{M}$ C L $\mathcal{F}'$ $\mathcal{F}$ $\mathcal{M}'$ $\mathcal{C}'$ U_{er})
thus *?thesis*
using *invars*
by (*smt* (*verit*) *Pair-inject assms*(*3*) *fimage-iff finsert-fsubset iefac-def*
implicitly-factorized-clauses-subset-def literal.collapse(*1*)
next-clause-in-factorized-clause-def ex1-efac-eq-factoring-chain ex-ground-factoringI)
next
case (*resolution* $\mathcal{M}$ C L D $U_{er}{}'$ U_{er} $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$)
thus *?thesis*
using *invars*
by (*simp add*: *fsubset-funion-eq implicitly-factorized-clauses-subset-def*)
qed

lemma *interp-eq-Interp-if-least-greater*:
assumes
C-in: C $|\in|$ *NN* **and**
D-least-gt-C: *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) C) *NN*) D
shows *ord-res.interp* (*fset NN*) D = *ord-res.interp* (*fset NN*) C ∪ *ord-res.production* (*fset NN*) C
proof −
have *nex-between-C-and-D*: ¬ (∃ *CD* $|\in|$ *NN*. $C \prec_c CD \wedge CD \prec_c D$)
using *D-least-gt-C*
unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *auto*

have *ord-res.interp* (*fset NN*) D = *ord-res.interp* (*fset* {|B $|\in|$ *NN*. $B \preceq_c C$|}) D
proof (*rule ord-res.interp-swap-clause-set*)
have *NN* = {|B $|\in|$ *NN*. $B \preceq_c C$|} |∪| {|E $|\in|$ *NN*. $D \preceq_c E$|}
using *nex-between-C-and-D* **by** *force*
show {*Da*. *Da* $|\in|$ *NN* ∧ *Da* $\prec_c D$} = {*Da*. *Da* $|\in|$ {|B $|\in|$ *NN*. $(\prec_c)^{==}$ B C|} ∧ *Da* $\prec_c D$}
using ‹*NN* = {|B $|\in|$ *NN*. $(\prec_c)^{==}$ B C|} |∪| *ffilter* (($\prec_c$)$^{==}$ D) *NN*›

linorder-cls.leD **by** *auto*
qed

also have ... = $\bigcup$ (*ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) ‘
{*Da*. *Da* |∈| {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|} ∧ *Da* ≺$_c$ *D*})
unfolding *ord-res.interp-def* **..**

also have ... = $\bigcup$ (*ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) ‘
{*Da* ∈ *fset NN*. (≺$_c$)$^{==}$ *Da C* ∧ *Da* ≺$_c$ *D*})
by *auto*

also have ... = $\bigcup$ (*ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) ‘
{*Da* ∈ *fset NN*. (≺$_c$)$^{==}$ *Da C*})
proof −
have {|*Da* |∈| *NN*. *Da* ⪯$_c$ *C* ∧ *Da* ≺$_c$ *D*|} = {|*Da* |∈| *NN*. *Da* ⪯$_c$ *C*|}
using *nex-between-C-and-D*
by (*metis* (*no-types*, *opaque-lifting*) *D-least-gt-C linorder-cls.is-least-in-fset-ffilterD*(*2*)
linorder-cls.le-less-trans)
thus *?thesis*
by *fastforce*
qed

also have ... = $\bigcup$ (*ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) ‘
{*Da* ∈ *fset NN*. *Da* ≺$_c$ *C*}) ∪ *ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B*
C|}) *C*
proof −
have {*Da*. *Da* |∈| *NN* ∧ (≺$_c$)$^{==}$ *Da C*} = *insert C* {*Da*. *Da* |∈| *NN* ∧ *Da* ≺$_c$
C}
using *C-in* **by** *auto*
thus *?thesis*
by *blast*
qed

also have ... = *ord-res-Interp* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) *C*
unfolding *ord-res.interp-def* **by** *auto*

also have ... = *ord-res-Interp* (*fset NN*) *C*
proof −
have *ord-res.interp* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) *C* = *ord-res.interp* (*fset*
NN) *C*
using *ord-res.interp-swap-clause-set*[*of fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|} *C fset*
NN]
by *auto*

moreover have *ord-res.production* (*fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}) *C* =
ord-res.production (*fset NN*) *C*
using *ord-res.production-swap-clause-set*[*of fset* {|*B* |∈| *NN*. (≺$_c$)$^{==}$ *B C*|}
C fset NN]
by *auto*

```
    ultimately show ?thesis
      by simp
  qed

  finally show ?thesis .
qed

lemma interp-eq-empty-if-least-in-set:
  assumes linorder-cls.is-least-in-set N C
  shows ord-res.interp N C = {}
  using assms
  unfolding ord-res.interp-def
  unfolding linorder-cls.is-least-in-set-iff
  by auto

definition model-eq-interp-upto-next-clause where
  model-eq-interp-upto-next-clause N s ⟷
    (∀ U_er F M C. s = (U_er, F, M, Some C) ⟶
      dom M = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) C)

lemma model-eq-interp-upto-next-clause:
  assumes step: ord-res-5 N s s' and
    invars:
      model-eq-interp-upto-next-clause N s
      next-clause-in-factorized-clause N s
  shows model-eq-interp-upto-next-clause N s'
  using step
proof (cases N s s' rule: ord-res-5.cases)
  case step-hyps: (skip M C C' F U_er)

  have dom M = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) D if C' = Some
D for D
  proof −
    have dom M = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) C
      using invars(1)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
          OF ‹s = (U_er, F, M, Some C)›] .

    also have ... = ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) C
    proof −
      have ord-res.production (fset (iefac F |`| (N |∪| U_er))) C = {}
        using ‹dom M ⊨ C›
      unfolding invars(1)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
          OF ‹s = (U_er, F, M, Some C)›]
        by (simp add: ord-res.production-unfold)
      thus ?thesis
        by simp
    qed
```

```
    also have ... = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) D
    proof (rule interp-eq-Interp-if-least-greater[symmetric])
      show C |∈| iefac F |‘| (N |∪| U_er)
        using invars(2)[unfolded next-clause-in-factorized-clause-def, rule-format,
            OF ‹s = (U_er, F, M, Some C)›] .
    next
      show linorder-cls.is-least-in-fset (ffilter ((≺_c) C) (iefac F |‘| (N |∪| U_er)))
D
        using step-hyps(3−) that by (metis Some-eq-The-optionalD)
    qed

    finally show ?thesis .
  qed

  thus ?thesis
    using step-hyps by (simp add: model-eq-interp-upto-next-clause-def)
next
  case step-hyps: (production M C L M' C' F U_er)

  have dom M' = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) D if C' = Some
D for D
  proof −
    have dom M' = dom M ∪ {atm-of L}
      unfolding ‹M' = M(atm-of L ↦ C)› by simp

    also have ... = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) C ∪ {atm-of L}
     unfolding invars(1)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
          OF ‹s = (U_er, F, M, Some C)›] ..

    also have ... = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) C ∪
      ord-res.production (fset (iefac F |‘| (N |∪| U_er))) C
    proof −
      have ¬ ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) C ⊨ C
        using ‹¬ dom M ⊨ C›
      unfolding invars(1)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
          OF ‹s = (U_er, F, M, Some C)›] .
      hence atm-of L ∈ ord-res.production (fset (iefac F |‘| (N |∪| U_er))) C
        using ‹is-pos L› ‹ord-res.is-strictly-maximal-lit L C›
        using invars(2)[unfolded next-clause-in-factorized-clause-def, rule-format,
            OF ‹s = (U_er, F, M, Some C)›]
        unfolding ord-res.production-unfold mem-Collect-eq
      by (metis linorder-lit.is-greatest-in-mset-iff literal.collapse(1) multi-member-split)
      hence ord-res.production (fset (iefac F |‘| (N |∪| U_er))) C = {atm-of L}
        by (metis empty-iff insertE ord-res.production-eq-empty-or-singleton)
      thus ?thesis
        by argo
    qed

    also have ... = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) D
```

```
    proof (rule interp-eq-Interp-if-least-greater[symmetric])
      show C |∈| iefac F |‘| (N |∪| U_er)
        using invars(2)[unfolded next-clause-in-factorized-clause-def, rule-format,
            OF ‹s = (U_er, F, M, Some C)›] .
    next
      show linorder-cls.is-least-in-fset (ffilter ((≺_c) C) (iefac F |‘| (N |∪| U_er)))
D
        using step-hyps(3−) that by (metis Some-eq-The-optionalD)
    qed

    finally show ?thesis .
  qed

  thus ?thesis
    using step-hyps by (simp add: model-eq-interp-upto-next-clause-def)
next
  case step-hyps: (factoring M C L F' F M' C' U_er)

  have dom M' = ord-res.interp (fset (iefac F' |‘| (N |∪| U_er))) D if C' = Some
D for D
  proof −
    have dom M' = {}
      using step-hyps(3−) by simp
    also have ... = ord-res.interp (fset (iefac F' |‘| (N |∪| U_er))) D
    proof (rule interp-eq-empty-if-least-in-set[symmetric])
      show linorder-cls.is-least-in-set (fset (iefac F' |‘| (N |∪| U_er))) D
        using step-hyps(3−) that
        by (metis Some-eq-The-optionalD linorder-cls.is-least-in-fset-iff
            linorder-cls.is-least-in-set-iff)
    qed
    finally show ?thesis .
  qed

  thus ?thesis
    using step-hyps by (simp add: model-eq-interp-upto-next-clause-def)
next
  case step-hyps: (resolution M C L D U_er' U_er M' C' F)

  have dom M' = ord-res.interp (fset (iefac F |‘| (N |∪| U_er'))) D if C' = Some
D for D
  proof −
    have dom M' = {}
      using step-hyps(3−) by simp
    also have ... = ord-res.interp (fset (iefac F |‘| (N |∪| U_er'))) D
    proof (rule interp-eq-empty-if-least-in-set[symmetric])
      show linorder-cls.is-least-in-set (fset (iefac F |‘| (N |∪| U_er'))) D
        using step-hyps(3−) that
        by (metis Some-eq-The-optionalD linorder-cls.is-least-in-fset-iff
            linorder-cls.is-least-in-set-iff)
```

```
    qed
    finally show ?thesis .
  qed

  thus ?thesis
    using step-hyps by (simp add: model-eq-interp-upto-next-clause-def)
qed

definition all-smaller-clauses-true-wrt-respective-Interp where
  all-smaller-clauses-true-wrt-respective-Interp N s ⟷
    (∀ U_er F M C. s = (U_er, F, M, C) ⟶
      (∀ C |∈| iefac F |`| (N |∪| U_er). (∀ D. C = Some D ⟶ C ≺_c D) ⟶
        ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) C ⊨ C))

lemma all-smaller-clauses-true-wrt-respective-Interp:
  assumes step: ord-res-5 N s s' and
    invars:
      all-smaller-clauses-true-wrt-respective-Interp N s
      model-eq-interp-upto-next-clause N s
      next-clause-in-factorized-clause N s
  shows all-smaller-clauses-true-wrt-respective-Interp N s'
  using step
proof (cases N s s' rule: ord-res-5.cases)
  case step-hyps: (skip M D C' F U_er)

  have D-true: ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) D ⊨ D
    using invars(2) ord-res.production-unfold step-hyps(1) step-hyps(3)
    by (auto simp: model-eq-interp-upto-next-clause-def)

  have ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) C ⊨ C
    if C' = Some E and C-in: C |∈| iefac F |`| (N |∪| U_er) and C ≺_c E for C E
  proof -
    have linorder-cls.is-least-in-fset (ffilter ((≺_c) D) (iefac F |`| (N |∪| U_er))) E
      using step-hyps ‹C' = Some E› by (metis Some-eq-The-optionalD)
    hence C ⪯_c D
      using C-in ‹C ≺_c E›
      by (metis asympD linorder-cls.is-least-in-ffilter-iff linorder-cls.le-less-linear
          ord-res.asymp-less-cls)
    thus ?thesis
      using D-true
        using invars(1)[unfolded all-smaller-clauses-true-wrt-respective-Interp-def,
rule-format,
          OF ‹s = (U_er, F, M, Some D)› C-in] by auto
  qed

  moreover have ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) C ⊨ C
    if C' = None and C-in: C |∈| iefac F |`| (N |∪| U_er) for C
  proof -
    have ∄ x. linorder-cls.is-least-in-fset (ffilter ((≺_c) D) (iefac F |`| (N |∪| U_er)))
```

x
    using *step-hyps* ‹$\mathcal{C}' = None$›
   by (*metis* (*no-types*, *opaque-lifting*) *None-eq-The-optionalD linorder-cls.Uniq-is-least-in-fset the1-equality'*)
   hence ¬ ($\exists x$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $D \prec_c x$)
    by (*metis femptyE ffmember-filter linorder-cls.ex1-least-in-fset*)
   hence $C \prec_c D \vee C = D$
    using *C-in* **by** *force*
   thus *?thesis*
   proof (*elim disjE*)
    assume $C \prec_c D$
    then show *?thesis*
     using *invars*(*1*) ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$› *C-in*
     by (*auto simp*: *all-smaller-clauses-true-wrt-respective-Interp-def*)
   next
    assume $C = D$
    then show *?thesis*
     using *D-true* **by** *argo*
   qed
  qed

  ultimately show *?thesis*
   using *step-hyps*
  by (*smt* (*verit*, *best*) *all-smaller-clauses-true-wrt-respective-Interp-def old.prod.inject option.exhaust*)
next
  case *step-hyps*: (*production* $\mathcal{M}$ D K $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$ U_{er})

  have K ∈# D
   using *step-hyps*(*3*−) **by** (*metis linorder-lit.is-greatest-in-mset-iff*)

  have ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⟙ D
   using ‹¬ *dom* $\mathcal{M}$ ⟙ D›
  unfolding *invars*(*2*)[*unfolded model-eq-interp-upto-next-clause-def*, *rule-format*, *OF* ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$›] **.**
  hence *atm-of* $K \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D
   using ‹*is-pos* K› ‹*ord-res.is-strictly-maximal-lit* K D› ‹K ∈# D›
   using *invars*(*3*)[*unfolded next-clause-in-factorized-clause-def*, *rule-format*, *OF* ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$›]
   unfolding *ord-res.production-unfold mem-Collect-eq*
   by (*metis literal.collapse*(*1*) *multi-member-split*)
  hence *prod-D-eq*: *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D = {*atm-of* K}
   by (*metis empty-iff insertE ord-res.production-eq-empty-or-singleton*)
  hence *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⟙*l* K
   using ‹*is-pos* K› **by** *force*
  hence *D-true*: *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⟙ D
   using ‹K ∈# D› **by** *auto*

have *dom* $\mathcal{M}' = dom\ \mathcal{M} \cup \{atm\text{-}of\ K\}$
unfolding ‹$\mathcal{M}' = \mathcal{M}(atm\text{-}of\ K \mapsto D)$› **by** *simp*

also have ... = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $D \cup \{atm\text{-}of\ K\}$
unfolding *invars*(*2*)[*unfolded model-eq-interp-upto-next-clause-def*, *rule-format*, *OF* ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$›] **..**

also have ... = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $D \cup$
ord-res.production (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D
using *prod-D-eq* **by** *argo*

finally have *dom-*$\mathcal{M}'$*-eq*: *dom* $\mathcal{M}'$ = *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D **.**

have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $C \models C$
if $\mathcal{C}'$ = *Some* E **and** *C-in*: C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and** $C \prec_c E$ **for** $C\ E$
proof −
have *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) D) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E
using *step-hyps* ‹$\mathcal{C}'$ = *Some* E› **by** (*metis Some-eq-The-optionalD*)
hence $C \preceq_c D$
using *C-in* ‹$C \prec_c E$›
by (*metis asympD linorder-cls.is-least-in-ffilter-iff linorder-cls.le-less-linear ord-res.asymp-less-cls*)
thus *?thesis*
using *D-true*
using *invars*(*1*)[*unfolded all-smaller-clauses-true-wrt-respective-Interp-def*, *rule-format*, *OF* ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$› *C-in*] **by** *auto*
qed

moreover have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $C \models C$
if $\mathcal{C}'$ = *None* **and** *C-in*: C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **for** C
proof −
have ∄x. *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) D) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) x
using *step-hyps* ‹$\mathcal{C}'$ = *None*›
by (*metis* (*no-types*, *opaque-lifting*) *None-eq-The-optionalD linorder-cls.Uniq-is-least-in-fset the1-equality'*)
hence ¬ (∃x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $D \prec_c x$)
by (*metis femptyE ffmember-filter linorder-cls.ex1-least-in-fset*)
hence $C \prec_c D \vee C = D$
using *C-in* **by** *force*
thus *?thesis*
proof (*elim disjE*)
assume $C \prec_c D$
then show *?thesis*
using *invars*(*1*) ‹$s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D)$› *C-in*
by (*auto simp*: *all-smaller-clauses-true-wrt-respective-Interp-def*)
next

```
      assume C = D
      thus ?thesis
        using D-true by argo
    qed
  qed

  ultimately show ?thesis
    unfolding step-hyps(2) all-smaller-clauses-true-wrt-respective-Interp-def
    by (metis prod.inject option.exhaust)
next
  case step-hyps: (factoring M D K F' F M' C' U_er)
  have D |∈| iefac F |‘| (N |∪| U_er)
    using invars(2−) ‹s = (U_er, F, M, Some D)›
    by (metis next-clause-in-factorized-clause-def)
  hence D |∈| N |∪| U_er
    using step-hyps(3−)
    by (smt (verit, ccfv-threshold) fimage-iff iefac-def literal.collapse(1)
        ex1-efac-eq-factoring-chain ex-ground-factoringI)
  hence iefac F' D |∈| iefac F' |‘| (N |∪| U_er)
    unfolding ‹F' = finsert D F› by simp
  hence C' ≠ None
    using step-hyps(3−)
   by (metis The-optional-eq-NoneD finsert-not-fempty linorder-cls.ex1-least-in-fset
set-finsert)
  then obtain E where
    C' = Some E
    by auto

  have ¬ (∃ C |∈| iefac F' |‘| (N |∪| U_er). C ≺_c E)
    using ‹C' = Some E› step-hyps(9)
    by (metis The-optional-eq-SomeD linorder-cls.is-least-in-fset-iff
        linorder-cls.less-imp-not-less)

  thus ?thesis
    unfolding step-hyps(1,2) all-smaller-clauses-true-wrt-respective-Interp-def
    using ‹C' = Some E› by simp
next
  case step-hyps: (resolution M C L D U_er' U_er M' C' F)
  hence eres D C |∈| N |∪| U_er'
    by simp
  hence iefac F (eres D C) |∈| iefac F |‘| (N |∪| U_er')
    by simp
  hence C' ≠ None
    using step-hyps(3−)
   by (metis The-optional-eq-NoneD finsert-not-fempty linorder-cls.ex1-least-in-fset
set-finsert)
  then obtain E where
    C' = Some E
    by auto
```

have ¬ (∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| $U_{er}{}'$). *C* ≺$_c$ *E*)
using ‹$\mathcal{C}'$ = *Some E*› *step-hyps*(*9*)
by (*metis The-optional-eq-SomeD linorder-cls.is-least-in-fset-iff linorder-cls.less-imp-not-less*)

thus *?thesis*
unfolding *step-hyps*(*1*,*2*) *all-smaller-clauses-true-wrt-respective-Interp-def*
using ‹$\mathcal{C}'$ = *Some E*› **by** *simp*
qed

lemma *all-smaller-clauses-true-wrt-model*:
assumes
invars:
all-smaller-clauses-true-wrt-respective-Interp N s
model-eq-interp-upto-next-clause N s
shows ∀ U_{er} $\mathcal{F}$ $\mathcal{M}$ *D*. *s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some D*) ⟶
(∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶ *dom* $\mathcal{M}$ ⊨ *C*)
proof (*intro allI impI ballI*)
fix U_{er} $\mathcal{F}$ $\mathcal{M}$ *D C*
assume
s-def: *s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some D*) **and**
C-in: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
C-lt: *C* ≺$_c$ *D*

hence *C-true*: *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⊨ *C*
using *invars*(*1*)[*unfolded all-smaller-clauses-true-wrt-respective-Interp-def s-def*]
by *auto*

moreover have *dom* $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*
using *invars*(*2*) *s-def* **by** (*metis model-eq-interp-upto-next-clause-def*)

ultimately show *dom* $\mathcal{M}$ ⊨ *C*
using *ord-res.entailed-clause-stays-entailed' C-lt* **by** *metis*
qed

definition *model-eq-sublocale* **where**
model-eq-sublocale N s ⟷
(∀ U_{er} $\mathcal{F}$ $\mathcal{M}$. *s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *None*) ⟶
(*let NN* = *fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *in dom* $\mathcal{M}$ = ⋃ (*ord-res.production NN* ‘ *NN*)))

lemma *all-smaller-clauses-true-wrt-model-strong*:
assumes
invars:
all-smaller-clauses-true-wrt-respective-Interp N s
model-eq-interp-upto-next-clause N s
model-eq-sublocale N s
shows ∀ U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$. *s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) ⟶

(∀ C |∈| iefac $\mathcal{F}$ |‘| (N |∪| U_{er}). (∀ D. $\mathcal{C}$ = Some D ⟶ C $\prec_c$ D) ⟶ dom $\mathcal{M}$ ⊨ C)
proof (*intro allI impI ballI*)
fix U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$ C
assume
s-def: s = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) **and**
C-in: C |∈| iefac $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
C-lt: ∀ D. $\mathcal{C}$ = Some D ⟶ C $\prec_c$ D
hence *C-true*: *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊨ C
using *invars*(*1*) **by** (*metis all-smaller-clauses-true-wrt-respective-Interp-def*)

show dom $\mathcal{M}$ ⊨ C
proof (*cases* $\mathcal{C}$)
case $\mathcal{C}$-*def*: *None*
have *let NN* = *fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) *in dom* $\mathcal{M}$ = ⋃ (*ord-res.production NN ‘ NN*)
using *invars*(*3*) *s-def* $\mathcal{C}$-*def*
by (*metis model-eq-sublocale-def*)
then show *?thesis*
using *C-true*
by (*smt* (*verit*, *ccfv-SIG*) *C-in UN-I insertCI linorder-lit.is-greatest-in-mset-iff ord-res.lift-entailment-to-Union ord-res.mem-productionE ord-res.production-eq-empty-or-singleton pos-literal-in-imp-true-cls sup-bot.right-neutral*)
next
case $\mathcal{C}$-*def*: (*Some D*)
have C $\prec_c$ D
using *C-lt* $\mathcal{C}$-*def* **by** *metis*
moreover have dom $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D
using *invars*(*2*) *s-def* $\mathcal{C}$-*def* **by** (*metis model-eq-interp-upto-next-clause-def*)
ultimately show *?thesis*
using *ord-res.entailed-clause-stays-entailed′ C-true* **by** *metis*
qed
qed

lemma *next-clause-lt-least-false-clause*:
assumes
invars:
all-smaller-clauses-true-wrt-respective-Interp N s
model-eq-interp-upto-next-clause N s
shows ∀ U_{er} $\mathcal{F}$ $\mathcal{M}$ C. s = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*) ⟶
(∀ D. *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) D ⟶ C $\preceq_c$ D)
proof (*intro allI impI ballI*)
fix U_{er} $\mathcal{F}$ $\mathcal{M}$ C D
assume
s-def: s = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*) **and**
D-least-false: *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) D
then show C $\preceq_c$ D
using *invars*[*unfolded model-eq-interp-upto-next-clause-def*

all-smaller-clauses-true-wrt-respective-Interp-def, *rule-format*, *OF s-def*, *simplified*]
by (*metis* (*no-types*, *lifting*) *fimage.rep-eq is-least-false-clause-def*
linorder-cls.is-least-in-fset-ffilterD(*1*) *linorder-cls.is-least-in-fset-ffilterD*(*2*)
linorder-cls.le-less-linear sup-fset.rep-eq)
qed

definition *atoms-in-model-were-produced* **where**
atoms-in-model-were-produced N s $\longleftrightarrow$
($\forall$ U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$. $s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow (\forall A\ C.\ \mathcal{M}\ A = Some\ C \longrightarrow$
$A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*))

lemma *atoms-in-model-were-produced*:
assumes *step*: *ord-res-5 N s s′* **and**
invars:
atoms-in-model-were-produced N s
model-eq-interp-upto-next-clause N s
next-clause-in-factorized-clause N s
shows *atoms-in-model-were-produced N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-5.cases*)
case (*skip* $\mathcal{M}$ *C* $\mathcal{C}'$ $\mathcal{F}$ U_{er})
thus *?thesis*
using *invars*(*1*) **by** (*simp add*: *atoms-in-model-were-produced-def*)
next
case (*production* $\mathcal{M}$ *C L* $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$ U_{er})
obtain *A* **where** $L = Pos\ A$
using ‹*is-pos L*› **by** (*cases L*) *simp-all*

have *atm-of L* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
unfolding *ord-res.production-unfold mem-Collect-eq*
proof (*intro exI conjI*)
show *atm-of L* = *A*
unfolding ‹*L* = *Pos A*› *literal.sel* **..**
next
show *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using *invars*(*3*)[*unfolded next-clause-in-factorized-clause-def*, *rule-format*,
OF ‹*s* = (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some C*)›] **.**
next
have *L* ∈# *C*
using ‹*linorder-lit.is-maximal-in-mset C L*›
unfolding *linorder-lit.is-maximal-in-mset-iff* **..**
thus *C* = *add-mset* (*Pos A*) (*C* − {#*Pos A*#})
unfolding ‹*L* = *Pos A*› **by** *simp*
next
show *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
using ‹*ord-res.is-strictly-maximal-lit L C*›
unfolding ‹*L* = *Pos A*› **.**
next

show ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⊫ *C*
using ‹¬ *dom* $\mathcal{M}$ ⊫ *C*›
unfolding *invars*(*2*)[*unfolded model-eq-interp-upto-next-clause-def*, *rule-format*, *OF* ‹*s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*)›] **.**
qed *simp-all*

thus *?thesis*
using *invars*(*1*)
by (*simp add*: *atoms-in-model-were-produced-def*
‹*s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*)› ‹*s'* = (U_{er}, $\mathcal{F}$, $\mathcal{M}'$, $\mathcal{C}'$)› ‹$\mathcal{M}'$ = $\mathcal{M}$(*atm-of L* ↦ *C*)›)
qed (*simp-all add*: *atoms-in-model-were-produced-def*)

definition *all-produced-atoms-in-model* **where**
all-produced-atoms-in-model N s ⟷
(∀ U_{er} $\mathcal{F}$ $\mathcal{M}$ *D*. *s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some D*) ⟶ (∀ *C A*. *C* ≺$_c$ *D* ⟶
A ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⟶ $\mathcal{M}$ *A* = *Some C*))

lemma *all-produced-atoms-in-model*:
assumes *step*: *ord-res-5 N s s'* **and**
invars:
all-produced-atoms-in-model N s
model-eq-interp-upto-next-clause N s
next-clause-in-factorized-clause N s
shows *all-produced-atoms-in-model N s'*
using *step*
proof (*cases N s s' rule*: *ord-res-5.cases*)
case (*skip* $\mathcal{M}$ *C* $\mathcal{C}'$ $\mathcal{F}$ U_{er})
have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* = {}
using *invars*
by (*metis ex-in-conv model-eq-interp-upto-next-clause-def local.skip*(*1*) *local.skip*(*3*) *ord-res.mem-productionE*)
thus *?thesis*
using *invars*(*1*) ‹*s* = (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*)›
unfolding *all-produced-atoms-in-model-def*
by (*smt* (*verit*, *del-insts*) *Pair-inject The-optional-eq-SomeD empty-iff linorder-cls.is-least-in-ffilter-iff linorder-cls.not-less-iff-gr-or-eq local.skip*(*2*) *local.skip*(*4*) *ord-res.mem-productionE*)
next
case *step-hyps*: (*production* $\mathcal{M}$ *C L* $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$ U_{er})
obtain *A* **where** *L* = *Pos A*
using ‹*is-pos L*› **by** (*cases L*) *simp-all*

have *atm-of L* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
unfolding *ord-res.production-unfold mem-Collect-eq*
proof (*intro exI conjI*)
show *atm-of L* = *A*
unfolding ‹*L* = *Pos A*› *literal.sel* **..**

```
  next
    show C |∈| iefac F |‘| (N |∪| U_er)
    using invars ‹s = (U_er, F, M, Some C)› by (metis next-clause-in-factorized-clause-def)
  next
    have L ∈# C
      using ‹linorder-lit.is-maximal-in-mset C L›
      unfolding linorder-lit.is-maximal-in-mset-iff ..
    thus C = add-mset (Pos A) (C − {#Pos A#})
      unfolding ‹L = Pos A› by simp
  next
    show ord-res.is-strictly-maximal-lit (Pos A) C
      using ‹ord-res.is-strictly-maximal-lit L C›
      unfolding ‹L = Pos A› .
  next
    show ¬ ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) C ⟙ C
      using ‹¬ dom M ⟙ C›
    using invars ‹s = (U_er, F, M, Some C)› by (metis model-eq-interp-upto-next-clause-def)
  qed simp-all

  thus ?thesis
    using invars ‹s = (U_er, F, M, Some C)›
    unfolding all-produced-atoms-in-model-def
    using ‹s' = (U_er, F, M', C')› ‹M' = M(atm-of L ↦ C)›
    using prod.inject
   by (smt (verit, del-insts) Some-eq-The-optionalD asympD ord-res.mem-productionE
       linorder-cls.antisym-conv3 linorder-cls.is-least-in-ffilter-iff
       linorder-trm.not-less-iff-gr-or-eq step-hyps(8) map-upd-Some-unfold
       ord-res.asymp-less-cls ord-res.less-trm-iff-less-cls-if-mem-production)
next
  case step-hyps: (factoring M C L F' F M' C' U_er)
  have ¬ (∃ C |∈| iefac F' |‘| (N |∪| U_er). C ≺_c D) if C' = Some D for D
  proof (rule nbex-less-than-least-in-fset)
    show linorder-cls.is-least-in-fset (iefac F' |‘| (N |∪| U_er)) D
      using step-hyps that by (metis The-optional-eq-SomeD)
  qed
  thus ?thesis
    unfolding all-produced-atoms-in-model-def
    by (metis step-hyps(2) ord-res.mem-productionE prod.inject)
next
  case step-hyps: (resolution M C L D U_er' U_er M' C' F)
  have ¬ (∃ C |∈| iefac F |‘| (N |∪| U_er'). C ≺_c D) if C' = Some D for D
  proof (rule nbex-less-than-least-in-fset)
    show linorder-cls.is-least-in-fset (iefac F |‘| (N |∪| U_er')) D
      using step-hyps that by (metis The-optional-eq-SomeD)
  qed
  thus ?thesis
    unfolding all-produced-atoms-in-model-def
    by (metis step-hyps(2) ord-res.mem-productionE prod.inject)
qed
```

definition *ord-res-5-invars* **where**
ord-res-5-invars N s $\longleftrightarrow$
next-clause-in-factorized-clause N s $\wedge$
implicitly-factorized-clauses-subset N s $\wedge$
model-eq-interp-upto-next-clause N s $\wedge$
all-smaller-clauses-true-wrt-respective-Interp N s $\wedge$
atoms-in-model-were-produced N s $\wedge$
all-produced-atoms-in-model N s

lemma *ord-res-5-invars-initial-state*:
assumes
$\mathcal{F}$*-subset*: $\mathcal{F}$ $|\subseteq|$ *N* $|\cup|$ U_{er} **and**
C-least: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) *C*
shows *ord-res-5-invars N* (U_{er}, $\mathcal{F}$, *Map.empty*, *Some C*)
unfolding *ord-res-5-invars-def*
proof (*intro conjI*)
show *next-clause-in-factorized-clause N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *next-clause-in-factorized-clause-def*
using *C-least*[*unfolded linorder-cls.is-least-in-fset-iff*] **by** *simp*
next
show *implicitly-factorized-clauses-subset N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *implicitly-factorized-clauses-subset-def*
using $\mathcal{F}$*-subset* **by** *simp*
next
have *ord-res.interp* (*iefac* $\mathcal{F}$ ‘ (*fset N* $\cup$ *fset* U_{er})) *C* = {}
using *C-least*[*unfolded linorder-cls.is-least-in-fset-iff*]
by (*simp add*: *interp-eq-empty-if-least-in-set linorder-cls.is-least-in-set-iff*)
thus *model-eq-interp-upto-next-clause N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *model-eq-interp-upto-next-clause-def* **by** *simp*
next
have $\neg(\exists$ *Ca* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}). *Ca* $\prec_c$ *C*)
using *C-least*[*unfolded linorder-cls.is-least-in-fset-iff*]
by (*metis linorder-cls.less-asym*)
thus *all-smaller-clauses-true-wrt-respective-Interp N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def* **by** *simp*
next
show *atoms-in-model-were-produced N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *atoms-in-model-were-produced-def* **by** *simp*
next
have $\forall$ *Ca*. *Ca* $\prec_c$ *C* $\longrightarrow$ *Ca* $|\notin|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *C-least*[*unfolded linorder-cls.is-least-in-fset-iff*]
by (*metis linorder-cls.order.asym*)
thus *all-produced-atoms-in-model N* (U_{er}, $\mathcal{F}$, λx. *None*, *Some C*)
unfolding *all-produced-atoms-in-model-def*
by (*simp add*: *ord-res.production-unfold*)
qed

lemma *ord-res-5-preserves-invars*:
  assumes *step*: *ord-res-5 N s s′* **and** *invars*: *ord-res-5-invars N s*
  shows *ord-res-5-invars N s′*
proof −
  obtain U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$ **where** *s-def*: $s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$
    by (*metis prod.exhaust*)

  then show *?thesis*
    unfolding *ord-res-5-invars-def*
    using *invars*[*unfolded ord-res-5-invars-def*]
    using *next-clause-in-factorized-clause*[*OF step*]
      ord-res-5-preserves-implicitly-factorized-clauses-subset[*OF step*]
      model-eq-interp-upto-next-clause[*OF step*]
      all-smaller-clauses-true-wrt-respective-Interp[*OF step*]
      atoms-in-model-were-produced[*OF step*]
      all-produced-atoms-in-model[*OF step*]
    by *metis*
qed

lemma *rtranclp-ord-res-5-preserves-invars*:
  assumes *steps*: $(\textit{ord-res-5 } N)^{**}$ *s s′* **and** *invars*: *ord-res-5-invars N s*
  shows *ord-res-5-invars N s′*
  using *steps invars*
 by (*induction s rule*: *converse-rtranclp-induct*) (*auto intro*: *ord-res-5-preserves-invars*)

lemma *tranclp-ord-res-5-preserves-invars*:
  assumes *steps*: $(\textit{ord-res-5 } N)^{++}$ *s s′* **and** *invars*: *ord-res-5-invars N s*
  shows *ord-res-5-invars N s′*
  using *rtranclp-ord-res-5-preserves-invars*
  by (*metis invars steps tranclp-into-rtranclp*)

lemma *le-least-false-clause*:
  fixes *N s* U_{er} $\mathcal{F}$ $\mathcal{M}$ *C D*
  assumes
    invars: *ord-res-5-invars N s* **and**
    s-def: $s = (U_{er}, \mathcal{F}, \mathcal{M}, \textit{Some } C)$ **and**
    D-least-false: *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *D*
  shows $C \preceq_c D$
proof −
  have *D-in*: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
    using *D-least-false*
    unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
    by *argo*

  show $C \preceq_c D$
  proof (*rule next-clause-lt-least-false-clause*[*rule-format*])
    show *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *D*
      using *D-least-false* **.**

```
  next
    show (U_er, F, M, Some C) = (U_er, F, M, Some C) ..
  next
    show all-smaller-clauses-true-wrt-respective-Interp N (U_er, F, M, Some C)
      using invars by (metis s-def ord-res-5-invars-def)
  next
    show model-eq-interp-upto-next-clause N (U_er, F, M, Some C)
      using invars by (metis s-def ord-res-5-invars-def)
  qed
qed

lemma ex-ord-res-5-if-not-final:
  assumes
    not-final: ¬ ord-res-5-final S and
    invars: ∀ N s. S = (N, s) ⟶ ord-res-5-invars N s
  shows ∃ S'. ord-res-5-step S S'
proof −
  from not-final obtain N U_er F M C where
    S-def: S = (N, (U_er, F, M, Some C)) and C ≠ {#}
    unfolding ord-res-5-final.simps de-Morgan-disj not-ex
    by (metis option.exhaust surj-pair)

  note invars' = invars[unfolded ord-res-5-invars-def, rule-format, OF S-def]

  have ∃ s'. ord-res-5 N (U_er, F, M, Some C) s'
  proof (cases dom M ⊨ C)
    case True
    thus ?thesis
      using ord-res-5.skip by metis
  next
    case C-false: False
    obtain L where L-max: linorder-lit.is-maximal-in-mset C L
      using linorder-lit.ex-maximal-in-mset[OF ‹C ≠ {#}›] ..

    show ?thesis
    proof (cases L)
      case (Pos A)
      hence L-pos: is-pos L
        by simp
      show ?thesis
      proof (cases ord-res.is-strictly-maximal-lit L C)
        case True
        then show ?thesis
          using ord-res-5.production[OF C-false L-max L-pos] by metis
      next
        case L-not-striclty-max: False
        thus ?thesis
          using ord-res-5.factoring[OF C-false L-max L-pos L-not-striclty-max - refl
refl] by metis
```

```
    qed
  next
    case (Neg A)
    hence L-neg: is-neg L
      by simp

    have C-least-false: is-least-false-clause (iefac F |`| (N |∪| U_er)) C
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      show C |∈| iefac F |`| (N |∪| U_er)
        using invars' by (metis next-clause-in-factorized-clause-def)
    next
      have ¬ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) C ⟹ C
        using invars' C-false by (metis model-eq-interp-upto-next-clause-def)
      moreover have ord-res.production (fset (iefac F |`| (N |∪| U_er))) C = {}
      proof −
        have ∄L. is-pos L ∧ ord-res.is-strictly-maximal-lit L C
          using L-max L-neg
          by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset
              linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
        thus ?thesis
          using unproductive-if-nex-strictly-maximal-pos-lit by metis
      qed
      ultimately show ¬ ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) C ⟹ C
        by simp
    next
      fix D
      assume D-in: D |∈| iefac F |`| (N |∪| U_er) and D ≠ C and
        C-false: ¬ ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) D ⟹ D
      have ¬ D ≺_c C
        using C-false
        using invars' D-in
        unfolding all-smaller-clauses-true-wrt-respective-Interp-def
        by auto
      thus C ≺_c D
        using ‹D ≠ C› by order
    qed
    then obtain D where D|∈|iefac F |`| (N |∪| U_er) and
      D ≺_c C and
      ord-res.is-strictly-maximal-lit (Pos A) D and
      D-prod: ord-res.production (fset (iefac F |`| (N |∪| U_er))) D = {A}
      using bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit
        L-max[unfolded Neg] by metis

    have M (atm-of L) = Some D
      using invars'
      by (metis Neg ‹D ≺_c C› all-produced-atoms-in-model-def D-prod insertI1
literal.sel(2))
```

```
    then show ?thesis
      using ord-res-5.resolution[OF C-false L-max L-neg] by metis
  qed
 qed
 thus ?thesis
  using S-def ord-res-5-step.simps by metis
qed

lemma ord-res-5-safe-state-if-invars:
 fixes N s
 assumes invars: ord-res-5-invars N s
 shows safe-state ord-res-5-step ord-res-5-final (N, s)
proof −
 {
  fix S'
  assume ord-res-5-semantics.eval (N, s) S' and stuck: stuck-state ord-res-5-step
S'
  then obtain s' where S' = (N, s') and (ord-res-5 N)** s s'
  proof (induction (N, s) arbitrary: N s rule: converse-rtranclp-induct)
    case base
    thus ?case by simp
  next
    case (step z)
    thus ?case
     by (smt (verit, ccfv-SIG) converse-rtranclp-into-rtranclp ord-res-5-step.cases
prod.inject)
  qed
  hence ord-res-5-invars N s'
    using invars rtranclp-ord-res-5-preserves-invars by metis
  hence ¬ ord-res-5-final S' ⟹ ∃ S''. ord-res-5-step S' S''
    using ex-ord-res-5-if-not-final[of S'] ‹S' = (N, s')› by blast
  hence ord-res-5-final S'
    using stuck[unfolded stuck-state-def] by argo
 }
 thus ?thesis
  unfolding safe-state-def stuck-state-def by metis
qed

lemma MAGIC1:
 assumes invars: ord-res-5-invars N (U_er, F, M, C)
 shows ∃ M' C'. (ord-res-5 N)** (U_er, F, M, C) (U_er, F, M', C') ∧
  (∄ M'' C''. ord-res-5 N (U_er, F, M', C') (U_er, F, M'', C''))
proof −
 define R where
  R = (λ(M, C) (M', C'). ord-res-5 N (U_er, F, M, C) (U_er, F, M', C'))

 define f :: ('f gterm ⇒ 'f gclause option) × 'f gclause option ⇒ 'f gclause fset
where
  f = (λ(M, C). case C of None ⇒ {||} | Some C ⇒ finsert C {|D |∈| iefac F
```

$|`|\ (N\ |\cup|\ U_{er}).\ C \preceq_c D|\})$

have *Wellfounded.wfp-on* $\{x'.\ R^{**}\ (\mathcal{M},\ \mathcal{C})\ x'\}\ R^{-1-1}$
proof (*rule wfp-on-if-convertible-to-wfp-on*)
have *wfp* $(|\subset|)$
by *auto*
thus *Wellfounded.wfp-on* $(f\ `\ \{x'.\ R^{**}\ (\mathcal{M},\ \mathcal{C})\ x'\})\ (|\subset|)$
using *Wellfounded.wfp-on-subset subset-UNIV* **by** *metis*
next
fix $x\ y$ :: $('f\ gterm \Rightarrow 'f\ gclause\ option) \times 'f\ gclause\ option$

obtain $\mathcal{M}_x\ \mathcal{C}_x$ **where** *x-def*: $x = (\mathcal{M}_x,\ \mathcal{C}_x)$
by *force*

obtain $\mathcal{M}_y\ \mathcal{C}_y$ **where** *y-def*: $y = (\mathcal{M}_y,\ \mathcal{C}_y)$
by *force*

have *rtranclp-with-constsD*: $(\lambda(y,\ z)\ (y',\ z').\ R\ (x,\ y,\ z)\ (x,\ y',\ z'))^{**}\ (y,\ z)\ (y',\ z') \Longrightarrow$
$R^{**}\ (x,\ y,\ z)\ (x,\ y',\ z')$ **for** $R\ x\ y\ z\ y'\ z'$
proof (*induction* $(y,\ z)$ *arbitrary*: $y\ z$ *rule*: *converse-rtranclp-induct*)
case *base*
show *?case*
by *simp*
next
case (*step w*)
thus *?case*
by *force*
qed

assume $x \in \{x'.\ R^{**}\ (\mathcal{M},\ \mathcal{C})\ x'\}$ **and** $y \in \{x'.\ R^{**}\ (\mathcal{M},\ \mathcal{C})\ x'\}$
hence
(*ord-res-5 N*)** $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})\ (U_{er},\ \mathcal{F},\ \mathcal{M}_x,\ \mathcal{C}_x)$ **and**
(*ord-res-5 N*)** $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})\ (U_{er},\ \mathcal{F},\ \mathcal{M}_y,\ \mathcal{C}_y)$
unfolding *atomize-conj mem-Collect-eq R-def x-def y-def*
by (*auto intro*: *rtranclp-with-constsD*)
hence
x-invars: *ord-res-5-invars N* $(U_{er},\ \mathcal{F},\ \mathcal{M}_x,\ \mathcal{C}_x)$ **and**
y-invars: *ord-res-5-invars N* $(U_{er},\ \mathcal{F},\ \mathcal{M}_y,\ \mathcal{C}_y)$
using *invars* **by** (*metis rtranclp-ord-res-5-preserves-invars*)+

assume $R^{-1-1}\ y\ x$
hence *ord-res-5 N* $(U_{er},\ \mathcal{F},\ \mathcal{M}_x,\ \mathcal{C}_x)\ (U_{er},\ \mathcal{F},\ \mathcal{M}_y,\ \mathcal{C}_y)$
unfolding *conversep-iff R-def x-def y-def prod.case* **.**
thus $f\ y\ |\subset|\ f\ x$
proof (*cases N* $(U_{er},\ \mathcal{F},\ \mathcal{M}_x,\ \mathcal{C}_x)\ (U_{er},\ \mathcal{F},\ \mathcal{M}_y,\ \mathcal{C}_y)$)
case *step-hyps*: (*skip C*)

have $f\ y\ |\subseteq|\ \{|D\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D|\}$

proof (*cases* $\mathcal{C}_y$)
case *None*
thus *?thesis*
unfolding *f-def y-def prod.case* **by** *simp*
next
case $\mathcal{C}_y$*-def*: (*Some D*)

have *D-least*: *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*
using *step-hyps* $\mathcal{C}_y$*-def* **by** (*metis The-optional-eq-SomeD*)
hence *f-y-eq*: *f y* = {|*E* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *D* $\preceq_c$ *E*|}
unfolding *f-def y-def prod.case* $\mathcal{C}_y$*-def option.case linorder-cls.is-least-in-ffilter-iff*
by *auto*

show *?thesis*
unfolding *f-y-eq subset-ffilter*
using *D-least*
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *auto*
qed

also have ... |⊂| *finsert C* {|*D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* $\preceq_c$ *D*|}
by *auto*

also have ... = *f x*
unfolding *f-def x-def y-def prod.case step-hyps option.case* **by** *metis*

finally show *?thesis* **.**
next
case *step-hyps*: (*production C L*)

have *f y* |⊆| {|*D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* $\prec_c$ *D*|}
proof (*cases* $\mathcal{C}_y$)
case *None*
thus *?thesis*
unfolding *f-def y-def prod.case* **by** *simp*
next
case $\mathcal{C}_y$*-def*: (*Some D*)

have *D-least*: *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*
using *step-hyps* $\mathcal{C}_y$*-def* **by** (*metis The-optional-eq-SomeD*)
hence *f-y-eq*: *f y* = {|*E* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *D* $\preceq_c$ *E*|}
unfolding *f-def y-def prod.case* $\mathcal{C}_y$*-def option.case linorder-cls.is-least-in-ffilter-iff*
by *auto*

show *?thesis*
unfolding *f-y-eq subset-ffilter*
using *D-least*

unfolding *linorder-cls.is-least-in-ffilter-iff*
by *auto*
qed

also have ... $|\subset|$ *finsert C* $\{|D \mid\in|$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup|\ U_{er}).\ C \preceq_c D|\}$
by *auto*

also have ... $= f\ x$
unfolding *f-def x-def y-def prod.case step-hyps option.case* **by** *metis*

finally show *?thesis* **.**
next
case *step-hyps*: (*factoring C L*)

have $C \mid\in|$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup|\ U_{er})$
using *x-invars* **unfolding** ‹$\mathcal{C}_x$ = *Some C*›
by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)
hence $C \mid\notin|\ \mathcal{F}$
using *step-hyps*(*3*,*4*,*5*)
by (*smt* (*verit*, *ccfv-SIG*) *fimage-iff iefac-def literal.collapse*(*1*)
ex1-efac-eq-factoring-chain ex-ground-factoringI)
moreover have $C \mid\in|\ \mathcal{F}$
using ‹$\mathcal{F}$ = *finsert C* $\mathcal{F}$› **by** *auto*
ultimately have *False*
by *contradiction*
thus *?thesis* **..**
next
case *step-hyps*: (*resolution C L D*)

have *D-productive*: *atm-of L* $\in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup| U_{er})$)) *D*
using *x-invars step-hyps*
by (*metis ord-res-5-invars-def atoms-in-model-were-produced-def*)

hence $\exists DC.$ *ground-resolution D C DC*
unfolding *ground-resolution-def*
using *step-hyps*
by (*metis Neg-atm-of-iff ord-res.mem-productionE linorder-cls.le-less-linear linorder-lit.is-maximal-in-mset-iff linorder-trm.dual-order.order-iff-strict linorder-trm.not-less ord-res.less-trm-if-neg ex-ground-resolutionI*)

hence *eres D C* $\neq$ *C*
unfolding *eres-ident-iff* **by** *metis*

have *eres D C* $\mid\notin|\ U_{er}$
proof (*rule notI*)
assume *eres D C* $\mid\in|\ U_{er}$
hence *iefac* $\mathcal{F}$ (*eres D C*) $\mid\in|$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup|\ U_{er})$
by *simp*

moreover have *iefac* $\mathcal{F}$ (*eres* D C) $\prec_c$ C
proof −
have *iefac* $\mathcal{F}$ C $\prec_c$ D **if** C $\prec_c$ D **for** $\mathcal{F}$ C D
proof (*cases* C |∈| $\mathcal{F}$)
case *True*
hence *iefac* $\mathcal{F}$ C = *efac* C
by (*simp add*: *iefac-def*)
also have ... $\preceq_c$ C
by (*metis efac-subset subset-implies-reflclp-multp*)
also have ... $\prec_c$ D
using *that* **.**
finally show *?thesis* **.**
next
case *False*
thus *?thesis*
using *that* **by** (*simp add*: *iefac-def*)
qed

moreover have *eres* D C $\prec_c$ C
proof −
have *eres* D C $\preceq_c$ C
using *eres-le* **by** *metis*
thus *eres* D C $\prec_c$ C
using ‹*eres* D C ≠ C› **by** *order*
qed

ultimately show *?thesis*
by *metis*
qed

ultimately have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) (*iefac* $\mathcal{F}$ (*eres* D C)) ⊨ *iefac* $\mathcal{F}$ (*eres* D C)
using *x-invars* **unfolding** ‹$\mathcal{C}_x$ = *Some* C›
unfolding *ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def*
by *fast*
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊨ *iefac* $\mathcal{F}$ (*eres* D C)
using *ord-res.entailed-clause-stays-entailed′* ‹*iefac* $\mathcal{F}$ (*eres* D C) $\prec_c$ C› **by** *metis*
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊨ *eres* D C
proof (*rule true-cls-iff-set-mset-eq*[*THEN iffD1*, *rotated*])
show *set-mset* (*iefac* $\mathcal{F}$ (*eres* D C)) = *set-mset* (*eres* D C)
using *iefac-def* **by** *auto*
qed
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊨ C
proof −
obtain m A D' C' **where**
ord-res.is-strictly-maximal-lit (*Pos* A) D **and**

```
      D-def: D = add-mset (Pos A) D' and
      C-def: C = replicate-mset (Suc m) (Neg A) + C' and
      Neg A ∉# C' and
      eres-D-C-eq: eres D C = repeat-mset (Suc m) D' + C'
      using ‹eres D C ≠ C›[THEN eres-not-identD] by metis

    hence
      atm-of L = A and
      D-in: D |∈| iefac F |‘| (N |∪| U_er) and
      D-false: ¬ ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) D ⊨ D
      unfolding atomize-conj
      using D-productive
      unfolding ord-res.production-unfold mem-Collect-eq
    by (metis linorder-lit.Uniq-is-greatest-in-mset literal.inject(1) the1-equality')

    have D'-false: ¬ ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) D ⊨ D'
      using D-false D-def by fastforce

    have D ≺_c C
    using ‹∃DC. ground-resolution D C DC› left-premise-lt-right-premise-if-ground-resolution
by blast

    let ?I = ord-res.interp (fset (iefac F |‘| (N |∪| U_er))) C

    assume ?I ⊨ eres D C
    hence ?I ⊨ D' ∨ ?I ⊨ C'
      unfolding eres-D-C-eq true-cls-union true-cls-repeat-mset-Suc .

    moreover have ¬ ?I ⊨ D'
      using ‹D ≺_c C›
      by (smt (verit) D-def D-productive ‹ord-res.is-strictly-maximal-lit (Pos
A) D›
        diff-single-eq-union ord-res.mem-productionE linorder-lit.is-greatest-in-mset-iff
          ord-res.Uniq-striclty-maximal-lit-in-ground-cls
          ord-res.false-cls-if-productive-production ord-res-5-invars-def
        next-clause-in-factorized-clause-def step-hyps(1) the1-equality' x-invars)

    ultimately show ?I ⊨ C
      by (simp add: C-def)
  qed
  hence dom M_x ⊨ C
    using x-invars ‹C_x = Some C›
    by (metis model-eq-interp-upto-next-clause-def ord-res-5-invars-def)
  thus False
    using ‹¬ dom M_x ⊨ C› by contradiction
qed
hence False
  using ‹U_er = finsert (eres D C) U_er› by auto
thus ?thesis ..
```

```
    qed
  qed

  then obtain M' C' where R** (M, C) (M', C') and ∄z. R (M', C') z
    using ex-terminating-rtranclp-strong by (metis surj-pair)

  show ?thesis
  proof (intro exI conjI)
    show (ord-res-5 N)** (U_er, F, M, C) (U_er, F, M', C')
      using ‹R** (M, C) (M', C')›
       by (induction (M, C) arbitrary: M C rule: converse-rtranclp-induct) (auto
simp: R-def)
  next
    show ∄M'' C''. ord-res-5 N (U_er, F, M', C') (U_er, F, M'', C'')
      using ‹∄z. R (M', C') z›
      by (simp add: R-def)
  qed
qed

lemma MAGIC2:
  assumes invars: ord-res-5-invars N (U_er, F, M, Some C)
  assumes C ≠ {#}
  shows ∃s'. ord-res-5 N (U_er, F, M, Some C) s'
proof (cases (dom M) ⊨ C)
  case C-true: True
  thus ?thesis
    using ord-res-5.skip by metis
next
  case C-false: False
  obtain L where L-max: linorder-lit.is-maximal-in-mset C L
    using ‹C ≠ {#}› linorder-lit.ex-maximal-in-mset by metis

  show ?thesis
  proof (cases L)
    case (Pos A)
    hence L-pos: is-pos L
      by simp

    show ?thesis
    proof (cases linorder-lit.is-greatest-in-mset C L)
      case L-greatest: True
      thus ?thesis
        using C-false L-max L-pos ord-res-5.production by metis
    next
      case L-not-greatest: False
      thus ?thesis
        using C-false L-max L-pos ord-res-5.factoring by metis
    qed
  next
```

case (*Neg A*)
hence *L-neg*: *is-neg L*
by *simp*

have *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *C*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using *invars* **unfolding** *ord-res-5-invars-def next-clause-in-factorized-clause-def*
by *metis*
next
have *dom* $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
using *invars* **unfolding** *ord-res-5-invars-def model-eq-interp-upto-next-clause-def*
by *metis*

moreover have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* = {}
proof −
have ∄*L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*
using *L-max L-neg*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
thus *?thesis*
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

ultimately show ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⊫ *C*
using *C-false model-eq-interp-upto-next-clause-def* **by** *simp*
next
fix *D*
assume
D |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
D ≠ *C* **and**
¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D* ⊫ *D*

moreover have ∀ *B* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *B* ≺$_c$ *C* ⟶
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *B* ⊫ *B*
using *invars*
unfolding *ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def*
by *simp*

ultimately show *C* ≺$_c$ *D*
by *force*
qed
then obtain *D* **where** *D*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
D ≺$_c$ *C* **and**
ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**
D-prod: *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D* = {*A*}
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
L-max[*unfolded Neg*] **by** *metis*

```
    have M (atm-of L) = Some D
      using invars
      unfolding ord-res-5-invars-def all-produced-atoms-in-model-def
      by (metis Neg ‹D ≺c C› D-prod insertI1 literal.sel(2))

    thus ?thesis
      using ord-res-5.resolution C-false L-max L-neg by metis
  qed
qed

lemma MAGIC3:
  assumes invars: ord-res-5-invars N (U_er, F, M, C) and
    steps: (ord-res-5 N)** (U_er, F, M, C) (U_er, F, M', C') and
    no-more-steps: (∄M'' C''. ord-res-5 N (U_er, F, M', C') (U_er, F, M'', C''))
  shows (∀ C. C' = Some C ⟷ is-least-false-clause (iefac F |`| (N |∪| U_er)) C)
proof −
  have invars': ord-res-5-invars N (U_er, F, M', C')
    using steps invars rtranclp-ord-res-5-preserves-invars by metis

  show ?thesis
  proof (cases C')
    case None

    moreover have ∄C. is-least-false-clause (iefac F |`| (N |∪| U_er)) C
    proof (rule notI, elim exE)
      fix C

      have all-smaller-clauses-true-wrt-respective-Interp N (U_er, F, M', C')
        using invars' unfolding ord-res-5-invars-def by metis
      hence (∀ C|∈|iefac F |`| (N |∪| U_er). ord-res-Interp (fset (iefac F |`| (N |∪|
U_er))) C ⊨ C)
        by (simp add: ‹C' = None› all-smaller-clauses-true-wrt-respective-Interp-def)

      moreover assume is-least-false-clause (iefac F |`| (N |∪| U_er)) C

      ultimately show False
        unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff by metis
    qed

    ultimately show ?thesis
      by simp
  next
    case (Some D)

    moreover have is-least-false-clause (iefac F |`| (N |∪| U_er)) D
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      show D |∈| iefac F |`| (N |∪| U_er)
```

using *invars′* ‹$\mathcal{C}' = Some\ D$›
unfolding *ord-res-5-invars-def next-clause-in-factorized-clause-def*
by *metis*
next
have $D \neq \{\#\} \Longrightarrow \exists\, s'.$ *ord-res-5* N $(U_{er}, \mathcal{F}, \mathcal{M}', Some\ D)$ s'
using *MAGIC2 invars′* ‹$\mathcal{C}' = Some\ D$› **by** *metis*

thus ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⊫ D
by (*smt* (*verit*) *Pair-inject Un-empty-right Uniq-D calculation empty-iff invars′*
linorder-lit.Uniq-is-maximal-in-mset
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset no-more-steps option.inject
ord-res-5.cases set-mset-empty model-eq-interp-upto-next-clause-def ord-res-5-invars-def
true-cls-def unproductive-if-nex-strictly-maximal-pos-lit)
next
fix E
assume
E |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
$E \neq D$ **and**
¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E ⊫ E

moreover have ∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $C \prec_c D \longrightarrow$
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊫ C
using *invars′* ‹$\mathcal{C}' = Some\ D$›
unfolding *ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def*
by *simp*

ultimately show $D \prec_c E$
by *force*
qed

ultimately show *?thesis*
by (*metis Uniq-D Uniq-is-least-false-clause option.inject*)
qed
qed

lemma *ord-res-5-construct-model-upto-least-false-clause*:
assumes *invars*: *ord-res-5-invars* N $(U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$
shows ∃ $\mathcal{M}'$ $\mathcal{C}'$. (*ord-res-5* N)** $(U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$ $(U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}')$ ∧
(∀ C. $\mathcal{C}' = Some\ C \longleftrightarrow$ *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) C)
using *MAGIC1*[*OF invars*] *MAGIC3*[*OF invars*] **by** *metis*

end

end
theory *ORD-RES-6*
imports

ORD-RES-5
begin

21 ORD-RES-6 (model backjump)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-6* **where**
skip:
$(dom\ \mathcal{M}) \models C \Longrightarrow$
$\mathcal{C}' = The\text{-}optional\ (linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset\ \{|D\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D|\}) \Longrightarrow$
$ord\text{-}res\text{-}6\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}')$ |

production:
$\neg\ (dom\ \mathcal{M}) \models C \Longrightarrow$
$linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$is\text{-}pos\ L \Longrightarrow$
$linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$\mathcal{M}' = \mathcal{M}(atm\text{-}of\ L := Some\ C) \Longrightarrow$
$\mathcal{C}' = The\text{-}optional\ (linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset\ \{|D\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D|\}) \Longrightarrow$
$ord\text{-}res\text{-}6\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}')$ |

factoring:
$\neg\ (dom\ \mathcal{M}) \models C \Longrightarrow$
$linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$is\text{-}pos\ L \Longrightarrow$
$\neg\ linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$\mathcal{F}' = finsert\ C\ \mathcal{F} \Longrightarrow$
$ord\text{-}res\text{-}6\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}', \mathcal{M}, Some\ (efac\ C))$ |

resolution-bot:
$\neg\ (dom\ \mathcal{M}) \models C \Longrightarrow$
$linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$is\text{-}neg\ L \Longrightarrow$
$\mathcal{M}\ (atm\text{-}of\ L) = Some\ D \Longrightarrow$
$U_{er}' = finsert\ (eres\ D\ C)\ U_{er} \Longrightarrow$
$eres\ D\ C = \{\#\} \Longrightarrow$
$\mathcal{M}' = (\lambda\text{-}.\ None) \Longrightarrow$
$ord\text{-}res\text{-}6\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', Some\ \{\#\})$ |

resolution-pos:
$\neg\ (dom\ \mathcal{M}) \models C \Longrightarrow$
$linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ L \Longrightarrow$
$is\text{-}neg\ L \Longrightarrow$
$\mathcal{M}\ (atm\text{-}of\ L) = Some\ D \Longrightarrow$
$U_{er}' = finsert\ (eres\ D\ C)\ U_{er} \Longrightarrow$
$eres\ D\ C \neq \{\#\} \Longrightarrow$

$\mathcal{M}' = \mathit{restrict\text{-}map}\ \mathcal{M}\ \{A.\ A \prec_t \mathit{atm\text{-}of}\ K\} \Longrightarrow$
$\mathit{linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset}\ (\mathit{eres}\ D\ C)\ K \Longrightarrow$
$\mathit{is\text{-}pos}\ K \Longrightarrow$
$\mathit{ord\text{-}res\text{-}6}\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathit{Some}\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', \mathit{Some}\ (\mathit{eres}\ D\ C))\ |$

resolution-neg:
$\neg\ (\mathit{dom}\ \mathcal{M}) \models C \Longrightarrow$
$\mathit{linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset}\ C\ L \Longrightarrow$
$\mathit{is\text{-}neg}\ L \Longrightarrow$
$\mathcal{M}\ (\mathit{atm\text{-}of}\ L) = \mathit{Some}\ D \Longrightarrow$
$U_{er}' = \mathit{finsert}\ (\mathit{eres}\ D\ C)\ U_{er} \Longrightarrow$
$\mathit{eres}\ D\ C \neq \{\#\} \Longrightarrow$
$\mathcal{M}' = \mathit{restrict\text{-}map}\ \mathcal{M}\ \{A.\ A \prec_t \mathit{atm\text{-}of}\ K\} \Longrightarrow$
$\mathit{linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset}\ (\mathit{eres}\ D\ C)\ K \Longrightarrow$
$\mathit{is\text{-}neg}\ K \Longrightarrow$
$\mathcal{M}\ (\mathit{atm\text{-}of}\ K) = \mathit{Some}\ E \Longrightarrow$
$\mathit{ord\text{-}res\text{-}6}\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathit{Some}\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', \mathit{Some}\ E)$

inductive *ord-res-6-step* **where**
$\mathit{ord\text{-}res\text{-}6}\ N\ s\ s' \Longrightarrow \mathit{ord\text{-}res\text{-}6\text{-}step}\ (N, s)\ (N, s')$

lemma *tranclp-ord-res-6-step-if-tranclp-ord-res-6*:
$(\mathit{ord\text{-}res\text{-}6}\ N)^{++}\ s\ s' \Longrightarrow \mathit{ord\text{-}res\text{-}6\text{-}step}^{++}\ (N, s)\ (N, s')$
by (*induction* s' *rule*: *tranclp-induct*)
(*auto intro*: *ord-res-6-step.intros tranclp.intros*)

lemma *right-unique-ord-res-6*: *right-unique* (*ord-res-6* N)
proof (*rule right-uniqueI*)
fix $s\ s'\ s''$
assume *step1*: *ord-res-6* $N\ s\ s'$ **and** *step2*: *ord-res-6* $N\ s\ s''$
thus $s' = s''$
by (*smt* (*verit*) *Pair-inject linorder-lit.Uniq-is-maximal-in-mset option.inject ord-res-6.cases*
the1-equality')
qed

lemma *right-unique-ord-res-6-step*: *right-unique ord-res-6-step*
proof (*rule right-uniqueI*)
fix $x\ y\ z$
show *ord-res-6-step* $x\ y \Longrightarrow$ *ord-res-6-step* $x\ z \Longrightarrow y = z$
using *right-unique-ord-res-6*[*THEN right-uniqueD*]
by (*elim ord-res-6-step.cases*) (*metis prod.inject*)
qed

inductive *ord-res-6-final* **where**
model-found:
ord-res-6-final $(N, U_{er}, \mathcal{F}, \mathcal{M}, \mathit{None})\ |$

contradiction-found:

ord-res-6-final (N, U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* $\{\#\}$)

sublocale *ord-res-6-semantics*: *semantics* **where**
step = *ord-res-6-step* **and**
final = *ord-res-6-final*
proof *unfold-locales*
fix S :: *'f gclause fset* × *'f gclause fset* × *'f gclause fset* ×
(*'f gterm* ⇒ *'f gclause option*) × *'f gclause option*

obtain
N U_{er} $\mathcal{F}$ :: *'f gterm clause fset* **and**
$\mathcal{M}$ :: *'f gterm* ⇒ *'f gclause option* **and**
$\mathcal{C}$ :: *'f gclause option* **where**
S-def: $S = (N, (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}))$
by (*metis prod.exhaust*)

assume *ord-res-6-final S*
hence $\mathcal{C} = None \vee \mathcal{C} = Some\ \{\#\}$
by (*simp add*: *S-def ord-res-6-final.simps*)
hence $\nexists x.$ *ord-res-6 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) *x*
by (*auto simp*: *linorder-lit.is-maximal-in-mset-iff elim*: *ord-res-6.cases*)
thus *finished ord-res-6-step S*
by (*simp add*: *S-def finished-def ord-res-6-step.simps*)
qed

lemma *ord-res-6-preserves-invars*:
assumes *step*: *ord-res-6 N s s'* **and** *invars*: *ord-res-5-invars N s*
shows *ord-res-5-invars N s'*
using *step*
proof (*cases N s s' rule*: *ord-res-6.cases*)
case (*skip* $\mathcal{M}$ C $\mathcal{C}'$ $\mathcal{F}$ U_{er})
thus *?thesis*
by (*metis invars ord-res-5-preserves-invars ord-res-5.skip*)
next
case (*production* $\mathcal{M}$ C L $\mathcal{M}'$ $\mathcal{C}'$ $\mathcal{F}$ U_{er})
thus *?thesis*
by (*metis invars ord-res-5.production ord-res-5-preserves-invars*)
next
case *step-hyps*: (*factoring* $\mathcal{M}$ C L $\mathcal{F}'$ $\mathcal{F}$ U_{er})

have *efac C* ≠ *C*
by (*metis ex1-efac-eq-factoring-chain is-pos-def ex-ground-factoringI step-hyps*(*4*,*5*,*6*))
moreover have *efac C* $\preceq_c$ *C*
by (*metis efac-subset subset-implies-reflclp-multp*)
ultimately have *efac C* $\prec_c$ *C*
by *order*

show *?thesis*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def*

proof (*intro conjI*)
have $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *invars step-hyps*
by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)
hence $C \mid\in\mid N \mid\cup\mid U_{er}$
using ‹*efac* $C \neq C$›
by (*smt* (*verit*, *best*) *fimage-iff iefac-def ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)
hence *efac* $C \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$
using *step-hyps*(3−)
using *iefac-def* **by** *auto*
thus *next-clause-in-factorized-clause* N (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac* C))
unfolding *next-clause-in-factorized-clause-def* **by** *simp*

have $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$
using *invars*
unfolding *step-hyps*(1,2) *ord-res-5-invars-def implicitly-factorized-clauses-subset-def*
by *metis*

hence $\mathcal{F}' \mid\subseteq\mid N \mid\cup\mid U_{er}$
using ‹$C \mid\in\mid N \mid\cup\mid U_{er}$› ‹$\mathcal{F}' =$ *finsert* C $\mathcal{F}$› **by** *simp*

thus *implicitly-factorized-clauses-subset* N (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac* C))
unfolding *implicitly-factorized-clauses-subset-def* **by** *simp*

have *dom-*$\mathcal{M}$*-eq*: *dom* $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) C
using *invars step-hyps*
by (*simp add*: *model-eq-interp-upto-next-clause-def ord-res-5-invars-def*)

have *efac* $C \mid\notin\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
proof (*rule notI*)
assume *efac* $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
show *False*
proof (*cases atm-of* $L \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) (*efac* C))
assume *atm-of* $L \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) (*efac* C)
hence *atm-of* $L \in$ *ord-res.interp* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) C
using ‹*efac* $C \prec_c C$› *ord-res.production-subset-if-less-cls* **by** *blast*
hence *dom* $\mathcal{M} \models C$
using *step-hyps*
by (*metis dom-*$\mathcal{M}$*-eq linorder-lit.is-maximal-in-mset-iff literal.collapse*(1) *pos-literal-in-imp-true-cls*)
thus *False*
using ‹$\neg$ *dom* $\mathcal{M} \models C$› **by** *contradiction*
next
assume *atm-of* $L \notin$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) (*efac* C)
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) (*efac* C) $\models$ *efac* C

unfolding *ord-res.production-unfold mem-Collect-eq*
using ‹*efac C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})›
by (*metis Pos-atm-of-iff* ‹*efac C* ≠ *C*› *insert-DiffM*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.is-maximal-in-mset-iff linorder-lit.neqE
obtains-positive-greatest-lit-if-efac-not-ident set-mset-efac step-hyps(4))
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⊨ *efac C*
using ‹*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› ‹*efac C* ≺$_c$ *C*› ‹*efac C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})›
ord-res.lift-interp-entails **by** *metis*
hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C* ⊨ *C*
by (*simp add*: *true-cls-def*)
hence *dom* $\mathcal{M}$ ⊨ *C*
using *dom-*$\mathcal{M}$*-eq* **by** *argo*
thus *False*
using ‹¬ *dom* $\mathcal{M}$ ⊨ *C*› **by** *contradiction*
qed
qed

have *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}) = *finsert* (*iefac* $\mathcal{F}'$ *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) − {|*C*|}
proof (*intro fsubset-antisym fsubsetI*)
fix *x* :: *'f gclause*
assume *x* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
thus *x* |∈| *finsert* (*iefac* $\mathcal{F}'$ *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) |−| {|*C*|}
by (*smt* (*verit*) ‹*efac C* ≠ *C*› *factorizable-if-neq-efac fimage-iff finsert-iff fminusI*
fsingletonE iefac-def ex1-efac-eq-factoring-chain step-hyps(7))
next
fix *x* :: *'f gclause*
assume *x-in*: *x* |∈| *finsert* (*iefac* $\mathcal{F}'$ *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) |−| {|*C*|}
hence *x* = *iefac* $\mathcal{F}'$ *C* ∨ *x* |∈| (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) |−| {|*C*|}
by *blast*
thus *x* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
proof (*elim disjE*)
assume *x* = *iefac* $\mathcal{F}'$ *C*
thus *x* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
using ‹*C* |∈| *N* |∪| U_{er}› **by** *blast*
next
assume *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) |−| {|*C*|}
hence *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and** *x* ≠ *C*
by *simp-all*
then obtain *x'* **where** *x'* |∈| *N* |∪| U_{er} **and** *x* = *iefac* $\mathcal{F}$ *x'*
by *auto*
moreover have *x'* ≠ *C*
using ‹*x* ≠ *C*› ‹*x* = *iefac* $\mathcal{F}$ *x'*›
by (*metis* ‹*efac C* |∉| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› ‹*x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› *iefac-def*)
ultimately show *x* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})

```
      using iefac-def step-hyps(7) by simp
    qed
  qed

  have ord-res.interp (fset (iefac F' |`| (N |∪| U_er))) (efac C) =
    ord-res.interp (fset (iefac F |`| (N |∪| U_er))) (efac C)
  proof (rule ord-res.interp-swap-clause-set)
    show {D. D |∈| iefac F' |`| (N |∪| U_er) ∧ D ≺_c efac C} =
      {D. D |∈| iefac F |`| (N |∪| U_er) ∧ D ≺_c efac C}
      unfolding ‹iefac F' |`| (N |∪| U_er) = finsert (iefac F' C) (iefac F |`| (N
|∪| U_er)) − {|C|}›
      using ‹efac C ≺_c C›
      using iefac-def by force
  qed

  also have ... = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) C
  proof −
    have ∀ x |∈| iefac F |`| (N |∪| U_er). x ≺_c C ⟶
      ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) x ⟹ x
      using invars[unfolded ord-res-5-invars-def step-hyps(1)
          all-smaller-clauses-true-wrt-respective-Interp-def, simplified]
      by simp
    then have ord-res.production (fset (iefac F |`| (N |∪| U_er))) x = {}
      if x |∈| iefac F |`| (N |∪| U_er) and efac C ≺_c x and x ≺_c C for x
    proof −
      have x ⪯_l y ∧ y ⪯_l z
        if X ⪯_c Y and Y ⪯_c Z and
          linorder-lit.is-maximal-in-mset X x and
          linorder-lit.is-maximal-in-mset Y y and
          linorder-lit.is-maximal-in-mset Z z
        for x y z X Y Z
        using that
        unfolding linorder-lit.is-maximal-in-mset-iff
        by (metis ord-res.asymp-less-lit ord-res.transp-less-lit linorder-cls.leD
              linorder-lit.leI linorder-lit.multp_HO-if-maximal-less-that-maximal
multp-eq-multp_HO
            that(3) that(4) that(5))

      hence y = x
        if X ⪯_c Y and Y ⪯_c Z and
          linorder-lit.is-maximal-in-mset X x and
          linorder-lit.is-maximal-in-mset Y y and
          linorder-lit.is-maximal-in-mset Z x
        for x y X Y Z
        using that
        by (metis linorder-lit.order.ordering-axioms ordering.antisym)

      hence y = x
        if X ≺_c Y and Y ≺_c Z and
```

linorder-lit.is-maximal-in-mset X x **and**
linorder-lit.is-maximal-in-mset Y y **and**
linorder-lit.is-maximal-in-mset Z x
for *x y X Y Z*
using *that* **by** *blast*

hence $K = L$
if *efac C* $\prec_c$ *x* **and** *x* $\prec_c$ *C* **and**
linorder-lit.is-maximal-in-mset (*efac C*) *L* **and**
linorder-lit.is-maximal-in-mset x K **and**
linorder-lit.is-maximal-in-mset C L
for *K*
using *that* **by** *metis*

hence $K = L$
if *linorder-lit.is-maximal-in-mset x K*
for *K*
using *that*
using ‹*ord-res.is-maximal-lit L C*›
using ‹*efac C* $\prec_c$ *x*› ‹*x* $\prec_c$ *C*› *ex1-efac-eq-factoring-chain*
ord-res.ground-factorings-preserves-maximal-literal **by** *blast*

hence *ord-res.is-maximal-lit L x*
by (*metis linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls that*(*2*))

have $\nexists A.\ A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *x*
proof (*rule notI* , *elim exE*)
fix *A* :: *'f gterm*
assume *A* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *x*
then obtain *x'* **where**
x |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
x = *add-mset* (*Pos A*) *x'* **and**
ord-res.is-strictly-maximal-lit (*Pos A*) *x* **and**
¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *x* ⊨ *x*
by (*metis ord-res.mem-productionE*)

have *A* ∈ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
using ‹*A* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *x*›
ord-res.production-subset-if-less-cls that(*3*) **by** *fastforce*

moreover have *L* = *Pos A*
using ‹*ord-res.is-maximal-lit L x*› ‹*ord-res.is-strictly-maximal-lit* (*Pos A*) *x*›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)

moreover have *L* ∈# *C*
using *step-hyps linorder-lit.is-maximal-in-mset-iff* **by** *metis*

ultimately have *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $C \Vdash C$
by *auto*

hence *dom* $\mathcal{M} \Vdash C$
using *dom-*$\mathcal{M}$*-eq* **by** *argo*

thus *False*
using ‹¬ *dom* $\mathcal{M} \Vdash C$› **by** *contradiction*
qed

thus *?thesis*
by *simp*
qed

moreover have {x. x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $x \prec_c C$} =
{x. x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $x \prec_c$ *efac* C} ∪
{x. x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ *efac* $C \prec_c x \wedge x \prec_c C$}
proof −
have {$w \in NN$. $w \prec_c z$} = {$w \in NN$. $w \prec_c x$} ∪ {$y \in NN$. $x \prec_c y \wedge y \prec_c z$}
if $x \notin NN$ **and** $x \prec_c z$ **for** NN x z
proof −
have {$w \in NN$. $w \prec_c z$} = {$w \in NN$. $w \preceq_c x \vee x \prec_c w \wedge w \prec_c z$}
using *that*(*2*) **by** *auto*
also have ... = {$w \in NN$. $w \prec_c x \vee x \prec_c w \wedge w \prec_c z$}
using *that*(*1*) **by** *auto*
also have ... = {$w \in NN$. $w \prec_c x$} ∪ {$y \in NN$. $x \prec_c y \wedge y \prec_c z$}
by *auto*
finally show *?thesis* **.**
qed
thus *?thesis*
using ‹*efac* $C \prec_c C$› ‹*efac* C |∉| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})› **by** (*simp only*:)
qed

ultimately show *?thesis*
unfolding *ord-res.interp-def* **by** *auto*
qed

finally show *model-eq-interp-upto-next-clause* N (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac* C))
unfolding *model-eq-interp-upto-next-clause-def*
using *dom-*$\mathcal{M}$*-eq*
by *simp*

have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}))) $x \Vdash x$
if x |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) **and** $x \prec_c$ *efac* C **for** x
proof −
have x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
using *that*

by (*metis* ‹*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) = *finsert* (*iefac* $\mathcal{F}'$ C) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) |−| {|C|}›
finsert-iff fminusE iefac-def linorder-cls.neq-iff)

moreover have $x \prec_c C$
using ‹$x \prec_c$ *efac* C› ‹*efac* $C \prec_c C$› **by** *order*

ultimately have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $x \Vdash x$
using *invars*
unfolding *ord-res-5-invars-def*
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def step-hyps*(*1*,*2*)
by *blast*

moreover have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) x =
ord-res-Interp (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}))) x
proof (*rule ord-res.Interp-swap-clause-set*)
show {D. D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ D x} =
{D. D |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ D x}
unfolding ‹*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) = *finsert* (*iefac* $\mathcal{F}'$ C) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) |−| {|C|}›
using ‹$x \prec_c$ *efac* C› ‹$x \prec_c C$›
by (*metis* (*no-types*, *opaque-lifting*) *finsertCI finsertE fminusE fminusI fsingleton-iff*
iefac-def linorder-cls.less-le-not-le)
qed

ultimately show *?thesis*
by *argo*
qed

thus *all-smaller-clauses-true-wrt-respective-Interp* N (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac* C))
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def* **by** *blast*

have *linorder-lit.is-greatest-in-mset* (*efac* C) L
using ‹*linorder-lit.is-maximal-in-mset* C L›
by (*metis* ‹*efac* $C \neq C$› *ex1-efac-eq-factoring-chain linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one
ord-res.ground-factorings-preserves-maximal-literal
obtains-positive-greatest-lit-if-efac-not-ident the1-equality')

have $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}))) D
if $\mathcal{M}$ A = *Some* D **for** A D
proof −
have $A \in$ *dom* $\mathcal{M}$
using ‹$\mathcal{M}$ A = *Some* D› **by** *blast*

hence $A \in$ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C
using *dom-*$\mathcal{M}$*-eq* **by** *argo*

have $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D
using *invars* ‹$\mathcal{M}$ A = *Some* D›
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
unfolding *atoms-in-model-were-produced-def*
by *simp*

hence *linorder-lit.is-greatest-in-mset* D (*Pos* A)
by (*metis ord-res.mem-productionE*)

moreover have *Pos* $A \prec_l L$
using ‹$A \in$ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C›
by (*smt* (*verit*, *del-insts*) *UN-E* ‹$A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D›
calculation dom-$\mathcal{M}$*-eq ord-res.interp-def ord-res.less-lit-simps*(*1*)
ord-res.totalp-less-lit linorder-cls.less-trans
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.is-maximal-in-mset-iff linorder-lit.less-irefl
linorder-lit.multp-if-maximal-less-that-maximal mem-Collect-eq
ord-res.less-trm-iff-less-cls-if-mem-production
pos-literal-in-imp-true-cls step-hyps(*3*) *step-hyps*(*4*) *totalpD*)

ultimately have $D \prec_c$ *efac* C
using ‹*linorder-lit.is-greatest-in-mset* (*efac* C) L›
by (*metis ord-res.asymp-less-lit ord-res.transp-less-lit*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.multp$_{HO}$*-if-maximal-less-that-maximal multp-eq-multp*$_{HO}$)

have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D =
ord-res.production (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}))) D
proof (*rule ord-res.production-swap-clause-set*)
have $D \prec_c C$
using ‹$D \prec_c$ *efac* C› ‹*efac* $C \prec_c C$› **by** *order*
thus {*Da*. *Da* |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ *Da* D} =
{*Da*. *Da* |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ *Da* D}
using ‹$D \prec_c$ *efac* C›
unfolding ‹*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}) = *finsert* (*iefac* $\mathcal{F}'$ C) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) |−| {|C|}›
by (*metis* (*no-types*, *opaque-lifting*) *finsertE finsertI2 fminus-iff fsingleton-iff*
iefac-def linorder-cls.leD)
qed

thus *?thesis*
using ‹$A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D› **by** *argo*
qed

thus *atoms-in-model-were-produced* N (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac* C))
unfolding *atoms-in-model-were-produced-def* **by** *simp*

```
    have M A = Some x
      if x ≺c efac C and A ∈ ord-res.production (fset (iefac F' |`| (N |∪| Uer))) x
      for x A
    proof −
      have x ≺c C
        using ‹x ≺c efac C› ‹efac C ≺c C› by order

      moreover have A ∈ ord-res.production (fset (iefac F |`| (N |∪| Uer))) x
      proof −
        have ord-res.production (fset (iefac F' |`| (N |∪| Uer))) x =
          ord-res.production (fset (iefac F |`| (N |∪| Uer))) x
        proof (rule ord-res.production-swap-clause-set)
         show {D. D |∈| iefac F' |`| (N |∪| Uer) ∧ (≺c)== D x} = {D. D |∈| iefac
F |`| (N |∪| Uer) ∧ (≺c)== D x}
           using ‹x ≺c efac C› ‹x ≺c C›
           unfolding ‹iefac F' |`| (N |∪| Uer) = finsert (iefac F' C) (iefac F |`|
(N |∪| Uer)) |−| {|C|}›
             by (metis (no-types, opaque-lifting) finsert-iff fminus-iff fsingleton-iff
iefac-def
              linorder-cls.dual-order.strict-iff-not)
        qed

        thus ?thesis
          using ‹A ∈ ord-res.production (fset (iefac F' |`| (N |∪| Uer))) x› by argo
      qed

      ultimately show ?thesis
        using invars
        unfolding ord-res-5-invars-def step-hyps(1,2)
        unfolding all-produced-atoms-in-model-def
        by simp
    qed

    thus all-produced-atoms-in-model N (Uer, F', M, Some (efac C))
      unfolding all-produced-atoms-in-model-def by simp
  qed
next
  case step-hyps: (resolution-bot M D L C Uer' Uer M' F)
  have F |⊆| N |∪| Uer
    using invars
   unfolding step-hyps(1,2) ord-res-5-invars-def implicitly-factorized-clauses-subset-def
    by metis

  hence F |⊆| N |∪| Uer'
    using step-hyps by blast

  moreover have linorder-cls.is-least-in-fset (iefac F |`| (N |∪| Uer')) {#}
    using step-hyps linorder-cls.is-least-in-fset-iff mempty-lesseq-cls by fastforce
```

ultimately show *?thesis*
using *step-hyps*
using *ord-res-5-invars-initial-state*
by (*metis ord-res-5-invars-initial-state*)
next
case *step-hyps*: (*resolution-pos* $\mathcal{M}$ *E L D* $U_{er}{}'$ U_{er} $\mathcal{M}'$ *K* $\mathcal{F}$)

hence
L-max: *ord-res.is-maximal-lit L E* **and**
L-neg: *is-neg L*
using *step-hyps* **by** *simp-all*

have $\mathcal{F}$*-subset*: $\mathcal{F}$ $|\subseteq|$ *N* $|\cup|$ U_{er}
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def implicitly-factorized-clauses-subset-def*
by *metis*

have *eres D E* $\neq$ *E*
using *step-hyps* **by** (*metis linorder-lit.Uniq-is-maximal-in-mset the1-equality'*)

moreover have *eres D E* $\preceq_c$ *E*
using *eres-le* **.**

ultimately have *eres D E* $\prec_c$ *E*
by *order*

have $\forall$ *F*. *is-least-false-clause* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) *F* $\longrightarrow$ *E* $\preceq_c$ *F*
using *invars*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
using *next-clause-lt-least-false-clause*[*of N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*)]
by *simp*

have *E-least-false*: *is-least-false-clause* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) *E*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *invars*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
by (*metis next-clause-in-factorized-clause-def*)
next
have $\neg$ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}))) *E* $\models$ *E*
using *invars*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
using ‹$\neg$ *dom* $\mathcal{M}$ $\models$ *E*› **by** (*metis model-eq-interp-upto-next-clause-def*)

moreover have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}))) *E* = {}
proof −
have $\nexists$ *L*. *is-pos L* $\wedge$ *ord-res.is-strictly-maximal-lit L E*
using ‹*ord-res.is-maximal-lit L E*› ‹*is-neg L*›

by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
thus *?thesis*
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

ultimately show ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *E* ⊨ *E*
by *simp*
next
fix *F*
assume *F-in*: *F* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and** *F* ≠ *E* **and**
F-false: ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *F* ⊨ *F*

have ¬ *F* ≺$_c$ *E*
using *invars*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def*
using *F-in F-false*
by (*metis option.inject*)

thus *E* ≺$_c$ *F*
using ‹*F* ≠ *E*› **by** *order*
qed

have *L-prod-by-D*: *atm-of L* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*
using *invars*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
by (*metis atoms-in-model-were-produced-def step-hyps*(*6*))

hence *D-prod*: *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D* ≠ {}
by (*metis empty-iff*)

have *ord-res.is-maximal-lit* (−*L*) *D*
using *L-prod-by-D L-neg*
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.collapse*(*2*)
ord-res.mem-productionE uminus-Neg)

moreover have −*L* ≺$_l$ *L*
using *L-neg*
by (*metis Neg-atm-of-iff atm-of-uminus linorder-lit.not-less-iff-gr-or-eq*
linorder-trm.less-imp-not-eq literal.collapse(*1*) *ord-res.less-lit-simps*(*4*) *uminus-not-id*)

ultimately have *D* ≺$_c$ *E*
using *L-max linorder-lit.multp-if-maximal-less-that-maximal* **by** *metis*

have *eres D E* |∉| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
proof (*rule notI*)

assume *eres D E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})$
moreover have $\neg$ $(E \prec_c$ *eres D E*$)$
using ‹*eres D E* $\prec_c$ *E*› **by** *order*
ultimately have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})))$ (*eres D E*) $\models$ *eres D E*
using *E-least-false* ‹*eres D E* $\neq$ *E*›
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *metis*
then show *False*
by (*metis* (*no-types*, *lifting*) *D-prod E-least-false clause-true-if-resolved-true ex1-eres-eq-full-run-ground-resolution full-run-def is-least-false-clause-def linorder-cls.is-least-in-fset-ffilterD*(*2*) *rtranclpD*)
qed

moreover have *efac* (*eres D E*) $|\notin|$ *iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})$
proof (*rule notI*)
have *efac* (*eres D E*) $\preceq_c$ *eres D E*
by (*meson efac-subset subset-implies-reflclp-multp*)
hence $\neg$ $(E \prec_c$ *efac* (*eres D E*))
using ‹*eres D E* $\prec_c$ *E*› **by** *order*
moreover assume *efac* (*eres D E*) $|\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})$
moreover have *efac* (*eres D E*) $\neq$ *E*
by (*metis* ‹*eres D E* $\prec_c$ *E*› *efac-properties-if-not-ident*(*1*) *linorder-cls.not-less-iff-gr-or-eq*)
ultimately have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})))$ (*efac* (*eres D E*)) $\models$ *efac* (*eres D E*)
using *E-least-false*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *metis*
hence *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})))$ (*eres D E*) $\models$ *eres D E*
using ‹*efac* (*eres D E*) $\preceq_c$ *eres D E*› *ord-res.entailed-clause-stays-entailed* **by** *fastforce*
hence *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})))$ *E* $\models$ *E*
using *clause-true-if-resolved-true*
by (*smt* (*verit*) *D-prod ex1-eres-eq-full-run-ground-resolution full-run-def rtranclpD*)
moreover have $\neg$ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ $|`|$ $(N$ $|\cup|$ $U_{er})))$ *E* $\models$ *E*
using *E-least-false is-least-false-clause-def linorder-cls.is-least-in-fset-ffilterD*(*2*)
by *blast*
ultimately show *False*
by *contradiction*
qed

ultimately have *eres D E* $|\notin|$ *N* $|\cup|$ U_{er}
unfolding *iefac-def* **by** *fastforce*

hence *iefac* $\mathcal{F}$ (*eres D E*) = *eres D E*
unfolding *iefac-def*
using $\mathcal{F}$*-subset* **by** *auto*

hence *iefac* $\mathcal{F}$ |‘| (N |∪| $U_{er}{}'$) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
unfolding ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}› **by** *simp*

show *?thesis*
unfolding *ord-res-5-invars-def step-hyps*(*1*,*2*)
proof (*intro conjI*)
have *eres D E* |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| $U_{er}{}'$)
unfolding ‹*iefac* $\mathcal{F}$ |‘| (N |∪| $U_{er}{}'$) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))› **by** *simp*

thus *next-clause-in-factorized-clause* N ($U_{er}{}'$, $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
unfolding *next-clause-in-factorized-clause-def* **by** *simp*

have $\mathcal{F}$ |⊆| N |∪| $U_{er}{}'$
unfolding ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}›
using ‹$\mathcal{F}$ |⊆| N |∪| U_{er}› **by** *blast*

thus *implicitly-factorized-clauses-subset* N ($U_{er}{}'$, $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
unfolding *implicitly-factorized-clauses-subset-def* **by** *simp*

have *dom-*$\mathcal{M}$*-eq*: *dom* $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def model-eq-interp-upto-next-clause-def*
by *metis*

have ∀ x ∈# E. ¬ *dom* $\mathcal{M}$ ⊨*l* x
using ‹¬ *dom* $\mathcal{M}$ ⊨ E› **by** (*simp add*: *true-cls-def*)

moreover have ∀ x ∈# D. $x \neq -L \longrightarrow$ ¬ *dom* $\mathcal{M}$ ⊨*l* x
proof −
have ∀ x ∈# D. $x \neq -L \longrightarrow$ ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⊨*l* x
using *L-prod-by-D* **by** (*metis ord-res.mem-productionE true-cls-def*)
moreover have ∀ x ∈# D. $x \neq -L \longrightarrow$ *atm-of* $x \prec_t$ *atm-of* (− L)
using ‹*ord-res.is-maximal-lit* (−L) D› *L-neg*
by (*smt* (*verit*, *best*) *L-prod-by-D atm-of-eq-atm-of linorder-cls.order-refl linorder-trm.antisym-conv1 ord-res.less-trm-if-neg ord-res.lesseq-trm-if-pos*)
ultimately have ∀ x ∈# D. $x \neq -L \longrightarrow$ ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E ⊨*l* x
using *ord-res.interp-fixed-for-smaller-literals*[
OF ‹*ord-res.is-maximal-lit* (−L) D› - ‹$D \prec_c E$›]
by *fastforce*

then show *?thesis*
unfolding *dom-*$\mathcal{M}$*-eq*[*symmetric*] **.**
qed

moreover have K ∈# *eres D E*
using ‹*ord-res.is-maximal-lit* K (*eres D E*)›

using *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

moreover have $\forall\, x \in\# \mathit{eres}\ D\ E.\ x \in\# D \vee x \in\# E$
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have $\forall\, x \in\# \mathit{eres}\ D\ E.\ x \neq -\ L$
proof (*intro ballI notI*)
fix x **assume** $x \in\# \mathit{eres}\ D\ E\ x = -\ L$
obtain $m\ A\ D'\ E'$ **where**
ord-res.is-strictly-maximal-lit $(\mathit{Pos}\ A)\ D$ **and**
$D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$ **and**
$E = \mathit{replicate\text{-}mset}\ (\mathit{Suc}\ m)\ (\mathit{Neg}\ A) + E'$ **and**
$\mathit{Neg}\ A \notin\# E'$ **and**
$\mathit{eres}\ D\ E = \mathit{repeat\text{-}mset}\ (\mathit{Suc}\ m)\ D' + E'$
using ‹*eres D E ≠ E*›[*THEN eres-not-identD*] **by** *metis*

have $L = \mathit{Neg}\ A$
using ‹*ord-res.is-strictly-maximal-lit* $(\mathit{Pos}\ A)\ D$›
by (*metis L-neg L-prod-by-D Uniq-D ord-res.mem-productionE*
linorder-lit.Uniq-is-greatest-in-mset literal.collapse(*2*) *uminus-Pos*)

have $x \in\# D' \vee x \in\# E'$
using ‹$x \in\#$ *eres D E*›
unfolding ‹*eres D E* = *repeat-mset* $(\mathit{Suc}\ m)\ D' + E'$›
by (*metis member-mset-repeat-mset-Suc union-iff*)
thus *False*
proof (*elim disjE*)
assume $x \in\# D'$
hence $\mathit{Pos}\ A \in\# D'$
unfolding ‹$x = -\ L$› ‹$L = \mathit{Neg}\ A$› **by** *simp*
hence ¬ *ord-res.is-strictly-maximal-lit* $(\mathit{Pos}\ A)\ D$
using ‹$D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$›
using *linorder-lit.is-greatest-in-mset-iff* **by** *auto*
thus *False*
using ‹*ord-res.is-strictly-maximal-lit* $(\mathit{Pos}\ A)\ D$› **by** *contradiction*
next
assume $x \in\# E'$
hence $\mathit{Pos}\ A \in\# E'$
unfolding ‹$x = -\ L$› ‹$L = \mathit{Neg}\ A$› **by** *simp*
hence $\mathit{Pos}\ A \in\# E$
unfolding ‹$E = \mathit{replicate\text{-}mset}\ (\mathit{Suc}\ m)\ (\mathit{Neg}\ A) + E'$› **by** *simp*
hence *ord-res.production* $(\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ D \Vdash l\ \mathit{Pos}\ A$
using *L-prod-by-D* ‹$L = \mathit{Neg}\ A$› **by** *auto*
hence *ord-res.interp* $(\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E \Vdash l\ \mathit{Pos}\ A$
by (*metis* ‹$L = \mathit{Neg}\ A$› *dom-*$\mathcal{M}$*-eq linorder-lit.is-maximal-in-mset-iff*
neg-literal-notin-imp-true-cls step-hyps(*3*) *step-hyps*(*4*) *true-lit-simps*(*1*))
hence *ord-res.interp* $(\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E \Vdash E$
using ‹$\mathit{Pos}\ A \in\# E$› **by** *blast*
hence $\mathit{dom}\ \mathcal{M} \Vdash E$

```
      using dom-M-eq by argo
    thus False
      using ‹¬ dom M ⊨ E› by contradiction
  qed
qed

ultimately have ¬ dom M ⊨l K
  by metis

have dom M' = ord-res.interp (fset (iefac F |`| (N |∪| U_er'))) (eres D E)
proof (intro subset-antisym subsetI)
  fix A :: 'f gterm
  assume A ∈ dom M'
  hence A ∈ dom M and A ≺t atm-of K
    unfolding ‹M' = restrict-map M {A. A ≺t atm-of K}› by simp-all

  have A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) E
    using ‹A ∈ dom M›
    unfolding ‹dom M = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) E› .
  hence A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) (eres D E)
   using ord-res.interp-fixed-for-smaller-literals ‹ord-res.is-maximal-lit K (eres
D E)›
      ‹A ≺t atm-of K› ‹eres D E ≺c E›
    by metis
  thus A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er'))) (eres D E)
   unfolding ‹iefac F |`| (N |∪| U_er') = finsert (eres D E) (iefac F |`| (N |∪|
U_er))›
    using ord-res.interp-insert-greater-clause-strong by simp
next
  fix A :: 'f gterm
  assume A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er'))) (eres D E)
  hence A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) (eres D E)
   unfolding ‹iefac F |`| (N |∪| U_er') = finsert (eres D E) (iefac F |`| (N |∪|
U_er))›
    using ord-res.interp-insert-greater-clause-strong by simp
  hence A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) E
    using ‹eres D E ≺c E› ord-res.interp-subset-if-less-cls by blast
  hence A ∈ dom M
    unfolding ‹dom M = ord-res.interp (fset (iefac F |`| (N |∪| U_er))) E› .

  moreover have A ≺t atm-of K
  proof −
    obtain C where
      C |∈| iefac F |`| (N |∪| U_er) and
      C ≺c eres D E and
      A-prod-by-C: A ∈ ord-res.production (fset (iefac F |`| (N |∪| U_er))) C
      using ‹A ∈ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) (eres D E)›
      unfolding ord-res.interp-def by blast
```

have *ord-res.is-maximal-lit* (*Pos A*) *C*
using *A-prod-by-C ord-res.mem-productionE* **by** *blast*

hence $A \preceq_t$ *atm-of K*
using ‹*ord-res.is-maximal-lit K* (*eres D E*)› ‹*C* $\prec_c$ *eres D E*›
by (*metis linorder-cls.dual-order.asym linorder-lit.multp-if-maximal-less-that-maximal*
linorder-trm.not-le-imp-less literal.collapse(*1*) *ord-res.less-lit-simps*(*1*)
step-hyps(*11*))

moreover have $A \neq$ *atm-of K*
using ‹$A \in$ *dom* $\mathcal{M}$› ‹$\neg$ *dom* $\mathcal{M} \models l$ *K*› ‹*is-pos K*› **by** *force*

ultimately show *?thesis*
by *order*
qed

ultimately show $A \in$ *dom* $\mathcal{M}'$
unfolding ‹$\mathcal{M}'$ = *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*}› **by** *simp*
qed

thus *model-eq-interp-upto-next-clause N* (*U*$_{er}$′, $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
unfolding *model-eq-interp-upto-next-clause-def* **by** *simp*

have $\forall$ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$). *C* $\prec_c$ *eres D E* $\longrightarrow$
ord-res-Interp (*iefac* $\mathcal{F}$ ‘ (*fset N* ∪ *fset U*$_{er}$′)) *C* $\models$ *C*
by (*smt* (*verit*, *ccfv-threshold*) *E-least-false Uniq-def Uniq-is-least-false-clause*
‹*eres D E* $\prec_c$ *E*› ‹*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$′) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$))›
finite-fset finsert.rep-eq finsertE finsert-absorb
fset.set-map ord-res.transp-less-cls
is-least-false-clause-finsert-smaller-false-clause linorder-cls.max.strict-order-iff
ord-res.interp-insert-greater-clause ord-res.production-insert-greater-clause transpE
true-cls-iefac-iff union-fset)

hence $\forall$ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$′). *C* $\prec_c$ *eres D E* $\longrightarrow$
ord-res-Interp (*iefac* $\mathcal{F}$ ‘ (*fset N* ∪ *fset U*$_{er}$′)) *C* $\models$ *C*
unfolding ‹*eres D E* $\prec_c$ *E*› ‹*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$′) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$))›
by *simp*

thus *all-smaller-clauses-true-wrt-respective-Interp N* (*U*$_{er}$′, $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def* **by** *simp*

have *A* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$′))) *C*
if $\mathcal{M}'$ *A* = *Some C* **for** *A C*
proof −
have $\mathcal{M}$ *A* = *Some C* **and** *A* $\prec_t$ *atm-of K*

unfolding *atomize-conj*
using ‹$\mathcal{M}'$ A = *Some* C› ‹$\mathcal{M}'$ = *restrict-map* $\mathcal{M}$ {A. $A \prec_t$ *atm-of* K}›
by (*metis Int-iff domI dom-restrict mem-Collect-eq restrict-in*)

hence *A-prod-by-C*: $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def atoms-in-model-were-produced-def*
by *metis*

moreover have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C =
ord-res.production (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'))) C
proof (*rule ord-res.production-swap-clause-set*)
have *ord-res.is-strictly-maximal-lit* (*Pos* A) C
using *A-prod-by-C ord-res.mem-productionE* **by** *metis*
moreover have *Pos* $A \prec_l K$
using ‹$A \prec_t$ *atm-of* K›
by (*metis Pos-atm-of-iff ord-res.less-lit-simps*(*1*) *step-hyps*(*11*))
ultimately have $C \prec_c$ *eres* D E
using *linorder-lit.multp-if-maximal-less-that-maximal step-hyps*(*10*) **by** *blast*
thus {D. D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ D C} =
{D. D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') ∧ $(\prec_c)^{==}$ D C}
unfolding ‹*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') = *finsert* (*eres* D E) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))›
by *auto*
qed

ultimately show *?thesis*
by *argo*
qed

thus *atoms-in-model-were-produced* N (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres* D E))
unfolding *atoms-in-model-were-produced-def* **by** *simp*

have $\mathcal{M}'$ A = *Some* C
if $C \prec_c$ *eres* D E **and** *A-in*: $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'))) C
for C A
proof −
have $C \prec_c E$
using ‹$C \prec_c$ *eres* D E› ‹*eres* D $E \prec_c E$› **by** *order*

moreover have *A-prod-by-C*: $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C
proof −
have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C =
ord-res.production (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'))) C
proof (*rule ord-res.production-swap-clause-set*)
show {D. D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) ∧ $(\prec_c)^{==}$ D C} =

```
        {D. D |∈| iefac F |`| (N |∪| U_er') ∧ (≺_c)^== D C}
        unfolding ‹iefac F |`| (N |∪| U_er') = finsert (eres D E) (iefac F |`| (N
|∪| U_er))›
        using ‹C ≺_c eres D E›
        by (metis (no-types, opaque-lifting) finsert-iff linorder-cls.less-le-not-le)
    qed

    thus ?thesis
      using A-in by argo
  qed

  ultimately have M A = Some C
    using invars
  unfolding step-hyps(1,2) ord-res-5-invars-def all-produced-atoms-in-model-def
    by metis

  moreover have A ≺_t atm-of K
  proof -
    have A ∈ dom M
      using ‹M A = Some C› by auto

    have ord-res.is-maximal-lit (Pos A) C
      using A-prod-by-C ord-res.mem-productionE by blast

    hence A ⪯_t atm-of K
      using ‹ord-res.is-maximal-lit K (eres D E)› ‹C ≺_c eres D E›
    by (metis linorder-cls.dual-order.asym linorder-lit.multp-if-maximal-less-that-maximal
        linorder-trm.not-le-imp-less literal.collapse(1) ord-res.less-lit-simps(1)
        step-hyps(11))

    moreover have A ≠ atm-of K
      using ‹A ∈ dom M› ‹¬ dom M ⟦=⟧l K› ‹is-pos K› by force

    ultimately show ?thesis
      by order
  qed

  ultimately show M' A = Some C
    unfolding ‹M' = restrict-map M {A. A ≺_t atm-of K}› by simp
qed

thus all-produced-atoms-in-model N (U_er', F, M', Some (eres D E))
  unfolding all-produced-atoms-in-model-def by simp
qed
next
case step-hyps: (resolution-neg M E L D U_er' U_er M' K C F)

obtain A_L where L-def: L = Neg A_L
  using ‹is-neg L› by (cases L) simp-all
```

have $A_L \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D
using *invars* ‹$\mathcal{M}$ (*atm-of* L) = *Some* D›
unfolding *step-hyps*(1,2) *ord-res-5-invars-def atoms-in-model-were-produced-def*
unfolding *L-def literal.sel*
by *metis*

hence D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
ord-res.is-strictly-maximal-lit (*Pos* A_L) D **and**
D-false: ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) D ⊨ D
unfolding *atomize-conj* **by** (*metis ord-res.mem-productionE*)

obtain A_K **where** *K-def*: K = *Neg* A_K
using ‹*is-neg* K› **by** (*cases* K) *simp-all*

have $A_K \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C
using *invars* ‹$\mathcal{M}$ (*atm-of* K) = *Some* C›
unfolding *step-hyps*(1,2) *ord-res-5-invars-def atoms-in-model-were-produced-def*
unfolding *K-def literal.sel*
by *metis*

hence C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
ord-res.is-strictly-maximal-lit (*Pos* A_K) C **and**
C-false: ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) C ⊨ C
unfolding *atomize-conj* **by** (*metis ord-res.mem-productionE*)

have $D \prec_c E$
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* A_L) D› ‹*ord-res.is-maximal-lit* L E›[*unfolded L-def*]
using *linorder-lit.multp-if-maximal-less-that-maximal ord-res.less-lit-simps*(2)
by *blast*

have *eres* D $E \neq E$
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* A_L) D› ‹*ord-res.is-maximal-lit* L E›[*unfolded L-def*]
by (*metis L-def eres-ident-iff ex-ground-resolutionI is-pos-def*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one
linorder-lit.multp-if-maximal-less-that-maximal linorder-lit.neq-iff
linorder-trm.dual-order.asym ord-res.less-lit-simps(4) *ground-resolution-def step-hyps*(5))

moreover have *eres* D $E \preceq_c E$
using *eres-le* **.**

ultimately have *eres* D $E \prec_c E$
by *order*

have *iefac* $\mathcal{F}$ (*eres* D E) = *eres* D E
by (*metis* (*mono-tags*, *lifting*) *Uniq-D efac-spec iefac-def is-pos-def*

linorder-lit.Uniq-is-maximal-in-mset step-hyps(10) step-hyps(11))

hence *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
unfolding ‹U_{er}' = *finsert* (*eres D E*) U_{er}› **by** *simp*

hence {|C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). C ≺$_c$ *eres D E*|} =
{|C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). C ≺$_c$ *eres D E*|}
by *auto*

show *?thesis*
unfolding *step-hyps(1,2) ord-res-5-invars-def*
proof (*intro conjI*)
have C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}')
using ‹C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})›
unfolding ‹*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))›
by *simp*

thus *next-clause-in-factorized-clause* N (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* C)
unfolding *next-clause-in-factorized-clause-def* **by** *simp*

have $\mathcal{F}$ |⊆| N |∪| U_{er}
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def implicitly-factorized-clauses-subset-def*
by *metis*

hence $\mathcal{F}$ |⊆| N |∪| U_{er}'
unfolding ‹U_{er}' = *finsert* (*eres D E*) U_{er}› **by** *blast*

thus *implicitly-factorized-clauses-subset* N (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* C)
unfolding *implicitly-factorized-clauses-subset-def* **by** *simp*

have *Pos* A_K ≺$_l$ *Neg* A_K
by *simp*

hence C ≺$_c$ *eres D E*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* A_K) C›
‹*ord-res.is-maximal-lit K* (*eres D E*)›[*unfolded K-def*]
using *linorder-lit.multp-if-maximal-less-that-maximal* **by** *blast*

have C ≺$_c$ E
using ‹C ≺$_c$ *eres D E*› ‹*eres D E* ≺$_c$ E› **by** *order*

have {|x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). x ≺$_c$ C|} = {|x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). x ≺$_c$ C|}
using ‹C ≺$_c$ *eres D E*›
unfolding ‹*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))›
by (*metis ffilter-eq-ffilter-minus-singleton finsert-absorb fminus-finsert-absorb*

linorder-cls.less-asym)

hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}′))) *C* =
ord-res.interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
using ‹*C* ≺$_c$ *eres D E*›
by (*simp add*: ‹*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}′) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))›
ord-res.interp-insert-greater-clause)

have *dom-*$\mathcal{M}$*-eq*: *dom* $\mathcal{M}$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *E*
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def model-eq-interp-upto-next-clause-def*
by *metis*

have ∀ *x* ∈# *E*. ¬ *dom* $\mathcal{M}$ ⊨*l* *x*
using ‹¬ *dom* $\mathcal{M}$ ⊨ *E*› **by** (*simp add*: *true-cls-def*)

moreover have ∀ *x* ∈# *D*. *x* ≠ −*L* ⟶ ¬ *dom* $\mathcal{M}$ ⊨*l* *x*
proof −
have ∀ *x* ∈# *D*. *x* ≠ −*L* ⟶ ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D* ⊨*l* *x*
using *D-false* **by** *blast*
moreover have ∀ *x* ∈# *D*. *x* ≠ −*L* ⟶ *atm-of x* ≺$_t$ *atm-of* (− *L*)
unfolding *L-def uminus-Neg literal.sel*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* A_L) *D*›
by (*metis Pos-atm-of-iff* ‹A_L ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*›
ord-res.less-trm-if-neg ord-res.lesseq-trm-if-pos reflclp-iff)
ultimately have ∀ *x* ∈# *D*. *x* ≠ −*L* ⟶ ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *E* ⊨*l* *x*
using *ord-res.interp-fixed-for-smaller-literals*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* A_L) *D*› ‹*D* ≺$_c$ *E*›
using *L-def* **by** *fastforce*
thus *?thesis*
unfolding *dom-*$\mathcal{M}$*-eq*[*symmetric*] **.**
qed

moreover have *K* ∈# *eres D E*
using ‹*ord-res.is-maximal-lit K* (*eres D E*)›
using *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

moreover have ∀ *x* ∈# *eres D E*. *x* ∈# *D* ∨ *x* ∈# *E*
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have ∀ *x* ∈# *eres D E*. *x* ≠ − *L*
proof (*intro ballI notI*)
fix *x* **assume** *x* ∈# *eres D E x* = − *L*
obtain *m A D*′ *E*′ **where**
ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**

$D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$ **and**
$E = \mathit{replicate\text{-}mset}\ (\mathit{Suc}\ m)\ (\mathit{Neg}\ A) + E'$ **and**
$\mathit{Neg}\ A \notin\# E'$ **and**
$\mathit{eres}\ D\ E = \mathit{repeat\text{-}mset}\ (\mathit{Suc}\ m)\ D' + E'$
using ‹$\mathit{eres}\ D\ E \neq E$›[*THEN eres-not-identD*] **by** *metis*

have $L = \mathit{Neg}\ A$
using ‹$\mathit{ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit}\ (\mathit{Pos}\ A)\ D$›
by (*metis L-def Uniq-D* ‹$\mathit{ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit}\ (\mathit{Pos}\ A_L)\ D$›
linorder-lit.Uniq-is-greatest-in-mset literal.inject(*1*))

have $x \in\# D' \vee x \in\# E'$
using ‹$x \in\# \mathit{eres}\ D\ E$›
unfolding ‹$\mathit{eres}\ D\ E = \mathit{repeat\text{-}mset}\ (\mathit{Suc}\ m)\ D' + E'$›
by (*metis member-mset-repeat-mset-Suc union-iff*)
thus *False*
proof (*elim disjE*)
assume $x \in\# D'$
hence $\mathit{Pos}\ A \in\# D'$
unfolding ‹$x = -\ L$› ‹$L = \mathit{Neg}\ A$› **by** *simp*
hence ¬ $\mathit{ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit}\ (\mathit{Pos}\ A)\ D$
using ‹$D = \mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ D'$›
using *linorder-lit.is-greatest-in-mset-iff* **by** *auto*
thus *False*
using ‹$\mathit{ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit}\ (\mathit{Pos}\ A)\ D$› **by** *contradiction*
next
assume $x \in\# E'$
hence $\mathit{Pos}\ A \in\# E'$
unfolding ‹$x = -\ L$› ‹$L = \mathit{Neg}\ A$› **by** *simp*
hence $\mathit{Pos}\ A \in\# E$
unfolding ‹$E = \mathit{replicate\text{-}mset}\ (\mathit{Suc}\ m)\ (\mathit{Neg}\ A) + E'$› **by** *simp*
hence $\mathit{ord\text{-}res.production}\ (\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ D \models l\ \mathit{Pos}\ A$
using *L-def* ‹$A_L \in \mathit{ord\text{-}res.production}\ (\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ D$›
‹$L = \mathit{Neg}\ A$›
by *blast*
hence $\mathit{ord\text{-}res.interp}\ (\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E \models l\ \mathit{Pos}\ A$
by (*metis* ‹$L = \mathit{Neg}\ A$› *dom-*$\mathcal{M}$*-eq linorder-lit.is-maximal-in-mset-iff*
neg-literal-notin-imp-true-cls step-hyps(*3*) *step-hyps*(*4*) *true-lit-simps*(*1*))
hence $\mathit{ord\text{-}res.interp}\ (\mathit{fset}\ (\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E \models E$
using ‹$\mathit{Pos}\ A \in\# E$› **by** *blast*
hence $\mathit{dom}\ \mathcal{M} \models E$
using *dom-*$\mathcal{M}$*-eq* **by** *argo*
thus *False*
using ‹¬ $\mathit{dom}\ \mathcal{M} \models E$› **by** *contradiction*
qed
qed

ultimately have ¬ $\mathit{dom}\ \mathcal{M} \models l\ K$
by *metis*

have *dom* $\mathcal{M}' = ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}'))))\ C$
proof (*intro subset-antisym subsetI*)
fix $A :: 'f\ gterm$
assume $A \in dom\ \mathcal{M}'$
hence $A \in dom\ \mathcal{M}$ **and** $A \prec_t atm\text{-}of\ K$
unfolding $\langle \mathcal{M}' = restrict\text{-}map\ \mathcal{M}\ \{A.\ A \prec_t atm\text{-}of\ K\}\rangle$ **by** *simp-all*

have $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E$
using $\langle A \in dom\ \mathcal{M}\rangle$
unfolding $\langle dom\ \mathcal{M} = ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E\rangle$ **.**
hence $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ C$
using *ord-res.interp-fixed-for-smaller-literals* $\langle ord\text{-}res.is\text{-}maximal\text{-}lit\ K\ (eres\ D\ E)\rangle$
$\langle A \prec_t atm\text{-}of\ K\rangle\ \langle C \prec_c E\rangle$
by (*smt* (*verit*, *del-insts*) *K-def*
$\langle A_K \in ord\text{-}res.production\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ C\rangle\ \langle eres\ D\ E \prec_c E\rangle$
literal.sel(*2*) *ord-res.lesser-atoms-in-previous-interp-are-in-final-interp*
ord-res.lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-productive)
thus $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}'))\ C$
unfolding $\langle iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\rangle$
using *ord-res.interp-insert-greater-clause-strong*
by (*simp add*: $\langle C \prec_c eres\ D\ E\rangle$)
next
fix $A :: 'f\ gterm$
assume $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}'))\ C$
hence $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ C$
unfolding $\langle iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\rangle$
using *ord-res.interp-insert-greater-clause-strong* **by** (*simp add*: $\langle C \prec_c eres\ D\ E\rangle$)
hence $A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E$
using $\langle C \prec_c E\rangle$ *ord-res.interp-subset-if-less-cls* **by** *blast*
hence $A \in dom\ \mathcal{M}$
unfolding $\langle dom\ \mathcal{M} = ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ E\rangle$ **.**

moreover have $A \prec_t atm\text{-}of\ K$
proof −
obtain B **where**
$B\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and**
$B \prec_c C$ **and**
A-prod-by-B: $A \in ord\text{-}res.production\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ B$
using $\langle A \in ord\text{-}res.interp\ (fset\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})))\ C\rangle$
unfolding *ord-res.interp-def* **by** *blast*

have *ord-res.is-maximal-lit* $(Pos\ A)\ B$
using *A-prod-by-B ord-res.mem-productionE* **by** *blast*

hence $A \preceq_t$ *atm-of K*
using ‹*ord-res.is-maximal-lit K* (*eres D E*)› ‹$C \prec_c$ *eres D E*›
by (*metis K-def* ‹$B \prec_c C$› *asympD*
linorder-cls.less-trans linorder-lit.multp-if-maximal-less-that-maximal
linorder-trm.le-less-linear literal.sel(*2*)
ord-res.asymp-less-cls ord-res.less-lit-simps(*4*))

moreover have $A \neq$ *atm-of K*
using ‹$A \in$ *dom* $\mathcal{M}$› ‹$\neg$ *dom* $\mathcal{M} \models l K$›
unfolding *K-def*
by (*metis* ‹$A \in$ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*›
‹$A_K \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*› *atm-of-uminus*
linorder-lit.is-greatest-in-mset-iff literal.sel(*1*) *ord-res.mem-productionE*
pos-literal-in-imp-true-cls uminus-Neg)

ultimately show *?thesis*
by *order*
qed

ultimately show $A \in$ *dom* $\mathcal{M}'$
unfolding ‹$\mathcal{M}'$ = *restrict-map* $\mathcal{M}$ {*A*. $A \prec_t$ *atm-of K*}› **by** *simp*
qed

thus *model-eq-interp-upto-next-clause N* (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)
unfolding *model-eq-interp-upto-next-clause-def* **by** *simp*

have ∀ *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). $x \prec_c E \longrightarrow$
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) $x \models x$
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def*
by *simp*

hence ∀ *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). $x \prec_c C \longrightarrow$
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) $x \models x$
using ‹$C \prec_c E$› **by** *simp*

moreover have ∀ *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). $x \prec_c C \longrightarrow$
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *x* =
ord-res-Interp (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}'))) *x*
unfolding ‹*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))›
by (*metis* (*no-types*, *lifting*) ‹$C \prec_c$ *eres D E*› *finite-fset finsert.rep-eq*
linorder-cls.dual-order.strict-trans2 ord-res.Interp-insert-greater-clause sup2CI)

ultimately have ∀ *x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). $x \prec_c C \longrightarrow$
ord-res-Interp (*iefac* $\mathcal{F}$ ‘ (*fset N* ∪ *fset* U_{er}')) $x \models x$
by *simp*

hence $\forall x \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}'). x \prec_c C \longrightarrow$
ord-res-Interp (*iefac* $\mathcal{F}$ ‘ (*fset* $N \cup$ *fset* $U_{er}{}'$)) $x \Vdash x$
unfolding ‹*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}') =$ *finsert* (*eres* $D$ $E$) (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)›
using ‹$C \prec_c$ *eres* D E› **by** *auto*

thus *all-smaller-clauses-true-wrt-respective-Interp* N ($U_{er}{}'$, $\mathcal{F}$, $\mathcal{M}'$, *Some* C)
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def*
by *simp*

have $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}'))) C$
if $\mathcal{M}'$ A = *Some* C **for** A C
proof −
have $\mathcal{M}$ A = *Some* C **and** $A \prec_t$ *atm-of* K
unfolding *atomize-conj*
using ‹$\mathcal{M}'$ A = *Some* C› ‹$\mathcal{M}'$ = *restrict-map* $\mathcal{M}$ {A. $A \prec_t$ *atm-of* K}›
by (*metis Int-iff domI dom-restrict mem-Collect-eq restrict-in*)

hence *A-prod-by-C*: $A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))) C$
using *invars*
unfolding *step-hyps*(*1*,*2*) *ord-res-5-invars-def atoms-in-model-were-produced-def*
by *metis*

moreover have *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))) C =$
ord-res.production (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}'))) C$
proof (*rule ord-res.production-swap-clause-set*)
have *ord-res.is-strictly-maximal-lit* (*Pos* A) C
using *A-prod-by-C ord-res.mem-productionE* **by** *metis*
moreover have *Pos* $A \prec_l K$
using ‹$A \prec_t$ *atm-of* K›
by (*simp add*: *K-def*)
ultimately have $C \prec_c$ *eres* D E
using *linorder-lit.multp-if-maximal-less-that-maximal step-hyps*(*10*) **by** *blast*
thus {D. $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}) \wedge (\prec_c)^{==} D C$} =
{D. $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}') \wedge (\prec_c)^{==} D C$}
unfolding ‹*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}') =$ *finsert* (*eres* $D$ $E$) (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)›
by *auto*
qed

ultimately show *?thesis*
by *argo*
qed

thus *atoms-in-model-were-produced* N ($U_{er}{}'$, $\mathcal{F}$, $\mathcal{M}'$, *Some* C)
unfolding *atoms-in-model-were-produced-def* **by** *simp*

```
  have M' A = Some B
   if B ≺c C and A-in: A ∈ ord-res.production (fset (iefac F |`| (N |∪| U_er'))) B
   for B A
  proof −
   have B ≺c eres D E
    using ‹B ≺c C› ‹C ≺c eres D E› by order
   hence B ≺c E
    using ‹eres D E ≺c E› by order

   moreover have A-prod-by-B: A ∈ ord-res.production (fset (iefac F |`| (N |∪| U_er))) B
   proof −
    have ord-res.production (fset (iefac F |`| (N |∪| U_er))) B =
      ord-res.production (fset (iefac F |`| (N |∪| U_er'))) B
    proof (rule ord-res.production-swap-clause-set)
     show {D. D |∈| iefac F |`| (N |∪| U_er) ∧ (≺c)== D B} =
       {D. D |∈| iefac F |`| (N |∪| U_er') ∧ (≺c)== D B}
      unfolding ‹iefac F |`| (N |∪| U_er') = finsert (eres D E) (iefac F |`| (N |∪| U_er))›
       using ‹B ≺c eres D E›
       by (metis (no-types, opaque-lifting) finsert-iff linorder-cls.less-le-not-le)
    qed

    thus ?thesis
     using A-in by argo
   qed

   ultimately have M A = Some B
    using invars
   unfolding step-hyps(1,2) ord-res-5-invars-def all-produced-atoms-in-model-def
    by metis

   moreover have A ≺t atm-of K
   proof −
    have A ∈ dom M
     using ‹M A = Some B› by auto

    have ord-res.is-maximal-lit (Pos A) B
     using A-prod-by-B ord-res.mem-productionE by blast

    hence A ⪯t atm-of K
     using ‹ord-res.is-maximal-lit K (eres D E)› ‹B ≺c eres D E›
     by (metis K-def asympD linorder-lit.multp-if-maximal-less-that-maximal
         linorder-trm.le-less-linear literal.sel(2) ord-res.asymp-less-cls
         ord-res.less-lit-simps(4))

    moreover have A ≠ atm-of K
     using ‹A ∈ dom M› ‹¬ dom M ⊨l K›
```

```
        using ‹M A = Some B› step-hyps(12) that(1) by force

      ultimately show ?thesis
        by order
    qed

    ultimately show M' A = Some B
      unfolding ‹M' = restrict-map M {A. A ≺t atm-of K}› by simp
  qed

  thus all-produced-atoms-in-model N (Uer', F, M', Some C)
    unfolding all-produced-atoms-in-model-def by simp
 qed
qed

lemma rtranclp-ord-res-6-preserves-invars:
 assumes steps: (ord-res-6 N)** s s' and invars: ord-res-5-invars N s
 shows ord-res-5-invars N s'
 using steps invars
 by (induction s rule: converse-rtranclp-induct) (auto intro: ord-res-6-preserves-invars)

lemma ex-ord-res-6-if-not-final:
 assumes
   not-final: ¬ ord-res-6-final S and
   invars: ∀ N s. S = (N, s) ⟶ ord-res-5-invars N s
 shows ∃ S'. ord-res-6-step S S'
proof −
 from not-final obtain N Uer F M C where
   S-def: S = (N, (Uer, F, M, Some C)) and C ≠ {#}
   unfolding ord-res-6-final.simps de-Morgan-disj not-ex
   by (metis option.exhaust surj-pair)

 note invars' = invars[unfolded ord-res-5-invars-def, rule-format, OF S-def]

 have ∃ s'. ord-res-6 N (Uer, F, M, Some C) s'
 proof (cases dom M ⊨ C)
   case True
   thus ?thesis
     using ord-res-6.skip by metis
 next
   case C-false: False
   obtain L where L-max: linorder-lit.is-maximal-in-mset C L
     using linorder-lit.ex-maximal-in-mset[OF ‹C ≠ {#}›] ..

   show ?thesis
   proof (cases L)
     case (Pos A)
     hence L-pos: is-pos L
       by simp
```

```
    show ?thesis
    proof (cases ord-res.is-strictly-maximal-lit L C)
      case True
      then show ?thesis
        using ord-res-6.production[OF C-false L-max L-pos] by metis
    next
      case L-not-striclty-max: False
      thus ?thesis
        using ord-res-6.factoring[OF C-false L-max L-pos L-not-striclty-max refl]
by metis
    qed
  next
    case (Neg A)
    hence L-neg: is-neg L
      by simp

    have C-least-false: is-least-false-clause (iefac \mathcal{F} |`| (N |\cup| U_er)) C
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      show C |\in| iefac \mathcal{F} |`| (N |\cup| U_er)
        using invars' by (metis next-clause-in-factorized-clause-def)
    next
      have ¬ ord-res.interp (fset (iefac \mathcal{F} |`| (N |\cup| U_er))) C \models C
        using invars' C-false by (metis model-eq-interp-upto-next-clause-def)
      moreover have ord-res.production (fset (iefac \mathcal{F} |`| (N |\cup| U_er))) C = {}
      proof -
        have \nexists L. is-pos L \wedge ord-res.is-strictly-maximal-lit L C
          using L-max L-neg
          by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset
              linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
        thus ?thesis
          using unproductive-if-nex-strictly-maximal-pos-lit by metis
      qed
      ultimately show ¬ ord-res-Interp (fset (iefac \mathcal{F} |`| (N |\cup| U_er))) C \models C
        by simp
    next
      fix D
      assume D-in: D |\in| iefac \mathcal{F} |`| (N |\cup| U_er) and D \neq C and
        C-false: ¬ ord-res-Interp (fset (iefac \mathcal{F} |`| (N |\cup| U_er))) D \models D
      have ¬ D \prec_c C
        using C-false
        using invars' D-in
        unfolding all-smaller-clauses-true-wrt-respective-Interp-def
        by auto
      thus C \prec_c D
        using ‹D \neq C› by order
    qed
    then obtain D where D|\in|iefac \mathcal{F} |`| (N |\cup| U_er) and
      D \prec_c C and
```

ord-res.is-strictly-maximal-lit (*Pos A*) *D* **and**
D-prod: *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D* = {*A*}
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
L-max[*unfolded Neg*] **by** *metis*

hence ∃ *DC*. *ground-resolution D C DC*
unfolding *ground-resolution-def*
using *L-max Neg ex-ground-resolutionI* **by** *blast*

hence *eres D C* ≠ *C*
unfolding *eres-ident-iff* **by** *metis*

hence *eres D C* ≺$_c$ *C*
using *eres-le*[*of D C*] **by** *order*

have $\mathcal{M}$ (*atm-of L*) = *Some D*
using *invars′*
by (*metis Neg* ‹*D* ≺$_c$ *C*› *all-produced-atoms-in-model-def D-prod insertI1 literal.sel*(*2*))

show *?thesis*
proof (*cases eres D C* = {#})
case *True*
then show *?thesis*
using *ord-res-6.resolution-bot*[*OF C-false L-max L-neg* ‹$\mathcal{M}$ (*atm-of L*) = *Some D*›] **by** *metis*
next
case *False*
then obtain *K* **where** *K-max*: *ord-res.is-maximal-lit K* (*eres D C*)
using *linorder-lit.ex-maximal-in-mset* **by** *metis*
show *?thesis*
proof (*cases K*)
case *K-def*: (*Pos* A_K)
hence *is-pos K*
by *simp*
thus *?thesis*
using *ord-res-6.resolution-pos*
using *C-false L-max L-neg* ‹$\mathcal{M}$ (*atm-of L*) = *Some D*› ‹*eres D C* ≠ {#}› *K-max* **by** *metis*
next
case *K-def*: (*Neg* A_K)
hence *K-neg*: *is-neg K*
by *simp*

have ¬ *ord-res-Interp* (*fset* (*finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})))) (*eres D C*) ⊨ *eres D C*
by (*smt* (*verit*) *C-least-false D-prod Interp-insert-unproductive K-max K-neg Uniq-D*
‹*eres D C* ≠ *C*› *clause-true-if-resolved-true ex1-eres-eq-full-run-ground-resolution*

finite-fset fset-simps(2) *full-run-def insert-not-empty is-least-false-clause-def*
linorder-cls.is-least-in-fset-ffilterD(2) *linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset rtranclpD
unproductive-if-nex-strictly-maximal-pos-lit)

hence *eres-least*:
is-least-false-clause (*finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) (*eres D C*)
using *C-least-false* ‹*eres D C* $\prec_c$ *C*›
by (*metis is-least-false-clause-finsert-smaller-false-clause*)

then obtain *E* **where** *E* |∈| *finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) **and**
E $\prec_c$ *eres D C* **and**
ord-res.is-strictly-maximal-lit (*Pos* A_K) *E* **and**
E-prod: *ord-res.production* (*fset* (*finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})))) *E* = {A_K}
using *bex-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
K-max K-def
by *metis*

have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *E* =
ord-res.production (*fset* (*finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})))) *E*
proof (*rule ord-res.production-swap-clause-set*)
have *eres D C* |∉| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
proof (*rule notI*)
assume *eres D C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
hence *finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) = *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
by *blast*
hence *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) (*eres D C*)
using *eres-least* **by** *argo*
thus *False*
using *C-least-false* ‹*eres D C* ≠ *C*›
by (*metis Uniq-D Uniq-is-least-false-clause*)
qed

thus {*D*. *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) ∧ ($\prec_c$)$^{==}$ *D E*} =
{*Da*. *Da* |∈| *finsert* (*eres D C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) ∧ ($\prec_c$)$^{==}$ *Da E*}
by (*metis* (*mono-tags*, *lifting*) ‹*E* $\prec_c$ *eres D C*› *finsert-iff linorder-cls.leD*)
qed

also have ... = {A_K}
using *E-prod* **.**

finally have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *E* = {A_K}
.

```
        hence M (atm-of K) = Some E
          using invars'
          unfolding ord-res-5-invars-def all-produced-atoms-in-model-def
        by (metis K-def ‹E ≺c eres D C› eres-le insertI1 linorder-cls.dual-order.strict-trans1
            literal.sel(2))

        thus ?thesis
          using ord-res-6.resolution-neg
         using C-false L-max L-neg ‹M (atm-of L) = Some D› ‹eres D C ≠ {#}›
K-max K-neg by metis
       qed
     qed
   qed
  qed
  thus ?thesis
    using S-def ord-res-6-step.simps by metis
qed

lemma ord-res-6-safe-state-if-invars:
  safe-state ord-res-6-step ord-res-6-final (N, s) if invars: ord-res-5-invars N s for
N s
proof −
  {
    fix S'
   assume ord-res-6-semantics.eval (N, s) S' and stuck: stuck-state ord-res-6-step
S'
    then obtain s' where S' = (N, s') and (ord-res-6 N)** s s'
    proof (induction (N, s) arbitrary: N s rule: converse-rtranclp-induct)
      case base
      thus ?case by simp
    next
      case (step z)
      thus ?case
       by (smt (verit, ccfv-SIG) converse-rtranclp-into-rtranclp ord-res-6-step.cases
prod.inject)
    qed
    hence ord-res-5-invars N s'
      using invars rtranclp-ord-res-6-preserves-invars by metis
    hence ¬ ord-res-6-final S' ⟹ ∃ S''. ord-res-6-step S' S''
      using ex-ord-res-6-if-not-final[of S'] ‹S' = (N, s')› by blast
    hence ord-res-6-final S'
      using stuck[unfolded stuck-state-def] by argo
  }
  thus ?thesis
    unfolding safe-state-def stuck-state-def by metis
qed

lemma ex-model-build-from-least-clause-to-any-less-than-least-false:
  assumes
```

$\mathcal{F}$-*subset*: $\mathcal{F}$ |⊆| N |∪| U_{er} **and**
C-*least*: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) C **and**
D-*in*: D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
D-*lt-least-false*: ∀E. *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) E ⟶ $D \preceq_c E$ **and**
$C \preceq_c D$
shows ∃$\mathcal{M}$. (*ord-res-5* N)** (U_{er}, $\mathcal{F}$, *Map.empty*, *Some* C) (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* D)
using *ord-res.wfP-less-cls* D-*in* ‹$C \preceq_c D$› D-*lt-least-false*
proof (*induction* D *rule*: *wfp-induct-rule*)
case (*less* D)

have *invars*: *ord-res-5-invars* N (U_{er}, $\mathcal{F}$, *Map.empty*, *Some* C)
using *ord-res-5-invars-initial-state* $\mathcal{F}$-*subset* C-*least* **by** *metis*

define *clauses-lt-D* :: *'f gclause fset* **where**
clauses-lt-D = {|C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $C \prec_c D$|}

show *?case*
proof (*cases clauses-lt-D* = {||})
case *True*
hence $C = D$
unfolding *clauses-lt-D-def*
using C-*least* ‹$C \preceq_c D$›
by (*metis fempty-iff ffmember-filter linorder-cls.antisym-conv3*
linorder-cls.is-least-in-fset-iff linorder-cls.less-le-not-le)
thus *?thesis*
by *blast*
next
case *False*

obtain x **where** x-*greatest*: *linorder-cls.is-greatest-in-fset clauses-lt-D* x
using *False linorder-cls.ex-greatest-in-fset* **by** *metis*

have $x \prec_c D$ **and** x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
using x-*greatest* **by** (*simp-all add*: *clauses-lt-D-def linorder-cls.is-greatest-in-fset-iff*)

moreover have $C \preceq_c x$
using x-*greatest* C-*least*
by (*metis clauses-lt-D-def ffmember-filter linorder-cls.is-greatest-in-fset-iff*
linorder-cls.not-less nbex-less-than-least-in-fset)

moreover have ⋀E. *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) E ⟹ $x \prec_c E$
using ‹$x \prec_c D$› *less.prems* **by** *force*

ultimately obtain $\mathcal{M}$ **where**
IH: (*ord-res-5* N)** (U_{er}, $\mathcal{F}$, *Map.empty*, *Some* C) (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* x)
using *less.IH* **by** *blast*

```
    moreover have ∃M'. ord-res-5 N (U_er, F, M, Some x) (U_er, F, M', Some
D)
    proof −
      have linorder-cls.is-least-in-fset (ffilter ((≺_c) x) (iefac F |`| (N |∪| U_er))) D
        using x-greatest[unfolded clauses-lt-D-def]
       by (smt (verit) ffmember-filter less.prems(1) linorder-cls.is-greatest-in-fset-iff
           linorder-cls.is-least-in-ffilter-iff linorder-cls.not-less-iff-gr-or-eq)
      hence next-clause-eq: The-optional (linorder-cls.is-least-in-fset
        (ffilter ((≺_c) x) (iefac F |`| (N |∪| U_er)))) = Some D
        by (metis linorder-cls.Uniq-is-least-in-fset The-optional-eq-SomeI)

      have x-true: ord-res-Interp (fset (iefac F |`| (N |∪| U_er))) x ⊨ x
        using less.prems
        unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff

        by (metis ‹⋀E. is-least-false-clause (iefac F |`| (N |∪| U_er)) E ⟹ x ≺_c
E› ‹x |∈| iefac F |`| (N |∪| U_er)›
          ex-false-clause-iff finsert-absorb is-least-false-clause-finsert-smaller-false-clause
            linorder-cls.order.irrefl ex-false-clause-def)

      show ?thesis
      proof (cases dom M ⊨ x)
        case True
        thus ?thesis
          using ord-res-5.skip next-clause-eq by metis
      next
        case False
        hence ¬ ord-res.interp (fset (iefac F |`| (N |∪| U_er))) x ⊨ x
       using rtranclp-ord-res-5-preserves-invars[OF IH invars, unfolded ord-res-5-invars-def,
            simplified]
          by (simp add: model-eq-interp-upto-next-clause-def)

        thus ?thesis
          using ord-res-5.production[OF False] next-clause-eq
          using x-true
       by (metis Un-empty-right linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
            unproductive-if-nex-strictly-maximal-pos-lit)
      qed
    qed

    ultimately show ?thesis
      by (smt (verit) rtranclp.rtrancl-into-rtrancl)
  qed
qed

lemma full-rtranclp-ord-res-5-run-upto:
  assumes
    ord-res-6 N (U_er, F, M, Some E) (U_er', F', M', Some D) and
```

invars: *ord-res-5-invars N* (U_{er}', $\mathcal{F}'$, $\mathcal{M}'$, *Some D*) **and**
$\mathcal{M}'$*-def*: $\mathcal{M}'$ = *restrict-map* $\mathcal{M}$ {*A*. ∃ *K*. *linorder-lit.is-maximal-in-mset D K* ∧ *A* ≺$_t$ *atm-of K*} **and**
C-least: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}')) *C*
shows (*ord-res-5 N*)** (U_{er}', $\mathcal{F}'$, *Map.empty*, *Some C*) (U_{er}', $\mathcal{F}'$, $\mathcal{M}'$, *Some D*)
proof −
have *D-in*: *D* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}')
using *invars*
by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)

have $\mathcal{F}'$ |⊆| *N* |∪| U_{er}'
using *invars*
by (*metis implicitly-factorized-clauses-subset-def ord-res-5-invars-def*)

moreover have *C* ≼$_c$ *D*
using *C-least D-in*
by (*metis linorder-cls.dual-order.strict-iff-order linorder-cls.is-least-in-fset-iff linorder-cls.le-cases*)

moreover have ∀ *F*. *is-least-false-clause* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}')) *F* ⟶ *D* ≼$_c$ *F*
using *invars le-least-false-clause* **by** *metis*

ultimately obtain $\mathcal{M}''$ **where**
steps: (*ord-res-5 N*)** (U_{er}', $\mathcal{F}'$, *Map.empty*, *Some C*) (U_{er}', $\mathcal{F}'$, $\mathcal{M}''$, *Some D*)
using *C-least D-in*
by (*metis ex-model-build-from-least-clause-to-any-less-than-least-false*)

have *ord-res-5-invars N* (U_{er}', $\mathcal{F}'$, *Map.empty*, *Some C*)
using ‹$\mathcal{F}'$ |⊆| *N* |∪| U_{er}'› *C-least ord-res-5-invars-initial-state* **by** *metis*

hence *ord-res-5-invars N* (U_{er}', $\mathcal{F}'$, $\mathcal{M}''$, *Some D*)
using ‹(*ord-res-5 N*)** (U_{er}', $\mathcal{F}'$, λ*x*. *None*, *Some C*) (U_{er}', $\mathcal{F}'$, $\mathcal{M}''$, *Some D*)›
rtranclp-ord-res-5-preserves-invars **by** *metis*

hence $\mathcal{M}''$*-spec*:
dom $\mathcal{M}''$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}'))) *D*
∀ *A C*. $\mathcal{M}''$ *A* = *Some C* ⟶ *A* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}'))) *C*
∀ *C A*. *C* ≺$_c$ *D* ⟶ *A* ∈ *ord-res.production* (*fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}'))) *C* ⟶ $\mathcal{M}''$ *A* = *Some C*
unfolding *ord-res-5-invars-def*
unfolding *model-eq-interp-upto-next-clause-def atoms-in-model-were-produced-def all-produced-atoms-in-model-def*
by *metis*+

have $\mathcal{M}'$ = $\mathcal{M}''$

proof (*cases* $D = \{\#\}$)
case *True*
have $\mathcal{M}' = Map.empty$
proof −
have $\nexists K.$ *ord-res.is-maximal-lit* K $D \wedge A \prec_t$ *atm-of* K **for** A
unfolding ‹$D = \{\#\}$›
by (*simp add*: *linorder-lit.is-maximal-in-mset-iff*)
hence $\{A.\ \exists K.$ *ord-res.is-maximal-lit* K $D \wedge A \prec_t$ *atm-of* $K\} = \{\}$
by *simp*
thus *?thesis*
unfolding $\mathcal{M}'$*-def* **by** *auto*
qed

also have $Map.empty = \mathcal{M}''$
proof −
have *ord-res.interp* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}'))) $D = \{\}$
unfolding ‹$D = \{\#\}$› **by** *simp*
thus *?thesis*
using $\mathcal{M}''$*-spec*(*1*) **by** *simp*
qed

finally show *?thesis* **.**
next
case *False*
then obtain K **where** *ord-res.is-maximal-lit* K D
using *linorder-lit.ex-maximal-in-mset* **by** *metis*
hence $\{A.\ \exists K.$ *ord-res.is-maximal-lit* K $D \wedge A \prec_t$ *atm-of* $K\} = \{A.\ A \prec_t$ *atm-of* $K\}$
by (*metis* (*no-types*, *lifting*) *linorder-lit.Uniq-is-maximal-in-mset the1-equality'*)
hence $\mathcal{M}'$*-def'*: $\mathcal{M}' =$ *restrict-map* $\mathcal{M}$ $\{A.\ A \prec_t$ *atm-of* $K\}$
unfolding $\mathcal{M}'$*-def* **by** *argo*

show *?thesis*
proof (*intro ext*)
fix x
have $\mathcal{M}'$*-spec*:
dom $\mathcal{M}'$ = *ord-res.interp* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}'))) D
$\forall A\ C.\ \mathcal{M}'\ A =$ *Some* $C \longrightarrow A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}'))) C
$\forall C\ A.\ C \prec_c D \longrightarrow A \in$ *ord-res.production* (*fset* (*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}'))) $C \longrightarrow \mathcal{M}'\ A =$ *Some* C
using *invars*
unfolding *ord-res-5-invars-def*
unfolding *model-eq-interp-upto-next-clause-def atoms-in-model-were-produced-def all-produced-atoms-in-model-def*
by *metis+*

have *dom* $\mathcal{M}'$ = *dom* $\mathcal{M}''$
using $\mathcal{M}'$*-spec*(*1*) $\mathcal{M}''$*-spec*(*1*) **by** *argo*

moreover have $\forall A\ C.\ \mathcal{M}'\ A = \mathit{Some}\ C \longleftrightarrow \mathcal{M}''\ A = \mathit{Some}\ C$
using $\mathcal{M}'$-*spec*(2) $\mathcal{M}''$-*spec*(2)
by (*smt* (*verit*, *del-insts*) *calculation domD domI linorder-cls.less-irrefl linorder-cls.neqE*
ord-res.less-trm-iff-less-cls-if-mem-production)

ultimately show $\mathcal{M}'\ x = \mathcal{M}''\ x$
by (*metis domD domIff*)
qed
qed

thus *?thesis*
using ‹(*ord-res-5 N*)** (U_{er}', $\mathcal{F}'$, *Map.empty*, *Some C*) (U_{er}', $\mathcal{F}'$, $\mathcal{M}''$, *Some D*)› **by** *argo*
qed

end

end
theory *ORD-RES-7*
imports
Background
Implicit-Exhaustive-Factorization
Exhaustive-Resolution
begin

22 ORD-RES-7 (clause-guided literal trail construction)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-7* **where**
decide-neg:
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
linorder-trm.is-least-in-fset {|*A* |∈| *atms-of-clss* (*N* |∪| U_{er}).
A ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ|} *A* ⟹
Γ′ = (*Neg A*, *None*) # Γ ⟹
ord-res-7 N (U_{er}, $\mathcal{F}$, Γ, *Some C*) (U_{er}, $\mathcal{F}$, Γ′, *Some C*) |

skip-defined:
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
¬(∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ) ⟹
trail-defined-lit Γ *L* ⟹
$\mathcal{C}'$ = *The-optional* (*linorder-cls.is-least-in-fset* {|*D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}).
C ≺$_c$ *D*|}) ⟹

ord-res-7 N $(U_{er}, \mathcal{F}, \Gamma,$ *Some C*$)$ $(U_{er}, \mathcal{F}, \Gamma, \mathcal{C}')$ |

skip-undefined-neg:
¬ *trail-false-cls* Γ $C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
$\neg(\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).$ $A \prec_t$ *atm-of* $L \wedge A \mid\notin\mid$ *trail-atms* $\Gamma) \Longrightarrow$
¬ *trail-defined-lit* Γ $L \Longrightarrow$
is-neg $L \Longrightarrow$
$\Gamma' = (L,$ *None*$)$ # $\Gamma \Longrightarrow$
$\mathcal{C}' =$ *The-optional* (*linorder-cls.is-least-in-fset* $\{|D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ $C \prec_c D|\}) \Longrightarrow$
ord-res-7 N $(U_{er}, \mathcal{F}, \Gamma,$ *Some C*$)$ $(U_{er}, \mathcal{F}, \Gamma', \mathcal{C}')$ |

skip-undefined-pos:
¬ *trail-false-cls* Γ $C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
$\neg(\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).$ $A \prec_t$ *atm-of* $L \wedge A \mid\notin\mid$ *trail-atms* $\Gamma) \Longrightarrow$
¬ *trail-defined-lit* Γ $L \Longrightarrow$
is-pos $L \Longrightarrow$
¬ *trail-false-cls* Γ $\{\#K \in\# C.$ $K \neq L\#\} \Longrightarrow$
linorder-cls.is-least-in-fset $\{|D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ $C \prec_c D|\}$ $D \Longrightarrow$
ord-res-7 N $(U_{er}, \mathcal{F}, \Gamma,$ *Some C*$)$ $(U_{er}, \mathcal{F}, \Gamma,$ *Some D*$)$ |

skip-undefined-pos-ultimate:
¬ *trail-false-cls* Γ $C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
$\neg(\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).$ $A \prec_t$ *atm-of* $L \wedge A \mid\notin\mid$ *trail-atms* $\Gamma) \Longrightarrow$
¬ *trail-defined-lit* Γ $L \Longrightarrow$
is-pos $L \Longrightarrow$
¬ *trail-false-cls* Γ $\{\#K \in\# C.$ $K \neq L\#\} \Longrightarrow$
$\Gamma' = (-$ $L,$ *None*$)$ # $\Gamma \Longrightarrow$
$\neg(\exists D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ $C \prec_c D) \Longrightarrow$
ord-res-7 N $(U_{er}, \mathcal{F}, \Gamma,$ *Some C*$)$ $(U_{er}, \mathcal{F}, \Gamma',$ *None*$)$ |

production:
¬ *trail-false-cls* Γ $C \Longrightarrow$
linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
¬ $(\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).$ $A \prec_t$ *atm-of* $L \wedge A \mid\notin\mid$ *trail-atms* $\Gamma) \Longrightarrow$
¬ *trail-defined-lit* Γ $L \Longrightarrow$
is-pos $L \Longrightarrow$
trail-false-cls Γ $\{\#K \in\# C.$ $K \neq L\#\} \Longrightarrow$
linorder-lit.is-greatest-in-mset C $L \Longrightarrow$
$\Gamma' = (L,$ *Some C*$)$ # $\Gamma \Longrightarrow$
$\mathcal{C}' =$ *The-optional* (*linorder-cls.is-least-in-fset* $\{|D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ $C \prec_c D|\}) \Longrightarrow$
ord-res-7 N $(U_{er}, \mathcal{F}, \Gamma,$ *Some C*$)$ $(U_{er}, \mathcal{F}, \Gamma', \mathcal{C}')$ |

factoring:
¬ *trail-false-cls* Γ $C \Longrightarrow$

linorder-lit.is-maximal-in-mset C $L \Longrightarrow$
$\neg$ ($\exists$ A $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $L \wedge$ A $|\notin|$ *trail-atms* Γ) $\Longrightarrow$
$\neg$ *trail-defined-lit* Γ $L \Longrightarrow$
is-pos $L \Longrightarrow$
trail-false-cls Γ $\{\#K \in\# C.\ K \neq L\#\} \Longrightarrow$
$\neg$ *linorder-lit.is-greatest-in-mset* C $L \Longrightarrow$
$\mathcal{F}' =$ *finsert* C $\mathcal{F} \Longrightarrow$
ord-res-7 N (U_{er}, $\mathcal{F}$, Γ, *Some* C) (U_{er}, $\mathcal{F}'$, Γ, *Some* (*efac* C)) |

resolution-bot:
trail-false-cls Γ $E \Longrightarrow$
linorder-lit.is-maximal-in-mset E $L \Longrightarrow$
is-neg $L \Longrightarrow$
map-of Γ ($-$ L) = *Some* (*Some* D) $\Longrightarrow$
$U_{er}' =$ *finsert* (*eres* D E) $U_{er} \Longrightarrow$
eres D $E = \{\#\} \Longrightarrow$
$\Gamma' = [] \Longrightarrow$
ord-res-7 N (U_{er}, $\mathcal{F}$, Γ, *Some* E) (U_{er}', $\mathcal{F}$, Γ', *Some* $\{\#\}$) |

resolution-pos:
trail-false-cls Γ $E \Longrightarrow$
linorder-lit.is-maximal-in-mset E $L \Longrightarrow$
is-neg $L \Longrightarrow$
map-of Γ ($-$ L) = *Some* (*Some* D) $\Longrightarrow$
$U_{er}' =$ *finsert* (*eres* D E) $U_{er} \Longrightarrow$
eres D $E \neq \{\#\} \Longrightarrow$
$\Gamma' =$ *dropWhile* ($\lambda Ln.$ *atm-of* $K \preceq_t$ *atm-of* (*fst* Ln)) $\Gamma \Longrightarrow$
linorder-lit.is-maximal-in-mset (*eres* D E) $K \Longrightarrow$
is-pos $K \Longrightarrow$
ord-res-7 N (U_{er}, $\mathcal{F}$, Γ, *Some* E) (U_{er}', $\mathcal{F}$, Γ', *Some* (*eres* D E)) |

resolution-neg:
trail-false-cls Γ $E \Longrightarrow$
linorder-lit.is-maximal-in-mset E $L \Longrightarrow$
is-neg $L \Longrightarrow$
map-of Γ ($-$ L) = *Some* (*Some* D) $\Longrightarrow$
$U_{er}' =$ *finsert* (*eres* D E) $U_{er} \Longrightarrow$
eres D $E \neq \{\#\} \Longrightarrow$
$\Gamma' =$ *dropWhile* ($\lambda Ln.$ *atm-of* $K \preceq_t$ *atm-of* (*fst* Ln)) $\Gamma \Longrightarrow$
linorder-lit.is-maximal-in-mset (*eres* D E) $K \Longrightarrow$
is-neg $K \Longrightarrow$
map-of Γ ($-$ K) = *Some* (*Some* C) $\Longrightarrow$
ord-res-7 N (U_{er}, $\mathcal{F}$, Γ, *Some* E) (U_{er}', $\mathcal{F}$, Γ', *Some* C)

lemma *right-unique-ord-res-7*:
fixes N :: *'f gclause fset*
shows *right-unique* (*ord-res-7* N)
proof (*rule right-uniqueI*)

fix *x y z*
assume *step1*: *ord-res-7 N x y* **and** *step2*: *ord-res-7 N x z*
show *y = z*
using *step1*
proof (*cases N x y rule*: *ord-res-7.cases*)
case *step-hyps1*: (*decide-neg* Γ *C L* U_{er} *A* Γ′ $\mathcal{F}$)
show *?thesis*
using *step2*
unfolding *step-hyps1*(*1*)
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, *Some C*) *z rule*: *ord-res-7.cases*)
case *step-hyps2*: (*decide-neg L2 A2* Γ*2′*)
have *L2 = L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*of C*, *THEN Uniq-D*] **by** *metis*
have *A2 = A*
using *step-hyps1 step-hyps2*
unfolding ‹*L2 = L*›
using *linorder-trm.Uniq-is-least-in-fset*[*THEN Uniq-D*] **by** *presburger*
have Γ*2′* = Γ′
using *step-hyps1 step-hyps2*
unfolding ‹*L2 = L*› ‹*A2 = A*› **by** *metis*
show *?thesis*
unfolding *step-hyps1*(*2*) *step-hyps2*(*1*)
unfolding ‹Γ*2′* = Γ′› **..**
next
case *step-hyps2*: (*skip-defined L2* $\mathcal{C}$*2′*)
have *L2 = L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2 = L*›
unfolding *linorder-trm.is-least-in-ffilter-iff* **by** *metis*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-neg L2* Γ*2′* $\mathcal{C}$*2′*)
have *L2 = L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2 = L*›
unfolding *linorder-trm.is-least-in-ffilter-iff* **by** *metis*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-pos L2 D2*)
have *L2 = L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*

```
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (production L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (factoring L2 F2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
  have False
    using step-hyps1 step-hyps2 by argo
```

```
    then show ?thesis ..
  qed
next
  case step-hyps1: (skip-defined Γ C L U_er C' F)
  show ?thesis
    using step2
    unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (skip-defined L2 C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have C2' = C'
      using step-hyps1 step-hyps2 by argo
    show ?thesis
      unfolding step-hyps1(2) step-hyps2(1)
      unfolding ‹C2' = C'› ..
  next
    case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-pos L2 D2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
```

```
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (production L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (factoring L2 F2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (skip-undefined-neg Γ C L U_er Γ' C' F)
  show ?thesis
    using step2 unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have L2 = L
```

```
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (skip-defined L2 C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have Γ2' = Γ'
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  have C2' = C'
    using step-hyps1 step-hyps2 by argo
  show ?thesis
    unfolding step-hyps1(2) step-hyps2(1)
    unfolding ‹C2' = C'› ‹Γ2' = Γ'› ..
next
  case step-hyps2: (skip-undefined-pos L2 D2)
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    unfolding linorder-trm.is-least-in-ffilter-iff by metis
  then show ?thesis ..
```

```
  next
    case step-hyps2: (production L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (factoring L2 F2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (skip-undefined-pos Γ C L U_er F D)
  show ?thesis
    using step2 unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
```

```
  case step-hyps2: (skip-defined L2 C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos L2 D2)
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have D2 = D
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    by (meson Uniq-D linorder-cls.Uniq-is-least-in-fset)
  show ?thesis
    unfolding step-hyps1(2) step-hyps2(1)
    unfolding ‹D2 = D› ..
next
  case step-hyps2: (skip-undefined-pos-ultimate L2 Γ')
  have False
    using step-hyps1 step-hyps2
    by (metis linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
next
  case step-hyps2: (production L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (factoring L2 F2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
```

```
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (skip-undefined-pos-ultimate Γ C L U_er Γ' F)
  show ?thesis
    using step2 unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (skip-defined L2 C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
```

using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-pos L2 D2*)
have *False*
using *step-hyps1 step-hyps2*
by (*metis linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-pos-ultimate L2* Γ*2′*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have Γ*2′* = Γ′
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
show *?thesis*
unfolding *step-hyps1*(*2*) *step-hyps2*(*1*)
unfolding ‹Γ*2′* = Γ′› **..**
next
case *step-hyps2*: (*production L2* Γ*2′* 𝒞*2′*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*factoring L2* ℱ*2′*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*resolution-bot L2 D2* U_{er}*2′* Γ*2′*)
have *False*
using *step-hyps1 step-hyps2* **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*resolution-pos L2 D2* U_{er}*2′* Γ*2′ K2*)
have *False*
using *step-hyps1 step-hyps2* **by** *argo*
then show *?thesis* **..**
next

```
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (production Γ C L U_er Γ' C' F)
  show ?thesis
    using step2
    unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (skip-defined L2 C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-pos L2 D2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
```

```
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    then show ?thesis ..
  next
    case step-hyps2: (production L2 Γ2' 𝒞2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have Γ2' = Γ'
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
    have 𝒞2' = 𝒞'
      using step-hyps1 step-hyps2 by argo
    show ?thesis
      unfolding step-hyps1(2) step-hyps2(1)
      unfolding ‹Γ2' = Γ'› ‹𝒞2' = 𝒞'› ..
  next
    case step-hyps2: (factoring L2 ℱ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (factoring Γ C L U_er ℱ' ℱ)
  show ?thesis
```

using *step2*
unfolding *step-hyps1*(*1*)
proof (*cases N* (*U*$_{er}$, $\mathcal{F}$, Γ, *Some C*) *z rule*: *ord-res-7.cases*)
case *step-hyps2*: (*decide-neg L2 A2* Γ*2'*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*of C*, *THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*›
unfolding *linorder-trm.is-least-in-ffilter-iff* **by** *metis*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-defined L2* $\mathcal{C}$*2'*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-neg L2* Γ*2'* $\mathcal{C}$*2'*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-pos L2 D2*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*skip-undefined-pos-ultimate L2* Γ*2'*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*› **by** *argo*
then show *?thesis* **..**
next

```
    case step-hyps2: (production L2 Γ2' C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of C, THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      unfolding linorder-trm.is-least-in-ffilter-iff by metis
    then show ?thesis ..
  next
    case step-hyps2: (factoring L2 F2')
    have F2' = F'
      using step-hyps1 step-hyps2
      by argo
    show ?thesis
      unfolding step-hyps1(2) step-hyps2(1)
      unfolding ‹F2' = F'› ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (resolution-bot Γ E L D U_er' U_er Γ' F)
  show ?thesis
    using step2
    unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some E) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-defined L2 C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
```

```
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos L2 D2)
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (production L2 Γ2' C2')
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (factoring L2 F2')
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
  have U_er2' = U_er'
    using step-hyps1 step-hyps2
    by argo
  have Γ2' = Γ'
    using step-hyps1 step-hyps2
    by argo
  show ?thesis
```

```
    unfolding step-hyps1(2) step-hyps2(1)
    unfolding ‹U_er2' = U_er'› ‹Γ2' = Γ'› ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
    have D2 = D
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      by (metis option.inject)
    have False
      using step-hyps1 step-hyps2
      unfolding ‹D2 = D›
      by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
    have D2 = D
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      by (metis option.inject)
    have False
      using step-hyps1 step-hyps2
      unfolding ‹D2 = D›
      by argo
    then show ?thesis ..
  qed
next
  case step-hyps1: (resolution-pos Γ E L D U_er' U_er Γ' K F)
  show ?thesis
    using step2
    unfolding step-hyps1(1)
  proof (cases N (U_er, F, Γ, Some E) z rule: ord-res-7.cases)
    case step-hyps2: (decide-neg L2 A2 Γ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (skip-defined L2 C2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L› by argo
```

```
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-neg L2 Γ2' C2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos L2 D2)
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (skip-undefined-pos-ultimate L2 Γ2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
  have False
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L› by argo
  then show ?thesis ..
next
  case step-hyps2: (production L2 Γ2' C2')
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (factoring L2 F2')
  have False
    using step-hyps1 step-hyps2 by argo
  then show ?thesis ..
next
  case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
  have L2 = L
    using step-hyps1 step-hyps2
    using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
  have D2 = D
    using step-hyps1 step-hyps2
    unfolding ‹L2 = L›
    by (metis option.inject)
  have False
    using step-hyps1 step-hyps2
```

unfolding ‹*D2* = *D*›
by *argo*
then show *?thesis* **..**
next
case *step-hyps2*: (*resolution-pos L2 D2* $U_{er}2'$ Γ*2′ K2*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*of E*, *THEN Uniq-D*] **by** *metis*
have *D2* = *D*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*›
by (*metis option.inject*)
have *K2* = *K*
using *step-hyps1 step-hyps2*
unfolding ‹*D2* = *D*›
using *linorder-lit.Uniq-is-maximal-in-mset*[*of eres D E*, *THEN Uniq-D*] **by** *metis*
have $U_{er}2' = U_{er}'$
using *step-hyps1 step-hyps2*
unfolding ‹*D2* = *D*›
by *argo*
have Γ*2′* = Γ′
using *step-hyps1 step-hyps2*
unfolding ‹*K2* = *K*›
by *argo*
show *?thesis*
unfolding *step-hyps1*(*2*) *step-hyps2*(*1*)
unfolding ‹$U_{er}2' = U_{er}'$› ‹Γ*2′* = Γ′› ‹*D2* = *D*› **..**
next
case *step-hyps2*: (*resolution-neg L2 D2* $U_{er}2'$ Γ*2′ K2 C2*)
have *L2* = *L*
using *step-hyps1 step-hyps2*
using *linorder-lit.Uniq-is-maximal-in-mset*[*of E*, *THEN Uniq-D*] **by** *metis*
have *D2* = *D*
using *step-hyps1 step-hyps2*
unfolding ‹*L2* = *L*›
by (*metis option.inject*)
have *K2* = *K*
using *step-hyps1 step-hyps2*
unfolding ‹*D2* = *D*›
using *linorder-lit.Uniq-is-maximal-in-mset*[*of eres D E*, *THEN Uniq-D*] **by** *metis*
have *False*
using *step-hyps1 step-hyps2*
unfolding ‹*K2* = *K*›
by *argo*
then show *?thesis* **..**
qed
next

case *step-hyps1*: (*resolution-neg* Γ *E L D* U_{er}' U_{er} Γ′ *K C* $\mathcal{F}$)
show *?thesis*
  using *step2*
  unfolding *step-hyps1*(*1*)
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, *Some E*) *z rule*: *ord-res-7.cases*)
  case *step-hyps2*: (*decide-neg L2 A2* Γ*2′*)
  have *False*
    using *step-hyps1 step-hyps2* **by** *argo*
  then show *?thesis* **..**
next
  case *step-hyps2*: (*skip-defined L2* $\mathcal{C}$*2′*)
  have *L2* = *L*
    using *step-hyps1 step-hyps2*
    using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
  have *False*
    using *step-hyps1 step-hyps2*
    unfolding ‹*L2* = *L*› **by** *argo*
  then show *?thesis* **..**
next
  case *step-hyps2*: (*skip-undefined-neg L2* Γ*2′* $\mathcal{C}$*2′*)
  have *L2* = *L*
    using *step-hyps1 step-hyps2*
    using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
  have *False*
    using *step-hyps1 step-hyps2*
    unfolding ‹*L2* = *L*› **by** *argo*
  then show *?thesis* **..**
next
  case *step-hyps2*: (*skip-undefined-pos L2 D2*)
  have *L2* = *L*
    using *step-hyps1 step-hyps2*
    using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
  have *False*
    using *step-hyps1 step-hyps2*
    unfolding ‹*L2* = *L*› **by** *argo*
  then show *?thesis* **..**
next
  case *step-hyps2*: (*skip-undefined-pos-ultimate L2* Γ*2′*)
  have *L2* = *L*
    using *step-hyps1 step-hyps2*
    using *linorder-lit.Uniq-is-maximal-in-mset*[*THEN Uniq-D*] **by** *metis*
  have *False*
    using *step-hyps1 step-hyps2*
    unfolding ‹*L2* = *L*› **by** *argo*
  then show *?thesis* **..**
next
  case *step-hyps2*: (*production L2* Γ*2′* $\mathcal{C}$*2′*)
  have *False*
    using *step-hyps1 step-hyps2* **by** *argo*

```
    then show ?thesis ..
  next
    case step-hyps2: (factoring L2 ℱ2')
    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-bot L2 D2 U_er2' Γ2')
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
    have D2 = D
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      by (metis option.inject)
    have False
      using step-hyps1 step-hyps2
      unfolding ‹D2 = D›
      by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-pos L2 D2 U_er2' Γ2' K2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
    have D2 = D
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      by (metis option.inject)
    have K2 = K
      using step-hyps1 step-hyps2
      unfolding ‹D2 = D›
      using linorder-lit.Uniq-is-maximal-in-mset[of eres D E, THEN Uniq-D] by
metis
    have False
      using step-hyps1 step-hyps2
      unfolding ‹K2 = K›
      by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 U_er2' Γ2' K2 C2)
    have L2 = L
      using step-hyps1 step-hyps2
      using linorder-lit.Uniq-is-maximal-in-mset[of E, THEN Uniq-D] by metis
    have D2 = D
      using step-hyps1 step-hyps2
      unfolding ‹L2 = L›
      by (metis option.inject)
    have K2 = K
```

using *step-hyps1 step-hyps2*
unfolding ‹*D2* = *D*›
using *linorder-lit.Uniq-is-maximal-in-mset*[*of eres D E*, *THEN Uniq-D*] **by** *metis*
have $U_{er}2' = U_{er}'$
using *step-hyps1 step-hyps2*
unfolding ‹*D2* = *D*›
by *argo*
have $\Gamma 2' = \Gamma'$
using *step-hyps1 step-hyps2*
unfolding ‹*K2* = *K*›
by *argo*
have *C2* = *C*
using *step-hyps1 step-hyps2*
unfolding ‹*K2* = *K*›
by (*metis option.inject*)
show *?thesis*
unfolding *step-hyps1*(*2*) *step-hyps2*(*1*)
unfolding ‹$U_{er}2' = U_{er}'$› ‹$\Gamma 2' = \Gamma'$› ‹*C2* = *C*› **..**
qed
qed
qed

inductive *ord-res-7-final* **where**
model-found:
ord-res-7-final (*N*, U_{er}, $\mathcal{F}$, Γ, *None*) |

contradiction-found:
ord-res-7-final (*N*, U_{er}, $\mathcal{F}$, Γ, *Some* {#})

sublocale *ord-res-7-semantics*: *semantics* **where**
step = *constant-context ord-res-7* **and**
final = *ord-res-7-final*
proof *unfold-locales*
fix *S* :: *'f gclause fset* ×*'f gclause fset* × *'f gclause fset* ×
(*'f gterm literal* × *'f gterm literal multiset option*) *list* ×
'f gterm literal multiset option

obtain
N U_{er} $\mathcal{F}$ :: *'f gterm clause fset* **and**
Γ :: (*'f gterm literal* × *'f gterm literal multiset option*) *list* **and**
$\mathcal{C}$ :: *'f gclause option*
where
S-def: *S* = (*N*, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)
by (*metis prod.exhaust*)

assume *ord-res-7-final S*

hence ∄*x*. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) *x*

unfolding *S-def*
proof (*cases* (N, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) *rule*: *ord-res-7-final.cases*)
case *model-found*
thus *?thesis*
by (*auto elim*: *ord-res-7.cases*)
next
case *contradiction-found*
thus *?thesis*
by (*auto simp*: *linorder-lit.is-maximal-in-mset-iff elim*: *ord-res-7.cases*)
qed

thus *finished* (*constant-context ord-res-7*) *S*
by (*simp add*: *S-def finished-def constant-context.simps*)
qed

inductive *ord-res-7-invars* **for** N **where**
ord-res-7-invars N (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) **if**
$\mathcal{F}$ |⊆| N |∪| U_{er} **and**
(∀ C. $\mathcal{C}$ = *Some* C ⟶ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) **and**
(∀ D. $\mathcal{C}$ = *Some* D ⟶
(∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). C ≺$_c$ D ⟶
(∀ L_C. *linorder-lit.is-maximal-in-mset* C L_C ⟶
¬ *trail-defined-lit* Γ L_C ⟶ (*trail-true-cls* Γ {#K ∈# C. K ≠ L_C#} ∧ *is-pos* L_C)))) **and**
(∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). (∀ D. $\mathcal{C}$ = *Some* D ⟶ C ≺$_c$ D) ⟶ *trail-true-cls* Γ C) **and**
(∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). (∀ D. $\mathcal{C}$ = *Some* D ⟶ C ≺$_c$ D) ⟶
(∀ K. *linorder-lit.is-maximal-in-mset* C K ⟶
¬ (∃ A |∈| *atms-of-clss* (N |∪| U_{er}). A ≺$_t$ *atm-of* K ∧ A |∉| *trail-atms* Γ))) **and**
sorted-wrt (λx y. *atm-of* (*fst* y) ≺$_t$ *atm-of* (*fst* x)) Γ **and**
linorder-trm.is-lower-fset (*trail-atms* Γ) (*atms-of-clss* (N |∪| U_{er})) **and**
($\mathcal{C}$ = *None* ⟶ *trail-atms* Γ = *atms-of-clss* (N |∪| U_{er})) **and**
(∀ C. $\mathcal{C}$ = *Some* C ⟶
(∀ A |∈| *trail-atms* Γ. ∃ L. *linorder-lit.is-maximal-in-mset* C L ∧ A ≼$_t$ *atm-of* L)) **and**
(∀ Ln ∈ *set* Γ. *is-neg* (*fst* Ln) ⟷ *snd* Ln = *None*) **and**
(∀ Ln ∈ *set* Γ. *snd* Ln = *None* ⟶
(∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). C ≺$_c$ {#*fst* Ln#} ⟶ *trail-true-cls* Γ C)) **and**
(∀ Ln ∈ *set* Γ. ∀ C. *snd* Ln = *Some* C ⟶ *linorder-lit.is-greatest-in-mset* C (*fst* Ln)) **and**
(∀ Ln ∈ *set* Γ. ∀ C. *snd* Ln = *Some* C ⟶ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) **and**
(∀ Γ_1 L C Γ_0. Γ = Γ_1 @ (L, *Some* C) # Γ_0 ⟶ *trail-false-cls* Γ_0 {#K ∈# C. K ≠ L#}) **and**
(∀ Γ_1 L D Γ_0. Γ = Γ_1 @ (L, *Some* D) # Γ_0 ⟶
(∀ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). C ≺$_c$ D ⟶ *trail-true-cls* Γ_0 C))

lemma *clause-almost-defined-if-lt-next-clause*:

```
  assumes ord-res-7-invars N (U_er, F, Γ, C)
  shows ∀ C |∈| iefac F |‘| (N |∪| U_er). (∀ D. C = Some D ⟶ C ≺_c D) ⟶
    (∀ K. linorder-lit.is-maximal-in-mset C K ⟶ trail-defined-cls Γ {#L ∈# C. L
≠ K#})
proof (intro ballI impI allI)
  fix C :: 'f gclause and K :: 'f gliteral
  assume
    C-in: C |∈| iefac F |‘| (N |∪| U_er) and
    C-lt: ∀ D. C = Some D ⟶ C ≺_c D and
    C-max-lit: ord-res.is-maximal-lit K C

  show trail-defined-cls Γ {#L ∈# C. L ≠ K#}
    using assms
  proof (cases N (U_er, F, Γ, C) rule: ord-res-7-invars.cases)
    case invars: 1

    have ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of K ∧ A |∉| trail-atms Γ)
      using invars C-in C-lt C-max-lit by metis

    hence C-almost-almost-defined: trail-defined-cls Γ {#L ∈# C. L ≠ K ∧ L ≠
- K#}
      using clause-almost-almost-definedI[OF C-in C-max-lit] by blast

    show ?thesis
    proof (cases trail-defined-lit Γ K)
      case True
      hence trail-defined-lit Γ (- K)
        unfolding trail-defined-lit-def uminus-of-uminus-id by argo
      then show ?thesis
        using C-almost-almost-defined
        unfolding trail-defined-cls-def
        by auto
    next
      case False
      show ?thesis
      proof (cases C)
        case None
        hence trail-atms Γ = atms-of-clss (N |∪| U_er)
          using invars by argo
        then show ?thesis
       by (metis C-in C-max-lit False atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff
              trail-defined-lit-iff-trail-defined-atm)
      next
        case (Some D)
        hence C ≺_c D
          using C-lt by simp
        then show ?thesis
          using invars
            by (smt (verit, ccfv-SIG) C-almost-almost-defined C-in C-max-lit False
```

Some
⟨¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| *U*$_{er}$). *A* ≺$_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ)⟩
atm-of-in-atms-of-clssI atm-of-uminus filter-mset-cong0
linorder-lit.is-greatest-in-set-iff linorder-lit.is-maximal-in-mset-iff
linorder-lit.is-maximal-in-set-eq-is-greatest-in-set
linorder-lit.is-maximal-in-set-iff literal.collapse(*1*) *literal.exhaust-sel*
ord-res.less-lit-simps(*4*) *trail-defined-lit-iff-trail-defined-atm*)
qed
qed
qed
qed

lemma *ord-res-7-invars-def*:
ord-res-7-invars N s ⟷
(∀ *U*$_{er}$ *F* Γ *C*. *s* = (*U*$_{er}$, *F*, Γ, *C*) ⟶
F |⊆| *N* |∪| *U*$_{er}$ ∧
(∀ *C*. *C* = *Some C* ⟶ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$)) ∧
(∀ *D*. *C* = *Some D* ⟶
(∀ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). *C* ≺$_c$ *D* ⟶
(∀ *L*$_C$. *linorder-lit.is-maximal-in-mset C L*$_C$ ⟶
¬ *trail-defined-lit* Γ *L*$_C$ ⟶ (*trail-true-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*$_C$#} ∧ *is-pos L*$_C$)))) ∧
(∀ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). (∀ *D*. *C* = *Some D* ⟶ *C* ≺$_c$ *D*) ⟶ *trail-true-cls* Γ *C*) ∧
(∀ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). (∀ *D*. *C* = *Some D* ⟶ *C* ≺$_c$ *D*) ⟶
(∀ *K*. *linorder-lit.is-maximal-in-mset C K* ⟶
¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$). *A* ≺$_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ))) ∧
sorted-wrt (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ ∧
linorder-trm.is-lower-fset (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| *U*$_{er}$)) ∧
(*C* = *None* ⟶ *trail-atms* Γ = *atms-of-clss* (*N* |∪| *U*$_{er}$)) ∧
(∀ *C*. *C* = *Some C* ⟶
(∀ *A* |∈| *trail-atms* Γ. ∃ *L*. *linorder-lit.is-maximal-in-mset C L* ∧ *A* ≼$_t$ *atm-of L*)) ∧
(∀ *Ln* ∈ *set* Γ. *is-neg* (*fst Ln*) ⟷ *snd Ln* = *None*) ∧
(∀ *Ln* ∈ *set* Γ. *snd Ln* = *None* ⟶
(∀ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). *C* ≺$_c$ {#*fst Ln*#} ⟶ *trail-true-cls* Γ *C*)) ∧
(∀ *Ln* ∈ *set* Γ. ∀ *C*. *snd Ln* = *Some C* ⟶ *linorder-lit.is-greatest-in-mset C* (*fst Ln*)) ∧
(∀ *Ln* ∈ *set* Γ. ∀ *C*. *snd Ln* = *Some C* ⟶ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$)) ∧
(∀Γ$_1$ *L C* Γ$_0$. Γ = Γ$_1$ @ (*L*, *Some C*) # Γ$_0$ ⟶ *trail-false-cls* Γ$_0$ {#*K* ∈# *C*. *K* ≠ *L*#}) ∧
(∀Γ$_1$ *L D* Γ$_0$. Γ = Γ$_1$ @ (*L*, *Some D*) # Γ$_0$ ⟶
(∀ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). *C* ≺$_c$ *D* ⟶ *trail-true-cls* Γ$_0$ *C*)))
(**is** *?NICE N s* ⟷ *?UGLY N s*)
proof (*rule iffI*)
show *?NICE N s* ⟹ *?UGLY N s*

```
    apply (intro allI impI)
    subgoal premises prems for U_er F Γ C
      using prems(1) unfolding prems(2)
      by (cases N (U_er, F, Γ, C) rule: ord-res-7-invars.cases) (simp only:)
    done
next
  assume ?UGLY N s
  obtain U_er F Γ C where s-def: s = (U_er, F, Γ, C)
    by (metis prod.exhaust)
  show ?NICE N s
    using ‹?UGLY N s›[rule-format, OF s-def]
    unfolding s-def
    by (intro ord-res-7-invars.intros) simp-all
qed

lemma ord-res-7-invars-implies-trail-consistent:
  assumes ord-res-7-invars N (U_er, F, Γ, C)
  shows trail-consistent Γ
  using assms unfolding ord-res-7-invars-def
  by (metis trail-consistent-if-sorted-wrt-atoms)

lemma ord-res-7-invars-implies-propagated-clause-almost-false:
  assumes invars: ord-res-7-invars N (U_er, F, Γ, C) and (L, Some C) ∈ set Γ
  shows trail-false-cls Γ {#K ∈# C. K ≠ L#}
proof −
  obtain Γ₁ Γ₀ where Γ-eq: Γ = Γ₁ @ (L, Some C) # Γ₀
    using ‹(L, Some C) ∈ set Γ› by (metis split-list)

  hence trail-false-cls Γ₀ {#K ∈# C. K ≠ L#}
    using invars by (simp-all add: ord-res-7-invars-def)

  moreover have suffix Γ₀ Γ
    using Γ-eq by (simp add: suffix-def)

  ultimately show ?thesis
    by (metis trail-false-cls-if-trail-false-suffix)
qed

lemma ord-res-7-preserves-invars:
  assumes step: ord-res-7 N s s' and invar: ord-res-7-invars N s
  shows ord-res-7-invars N s'
  using step
proof (cases N s s' rule: ord-res-7.cases)
  case step-hyps: (decide-neg Γ D L U_er A Γ' F)

  note D-max-lit = ‹ord-res.is-maximal-lit L D›

  have
    A |∈| atms-of-clss (N |∪| U_er) and
```

$A \prec_t$ *atm-of* L **and**
$A \mid\notin\mid$ *trail-atms* Γ
using *step-hyps* **unfolding** *atomize-conj linorder-trm.is-least-in-ffilter-iff* **by** *argo*

have *suffix* $\Gamma\ \Gamma'$
using *step-hyps* **unfolding** *suffix-def* **by** *simp*

have
$\mathcal{F}$*-subset*: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**
D-in: $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall L_C.$ *linorder-lit.is-maximal-in-mset* $C\ L_C \longrightarrow$
$\neg$ *trail-defined-lit* $\Gamma\ L_C \longrightarrow$ (*trail-true-cls* Γ $\{\#K \in\#\ C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* Γ)) **and**
Γ*-sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$) **and**
trail-atms-le0: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ \exists L.$ *ord-res.is-maximal-lit* $L\ D \wedge (A \preceq_t$ *atm-of* L) **and**
Γ*-deci-iff-neg*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ*-deci-ball-lt-true*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c \{\#$*fst* $Ln\#\} \longrightarrow$ *trail-true-cls* $\Gamma\ C$) **and**
Γ*-prop-in*: $\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
Γ*-prop-greatest*:
$\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst* Ln) **and**
Γ*-prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* C) # $\Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\#\ C.\ K \neq L\#\}$ **and**
Γ*-prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* D) # $\Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* $\Gamma_0\ C$))
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$› *ord-res-*γ*-invars-def*)

have *trail-atms-le*: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ A \preceq_t$ *atm-of* L
using *trail-atms-le0* ‹*ord-res.is-maximal-lit* $L\ D$›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have *clauses-lt-D-almost-defined*: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$ *trail-defined-cls* Γ $\{\#L \in\#\ C.\ L$

$\neq K\#\}$)
  using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$›] *clause-almost-defined-if-lt-next-clause*
  by *simp*

 have $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$
  using $\mathcal{F}$*-subset* **.**

 moreover have $\forall C'.$ *Some* $D =$ *Some* $C' \longrightarrow C' \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
  using *D-in* **by** *simp*

 moreover have *trail-true-cls* $\Gamma' \{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
  if *Some* $D =$ *Some* E **and**
   C-in: $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
   C-lt: $C \prec_c E$ **and**
   C-max-lit: *linorder-lit.is-maximal-in-mset* $C\ L_C$ **and**
   L_C*-undef*: $\neg$ *trail-defined-lit* $\Gamma'\ L_C$
  for $E\ C\ L_C$
 proof −
  have $E = D$
   using *that* **by** *simp*
  hence $C \prec_c D$
   using *C-lt* **by** *argo*

  moreover have $\neg$ *trail-defined-lit* $\Gamma\ L_C$
   using L_C*-undef* ‹*suffix* $\Gamma\ \Gamma'$›
   using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

  ultimately have *trail-true-cls* $\Gamma\ \{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
  using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in - C-max-lit*]
by *metis*

  thus *?thesis*
   using ‹*suffix* $\Gamma\ \Gamma'$› *trail-true-cls-if-trail-true-suffix* **by** *blast*
 qed

 moreover have
  trail-true-cls $\Gamma'\ C$
  $\bigwedge K.$ *linorder-lit.is-maximal-in-mset* $C\ K \Longrightarrow$
   $\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma')$
  if *C-lt*: $\bigwedge D'.$ *Some* $D =$ *Some* $D' \Longrightarrow C \prec_c D'$ **and** $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **for** C
 proof −
  have $C \prec_c D$
   using *C-lt* **by** *metis*

  hence *trail-true-cls* $\Gamma\ C$
   using *clauses-lt-D-true* ‹$C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$› **by** *metis*

  thus *trail-true-cls* $\Gamma'\ C$

using ‹*suffix* Γ Γ′›
by (*metis trail-true-cls-if-trail-true-suffix*)

fix *K*
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset C K*

have ¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-D*
using ‹*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› ‹*C* ≺$_c$ *D*› *C-max-lit* **by** *metis*

thus ¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of K* ∧ *A* |∉| *trail-atms* Γ′)
using *step-hyps*(*6*) **by** *auto*
qed

moreover have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ′
proof −
have ∀ *x* |∈| *trail-atms* Γ. *x* ≺$_t$ *A*
using *step-hyps*(*5*)[*unfolded linorder-trm.is-least-in-ffilter-iff*, *simplified*]
by (*metis* Γ*-lower linorder-trm.antisym-conv3 linorder-trm.is-lower-set-iff*)

hence ∀ *x*∈*set* Γ. *atm-of* (*fst x*) ≺$_t$ *A*
by (*simp add*: *fset-trail-atms*)

thus *?thesis*
using Γ*-sorted* **by** (*simp add*: ‹Γ′ = (*Neg A*, *None*) # Γ›)
qed

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| U_{er}))
proof −
have *linorder-trm.is-lower-set* (*insert A* (*fset* (*trail-atms* Γ)))
(*fset* (*atms-of-clss* (*N* |∪| U_{er})))
proof (*rule linorder-trm.is-lower-set-insertI*)
show *A* |∈| *atms-of-clss* (*N* |∪| U_{er})
using *step-hyps*(*5*)[*unfolded linorder-trm.is-least-in-ffilter-iff*, *simplified*]
by *argo*
next
show ∀ *w*|∈|*atms-of-clss* (*N* |∪| U_{er}). *w* ≺$_t$ *A* ⟶ *w* |∈| *trail-atms* Γ
using *step-hyps*(*5*)[*unfolded linorder-trm.is-least-in-ffilter-iff*, *simplified*]
by *fastforce*
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using Γ*-lower* **.**
qed

moreover have *trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)
by (*simp add*: ‹Γ′ = (*Neg A*, *None*) # Γ›)

ultimately show *?thesis*
by *simp*
qed

moreover have $\forall A \mid\in\mid$ *trail-atms* Γ'. $\exists L$. *ord-res.is-maximal-lit* L $D \wedge (A \preceq_t$ *atm-of* $L)$
proof (*intro ballI exI conjI*)
show *ord-res.is-maximal-lit* L D
using ‹*ord-res.is-maximal-lit* L D› **.**
next
fix x **assume** $x \mid\in\mid$ *trail-atms* Γ'

hence $x = A \vee x \mid\in\mid$ *trail-atms* Γ
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$› **by** *simp*

thus $x \preceq_t$ *atm-of* L
proof (*elim disjE*)
assume $x = A$
thus $x \preceq_t$ *atm-of* L
using *step-hyps*(*5*) **by** (*simp add*: *linorder-trm.is-least-in-ffilter-iff*)
next
assume $x \mid\in\mid$ *trail-atms* Γ
thus $x \preceq_t$ *atm-of* L
using *trail-atms-le* **by** *metis*
qed
qed

moreover have $\forall Ln \in$ *set* Γ'. *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln)
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$›
using Γ-*deci-iff-neg* **by** *simp*

moreover have *trail-true-cls* Γ' C
if $Ln \in$ *set* Γ' **and** *snd* $Ln =$ *None* **and** $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and** $C \prec_c \{\#fst\ Ln\#\}$
for Ln C
proof −
have $Ln = (Neg\ A,\ None) \vee Ln \in$ *set* Γ
using ‹$Ln \in$ *set* Γ'› **unfolding** ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$› **by** *simp*

hence *trail-true-cls* Γ C
proof (*elim disjE*)
assume $Ln = (Neg\ A,\ None)$

hence *fst* $Ln \prec_l L$
by (*metis* ‹$A \prec_t$ *atm-of* L› *fst-conv literal.exhaust-sel ord-res.less-lit-simps*(*3*) *ord-res.less-lit-simps*(*4*))

moreover have *linorder-lit.is-maximal-in-mset* $\{\#fst\ Ln\#\}$ (*fst* Ln)

unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *simp*

ultimately have $\{\#fst\ Ln\#\} \prec_c D$
using *D-max-lit*
using *linorder-lit.multp-if-maximal-less-that-maximal* **by** *metis*

hence $C \prec_c D$
using ‹$C \prec_c \{\#fst\ Ln\#\}$› **by** *order*

thus *trail-true-cls* Γ C
using ‹C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})›
using *clauses-lt-D-true* **by** *blast*
next
assume $Ln \in set\ \Gamma$

thus *trail-true-cls* Γ C
using Γ-*deci-ball-lt-true* ‹*snd Ln* = *None*› ‹C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})› ‹$C \prec_c \{\#fst\ Ln\#\}$›
by *metis*
qed

thus *trail-true-cls* Γ' C
using ‹*suffix* Γ Γ'› **by** (*metis trail-true-cls-if-trail-true-suffix*)
qed

moreover have $\forall Ln \in set\ \Gamma'.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$› **using** Γ-*prop-in* **by** *simp*

moreover have $\forall Ln \in set\ \Gamma'.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst Ln*)
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$› **using** Γ-*prop-greatest* **by** *simp*

moreover have $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma' = \Gamma_1$ @ $(L,\ Some\ C) \#\ \Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\#\ C.\ K \neq L\#\}$
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$› **using** Γ-*prop-almost-false*
by (*metis* (*no-types*, *lifting*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma' = \Gamma_1$ @ $(L,\ Some\ D) \#\ \Gamma_0 \longrightarrow$
$(\forall C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})$.\ C \prec_c D \longrightarrow$ *trail-true-cls* Γ_0 C))
using Γ-*prop-ball-lt-true*
unfolding ‹$\Gamma' = (Neg\ A,\ None) \#\ \Gamma$›
by (*smt* (*verit*, *best*) *append-eq-Cons-conv list.inject option.discI snd-conv*)

ultimately show *?thesis*
by (*auto simp add*: ‹$s' = (U_{er},\ \mathcal{F},\ \Gamma',\ Some\ D)$› *ord-res-7-invars-def*)
next
case *step-hyps*: (*skip-defined* Γ D K U_{er} $\mathcal{C}'$ $\mathcal{F}$)

note *D-max-lit* = ‹*ord-res.is-maximal-lit K D*›

have
F-subset: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**
D-in: $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall L_C.$ *linorder-lit.is-maximal-in-mset* $C\ L_C \longrightarrow$
$\neg$ *trail-defined-lit* $\Gamma\ L_C \longrightarrow$ (*trail-true-cls* Γ $\{\#K \in\#\ C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* Γ)) **and**
Γ-*sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))\ \Gamma$ **and**
Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$) **and**
trail-atms-le0: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ \exists L.$ *ord-res.is-maximal-lit* $L\ D \wedge (A \preceq_t$ *atm-of* L) **and**
Γ-*deci-iff-neg*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ-*deci-ball-lt-true*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c \{\#fst\ Ln\#\} \longrightarrow$ *trail-true-cls* $\Gamma\ C$) **and**
Γ-*prop-in*: $\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
Γ-*prop-greatest*:
$\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst* Ln) **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1$ @ $(L,$ *Some* $C)\ \#\ \Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\#\ C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1$ @ $(L,$ *Some* $D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* $\Gamma_0\ C$))
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$› *ord-res-7-invars-def*)

have *clauses-lt-D-almost-defined*: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$ *trail-defined-cls* Γ $\{\#L \in\#\ C.\ L \neq K\#\}$)
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

have $K \in\#\ D$ **and** *lit-in-D-le-K*: $\bigwedge L.\ L \in\#\ D \Longrightarrow L \preceq_l K$
using ‹*ord-res.is-maximal-lit K D*›
unfolding *atomize-imp atomize-all atomize-conj linorder-lit.is-maximal-in-mset-iff*

using *linorder-lit.leI* **by** *blast*

hence *atm-of K* $|\in|$ *atms-of-clss* (*N* $|\cup|$ U_{er})
using *D-in atm-of-in-atms-of-clssI* **by** *metis*

have *trail-atms-le*: $\forall A$ $|\in|$ *trail-atms* Γ. $A \preceq_t$ *atm-of K*
using *trail-atms-le0* ‹*ord-res.is-maximal-lit K D*›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have *trail-defined-cls* Γ $\{\#Ka \in\# D.\ Ka \neq K\#\}$
using *step-hyps*(*4*,*5*,*6*) *D-in* **by** (*metis clause-almost-definedI*)

show *?thesis*
unfolding ‹$s' = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}')$›
proof (*intro ord-res-7-invars.intros*)
show $\mathcal{F}$ $|\subseteq|$ *N* $|\cup|$ U_{er}
using $\mathcal{F}$*-subset* **.**

show $\forall C'.\ \mathcal{C}' = Some\ C' \longrightarrow C'$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *step-hyps*(*7*) **by** (*metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD*)

have *trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
if $\mathcal{C}' = Some\ E$ **and**
C-in: *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C* L_C **and**
L_C*-undef*: $\neg$ *trail-defined-lit* Γ L_C
for *E C* L_C
proof −
have *linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) *D*) (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}))) *E*
using ‹$\mathcal{C}' = Some\ E$› *step-hyps* **by** (*metis Some-eq-The-optionalD*)
hence $C \prec_c D \vee C = D$
unfolding *linorder-cls.is-least-in-ffilter-iff*
using *C-lt* **by** (*metis C-in linorder-cls.not-less-iff-gr-or-eq*)
thus *?thesis*
proof (*elim disjE*)
assume $C \prec_c D$
thus *?thesis*
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in C-max-lit* L_C*-undef*]
by *argo*
next
assume $C = D$
hence $L_C = K$
using *C-max-lit D-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
hence *False*
using L_C*-undef* ‹*trail-defined-lit* Γ *K*› **by** *argo*
thus *?thesis* **..**

```
    qed
  qed

  thus ∀ D. C' = Some D ⟶ (∀ C|∈|iefac F |‘| (N |∪| U_er). C ≺_c D ⟶
    (∀ L_C. ord-res.is-maximal-lit L_C C ⟶ ¬ trail-defined-lit Γ L_C ⟶
      trail-true-cls Γ {#K ∈# C. K ≠ L_C#} ∧ is-pos L_C))
    by metis

  have
    trail-true-cls Γ C
    ⋀K. linorder-lit.is-maximal-in-mset C K ⟹
      ¬ (∃ A |∈| atms-of-clss (N |∪| U_er). A ≺_t atm-of K ∧ A |∉| trail-atms Γ)
    if C-lt: ⋀E. C' = Some E ⟹ C ≺_c E and C-in: C |∈| iefac F |‘| (N |∪|
U_er) for C
  proof −
    have C ⪯_c D
      using step-hyps that by (metis clause-le-if-lt-least-greater)

    hence C ≺_c D ∨ C = D
      by simp

    thus trail-true-cls Γ C
    proof (elim disjE)
      assume C ≺_c D
      thus trail-true-cls Γ C
        using clauses-lt-D-true ‹C |∈| iefac F |‘| (N |∪| U_er)› by metis
    next
      assume C = D

      have trail-defined-cls Γ D
       using ‹trail-defined-lit Γ K› ‹trail-defined-cls Γ {#Ka ∈# D. Ka ≠ K#}›
       unfolding trail-defined-cls-def by auto

      hence trail-true-cls Γ D
        using Γ-consistent ‹¬ trail-false-cls Γ D›
        by (metis trail-true-or-false-cls-if-defined)

      thus trail-true-cls Γ C
        using ‹C = D› by simp
    qed

    fix K_c
    assume C-max-lit: linorder-lit.is-maximal-in-mset C K_c
    thus ¬ (∃ A |∈| atms-of-clss (N |∪| U_er). A ≺_t atm-of K_c ∧ A |∉| trail-atms
Γ)
      using ‹C ≺_c D ∨ C = D›
    proof (elim disjE)
      assume C ≺_c D
      thus ?thesis
```

using *no-undef-atm-lt-max-lit-if-lt-D* ‹C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})› *C-max-lit* **by** *metis*
next
assume $C = D$
thus *?thesis*
by (*metis C-max-lit D-max-lit Uniq-D linorder-lit.Uniq-is-maximal-in-mset step-hyps*(5))
qed
qed

thus
$\forall C|\in|$*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). ($\forall D.\ \mathcal{C}' =$ *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$ *trail-true-cls* Γ C
$\forall C|\in|$*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). ($\forall D.\ \mathcal{C}' =$ *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$
($\forall K.$ *ord-res.is-maximal-lit* K $C \longrightarrow$
$\neg$ ($\exists A|\in|$*atms-of-clss* (N |∪| U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ |∉| *trail-atms* Γ))
unfolding *atomize-conj* **by** *metis*

show *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ
using Γ-*sorted* **.**

show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N |∪| U_{er}))
using Γ-*lower* **.**

have *trail-atms* Γ = *atms-of-clss* (N |∪| U_{er}) **if** $\mathcal{C}' =$ *None*
proof (*intro fsubset-antisym*)
show *trail-atms* Γ |⊆| *atms-of-clss* (N |∪| U_{er})
using Γ-*lower* **unfolding** *linorder-trm.is-lower-set-iff* **by** *blast*
next
have *nbex-gt-D*: $\neg$ ($\exists E$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $D \prec_c E$)
using *step-hyps* ‹$\mathcal{C}' =$ *None*›
by (*metis clause-le-if-lt-least-greater linorder-cls.leD option.simps*(3))

have $\neg$ ($\exists A|\in|$*atms-of-clss* (N |∪| U_{er}). *atm-of* $K \prec_t A$)
proof (*intro notI* , *elim bexE*)
fix A :: *'f gterm*
assume A |∈| *atms-of-clss* (N |∪| U_{er}) **and** *atm-of* $K \prec_t A$

hence A |∈| *atms-of-clss* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))
by *simp*

then obtain E L **where** *E-in*: E |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and** $L \in\#$ E **and** A = *atm-of* L
unfolding *atms-of-clss-def atms-of-cls-def*
by (*smt* (*verit*, *del-insts*) *fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have $K \prec_l L$
using ‹*atm-of* $K \prec_t A$› ‹A = *atm-of* L›
by (*cases* K; *cases* L) *simp-all*

hence $D \prec_c E$
using *D-max-lit* ‹$L \in\# E$›
by (*metis empty-iff linorder-lit.ex-maximal-in-mset linorder-lit.is-maximal-in-mset-iff*
linorder-lit.less-linear linorder-lit.less-trans
linorder-lit.multp-if-maximal-less-that-maximal set-mset-empty)

thus *False*
using *E-in nbex-gt-D* **by** *metis*
qed

hence ¬ (∃ A|∈|*atms-of-clss* (N |∪| U_{er}). A |∉| *trail-atms* Γ)
using *step-hyps*(5) *step-hyps*(6)
by (*metis linorder-trm.linorder-cases*
trail-defined-lit-iff-trail-defined-atm)

then show *atms-of-clss* (N |∪| U_{er}) |⊆| *trail-atms* Γ
by *blast*
qed

thus $\mathcal{C}' = None \longrightarrow$ *trail-atms* Γ = *atms-of-clss* (N |∪| U_{er})
by *metis*

show ∀ D. $\mathcal{C}' = Some\ D \longrightarrow$ (∀ A |∈| *trail-atms* Γ.
∃ L. *ord-res.is-maximal-lit* L D ∧ $A \preceq_t$ *atm-of* L)
proof (*intro allI impI ballI*)
fix E :: *'f gterm literal multiset* **and** A :: *'f gterm*
assume $\mathcal{C}' = Some\ E$ **and** A |∈| *trail-atms* Γ

have *linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) D) (*iefac* $\mathcal{F}$ |`| (N |∪| U_{er}))) E
using *step-hyps*(7) ‹$\mathcal{C}' = Some\ E$› **by** (*metis Some-eq-The-optionalD*)

hence $D \prec_c E$
unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *argo*

obtain L **where** *linorder-lit.is-maximal-in-mset* E L
by (*metis* ‹$D \prec_c E$› *linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls*)

show ∃ L. *ord-res.is-maximal-lit* L E ∧ $A \preceq_t$ *atm-of* L
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit* L E
using ‹*ord-res.is-maximal-lit* L E› **.**
next
have $K \preceq_l L$
using *step-hyps*(4) ‹*ord-res.is-maximal-lit* L E›
by (*metis* ‹$D \prec_c E$› *linorder-cls.less-asym linorder-lit.leI*
linorder-lit.multp-if-maximal-less-that-maximal)

hence *atm-of* $K \preceq_t$ *atm-of* L

by (*cases K*; *cases L*) *simp-all*

moreover have $A \preceq_t$ *atm-of K*
using ‹A |∈| *trail-atms* Γ› *trail-atms-le* **by** *blast*

ultimately show $A \preceq_t$ *atm-of L*
by *order*
qed
qed

show ∀ *Ln*∈*set* Γ. *is-neg* (*fst Ln*) = (*snd Ln* = *None*)
using Γ-*deci-iff-neg* **by** *metis*

show ∀ *Ln* ∈ *set* Γ. *snd Ln* = *None* ⟶
(∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ {#*fst Ln*#} ⟶ *trail-true-cls* Γ *C*)
using Γ-*deci-ball-lt-true* **.**

show ∀ *Ln* ∈ *set* Γ. ∀ *C*. *snd Ln* = *Some C* ⟶ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using Γ-*prop-in* **by** *simp*

show ∀ *Ln* ∈ *set* Γ. ∀ *C*. *snd Ln* = *Some C* ⟶ *linorder-lit.is-greatest-in-mset* *C* (*fst Ln*)
using Γ-*prop-greatest* **by** *simp*

show ∀ Γ$_1$ *L C* Γ$_0$. Γ = Γ$_1$ @ (*L*, *Some C*) # Γ$_0$ ⟶ *trail-false-cls* Γ$_0$ {#*K* ∈# *C*. *K* ≠ *L*#}
using Γ-*prop-almost-false* **.**

show (∀ Γ$_1$ *L D* Γ$_0$. Γ = Γ$_1$ @ (*L*, *Some D*) # Γ$_0$ ⟶
(∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶ *trail-true-cls* Γ$_0$ *C*))
using Γ-*prop-ball-lt-true* **.**
qed
next
case *step-hyps*: (*skip-undefined-neg* Γ *D K* U_{er} Γ′ $\mathcal{C}'$ $\mathcal{F}$)

note *D-max-lit* = ‹*ord-res.is-maximal-lit K D*›

have *suffix* Γ Γ′
using *step-hyps* **unfolding** *suffix-def* **by** *simp*

have
$\mathcal{F}$-*subset*: $\mathcal{F}$ |⊆| *N* |∪| U_{er} **and**
D-in: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
clauses-lt-D-seldomly-have-undef-max-lit:
∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶
(∀ L_C. *linorder-lit.is-maximal-in-mset C* L_C ⟶
¬ *trail-defined-lit* Γ L_C ⟶ (*trail-true-cls* Γ {#*K* ∈# *C*. *K* ≠ L_C#} ∧ *is-pos* L_C)) **and**
clauses-lt-D-true: ∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶ *trail-true-cls* Γ

C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall\, C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. $C \prec_c D \longrightarrow$
$(\forall\, K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists\, A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$. $A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma))$
and
Γ-*sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))\ \Gamma$ **and**
Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$)
and
trail-atms-le0: $\forall\, A \mid\in\mid$ *trail-atms* $\Gamma.\ \exists\, L.$ *ord-res.is-maximal-lit* $L\ D \wedge (A \preceq_t$ *atm-of* $L)$ **and**
Γ-*deci-iff-neg*: $\forall\, Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ-*deci-ball-lt-true*: $\forall\, Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longrightarrow$
$(\forall\, C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. $C \prec_c \{\#fst\ Ln\#\} \longrightarrow$ *trail-true-cls* $\Gamma\ C)$
and
Γ-*prop-in*: $\forall\, Ln \in$ *set* $\Gamma.\ \forall\, C.$ *snd* $Ln =$ *Some* $C \longrightarrow C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
Γ-*prop-greatest*:
$\forall\, Ln \in$ *set* $\Gamma.\ \forall\, C.$ *snd* $Ln =$ *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst* Ln) **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,$ *Some* $C)\ \#\ \Gamma_0 \longrightarrow$ *trail-false-cls* $\Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,$ *Some* $D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall\, C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. $C \prec_c D \longrightarrow$ *trail-true-cls* $\Gamma_0\ C))$
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$› *ord-res-7-invars-def*)

have *clauses-lt-D-almost-defined*: $\forall\, C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. $C \prec_c D \longrightarrow$
$(\forall\, K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$ *trail-defined-cls* $\Gamma\ \{\#L \in\#\ C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

have $K \in\#\ D$
using ‹*ord-res.is-maximal-lit* $K\ D$›
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of* $K \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
using *D-in atm-of-in-atms-of-clssI* **by** *metis*

have *trail-atms-le*: $\forall\, A \mid\in\mid$ *trail-atms* $\Gamma.\ A \preceq_t$ *atm-of* K
using *trail-atms-le0* ‹*ord-res.is-maximal-lit* $K\ D$›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$
using $\mathcal{F}$-*subset* **.**

moreover have $\forall\, C'.\ \mathcal{C}' = \mathit{Some}\ C' \longrightarrow C' \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *step-hyps*(*9*) **by** (*metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD*)

moreover have *trail-true-cls* Γ' $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
if $\mathcal{C}' = \mathit{Some}\ E$ **and**
C-in: $C \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C\ L_C$ **and**
L_C*-undef*: $\neg$ *trail-defined-lit* $\Gamma'\ L_C$
for $E\ C\ L_C$
proof −
have *linorder-cls.is-least-in-fset* (*ffilter* $((\prec_c)\ D)$ (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) E
using ‹$\mathcal{C}' = \mathit{Some}\ E$› *step-hyps* **by** (*metis Some-eq-The-optionalD*)
hence $C \prec_c D \vee C = D$
unfolding *linorder-cls.is-least-in-ffilter-iff*
using *C-lt* **by** (*metis C-in linorder-cls.not-less-iff-gr-or-eq*)
thus *?thesis*
proof (*elim disjE*)
assume $C \prec_c D$

moreover have $\neg$ *trail-defined-lit* $\Gamma\ L_C$
using L_C*-undef* ‹*suffix* $\Gamma\ \Gamma'$›
using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

ultimately have *trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in C-max-lit*] **by** *metis*

thus *?thesis*
using ‹*suffix* $\Gamma\ \Gamma'$› *trail-true-cls-if-trail-true-suffix* **by** *blast*
next
assume $C = D$
hence $L_C = K$
using *C-max-lit D-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
moreover have *trail-defined-lit* $\Gamma'\ K$
by (*simp add*: *step-hyps*(*8*) *trail-defined-lit-def*)
ultimately have *False*
using L_C*-undef* **by** *argo*
thus *?thesis* **..**
qed
qed

moreover have *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ'
proof −
have *atm-of* $K \mid\notin\mid$ *trail-atms* Γ
using ‹$\neg$ *trail-defined-lit* $\Gamma\ K$›
by (*simp add*: *trail-defined-lit-iff-trail-defined-atm*)

```
    have x ≺t atm-of K if x-in: x |∈| trail-atms Γ for x
    proof −
      have x ⪯t atm-of K
        using x-in trail-atms-le by metis

      moreover have x ≠ atm-of K
        using x-in ‹atm-of K |∉| trail-atms Γ› by metis

      ultimately show ?thesis
        by order
    qed

    hence ∀ x∈set Γ. atm-of (fst x) ≺t atm-of K
      by (simp add: fset-trail-atms)

    thus ?thesis
      using Γ-sorted ‹Γ' = (K, None) # Γ› by simp
  qed

  moreover have Γ'-lower: linorder-trm.is-lower-fset (trail-atms Γ') (atms-of-clss
(N |∪| U_er))
  proof −
    have linorder-trm.is-lower-set (insert (atm-of K) (fset (trail-atms Γ)))
     (fset (atms-of-clss (N |∪| U_er)))
    proof (rule linorder-trm.is-lower-set-insertI)
      show atm-of K |∈| atms-of-clss (N |∪| U_er)
        using ‹atm-of K |∈| atms-of-clss (N |∪| U_er)› .
    next
      show ∀ w|∈|atms-of-clss (N |∪| U_er). w ≺t (atm-of K) ⟶ w |∈| trail-atms
Γ
        using step-hyps(5)[unfolded linorder-trm.is-least-in-ffilter-iff, simplified]
        by fastforce
    next
      show linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N |∪| U_er))
        using Γ-lower .
    qed

    moreover have trail-atms Γ' = finsert (atm-of K) (trail-atms Γ)
      by (simp add: ‹Γ' = (K, None) # Γ›)

    ultimately show ?thesis
      by simp
  qed

  moreover have trail-atms Γ' = atms-of-clss (N |∪| U_er) if C' = None
  proof (intro fsubset-antisym)
    show trail-atms Γ' |⊆| atms-of-clss (N |∪| U_er)
      using Γ'-lower unfolding linorder-trm.is-lower-set-iff by blast
  next
```

have *nbex-gt-D*: $\neg$ ($\exists$ *E* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}). *D* $\prec_c$ *E*)
using *step-hyps* ‹$\mathcal{C}'$ = *None*›
by (*metis clause-le-if-lt-least-greater linorder-cls.leD option.simps*(*3*))

have $\neg$ ($\exists$ *A*|$\in$|*atms-of-clss* (*N* |$\cup$| U_{er}). *atm-of K* $\prec_t$ *A*)
proof (*intro notI* , *elim bexE*)
fix *A* :: *'f gterm*
assume *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}) **and** *atm-of K* $\prec_t$ *A*

hence *A* |$\in$| *atms-of-clss* (*iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}))
by *simp*

then obtain *E L* **where** *E-in*: *E* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}) **and** *L* $\in\#$ *E*
and *A* = *atm-of L*
unfolding *atms-of-clss-def atms-of-cls-def*
by (*smt* (*verit*, *del-insts*) *fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have *K* $\prec_l$ *L*
using ‹*atm-of K* $\prec_t$ *A*› ‹*A* = *atm-of L*›
by (*cases K*; *cases L*) *simp-all*

hence *D* $\prec_c$ *E*
using *D-max-lit* ‹*L* $\in\#$ *E*›
by (*metis empty-iff linorder-lit.ex-maximal-in-mset linorder-lit.is-maximal-in-mset-iff*
linorder-lit.less-linear linorder-lit.less-trans
linorder-lit.multp-if-maximal-less-that-maximal set-mset-empty)

thus *False*
using *E-in nbex-gt-D* **by** *metis*
qed

hence $\neg$ ($\exists$ *A*|$\in$|*atms-of-clss* (*N* |$\cup$| U_{er}). *A* |$\notin$| *trail-atms* Γ')
using *step-hyps*
by (*metis finsert-iff linorder-trm.antisym-conv3 prod.sel*(*1*) *trail-atms.simps*(*2*))

then show *atms-of-clss* (*N* |$\cup$| U_{er}) |$\subseteq$| *trail-atms* Γ'
by *blast*
qed

moreover have $\forall$ *D*. $\mathcal{C}'$ = *Some D* $\longrightarrow$ ($\forall$ *A* |$\in$| *trail-atms* Γ'.
$\exists$ *L*. *ord-res.is-maximal-lit L D* $\wedge$ *A* $\preceq_t$ *atm-of L*)
proof (*intro allI impI ballI*)
fix *E* :: *'f gterm literal multiset* **and** *A* :: *'f gterm*
assume $\mathcal{C}'$ = *Some E* **and** *A* |$\in$| *trail-atms* Γ'

have *linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) *D*) (*iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}))) *E*
using *step-hyps*(*9*) ‹$\mathcal{C}'$ = *Some E*› **by** (*metis Some-eq-The-optionalD*)

hence *D* $\prec_c$ *E*

unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *argo*

obtain L **where** *linorder-lit.is-maximal-in-mset E L*
by (*metis* ‹$D \prec_c E$› *linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls*)

show $\exists L.$ *ord-res.is-maximal-lit L E* $\wedge$ $A \preceq_t$ *atm-of L*
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit L E*
using ‹*ord-res.is-maximal-lit L E*› **.**
next
have $K \preceq_l L$
using *step-hyps*(*4*) ‹*ord-res.is-maximal-lit L E*›
by (*metis* ‹$D \prec_c E$› *linorder-cls.less-asym linorder-lit.leI*
linorder-lit.multp-if-maximal-less-that-maximal)

hence *atm-of K* $\preceq_t$ *atm-of L*
by (*cases K*; *cases L*) *simp-all*

moreover have $A \preceq_t$ *atm-of K*
proof −
have $A =$ *atm-of K* $\vee$ A |$\in$| *trail-atms* Γ
using ‹A |$\in$| *trail-atms* Γ'›
unfolding ‹$\Gamma' = (K,$ *None*$)$ # Γ›
by (*metis* (*mono-tags*, *lifting*) *finsertE prod.sel*(*1*) *trail-atms.simps*(*2*))

thus $A \preceq_t$ *atm-of K*
using *trail-atms-le* **by** *blast*
qed

ultimately show $A \preceq_t$ *atm-of L*
by *order*
qed
qed

moreover have $\forall$ *Ln* $\in$ *set* $\Gamma'.$ *snd Ln* = *None* $\longleftrightarrow$ *is-neg* (*fst Ln*)
unfolding ‹$\Gamma' = (K,$ *None*$)$ # Γ›
using Γ*-deci-iff-neg* ‹*is-neg K*› **by** *simp*

moreover have *trail-true-cls* Γ' *C*
if *Ln* $\in$ *set* Γ' **and** *snd Ln* = *None* **and** *C* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}) **and** *C* $\prec_c$ {#*fst Ln*#}
for *Ln C*
proof −
have *Ln* $= (K,$ *None*$)$ $\vee$ *Ln* $\in$ *set* Γ
using ‹*Ln* $\in$ *set* Γ'› **unfolding** ‹$\Gamma' = (K,$ *None*$)$ # Γ› **by** *simp*

hence *trail-true-cls* Γ *C*
proof (*elim disjE*)
assume *Ln* $= (K,$ *None*$)$

hence $\forall\, x \in\# C.\ x \prec_l K$
using ‹$C \prec_c \{\#fst\ Ln\#\}$›
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $C \prec_c D$
using *D-max-lit*
by (*metis* ‹$K \in\# D$› *empty-iff ord-res.multp-if-all-left-smaller set-mset-empty*)

thus *trail-true-cls* Γ C
using ‹C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})›
using *clauses-lt-D-true* **by** *blast*
next
assume $Ln \in$ *set* Γ

thus *trail-true-cls* Γ C
using Γ-*deci-ball-lt-true* ‹*snd Ln* = *None*› ‹C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})›
‹$C \prec_c \{\#fst\ Ln\#\}$›
by *metis*
qed

thus *trail-true-cls* Γ' C
using ‹*suffix* Γ Γ'› **by** (*metis trail-true-cls-if-trail-true-suffix*)
qed

moreover have $\forall\, Ln \in$ *set* $\Gamma'.\ \forall\, C.$ *snd Ln* = *Some* $C \longrightarrow C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
using Γ-*prop-in* ‹Γ' = (K, *None*) # Γ› **by** *simp*

moreover have $\forall\, Ln \in$ *set* $\Gamma'.\ \forall\, C.$ *snd Ln* = *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst Ln*)
using Γ-*prop-greatest* ‹Γ' = (K, *None*) # Γ› **by** *simp*

moreover have $\forall\, \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma' = \Gamma_1$ @ (L, *Some* C) # $\Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\# C.\ K \neq L\#\}$
unfolding ‹Γ' = (K, *None*) # Γ› **using** Γ-*prop-almost-false*
by (*metis* (*no-types*, *lifting*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have ($\forall\, \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma' = \Gamma_1$ @ (L, *Some* D) # $\Gamma_0 \longrightarrow$
($\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})). $C \prec_c D \longrightarrow$ *trail-true-cls* Γ_0 C))
unfolding ‹Γ' = (K, *None*) # Γ› **using** Γ-*prop-ball-lt-true*
by (*smt* (*verit*, *ccfv-SIG*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

ultimately show *?thesis*
unfolding ‹s' = (U_{er}, $\mathcal{F}$, Γ', $\mathcal{C}'$)›
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ' C
$\bigwedge K_C.$ *linorder-lit.is-maximal-in-mset* C $K_C \Longrightarrow$

$\neg$ ($\exists$ *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}). *A* $\prec_t$ *atm-of* K_C $\wedge$ *A* |$\notin$| *trail-atms* Γ')
if *C-lt*: $\bigwedge$*E*. $\mathcal{C}'$ = *Some E* $\longrightarrow$ *C* $\prec_c$ *E* **and** *C-in*: *C* |$\in$| *iefac* $\mathcal{F}$ |`| (*N* |$\cup$| U_{er}) **for** *C*
proof −
have *C* $\preceq_c$ *D*
using *step-hyps that* **by** (*metis clause-le-if-lt-least-greater*)

hence *C* $\prec_c$ *D* $\vee$ *C* = *D*
by *simp*

thus *trail-true-cls* Γ' *C*
proof (*elim disjE*)
assume *C* $\prec_c$ *D*

hence *trail-true-cls* Γ *C*
using *clauses-lt-D-true* ‹*C* |$\in$| *iefac* $\mathcal{F}$ |`| (*N* |$\cup$| U_{er})› **by** *metis*

thus *trail-true-cls* Γ' *C*
using ‹*suffix* Γ Γ'› **by** (*metis trail-true-cls-if-trail-true-suffix*)
next
assume *C* = *D*

have *trail-true-lit* Γ' *K*
unfolding ‹Γ' = (*K*, *None*) # Γ› *trail-true-lit-def* **by** *simp*

thus *trail-true-cls* Γ' *C*
unfolding ‹*C* = *D*› *trail-true-cls-def*
using ‹*K* $\in$# *D*› **by** *metis*
qed

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset C* K_C
show $\neg$ ($\exists$ *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}). *A* $\prec_t$ *atm-of* K_C $\wedge$ *A* |$\notin$| *trail-atms* Γ')
using ‹*C* $\prec_c$ *D* $\vee$ *C* = *D*›
proof (*elim disjE*)
assume *C* $\prec_c$ *D*

hence $\neg$ ($\exists$ *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}). *A* $\prec_t$ *atm-of* K_C $\wedge$ *A* |$\notin$| *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-D* ‹*C* |$\in$| *iefac* $\mathcal{F}$ |`| (*N* |$\cup$| U_{er})› *C-max-lit* **by** *metis*

thus $\neg$ ($\exists$ *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}). *A* $\prec_t$ *atm-of* K_C $\wedge$ *A* |$\notin$| *trail-atms* Γ')
by (*metis finsert-iff step-hyps*(*8*) *trail-atms.simps*(*2*))
next
assume *C* = *D*
thus $\neg$ ($\exists$ *A* |$\in$| *atms-of-clss* (*N* |$\cup$| U_{er}). *A* $\prec_t$ *atm-of* K_C $\wedge$ *A* |$\notin$| *trail-atms*

Γ')
by (*metis* (*mono-tags*, *lifting*) *C-max-lit D-max-lit Uniq-D* Γ'*-lower finsert-iff linorder-lit.Uniq-is-maximal-in-mset linorder-trm.not-in-lower-setI prod.sel*(*1*) *step-hyps*(*8*) *trail-atms.simps*(*2*))
qed
qed

thus
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). ($\forall\, D.\ \mathcal{C}' =$ *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$ *trail-true-cls* Γ' C
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). ($\forall\, D.\ \mathcal{C}' =$ *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$
($\forall\, K.$ *ord-res.is-maximal-lit* K $C \longrightarrow$
$\neg$ ($\exists\, A|\in|$*atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ'))
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*skip-undefined-pos* Γ D K U_{er} $\mathcal{F}$ E)

note *D-max-lit* = ‹*ord-res.is-maximal-lit* K D›
note *E-least* = ‹*linorder-cls.is-least-in-fset* (*ffilter* (($\prec_c$) D) (*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}))) E›

have $D \prec_c E$
using *E-least* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *argo*

have
F-subset: $\mathcal{F}$ $|\subseteq|$ N $|\cup|$ U_{er} **and**
D-in: D $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}) **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$
($\forall\, L_C.$ *linorder-lit.is-maximal-in-mset* C $L_C \longrightarrow$
$\neg$ *trail-defined-lit* Γ $L_C \longrightarrow$ (*trail-true-cls* Γ {#K $\in\#$ C. $K \neq L_C$#} $\wedge$ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$
($\forall\, K.$ *linorder-lit.is-maximal-in-mset* C $K \longrightarrow$
$\neg$ ($\exists\, A$ $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ)) **and**
Γ*-sorted*: *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $|\cup|$ U_{er})) **and**
trail-atms-le0: $\forall\, A$ $|\in|$ *trail-atms* Γ. $\exists\, L.$ *ord-res.is-maximal-lit* L $D \wedge (A \preceq_t$ *atm-of* L) **and**
Γ*-deci-iff-neg*: $\forall\, Ln \in$ *set* Γ. *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ*-deci-ball-lt-true*: $\forall\, Ln \in$ *set* Γ. *snd* $Ln =$ *None* $\longrightarrow$
($\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c$ {#*fst* Ln#} $\longrightarrow$ *trail-true-cls* Γ C) **and**

Γ-*prop-in*: $\forall\, Ln \in set\ \Gamma.\ \forall\, C.\ snd\ Ln = Some\ C \longrightarrow C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and**
Γ-*prop-greatest*:
$\forall\, Ln \in set\ \Gamma.\ \forall\, C.\ snd\ Ln = Some\ C \longrightarrow linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ (fst\ Ln)$ **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L, Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L, Some\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C))$
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ D)$› *ord-res-7-invars-def*)

have *clauses-lt-D-almost-defined*: $\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow$
$(\forall\, K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow trail\text{-}defined\text{-}cls\ \Gamma\ \{\#L \in\#\ C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ D)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

have $K \in\#\ D$
using ‹*ord-res.is-maximal-lit K D*›
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of* $K\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er})$
using *D-in atm-of-in-atms-of-clssI* **by** *metis*

have *trail-defined-cls* $\Gamma\ \{\#L \in\#\ D.\ L \neq K \wedge L \neq -\ K\#\}$
using *clause-almost-almost-definedI*[*OF D-in step-hyps*(4,5)] **.**

moreover have $-\ K \notin\#\ D$
using ‹*is-pos K*› *D-max-lit*
by (*metis* (*no-types*, *opaque-lifting*) *is-pos-def linorder-lit.antisym-conv3*
linorder-lit.is-maximal-in-mset-iff linorder-trm.less-imp-not-eq ord-res.less-lit-simps(4)
uminus-Pos uminus-not-id)

ultimately have *D-almost-defined*: *trail-defined-cls* $\Gamma\ \{\#L \in\#\ D.\ L \neq K\#\}$
unfolding *trail-defined-cls-def* **by** *auto*

hence *trail-true-cls* $\Gamma\ \{\#L \in\#\ D.\ L \neq K\#\}$
using ‹$\neg$ *trail-false-cls* $\Gamma\ \{\#L \in\#\ D.\ L \neq K\#\}$›
using *trail-true-or-false-cls-if-defined* **by** *metis*

hence *D-true*: *trail-true-cls* $\Gamma\ D$
unfolding *trail-true-cls-def* **by** *auto*

have *trail-atms-le*: $\forall\, A\ |\in|\ trail\text{-}atms\ \Gamma.\ A \preceq_t atm\text{-}of\ K$
using *trail-atms-le0 D-max-lit*

by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

obtain L **where** *E-max-lit*: *linorder-lit.is-maximal-in-mset E L*
by (*metis* ‹$D \prec_c E$› *linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls*)

have $\mathcal{F}$ |⊆| N |∪| U_{er}
using $\mathcal{F}$*-subset* **.**

moreover have $\forall C'$. *Some E* = *Some* $C' \longrightarrow C'$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
using *step-hyps* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *simp*

moreover have *trail-true-cls* Γ {#K ∈# C. $K \neq L_C$#} ∧ *is-pos* L_C
if *Some E* = *Some* E' **and**
C-in: C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
C-lt: $C \prec_c E'$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C* L_C **and**
L_C*-undef*: ¬ *trail-defined-lit* Γ L_C
for E' C L_C
proof −
have $E' = E$
using *that* **by** *simp*
hence *linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) D) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E
using *step-hyps* **by** *metis*
hence $C \prec_c D \vee C = D$
unfolding *linorder-cls.is-least-in-ffilter-iff*
using *C-lt* ‹$E' = E$› **by** (*metis C-in linorder-cls.not-less-iff-gr-or-eq*)
thus *?thesis*
proof (*elim disjE*)
assume $C \prec_c D$
thus *trail-true-cls* Γ {#K ∈# C. $K \neq L_C$#} ∧ *is-pos* L_C
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in C-max-lit* L_C*-undef*]
by *metis*
next
assume $C = D$
hence $L_C = K$
using *C-max-lit D-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
thus *?thesis*
using ‹$C = D$› ‹*trail-true-cls* Γ {#L ∈# D. $L \neq K$#}› ‹*is-pos* K› **by** *metis*
qed
qed

moreover have *sorted-wrt* (λx y. *atm-of* (*fst* y) ≺$_t$ *atm-of* (*fst* x)) Γ
using Γ*-sorted* **.**

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N |∪| U_{er}))
using Γ*-lower* **.**

moreover have $\forall D.\ Some\ E = Some\ D \longrightarrow (\forall A \mid\in\mid trail\text{-}atms\ \Gamma.$
$\exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ D \wedge A \preceq_t atm\text{-}of\ L)$
proof −
have $\forall A \mid\in\mid trail\text{-}atms\ \Gamma.\ \exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ E \wedge A \preceq_t atm\text{-}of\ L$
proof (*intro ballI*)
fix $A :: 'f\ gterm$
assume $A \mid\in\mid trail\text{-}atms\ \Gamma$
show $\exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ E \wedge A \preceq_t atm\text{-}of\ L$
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit L E*
using *E-max-lit* **.**
next
have $K \preceq_l L$
using *D-max-lit E-max-lit*
by (*metis* ‹$D \prec_c E$› *linorder-cls.less-asym linorder-lit.leI*
linorder-lit.multp-if-maximal-less-that-maximal)

hence $atm\text{-}of\ K \preceq_t atm\text{-}of\ L$
by (*cases K*; *cases L*) *simp-all*

moreover have $A \preceq_t atm\text{-}of\ K$
using ‹$A \mid\in\mid trail\text{-}atms\ \Gamma$› *trail-atms-le* **by** *metis*

ultimately show $A \preceq_t atm\text{-}of\ L$
by *order*
qed
qed

thus *?thesis*
by *simp*
qed

moreover have $\forall Ln \in set\ \Gamma.\ snd\ Ln = None \longleftrightarrow is\text{-}neg\ (fst\ Ln)$
using Γ*-deci-iff-neg* **.**

moreover have $\forall Ln \in set\ \Gamma.\ snd\ Ln = None \longrightarrow$
$(\forall C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c \{\#fst\ Ln\#\} \longrightarrow trail\text{-}true\text{-}cls\ \Gamma\ C)$
using Γ*-deci-ball-lt-true* **.**

moreover have $\forall Ln \in set\ \Gamma.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using Γ*-prop-in* **.**

moreover have $\forall Ln \in set\ \Gamma.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ (fst\ Ln)$
using Γ*-prop-greatest* **.**

moreover have $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls$

Γ_0 $\{\#K \in\# C.\ K \neq L\#\}$
using Γ-*prop-almost-false* **.**

moreover have $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L, Some\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C))$
using Γ-*prop-ball-lt-true* **.**

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}, \mathcal{F}, \Gamma, Some\ E)\rangle$
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ C
$\bigwedge K_C.$ *linorder-lit.is-maximal-in-mset* $C\ K_C \Longrightarrow$
$\neg\ (\exists A\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ K_C \wedge A\ |\notin|\ trail\text{-}atms\ \Gamma)$
if *C-lt*: $\bigwedge E'.\ Some\ E = Some\ E' \Longrightarrow C \prec_c E'$ **and** *C-in*: $C\ |\in|\ iefac\ \mathcal{F}\ |`|$
$(N\ |\cup|\ U_{er})$ **for** C
proof −
have $C \prec_c E$
using *C-lt* **by** *simp*

hence $C \prec_c D \vee C = D$
using *E-least C-in*
by (*metis linorder-cls.is-least-in-ffilter-iff linorder-cls.less-imp-triv*
linorder-cls.linorder-cases)

thus *trail-true-cls* Γ C
proof (*elim disjE*)
assume $C \prec_c D$
thus *trail-true-cls* Γ C
using *clauses-lt-D-true* $\langle C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})\rangle$ **by** *metis*
next
assume $C = D$
thus *trail-true-cls* Γ C
using *D-true* **by** *argo*
qed

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset* $C\ K_C$
show $\neg\ (\exists A\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ K_C \wedge A\ |\notin|\ trail\text{-}atms$
$\Gamma)$
using $\langle C \prec_c D \vee C = D\rangle$
proof (*elim disjE*)
assume $C \prec_c D$
thus $\neg\ (\exists A\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ K_C \wedge A\ |\notin|\ trail\text{-}atms$
$\Gamma)$
using *no-undef-atm-lt-max-lit-if-lt-D* $\langle C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})\rangle$
C-max-lit **by** *metis*
next
assume $C = D$
thus $\neg\ (\exists A\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ K_C \wedge A\ |\notin|\ trail\text{-}atms$

Γ)
by (*metis C-max-lit D-max-lit Uniq-D linorder-lit.Uniq-is-maximal-in-mset step-hyps*(5))
qed
qed

thus
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). ($\forall\, D$. *Some* E = *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$ *trail-true-cls* Γ C
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). ($\forall\, D$. *Some* E = *Some* $D \longrightarrow C \prec_c D$) $\longrightarrow$
($\forall\, K$. *ord-res.is-maximal-lit* K $C \longrightarrow$
$\neg$ ($\exists\, A|\in|$*atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ))
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*skip-undefined-pos-ultimate* Γ D K U_{er} Γ′ $\mathcal{F}$)

note *D-max-lit* = ‹*ord-res.is-maximal-lit K D*›

have *suffix* Γ Γ′
using ‹Γ′ = (− K, *None*) # Γ› **by** (*simp add*: *suffix-def*)

have
$\mathcal{F}$*-subset*: $\mathcal{F}$ $|\subseteq|$ N $|\cup|$ U_{er} **and**
D-in: D $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}) **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$
($\forall\, L_C$. *linorder-lit.is-maximal-in-mset* C $L_C \longrightarrow$
$\neg$ *trail-defined-lit* Γ $L_C \longrightarrow$ (*trail-true-cls* Γ {#K ∈# C. $K \neq L_C$#} ∧ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c D \longrightarrow$
($\forall\, K$. *linorder-lit.is-maximal-in-mset* C $K \longrightarrow$
$\neg$ ($\exists\, A$ $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ)) **and**
Γ*-sorted*: *sorted-wrt* ($\lambda x\ y$. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $|\cup|$ U_{er})) **and**
trail-atms-le0: $\forall\, A$ $|\in|$ *trail-atms* Γ. $\exists\, L$. *ord-res.is-maximal-lit* L $D \wedge (A \preceq_t$ *atm-of* L) **and**
Γ*-deci-iff-neg*: $\forall$ *Ln* ∈ *set* Γ. *snd Ln* = *None* $\longleftrightarrow$ *is-neg* (*fst Ln*) **and**
Γ*-deci-ball-lt-true*: $\forall$ *Ln* ∈ *set* Γ. *snd Ln* = *None* $\longrightarrow$
($\forall\, C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c$ {#*fst Ln*#} $\longrightarrow$ *trail-true-cls* Γ C) **and**
Γ*-prop-in*: $\forall$ *Ln* ∈ *set* Γ. $\forall\, C$. *snd Ln* = *Some* $C \longrightarrow C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}) **and**
Γ*-prop-greatest*:
$\forall$ *Ln* ∈ *set* Γ. $\forall\, C$. *snd Ln* = *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C

(*fst Ln*) **and**
  Γ-*prop-almost-false*:
    $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$ **and**
  Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ D)\ \#\ \Gamma_0 \longrightarrow$
    $(\forall C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C))$
  using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,\ Some\ D)$› *ord-res-7-invars-def*)

 have *clauses-lt-D-almost-defined*: $\forall C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow$
  $(\forall K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow trail\text{-}defined\text{-}cls\ \Gamma\ \{\#L \in\#\ C.\ L \neq K\#\})$
  using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,\ Some\ D)$›] *clause-almost-defined-if-lt-next-clause*
  by *simp*

 have Γ-*consistent*: *trail-consistent* Γ
  using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

 have $K \in\#\ D$
  using ‹*ord-res.is-maximal-lit K D*›
  unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

 hence *atm-of K* $|\in|$ *atms-of-clss* $(N\ |\cup|\ U_{er})$
  using *D-in atm-of-in-atms-of-clssI* **by** *metis*

 have *trail-defined-cls* Γ $\{\#L \in\#\ D.\ L \neq K \wedge L \neq -\ K\#\}$
  using *clause-almost-almost-definedI*[*OF D-in step-hyps*(4,5)] **.**

 moreover have $-\ K \notin\#\ D$
  using ‹*is-pos K*› *D-max-lit*
  by (*metis* (*no-types*, *opaque-lifting*) *is-pos-def linorder-lit.antisym-conv3*
    linorder-lit.is-maximal-in-mset-iff linorder-trm.less-imp-not-eq ord-res.less-lit-simps(4)
      uminus-Pos uminus-not-id)

 ultimately have *D-almost-defined*: *trail-defined-cls* Γ $\{\#L \in\#\ D.\ L \neq K\#\}$
  unfolding *trail-defined-cls-def* **by** *auto*

 hence *trail-true-cls* Γ $\{\#L \in\#\ D.\ L \neq K\#\}$
  using ‹¬ *trail-false-cls* Γ $\{\#L \in\#\ D.\ L \neq K\#\}$›
  using *trail-true-or-false-cls-if-defined* **by** *metis*

 hence *D-true*: *trail-true-cls* Γ *D*
  unfolding *trail-true-cls-def* **by** *auto*

 have *trail-atms-le*: $\forall A\ |\in|\ trail\text{-}atms\ \Gamma.\ A \preceq_t atm\text{-}of\ K$
  using *trail-atms-le0 D-max-lit*
  by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

 have $\mathcal{F}\ |\subseteq|\ N\ |\cup|\ U_{er}$
  using $\mathcal{F}$-*subset* **.**

moreover have $\forall\, C'.\ None = Some\ C' \longrightarrow C' \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
by *simp*

moreover have *trail-true-cls* $\Gamma' \{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
if $None = Some\ E'$ **and**
C-in: $C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-lt: $C \prec_c E'$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C\ L_C$ **and**
L_C*-undef*: $\neg$ *trail-defined-lit* $\Gamma'\ L_C$
for $E'\ C\ L_C$
using *that* **by** *simp*

moreover have *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))\ \Gamma'$
proof −
have $\forall\, x{\in}set\ \Gamma.$ *atm-of* $(fst\ x) \prec_t$ *atm-of* K
by (*metis image-eqI linorder-trm.less-linear linorder-trm.not-le step-hyps*(6) *trail-atms-le*
trail-defined-lit-def trail-defined-lit-iff-trail-defined-atm)

thus *?thesis*
unfolding $\langle\Gamma' = (-\ K,\ None)\ \#\ \Gamma\rangle$
using Γ*-sorted* **by** *simp*
qed

moreover have Γ'*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* $(N \mid\cup\mid U_{er})$)
proof −
have *atm-of* $K \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
using $\langle$*atm-of* $K \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})\rangle$ **.**

moreover have $\forall\, w{\mid\in\mid}$*atms-of-clss* $(N \mid\cup\mid U_{er}).\ w \prec_t$ *atm-of* $K \longrightarrow w \mid\in\mid$ *trail-atms* Γ
using *step-hyps*(5) **by** *blast*

ultimately show *?thesis*
using Γ*-lower* **by** (*simp add*: $\langle\Gamma' = (-\ K,\ None)\ \#\ \Gamma\rangle$ *linorder-trm.is-lower-set-insertI*)
qed

moreover have *trail-atms* $\Gamma' =$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
proof (*intro fsubset-antisym*)
show *trail-atms* $\Gamma' \mid\subseteq\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
using Γ'*-lower* **unfolding** *linorder-trm.is-lower-set-iff* **by** *blast*
next
have *nbex-gt-D*: $\neg\ (\exists\, E \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ D \prec_c E)$
using *step-hyps* **by** *argo*

have $\neg\ (\exists\, A{\mid\in\mid}$*atms-of-clss* $(N \mid\cup\mid U_{er}).$ *atm-of* $K \prec_t A)$
proof (*intro notI* , *elim bexE*)

fix $A :: 'f\ gterm$
assume $A \mid\in\mid atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er})$ **and** $atm\text{-}of\ K \prec_t A$

hence $A \mid\in\mid atms\text{-}of\text{-}clss\ (iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))$
by *simp*

then obtain $E\ L$ **where** *E-in*: $E \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and** $L \in\#\ E$ **and** $A = atm\text{-}of\ L$
unfolding *atms-of-clss-def atms-of-cls-def*
by (*smt* (*verit*, *del-insts*) *fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have $K \prec_l L$
using ‹*atm-of K* $\prec_t$ *A*› ‹*A* = *atm-of L*›
by (*cases K*; *cases L*) *simp-all*

hence $D \prec_c E$
using *D-max-lit* ‹*L* $\in\#$ *E*›
by (*metis empty-iff linorder-lit.ex-maximal-in-mset linorder-lit.is-maximal-in-mset-iff*
linorder-lit.less-linear linorder-lit.less-trans
linorder-lit.multp-if-maximal-less-that-maximal set-mset-empty)

thus *False*
using *E-in nbex-gt-D* **by** *metis*
qed

hence $\neg\ (\exists\, A \mid\in\mid atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}).\ A \mid\notin\mid trail\text{-}atms\ \Gamma')$
using *step-hyps*
by (*metis atm-of-uminus finsert-iff fst-conv linorder-trm.antisym-conv3 trail-atms.simps*(*2*))

then show $atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}) \mid\subseteq\mid trail\text{-}atms\ \Gamma'$
by *blast*
qed

moreover have $\forall\, D.\ None = Some\ D \longrightarrow (\forall\, A \mid\in\mid trail\text{-}atms\ \Gamma.$
$\exists\, L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ D \wedge A \preceq_t atm\text{-}of\ L)$
by *simp*

moreover have $\forall\, Ln \in set\ \Gamma'.\ snd\ Ln = None \longleftrightarrow is\text{-}neg\ (fst\ Ln)$
using Γ*-deci-iff-neg* ‹*is-pos K*›
by (*simp add*: ‹$\Gamma' = (-\ K,\ None)\ \#\ \Gamma$› *is-pos-neg-not-is-pos*)

moreover have *trail-true-cls* $\Gamma'\ C$
if $Ln \in set\ \Gamma'$ **and** $snd\ Ln = None$ **and** $C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and** $C \prec_c \{\#fst\ Ln\#\}$
for *Ln C*
proof −
have $Ln = (-\ K,\ None) \vee Ln \in set\ \Gamma$
using ‹$Ln \in set\ \Gamma'$› **unfolding** ‹$\Gamma' = (-\ K,\ None)\ \#\ \Gamma$› **by** *simp*

hence *trail-true-cls* Γ *C*
proof (*elim disjE*)
assume *Ln* = (− *K*, *None*)

hence ∀ *x* ∈# *C*. *x* ≺$_l$ − *K*
using ‹*C* ≺$_c$ {#*fst Ln*#}›
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence *C* ⪯$_c$ *D*
using *D-max-lit step-hyps*
using *linorder-cls.leI that*(*3*) **by** *blast*

thus *trail-true-cls* Γ *C*
using ‹*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› *D-true*
using *clauses-lt-D-true* **by** *blast*
next
assume *Ln* ∈ *set* Γ

thus *trail-true-cls* Γ *C*
using Γ-*deci-ball-lt-true* ‹*snd Ln* = *None*› ‹*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})›
‹*C* ≺$_c$ {#*fst Ln*#}›
by *metis*
qed

thus *trail-true-cls* Γ′ *C*
using ‹*suffix* Γ Γ′› **by** (*metis trail-true-cls-if-trail-true-suffix*)
qed

moreover have ∀ *Ln* ∈ *set* Γ′. ∀ *C*. *snd Ln* = *Some C* ⟶ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N*
|∪| U_{er})
using Γ-*prop-in* **by** (*simp add*: ‹Γ′ = (− *K*, *None*) # Γ›)

moreover have ∀ *Ln* ∈ *set* Γ′. ∀ *C*. *snd Ln* = *Some C* ⟶ *linorder-lit.is-greatest-in-mset*
C (*fst Ln*)
using Γ-*prop-greatest* **by** (*simp add*: ‹Γ′ = (− *K*, *None*) # Γ›)

moreover have ∀ Γ$_1$ *L C* Γ$_0$. Γ′ = Γ$_1$ @ (*L*, *Some C*) # Γ$_0$ ⟶ *trail-false-cls*
Γ$_0$ {#*K* ∈# *C*. *K* ≠ *L*#}
using Γ-*prop-almost-false*
unfolding ‹Γ′ = (− *K*, *None*) # Γ›
by (*metis* (*no-types*, *lifting*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have (∀ Γ$_1$ *L D* Γ$_0$. Γ′ = Γ$_1$ @ (*L*, *Some D*) # Γ$_0$ ⟶
(∀ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶ *trail-true-cls* Γ$_0$ *C*))
using Γ-*prop-ball-lt-true*
unfolding ‹Γ′ = (− *K*, *None*) # Γ›
by (*metis D-true clauses-lt-D-true linorder-cls.neq-iff list.inject step-hyps*(*10*)
suffix-Cons

suffix-def)

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}, \mathcal{F}, \Gamma', None)$›
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ' C
$\bigwedge K_C$. *linorder-lit.is-maximal-in-mset* C $K_C \Longrightarrow$
$\neg$ ($\exists$ A |$\in$| *atms-of-clss* (N |$\cup$| U_{er}). $A \prec_t$ *atm-of* $K_C \wedge A$ |$\notin$| *trail-atms* Γ')
if *C-lt*: $\bigwedge E$. *None* = *Some* $E \Longrightarrow C \prec_c E$ **and** *C-in*: C |$\in$| *iefac* $\mathcal{F}$ |‘| (N |$\cup$| U_{er}) **for** C
proof −
have *None* = *The-optional* (*linorder-cls.is-least-in-fset*
(*ffilter* (($\prec_c$) D) (*iefac* $\mathcal{F}$ |‘| (N |$\cup$| U_{er}))))
using *step-hyps*
by (*metis* (*no-types*, *opaque-lifting*) *Some-eq-The-optionalD*
linorder-cls.is-least-in-ffilter-iff not-Some-eq)

hence $C \preceq_c D$
using *step-hyps that* **by** (*metis clause-le-if-lt-least-greater*)

hence $C \prec_c D \vee C = D$
by *simp*

hence *trail-true-cls* Γ C
proof (*elim disjE*)
assume $C \prec_c D$
thus *trail-true-cls* Γ C
using *clauses-lt-D-true* ‹C |$\in$| *iefac* $\mathcal{F}$ |‘| (N |$\cup$| U_{er})› **by** *metis*
next
assume $C = D$
thus *trail-true-cls* Γ C
using *D-true* **by** *argo*
qed

thus *trail-true-cls* Γ' C
using ‹*suffix* Γ Γ'› **by** (*metis trail-true-cls-if-trail-true-suffix*)

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset* C K_C
thus $\neg$ ($\exists$ A |$\in$| *atms-of-clss* (N |$\cup$| U_{er}). $A \prec_t$ *atm-of* $K_C \wedge A$ |$\notin$| *trail-atms* Γ')
using ‹$C \prec_c D \vee C = D$›
proof (*elim disjE*)
assume $C \prec_c D$
hence $\neg$ ($\exists$ A|$\in$|*atms-of-clss* (N |$\cup$| U_{er}). $A \prec_t$ *atm-of* $K_C \wedge A$ |$\notin$| *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-D C-in C-max-lit* **by** *force*
thus *?thesis*
using *step-hyps*(*9*) **by** *force*

next
assume $C = D$
thus *?thesis*
by (*metis C-max-lit D-max-lit finsert-iff linorder-cls.order.irrefl linorder-lit.antisym-conv3 linorder-lit.multp-if-maximal-less-that-maximal step-hyps*(5) *step-hyps*(9) *trail-atms.simps*(2))
qed
qed

thus
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $(\forall\, D.\ None = Some\ D \longrightarrow C \prec_c D) \longrightarrow$ *trail-true-cls* Γ' C
$\forall\, C|\in|$*iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $(\forall\, D.\ None = Some\ D \longrightarrow C \prec_c D) \longrightarrow$
$(\forall\, K.$ *ord-res.is-maximal-lit* $K\ C \longrightarrow$
$\neg\ (\exists\, A|\in|$*atms-of-clss* $(N\ |\cup|\ U_{er})$. $A \prec_t$ *atm-of* $K \wedge A\ |\notin|$ *trail-atms* $\Gamma')$)
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*production* $\Gamma\ D\ K\ U_{er}\ \Gamma'\ \mathcal{C}'\ \mathcal{F}$)

note *D-max-lit* = ‹*ord-res.is-maximal-lit K D*›

have *suffix* $\Gamma\ \Gamma'$
using *step-hyps* **by** (*simp add*: *suffix-def*)

have
$\mathcal{F}$*-subset*: $\mathcal{F}\ |\subseteq|\ N\ |\cup|\ U_{er}$ **and**
D-in: $D\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall\, C\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $C \prec_c D \longrightarrow$
$(\forall\, L_C.$ *linorder-lit.is-maximal-in-mset* $C\ L_C \longrightarrow$
$\neg$ *trail-defined-lit* $\Gamma\ L_C \longrightarrow$ (*trail-true-cls* Γ $\{\#K \in\#\ C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall\, C\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall\, C\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $C \prec_c D \longrightarrow$
$(\forall\, K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists\, A\ |\in|$ *atms-of-clss* $(N\ |\cup|\ U_{er})$. $A \prec_t$ *atm-of* $K \wedge A\ |\notin|$ *trail-atms* $\Gamma))$ **and**
Γ*-sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))\ \Gamma$ **and**
Γ*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N\ |\cup|\ U_{er})$) **and**
trail-atms-le0: $\forall\, A\ |\in|$ *trail-atms* $\Gamma.\ \exists\, L.$ *ord-res.is-maximal-lit* $L\ D \wedge (A \preceq_t$ *atm-of* $L)$ **and**
Γ*-deci-iff-neg*: $\forall\, Ln \in set\ \Gamma.$ *snd* $Ln = None \longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ*-deci-ball-lt-true*: $\forall\, Ln \in set\ \Gamma.$ *snd* $Ln = None \longrightarrow$
$(\forall\, C\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|\ U_{er})$. $C \prec_c \{\#fst\ Ln\#\} \longrightarrow$ *trail-true-cls* $\Gamma\ C)$ **and**
Γ*-prop-in*: $\forall\, Ln \in set\ \Gamma.\ \forall\, C.$ *snd* $Ln = Some\ C \longrightarrow C\ |\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N\ |\cup|$

U_{er}) **and**
Γ-*prop-greatest*:
$\forall\, Ln \in set\ \Gamma.\ \forall\, C.\ snd\ Ln = Some\ C \longrightarrow linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ (fst\ Ln)$ **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L, Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\# C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L, Some\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall\, C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C))$
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ D)$› *ord-res-7-invars-def*)

have *clauses-lt-D-almost-defined*: $\forall\, C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$
$(\forall\, K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow trail\text{-}defined\text{-}cls\ \Gamma\ \{\#L \in\# C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ D)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have $K \in\# D$
using ‹*ord-res.is-maximal-lit K D*›
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of K* $\mid\in\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$)
using *D-in atm-of-in-atms-of-clssI* **by** *metis*

have *atm-of K* $\mid\notin\mid$ *trail-atms* Γ
using ‹¬ *trail-defined-lit* Γ *K*›
by (*metis trail-defined-lit-iff-trail-defined-atm*)

hence *trail-atms-lt*: $\forall\, A \mid\in\mid trail\text{-}atms\ \Gamma.\ A \prec_t atm\text{-}of\ K$
using *trail-atms-le0* ‹*ord-res.is-maximal-lit K D*›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset linorder-trm.antisym-conv1*)

have $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$
using $\mathcal{F}$-*subset* **.**

moreover have $\forall\, C'.\ \mathcal{C}' = Some\ C' \longrightarrow C' \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *step-hyps*(*11*) **by** (*metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD*)

moreover have *trail-true-cls* Γ' $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
if $\mathcal{C}' = Some\ E$ **and**
C-in: $C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C* L_C **and**
L_C-*undef*: ¬ *trail-defined-lit* Γ' L_C
for *E C* L_C
proof −
have *linorder-cls.is-least-in-fset* (*ffilter* $((\prec_c)\ D)$ (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) *E*
using ‹$\mathcal{C}' = Some\ E$› *step-hyps* **by** (*metis Some-eq-The-optionalD*)
hence $C \prec_c D \vee C = D$

unfolding *linorder-cls.is-least-in-ffilter-iff*
using *C-lt* **by** (*metis C-in linorder-cls.not-less-iff-gr-or-eq*)
thus *?thesis*
proof (*elim disjE*)
assume $C \prec_c D$

moreover have $\neg$ *trail-defined-lit* Γ L_C
using L_C*-undef* ‹*suffix* Γ Γ'›
using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

ultimately have *trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in C-max-lit*] **by** *metis*

thus *?thesis*
using ‹*suffix* Γ Γ'› *trail-true-cls-if-trail-true-suffix* **by** *blast*
next
assume $C = D$
hence $L_C = K$
using *C-max-lit D-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)
moreover have *trail-defined-lit* Γ' K
by (*simp add*: *step-hyps trail-defined-lit-def*)
ultimately have *False*
using L_C*-undef* **by** *argo*
thus *?thesis* **..**
qed
qed

moreover have *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ'
proof −
have $x \prec_t$ *atm-of* K **if** *x-in*: x $|\in|$ *trail-atms* Γ **for** x
using *x-in trail-atms-lt* **by** *metis*

hence $\forall\, x \in set\ \Gamma.$ *atm-of* (*fst* x) $\prec_t$ *atm-of* K
by (*simp add*: *fset-trail-atms*)

thus *?thesis*
using Γ*-sorted*
by (*simp add*: ‹$\Gamma' = (K,$ *Some* $D)$ # Γ›)
qed

moreover have Γ'*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* (N $|\cup|$ U_{er}))
proof −
have *linorder-trm.is-lower-set* (*insert* (*atm-of* K) (*fset* (*trail-atms* Γ)))
(*fset* (*atms-of-clss* (N $|\cup|$ U_{er})))
proof (*rule linorder-trm.is-lower-set-insertI*)
show *atm-of* K $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er})

```
      using ‹atm-of K |∈| atms-of-clss (N |∪| U_er)› .
    next
      show ∀ w|∈|atms-of-clss (N |∪| U_er). w ≺_t (atm-of K) ⟶ w |∈| trail-atms
Γ
        using step-hyps(5)[unfolded linorder-trm.is-least-in-ffilter-iff, simplified]
        by fastforce
    next
      show linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N |∪| U_er))
        using Γ-lower .
    qed

    moreover have trail-atms Γ' = finsert (atm-of K) (trail-atms Γ)
      by (simp add: ‹Γ' = (K, Some D) # Γ›)

    ultimately show ?thesis
      by simp
  qed

  moreover have trail-atms Γ' = atms-of-clss (N |∪| U_er) if C' = None
  proof (intro fsubset-antisym)
    show trail-atms Γ' |⊆| atms-of-clss (N |∪| U_er)
      using Γ'-lower unfolding linorder-trm.is-lower-set-iff by blast
  next
    have nbex-gt-D: ¬ (∃ E |∈| iefac F |`| (N |∪| U_er). D ≺_c E)
      using step-hyps ‹C' = None›
      by (metis clause-le-if-lt-least-greater linorder-cls.leD option.simps(3))

    have ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). atm-of K ≺_t A)
    proof (intro notI , elim bexE)
      fix A :: 'f gterm
      assume A |∈| atms-of-clss (N |∪| U_er) and atm-of K ≺_t A

      hence A |∈| atms-of-clss (iefac F |`| (N |∪| U_er))
        by simp

      then obtain E L where E-in: E |∈| iefac F |`| (N |∪| U_er) and L ∈# E
and A = atm-of L
        unfolding atms-of-clss-def atms-of-cls-def
        by (smt (verit, del-insts) fimage-iff fmember-ffUnion-iff fset-fset-mset)

      have K ≺_l L
        using ‹atm-of K ≺_t A› ‹A = atm-of L›
        by (cases K; cases L) simp-all

      hence D ≺_c E
        using D-max-lit ‹L ∈# E›
      by (metis empty-iff linorder-lit.ex-maximal-in-mset linorder-lit.is-maximal-in-mset-iff
            linorder-lit.less-linear linorder-lit.less-trans
            linorder-lit.multp-if-maximal-less-that-maximal set-mset-empty)
```

```
    thus False
      using E-in nbex-gt-D by metis
  qed

  hence ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). A |∉| trail-atms Γ')
    using step-hyps
    by (metis finsert-iff fst-conv linorder-trm.antisym-conv3 trail-atms.simps(2))

  then show atms-of-clss (N |∪| U_er) |⊆| trail-atms Γ'
    by blast
qed

moreover have ∀ D. C' = Some D ⟶ (∀ A |∈| trail-atms Γ'.
  ∃ L. ord-res.is-maximal-lit L D ∧ A ⪯_t atm-of L)
proof (intro allI impI ballI)
  fix E :: 'f gterm literal multiset and A :: 'f gterm
  assume C' = Some E and A |∈| trail-atms Γ'

  have linorder-cls.is-least-in-fset (ffilter ((≺_c) D) (iefac F |`| (N |∪| U_er))) E
    using step-hyps(11) ‹C' = Some E› by (metis Some-eq-The-optionalD)

  hence D ≺_c E
    unfolding linorder-cls.is-least-in-ffilter-iff by argo

  obtain L where linorder-lit.is-maximal-in-mset E L
  by (metis ‹D ≺_c E› linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls)

  show ∃ L. ord-res.is-maximal-lit L E ∧ A ⪯_t atm-of L
  proof (intro exI conjI)
    show ord-res.is-maximal-lit L E
      using ‹ord-res.is-maximal-lit L E› .
  next
    have K ⪯_l L
      using step-hyps(4) ‹ord-res.is-maximal-lit L E›
      by (metis ‹D ≺_c E› linorder-cls.less-asym linorder-lit.leI
          linorder-lit.multp-if-maximal-less-that-maximal)

    hence atm-of K ⪯_t atm-of L
      by (cases K; cases L) simp-all

    moreover have A ⪯_t atm-of K
    proof −
      have A = atm-of K ∨ A |∈| trail-atms Γ
        using ‹A |∈| trail-atms Γ'›
        unfolding ‹Γ' = (K, Some D) # Γ›
        by simp

      thus A ⪯_t atm-of K
```

using *trail-atms-lt* **by** *blast*
qed

ultimately show $A \preceq_t$ *atm-of* L
by *order*
qed
qed

moreover have $\forall$ *Ln* $\in$ *set* Γ'. *snd Ln* = *None* $\longleftrightarrow$ *is-neg* (*fst Ln*)
unfolding ‹Γ' = (*K*, *Some D*) # Γ›
using Γ-*deci-iff-neg* ‹*is-pos K*› **by** *simp*

moreover have *trail-true-cls* Γ' *C*
if *Ln* $\in$ *set* Γ' **and** *snd Ln* = *None* **and** *C* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}) **and** *C* $\prec_c$ {#*fst Ln*#}
for *Ln C*
proof −
have *Ln* = (*K*, *Some D*) $\vee$ *Ln* $\in$ *set* Γ
using ‹*Ln* $\in$ *set* Γ'› **unfolding** ‹Γ' = (*K*, *Some D*) # Γ› **by** *simp*

hence *trail-true-cls* Γ *C*
proof (*elim disjE*)
assume *Ln* = (*K*, *Some D*)
hence *False*
using ‹*snd Ln* = *None*› **by** *simp*
thus *?thesis* **..**
next
assume *Ln* $\in$ *set* Γ

thus *trail-true-cls* Γ *C*
using Γ-*deci-ball-lt-true* ‹*snd Ln* = *None*› ‹*C* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er})› ‹*C* $\prec_c$ {#*fst Ln*#}›
by *metis*
qed

thus *trail-true-cls* Γ' *C*
using ‹*suffix* Γ Γ'› **by** (*metis trail-true-cls-if-trail-true-suffix*)
qed

moreover have $\forall$ *Ln* $\in$ *set* Γ'. $\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *C* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er})
using Γ-*prop-in step-hyps*(*10*) ‹*D* |$\in$| *iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er})› **by** *simp*

moreover have $\forall$ *Ln* $\in$ *set* Γ'. $\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *linorder-lit.is-greatest-in-mset C* (*fst Ln*)
using Γ-*prop-greatest step-hyps*(*8*,*9*,*10*) **by** *simp*

moreover have $\forall \Gamma_1$ *L C* Γ_0. Γ' = Γ_1 @ (*L*, *Some C*) # Γ_0 $\longrightarrow$ *trail-false-cls* Γ_0 {#*K* $\in$# *C*. *K* $\neq$ *L*#}

unfolding ‹$\Gamma' = (K,\ \mathit{Some}\ D) \#\ \Gamma$›
using Γ-*prop-almost-false* ‹*trail-false-cls* Γ $\{\#x \in\# D.\ x \neq K\#\}$›
by (*metis* (*no-types*, *lifting*) *append-eq-Cons-conv list.inject option.inject prod.inject*)

moreover have $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma' = \Gamma_1\ @\ (L,\ \mathit{Some}\ D) \#\ \Gamma_0 \longrightarrow$
$(\forall\ C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow \mathit{trail\text{-}true\text{-}cls}\ \Gamma_0\ C))$
proof −
have $\forall\ C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow \mathit{trail\text{-}true\text{-}cls}\ \Gamma\ C$
proof (*intro ballI impI*)
fix $C :: 'f\ \mathit{gterm\ literal\ multiset}$
assume $C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and** $C \prec_c D$
thus *trail-true-cls* Γ C
using *clauses-lt-D-true* **by** *metis*
qed

thus *?thesis*
unfolding ‹$\Gamma' = (K,\ \mathit{Some}\ D) \#\ \Gamma$›
by (*smt* (*verit*, *ccfv-SIG*) Γ-*prop-ball-lt-true append-eq-Cons-conv list.inject option.inject*
prod.inject)
qed

ultimately show *?thesis*
unfolding ‹$s' = (U_{er},\ \mathcal{F},\ \Gamma',\ \mathcal{C}')$›
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ' C
$\bigwedge K_C.$ *linorder-lit.is-maximal-in-mset* $C\ K_C \Longrightarrow$
$\neg\ (\exists A\ |\in|\ \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).\ A \prec_t \mathit{atm\text{-}of}\ K_C \wedge A\ |\notin|\ \mathit{trail\text{-}atms}\ \Gamma')$
if *C-lt*: $\bigwedge E.\ \mathcal{C}' = \mathit{Some}\ E \Longrightarrow C \prec_c E$ **and** *C-in*: $C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **for** C
proof −
have $C \preceq_c D$
using *step-hyps that* **by** (*metis clause-le-if-lt-least-greater*)

hence $C \prec_c D \vee C = D$
by *simp*

thus *trail-true-cls* Γ' C
proof (*elim disjE*)
assume $C \prec_c D$

hence *trail-true-cls* Γ C
using *clauses-lt-D-true* ‹$C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$› **by** *metis*

thus *trail-true-cls* Γ' C
using ‹*suffix* $\Gamma\ \Gamma'$› **by** (*metis trail-true-cls-if-trail-true-suffix*)
next
assume $C = D$

```
      have trail-true-lit Γ′ K
        using ‹Γ′ = (K, Some D) # Γ› ‹is-pos K›
        unfolding trail-true-lit-def by (cases K) simp-all

      thus trail-true-cls Γ′ C
        unfolding ‹C = D› trail-true-cls-def
        using ‹K ∈# D› by metis
    qed

    fix K_C
    assume C-max-lit: linorder-lit.is-maximal-in-mset C K_C
   show ¬ (∃ A |∈| atms-of-clss (N |∪| U_er). A ≺_t atm-of K_C ∧ A |∉| trail-atms
Γ′)
      using ‹C ≺_c D ∨ C = D›
    proof (elim disjE)
      assume C ≺_c D
     hence ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of K_C ∧ A |∉| trail-atms
Γ)
        using no-undef-atm-lt-max-lit-if-lt-D C-in C-max-lit by force
      thus ?thesis
        using step-hyps by force
    next
      assume C = D
      thus ?thesis
     by (metis C-max-lit D-max-lit ‹suffix Γ Γ′› linorder-lit.Uniq-is-maximal-in-mset
        literal.sel(2) step-hyps(5) the1-equality′ trail-defined-lit-if-trail-defined-suffix
            trail-defined-lit-iff-trail-defined-atm)
    qed
  qed

  thus
  ∀ C|∈|iefac F |‘| (N |∪| U_er). (∀ D. C′ = Some D ⟶ C ≺_c D) ⟶ trail-true-cls
Γ′ C
    ∀ C|∈|iefac F |‘| (N |∪| U_er). (∀ D. C′ = Some D ⟶ C ≺_c D) ⟶
    (∀ K. ord-res.is-maximal-lit K C ⟶
      ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of K ∧ A |∉| trail-atms Γ′))
    unfolding atomize-conj by metis
 qed simp-all
next
 case step-hyps: (factoring Γ D K U_er F′ F)

 note D-max-lit = ‹ord-res.is-maximal-lit K D›

 have
   F-subset: F |⊆| N |∪| U_er and
   D-in: D |∈| iefac F |‘| (N |∪| U_er) and
   clauses-lt-D-seldomly-have-undef-max-lit:
     ∀ C |∈| iefac F |‘| (N |∪| U_er). C ≺_c D ⟶
       (∀ L_C. linorder-lit.is-maximal-in-mset C L_C ⟶
```

$\neg$ *trail-defined-lit* Γ L_C $\longrightarrow$ (*trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\}$ $\wedge$ *is-pos* L_C)) **and**
clauses-lt-D-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}). $C \prec_c D \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}). $C \prec_c D \longrightarrow$
($\forall K$. *linorder-lit.is-maximal-in-mset* C $K \longrightarrow$
$\neg$ ($\exists A \mid\in\mid$ *atms-of-clss* (N $\mid\cup\mid$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* Γ)) **and**
Γ-*sorted*: *sorted-wrt* ($\lambda x\ y$. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $\mid\cup\mid$ U_{er})) **and**
trail-atms-le0: $\forall A \mid\in\mid$ *trail-atms* Γ. $\exists L$. *ord-res.is-maximal-lit* L $D \wedge (A \preceq_t$ *atm-of* L) **and**
Γ-*deci-iff-neg*: $\forall Ln \in$ *set* Γ. *snd* Ln = *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ-*deci-ball-lt-true*: $\forall Ln \in$ *set* Γ. *snd* Ln = *None* $\longrightarrow$
($\forall C \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}). $C \prec_c \{\#$*fst* $Ln\#\} \longrightarrow$ *trail-true-cls* Γ C) **and**
Γ-*prop-in*: $\forall Ln \in$ *set* Γ. $\forall D$. *snd* Ln = *Some* $D \longrightarrow D \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}) **and**
Γ-*prop-greatest*:
$\forall Ln \in$ *set* Γ. $\forall D$. *snd* Ln = *Some* $D \longrightarrow$ *linorder-lit.is-greatest-in-mset* D (*fst* Ln) **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0$. $\Gamma = \Gamma_1$ @ (L, *Some* C) # $\Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\# C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: ($\forall \Gamma_1\ L\ D\ \Gamma_0$. $\Gamma = \Gamma_1$ @ (L, *Some* D) # $\Gamma_0 \longrightarrow$
($\forall C \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}). $C \prec_c D \longrightarrow$ *trail-true-cls* Γ_0 C))
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$› *ord-res-7-invars-def*)

have *clauses-lt-D-almost-defined*: $\forall C \mid\in\mid$ *iefac* $\mathcal{F}$ $\mid`\mid$ (N $\mid\cup\mid$ U_{er}). $C \prec_c D \longrightarrow$
($\forall K$. *linorder-lit.is-maximal-in-mset* C $K \longrightarrow$ *trail-defined-cls* Γ $\{\#L \in\# C.\ L \neq K\#\}$)
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $D)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have *atm-of* $K \mid\notin\mid$ *trail-atms* Γ
using ‹$\neg$ *trail-defined-lit* Γ K›
by (*metis trail-defined-lit-iff-trail-defined-atm*)

hence *trail-atms-lt*: $\forall A \mid\in\mid$ *trail-atms* Γ. $A \prec_t$ *atm-of* K
using *trail-atms-le0* ‹*ord-res.is-maximal-lit* K D›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset linorder-trm.antisym-conv1*)

have *efac* $D \neq D$
using ‹$\neg$ *ord-res.is-strictly-maximal-lit* K D› *D-max-lit* ‹*is-pos* K›
by (*metis ex1-efac-eq-factoring-chain ex-ground-factoringI is-pos-def*)

hence *efac* $D \prec_c D$
by (*metis efac-properties-if-not-ident*(*1*))

hence *D-in-strong*: $D \mid\in\mid N \mid\cup\mid U_{er}$ **and** $D \mid\notin\mid \mathcal{F}$
using *D-in* ‹*efac* $D \neq D$›
unfolding *atomize-conj iefac-def*
by (*smt* (*verit*) *factorizable-if-neq-efac fimage-iff iefac-def ex1-efac-eq-factoring-chain*)

have $\mathcal{F}' \mid\subseteq\mid N \mid\cup\mid U_{er}$
unfolding ‹$\mathcal{F}' =$ *finsert* D $\mathcal{F}$›
using $\mathcal{F}$*-subset D-in-strong* **by** *simp*

moreover have $\forall C'.$ *Some* (*efac* D) = *Some* $C' \longrightarrow C' \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$
proof −
have *efac* $D \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$
using *D-in-strong* **by** (*simp add*: *iefac-def* ‹$\mathcal{F}' =$ *finsert* D $\mathcal{F}$›)

thus *?thesis*
by *simp*
qed

moreover have *trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
if *Some* (*efac* D) = *Some* E **and**
C-in: $C \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* C L_C **and**
L_C*-undef*: $\neg$ *trail-defined-lit* Γ L_C
for E C L_C
proof −
have $E =$ *efac* D
using *that* **by** *simp*
hence $C \prec_c$ *efac* D
using *C-lt* **by** *order*
hence $C \neq$ *efac* D
by *order*
hence $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *C-in iefac-def step-hyps*(*10*) **by** *auto*
moreover have $C \prec_c D$
using ‹$C \prec_c$ *efac* D› ‹*efac* $D \prec_c D$› **by** *order*
ultimately show *trail-true-cls* Γ $\{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C
using *clauses-lt-D-seldomly-have-undef-max-lit C-max-lit* L_C*-undef* **by** *metis*
qed

moreover have *trail-true-cls* Γ C
$\bigwedge K_C.$ *linorder-lit.is-maximal-in-mset* C $K_C \Longrightarrow$ *trail-defined-cls* Γ $\{\#L \in\# C.\ L \neq K_C\#\}$
if *C-lt*: $\bigwedge E.$ *Some* (*efac* D) = *Some* $E \Longrightarrow C \prec_c E$ **and** *C-in*: $C \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$ **for** C
proof −
have $C \prec_c$ *efac* D

using *C-lt* **by** *metis*

hence $C \neq efac\ D$
by *order*

hence $C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *C-in* **by** (*auto simp*: ‹$\mathcal{F}' = finsert\ D\ \mathcal{F}$› *iefac-def*)

moreover have $C \prec_c D$
using ‹$C \prec_c efac\ D$› ‹$efac\ D \prec_c D$› **by** *order*

ultimately show *trail-true-cls* $\Gamma\ C$
using *clauses-lt-D-true* **by** *metis*

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset* $C\ K_C$
show *trail-defined-cls* $\Gamma\ \{\#L \in\#\ C.\ L \neq K_C\#\}$
using *clauses-lt-D-almost-defined* ‹$C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$› ‹$C \prec_c D$› *C-max-lit*
by *metis*
qed

moreover have *sorted-wrt* $(\lambda x\ y.\ atm\text{-}of\ (fst\ y) \prec_t atm\text{-}of\ (fst\ x))\ \Gamma$
using Γ-*sorted* **.**

moreover have *linorder-trm.is-lower-fset* $(trail\text{-}atms\ \Gamma)\ (atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}))$
using Γ-*lower* **.**

moreover have $\forall D'.\ Some\ (efac\ D) = Some\ D' \longrightarrow (\forall A \mid\in\mid trail\text{-}atms\ \Gamma.\ \exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ D' \wedge A \preceq_t atm\text{-}of\ L)$
proof −
have $\forall A \mid\in\mid trail\text{-}atms\ \Gamma.\ \exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ (efac\ D) \wedge A \preceq_t atm\text{-}of\ L$
using *trail-atms-lt*
using *ex1-efac-eq-factoring-chain step-hyps*(*4*)
ord-res.ground-factorings-preserves-maximal-literal **by** *blast*

thus *?thesis*
by *simp*
qed

moreover have $\forall Ln \in set\ \Gamma.\ snd\ Ln = None \longleftrightarrow is\text{-}neg\ (fst\ Ln)$
using Γ-*deci-iff-neg* **.**

moreover have *trail-true-cls* $\Gamma\ C$
if $Ln \in set\ \Gamma$ **and** $snd\ Ln = None$ **and** $C \mid\in\mid iefac\ \mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er})$ **and** $C \prec_c \{\#fst\ Ln\#\}$
for $Ln\ C$

```
  proof −
    have C = efac D ∨ C |∈| iefac F |‘| (N |∪| U_er)
      using ‹C |∈| iefac F' |‘| (N |∪| U_er)› by (auto simp: iefac-def ‹F' = finsert
D F›)

    thus trail-true-cls Γ C
    proof (elim disjE)
      assume C = efac D

      hence linorder-lit.is-greatest-in-mset C K
        using D-max-lit ‹is-pos K›
        by (metis greatest-literal-in-efacI)

      hence K ≺_l fst Ln
        using ‹C ≺_c {#fst Ln#}›
        by (simp add: linorder-lit.is-greatest-in-mset-iff)

      hence atm-of K ⪯_t atm-of (fst Ln)
        by (cases K; cases fst Ln) simp-all

      moreover have atm-of (fst Ln) |∈| trail-atms Γ
        using ‹Ln ∈ set Γ› by (simp add: fset-trail-atms)

      moreover have atm-of K |∈| atms-of-clss (N |∪| U_er)
      by (meson D-in D-max-lit atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff)

      ultimately have atm-of K |∈| trail-atms Γ
        using Γ-lower
        unfolding linorder-trm.is-lower-set-iff
        by fastforce

      hence False
        using ‹atm-of K |∉| trail-atms Γ› by contradiction

      thus trail-true-cls Γ C ..
    next
      assume C |∈| iefac F |‘| (N |∪| U_er)
      thus trail-true-cls Γ C
       using Γ-deci-ball-lt-true ‹Ln ∈ set Γ› ‹snd Ln = None› ‹C ≺_c {#fst Ln#}›
        by metis
    qed
  qed

  moreover have C |∈| iefac F' |‘| (N |∪| U_er) if Ln ∈ set Γ and snd Ln =
Some C for Ln C
  proof −
    have atm-of (fst Ln) ≺_t atm-of K
      using trail-atms-lt[unfolded fset-trail-atms, simplified] ‹Ln ∈ set Γ› by metis
```

hence *atm-of* (*fst Ln*) ≠ *atm-of K*
by *order*

hence *fst Ln* ≠ *K*
by (*cases fst Ln*; *cases K*) *simp-all*

moreover have *ord-res.is-maximal-lit* (*fst Ln*) *C*
using Γ-*prop-greatest* ‹*Ln* ∈ *set* Γ› ‹*snd Ln* = *Some C*› **by** *blast*

ultimately have *C* ≠ *D*
using ‹*ord-res.is-maximal-lit K D*› **by** (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

moreover have *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using Γ-*prop-in* ‹*Ln* ∈ *set* Γ› ‹*snd Ln* = *Some C*› **by** *metis*

ultimately show *?thesis*
by (*auto simp*: ‹$\mathcal{F}'$ = *finsert D* $\mathcal{F}$› *iefac-def*)
qed

moreover have ∀ *Ln* ∈ *set* Γ. ∀ *D*. *snd Ln* = *Some D* ⟶ *linorder-lit.is-greatest-in-mset D* (*fst Ln*)
using Γ-*prop-greatest* **by** *simp*

moreover have ∀ Γ_1 *L C* Γ_0. Γ = Γ_1 @ (*L*, *Some C*) # Γ_0 ⟶ *trail-false-cls* Γ_0 {#*K* ∈# *C*. *K* ≠ *L*#}
using Γ-*prop-almost-false* **.**

moreover have (∀ Γ_1 *L D* Γ_0. Γ = Γ_1 @ (*L*, *Some D*) # Γ_0 ⟶
(∀ *C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D* ⟶ *trail-true-cls* Γ_0 *C*))
proof (*intro allI impI ballI*)
fix
Γ_1 Γ_0 :: (*'f gliteral* × *'f gclause option*) *list* **and**
L :: *'f gliteral* **and**
C_0 C_1 :: *'f gclause*
assume
Γ-*eq*: Γ = Γ_1 @ (*L*, *Some* C_1) # Γ_0 **and**
C_0-*in*: C_0 |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}) **and**
C_0 ≺$_c$ C_1

have C_0 = *efac D* ∨ C_0 |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using C_0-*in* **by** (*auto simp*: *iefac-def* ‹$\mathcal{F}'$ = *finsert D* $\mathcal{F}$›)

thus *trail-true-cls* Γ_0 C_0
proof (*elim disjE*)
assume C_0 = *efac D*

have *atm-of L* |∈| *trail-atms* Γ
using Γ-*eq* **unfolding** *fset-trail-atms* **by** *simp*

hence *atm-of L* $\prec_t$ *atm-of K*
using *trail-atms-lt* **by** *metis*

hence $L \prec_l K$
by (*cases L*; *cases K*) *simp-all*

moreover have *linorder-lit.is-greatest-in-mset* C_1 *L*
using Γ*-eq* Γ*-prop-greatest* **by** *simp*

moreover have *linorder-lit.is-greatest-in-mset* (*efac D*) *K*
using ‹*is-pos K*› *D-max-lit* **by** (*metis greatest-literal-in-efacI*)

ultimately have $C_1 \prec_c$ *efac D*
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*
linorder-lit.multp-if-maximal-less-that-maximal)

hence *False*
using ‹C_0 = *efac D*› ‹$C_0 \prec_c C_1$› **by** *order*

thus *?thesis* **..**
next
assume C_0 |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
thus *?thesis*
using Γ*-prop-ball-lt-true* Γ*-eq* ‹$C_0 \prec_c C_1$› **by** *metis*
qed
qed

ultimately show *?thesis*
unfolding ‹s' = (U_{er}, $\mathcal{F}'$, Γ, *Some* (*efac D*))›
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ *C*
⋀K_C. *linorder-lit.is-maximal-in-mset C* K_C ⟹
¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* $\prec_t$ *atm-of* K_C ∧ *A* |∉| *trail-atms* Γ)
if *C-lt*: ⋀*E*. *Some* (*efac D*) = *Some E* ⟹ *C* $\prec_c$ *E* **and** *C-in*: *C* |∈| *iefac* $\mathcal{F}'$
|‘| (*N* |∪| U_{er})
for *C*
proof −
have *C* $\prec_c$ *efac D*
using *C-lt* **by** *metis*

hence *C* ≠ *efac D*
by *order*

hence *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using *C-in* **by** (*auto simp*: ‹$\mathcal{F}'$ = *finsert D* $\mathcal{F}$› *iefac-def*)

moreover have *C* $\prec_c$ *D*
using ‹*C* $\prec_c$ *efac D*› ‹*efac D* $\prec_c$ *D*› **by** *order*

ultimately show *trail-true-cls* Γ C
using *clauses-lt-D-true* **by** *metis*

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset* C K_C
thus $\neg$ ($\exists$ A $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K_C \wedge A$ $|\notin|$ *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-D* ‹C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})› ‹$C \prec_c D$› **by** *metis*
qed

thus
$\forall$ $C|\in|$*iefac* $\mathcal{F}'$ $|`|$ (N $|\cup|$ U_{er}). ($\forall$ *Da*. *Some* (*efac* D) = *Some* *Da* $\longrightarrow$ $C \prec_c$ *Da*) $\longrightarrow$
trail-true-cls Γ C
$\forall$ $C|\in|$*iefac* $\mathcal{F}'$ $|`|$ (N $|\cup|$ U_{er}). ($\forall$ *Da*. *Some* (*efac* D) = *Some* *Da* $\longrightarrow$ $C \prec_c$ *Da*) $\longrightarrow$
($\forall$ K. *ord-res.is-maximal-lit* K C $\longrightarrow$
$\neg$ ($\exists$ $A|\in|$*atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ))
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*resolution-bot* Γ E K D U_{er}' U_{er} Γ' $\mathcal{F}$)

have
$\mathcal{F}$-*subset*: $\mathcal{F}$ $|\subseteq|$ N $|\cup|$ U_{er} **and**
E-in: E $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}) **and**
clauses-lt-E-seldomly-have-undef-max-lit:
$\forall$ C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c E \longrightarrow$
($\forall$ L_C. *linorder-lit.is-maximal-in-mset* C $L_C \longrightarrow$
$\neg$ *trail-defined-lit* Γ $L_C \longrightarrow$ (*trail-true-cls* Γ {#K $\in$# C. $K \neq L_C$#} $\wedge$ *is-pos* L_C)) **and**
clauses-lt-E-true: $\forall$ C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c E \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-E: $\forall$ C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c E \longrightarrow$
($\forall$ K. *linorder-lit.is-maximal-in-mset* C $K \longrightarrow$
$\neg$ ($\exists$ A $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}). $A \prec_t$ *atm-of* $K \wedge A$ $|\notin|$ *trail-atms* Γ)) **and**
Γ-*sorted*: *sorted-wrt* (λx y. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $|\cup|$ U_{er})) **and**
trail-atms-le0: $\forall$ A $|\in|$ *trail-atms* Γ. $\exists$ L. *ord-res.is-maximal-lit* L $E \wedge$ ($A \preceq_t$ *atm-of* L) **and**
Γ-*deci-iff-neg*: $\forall$ *Ln* $\in$ *set* Γ. *snd Ln* = *None* $\longleftrightarrow$ *is-neg* (*fst Ln*) **and**
Γ-*deci-ball-lt-true*: $\forall$ *Ln* $\in$ *set* Γ. *snd Ln* = *None* $\longrightarrow$
($\forall$ C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}). $C \prec_c$ {#*fst Ln*#} $\longrightarrow$ *trail-true-cls* Γ C) **and**
Γ-*prop-in*: $\forall$ *Ln* $\in$ *set* Γ. $\forall$ C. *snd Ln* = *Some* $C \longrightarrow C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}) **and**

Γ-*prop-greatest*:
$\forall Ln \in set\ \Gamma.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ (fst\ Ln)$ **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\# C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C)$
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ E)$› *ord-res-7-invars-def*)

have *clauses-lt-E-almost-defined*: $\forall C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow trail\text{-}defined\text{-}cls\ \Gamma\ \{\#L \in\# C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma, Some\ E)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}'$
unfolding ‹$U_{er}' = finsert\ (eres\ D\ E)\ U_{er}$›
using $\mathcal{F}$-*subset* **by** *blast*

moreover have $\forall C'.\ Some\ \{\#\} = Some\ C' \longrightarrow C' \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$
by (*simp add*: ‹$eres\ D\ E = \{\#\}$› ‹$U_{er}' = finsert\ (eres\ D\ E)\ U_{er}$›)

moreover have $trail\text{-}true\text{-}cls\ \Gamma'\ \{\#K \in\# C.\ K \neq L_C\#\} \wedge is\text{-}pos\ L_C$
if $Some\ \{\#\} = Some\ E$ **and**
C-*in*: $C \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$ **and**
C-*lt*: $C \prec_c E$ **and**
C-*max-lit*: $linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ L_C$ **and**
L_C-*undef*: $\neg\ trail\text{-}defined\text{-}lit\ \Gamma'\ L_C$
for $E\ C\ L_C$
proof −
have $E = \{\#\}$
using *that* **by** *simp*
hence *False*
using C-*lt linorder-cls.leD mempty-lesseq-cls* **by** *blast*
thus *?thesis* **..**
qed

moreover have $trail\text{-}defined\text{-}cls\ \Gamma'\ \{\#L \in\# x.\ L \neq K_x\#\}$
if $Some\ \{\#\} = Some\ y$ **and** x-*in*: $x \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$ **and** $x \prec_c y$ **and**
x-*max-lit*: $linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ x\ K_x$ **for** $x\ y\ K_x$
proof −
have *False*
using *linorder-cls.leD mempty-lesseq-cls that*(*1*) *that*(*3*) **by** *blast*
thus *?thesis* **..**
qed

moreover have $sorted\text{-}wrt\ (\lambda x\ y.\ atm\text{-}of\ (fst\ y) \prec_t atm\text{-}of\ (fst\ x))\ \Gamma'$

unfolding ‹$\Gamma' = []$› **by** *simp*

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* (N |∪| U_{er}'))
unfolding ‹$\Gamma' = []$›
by (*simp add*: *linorder-trm.is-lower-set-iff*)

moreover have $\forall D.$ *Some* $\{\#\}$ = *Some* $D \longrightarrow (\forall A$ |∈| *trail-atms* $\Gamma'.$
$\exists L.$ *ord-res.is-maximal-lit* L $D \wedge A \preceq_t$ *atm-of* L)
unfolding ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ *snd Ln* = *None* ⟷ *is-neg* (*fst Ln*)
unfolding ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ *snd Ln* = *None* ⟶
($\forall C$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). $C \prec_c$ {#*fst Ln*#} ⟶ *trail-true-cls* Γ' C)
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ *snd Ln* = *None* ⟶
¬($\exists C$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). *linorder-lit.is-maximal-in-mset* C (− (*fst Ln*)))
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ $\forall D.$ *snd Ln* = *Some* D ⟶
¬($\exists C$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). $C \prec_c D \wedge$ *fst Ln* ∈# C)
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ $\forall C.$ *snd Ln* = *Some* C ⟶ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}')
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall$ *Ln* ∈ *set* $\Gamma'.$ $\forall C.$ *snd Ln* = *Some* C ⟶ *linorder-lit.is-greatest-in-mset* C (*fst Ln*)
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall \Gamma_1$ L C $\Gamma_0.$ $\Gamma' = \Gamma_1$ @ (L, *Some* C) # Γ_0 ⟶ *trail-false-cls* Γ_0 {#K ∈# $C.$ $K \neq L$#}
using ‹$\Gamma' = []$› **by** *simp*

moreover have $\forall \Gamma_1$ L D $\Gamma_0.$ $\Gamma' = \Gamma_1$ @ (L, *Some* D) # Γ_0 ⟶
($\forall C$ |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). $C \prec_c D$ ⟶ *trail-true-cls* Γ_0 C)
using ‹$\Gamma' = []$› **by** *simp*

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}', \mathcal{F}, \Gamma',$ *Some* $\{\#\})$›
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ' x
$\bigwedge K_x.$ *linorder-lit.is-maximal-in-mset* x $K_x \Longrightarrow$
¬ ($\exists A$ |∈| *atms-of-clss* (N |∪| U_{er}'). $A \prec_t$ *atm-of* $K_x \wedge A$ |∉| *trail-atms* Γ')

if *C-lt*: $\bigwedge E.\ Some\ \{\#\} = Some\ E \Longrightarrow x \prec_c E$ **and** *C-in*: $x \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}')$
for x
proof −
have $x \prec_c \{\#\}$
using *C-lt* **by** *metis*
hence *False*
using *linorder-cls.leD mempty-lesseq-cls* **by** *blast*
thus
trail-true-cls $\Gamma'\ x$
$\bigwedge K_x.$ *linorder-lit.is-maximal-in-mset* $x\ K_x \Longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}{}').\ A \prec_t$ *atm-of* $K_x \wedge A \mid\notin\mid$ *trail-atms* $\Gamma')$
by *argo+*
qed

thus
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}').\ (\forall D.\ Some\ \{\#\} = Some\ D \longrightarrow C \prec_c D) \longrightarrow$ *trail-true-cls* $\Gamma'\ C$
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}{}').\ (\forall D.\ Some\ \{\#\} = Some\ D \longrightarrow C \prec_c D) \longrightarrow$
$(\forall K.$ *ord-res.is-maximal-lit* $K\ C \longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}{}').\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma'))$
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*resolution-pos* $\Gamma\ E\ L\ D\ U_{er}{}'\ U_{er}\ \Gamma'\ K\ \mathcal{F}$)

note *E-max-lit* = ‹*ord-res.is-maximal-lit* $L\ E$›
note *eres-max-lit* = ‹*ord-res.is-maximal-lit* K (*eres* $D\ E$)›

have
$\mathcal{F}$*-subset*: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**
E-in: $E \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall L_C.$ *linorder-lit.is-maximal-in-mset* $C\ L_C \longrightarrow$
$\neg$ *trail-defined-lit* $\Gamma\ L_C \longrightarrow$ (*trail-true-cls* $\Gamma\ \{\#K \in\#\ C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C)) **and**
clauses-lt-E-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$ *trail-true-cls* $\Gamma\ C$ **and**
no-undef-atm-lt-max-lit-if-lt-E: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma))$ **and**
Γ*-sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* $(fst\ y) \prec_t$ *atm-of* $(fst\ x))\ \Gamma$ **and**
Γ*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$) **and**
trail-atms-le0: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ \exists L.$ *ord-res.is-maximal-lit* $L\ E \wedge (A \preceq_t$ *atm-of* $L)$ **and**

Γ-*deci-iff-neg*: $\forall\, Ln \in set\ \Gamma.\ snd\ Ln = None \longleftrightarrow is\text{-}neg\ (fst\ Ln)$ **and**
Γ-*deci-ball-lt-true*: $\forall\, Ln \in set\ \Gamma.\ snd\ Ln = None \longrightarrow$
$(\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c \{\#fst\ Ln\#\} \longrightarrow trail\text{-}true\text{-}cls\ \Gamma\ C)$
and
Γ-*prop-in*: $\forall\, Ln \in set\ \Gamma.\ \forall\, C.\ snd\ Ln = Some\ C \longrightarrow C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and**
Γ-*prop-greatest*:
$\forall\, Ln \in set\ \Gamma.\ \forall\, C.\ snd\ Ln = Some\ C \longrightarrow linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset\ C\ (fst\ Ln)$ **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ C)\ \#\ \Gamma_0 \longrightarrow trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (L,\ Some\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma_0\ C)$
using *invar* **by** (*simp all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,\ Some\ E)$› *ord-res-7-invars-def*)

have *clauses-lt-E-almost-defined*: $\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c E \longrightarrow$
$(\forall\, K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow trail\text{-}defined\text{-}cls\ \Gamma\ \{\#L \in\#\ C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,\ Some\ E)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

have *trail-atms-le*: $\forall\, A\ |\in|\ trail\text{-}atms\ \Gamma.\ A \preceq_t atm\text{-}of\ L$
using *trail-atms-le0 E-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have $(-\ L,\ Some\ D) \in set\ \Gamma$
using ‹*map-of* Γ $(-\ L) = Some\ (Some\ D)$› **by** (*metis map-of-SomeD*)

hence *D-in*: $D\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
using Γ-*prop-in* **by** *simp*

have *D-max-lit*: *linorder-lit.is-greatest-in-mset D* $(-\ L)$
using Γ-*prop-greatest* ‹$(-\ L,\ Some\ D) \in set\ \Gamma$› **by** *fastforce*

have *suffix* $\Gamma'\ \Gamma$
using *step-hyps*(9) *suffix-dropWhile* **by** *metis*

hence *atms-of-cls* (*eres D E*) $|\subseteq|$ *atms-of-cls D* $|\cup|$ *atms-of-cls E*
using *lit-in-one-of-resolvents-if-in-eres*
unfolding *atms-of-cls-def* **by** *fastforce*

moreover have *atms-of-cls D* $|\subseteq|$ *atms-of-clss* $(N\ |\cup|\ U_{er})$
using ‹$D\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$›
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb funion-upper1*)

moreover have *atms-of-cls E* $|\subseteq|$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
using ‹*E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ $(N \mid\cup\mid U_{er})$›
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb funion-upper1*)

ultimately have *atms-of-clss* $(N \mid\cup\mid U_{er})$ = *atms-of-clss* $(N \mid\cup\mid U_{er}')$
unfolding ‹U_{er}' = *finsert* (*eres D E*) U_{er}› *atms-of-clss-def* **by** *auto*

obtain A_L **where** *L-def*: L = *Neg* A_L
using ‹*is-neg L*› **by** (*cases L*) *simp-all*

have $D \prec_c E$
using *clause-lt-clause-if-max-lit-comp*
using *E-max-lit* ‹*is-neg L*› *D-max-lit*
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*)

have *eres D E* $\prec_c$ *E*
using *eres-lt-if*
using *E-max-lit* ‹*is-neg L*› *D-max-lit* **by** *metis*

hence *eres D E* $\neq$ *E*
by *order*

have $L \in\#$ *E*
using *E-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

hence $-$ *L* $\notin\#$ *E*
using *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*
using Γ-*consistent* ‹*trail-false-cls* Γ *E*› **by** *metis*

hence $\forall$ *K* $\in\#$ *eres D E*. *atm-of K* $\prec_t$ *atm-of L*
using *lit-in-eres-lt-greatest-lit-in-grestest-resolvant*[*OF* ‹*eres D E* $\neq$ *E*› *E-max-lit*]
by *metis*

hence $\forall$ *K* $\in\#$ *eres D E*. *K* $\neq$ *L* $\wedge$ *K* $\neq -$ *L*
by *fastforce*

moreover have $\forall$ *L* $\in\#$ *eres D E*. *L* $\in\#$ *D* $\vee$ *L* $\in\#$ *E*
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have *D-almost-false*: *trail-false-cls* Γ {#*K* $\in\#$ *D*. *K* $\neq -$ *L*#}
using *ord-res-7-invars-implies-propagated-clause-almost-false*
using ‹(− *L*, *Some D*) $\in$ *set* Γ› *invar*[*unfolded* ‹*s* = (U_{er}, $\mathcal{F}$, Γ, *Some E*)›]
by *metis*

ultimately have *trail-false-cls* Γ (*eres D E*)
using ‹*trail-false-cls* Γ *E*› **unfolding** *trail-false-cls-def* **by** *fastforce*

hence *trail-false-lit* Γ *K*
using *eres-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff trail-false-cls-def*

by *metis*

have *eres D E* $|\notin|$ *N* $|\cup|$ U_{er}
using *eres-not-in-known-clauses-if-trail-false-cls*
using Γ*-consistent clauses-lt-E-true* ‹*eres D E* $\prec_c$ *E*› ‹*trail-false-cls* Γ (*eres D E*)› **by** *metis*

hence *eres D E* $|\notin|$ $\mathcal{F}$
using $\mathcal{F}$*-subset* **by** *blast*

hence *iefac* $\mathcal{F}$ (*eres D E*) = *eres D E*
by (*simp add*: *iefac-def*)

hence *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ $U_{er}{}'$) = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}))
unfolding ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}› **by** *simp*

have *trail-false-lit* Γ *K*
by (*meson* ‹*trail-false-cls* Γ (*eres D E*)› *linorder-lit.is-maximal-in-mset-iff step-hyps*(*10*)
trail-false-cls-def)

have *mem-set-*Γ′*-iff*: (*Ln* ∈ *set* Γ′) = (¬ *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*) ∧ *Ln* ∈ *set* Γ) **for** *Ln*
unfolding ‹Γ′ = *dropWhile* (λ*Ln*. *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)) Γ›
proof (*rule mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ*-sorted* **.**
next
show *monotone-on* (*set* Γ) (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) (λ*x y*. *y* ≤ *x*)
(λ*Ln*. *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*))
by (*rule monotone-onI*) *auto*
qed

hence *atms-in-*Γ′*-lt-atm-K*: ∀ *x* $|\in|$ *trail-atms* Γ′. *x* $\prec_t$ *atm-of K*
by (*auto simp add*: *fset-trail-atms*)

have $\mathcal{F}$ $|\subseteq|$ *N* $|\cup|$ $U_{er}{}'$
unfolding ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}›
using $\mathcal{F}$*-subset* **by** *blast*

moreover have ∀ *C*′. *Some* (*eres D E*) = *Some C*′ ⟶ *C*′ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ $U_{er}{}'$)
proof −
have *eres D E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ $U_{er}{}'$)
using ‹*iefac* $\mathcal{F}$ (*eres D E*) = *eres D E*›
by (*simp add*: ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}›)

thus *?thesis*

```
    by simp
  qed

  moreover have sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ'
    using Γ-sorted step-hyps(9) by (metis sorted-wrt-dropWhile)

  moreover have Γ'-lower: linorder-trm.is-lower-fset (trail-atms Γ') (atms-of-clss
(N |∪| U_er'))
  proof −
    have linorder-trm.is-lower-fset (trail-atms Γ') (trail-atms Γ)
      unfolding linorder-trm.is-lower-set-iff
    proof (intro conjI ballI impI)
      show fset (trail-atms Γ') ⊆ fset (trail-atms Γ)
          unfolding fset-trail-atms using ‹suffix Γ' Γ› by (metis image-mono
set-mono-suffix)
    next
      obtain Γ'' where Γ = Γ'' @ Γ'
        using ‹suffix Γ' Γ› unfolding suffix-def by metis

      fix l x
      assume l |∈| trail-atms Γ' and x |∈| trail-atms Γ and x ≺t l

      have x |∈| trail-atms Γ'' ∨ x |∈| trail-atms Γ'
       using ‹x |∈| trail-atms Γ› unfolding ‹Γ = Γ'' @ Γ'› fset-trail-atms by auto

      thus x |∈| trail-atms Γ'
      proof (elim disjE)
        assume x |∈| trail-atms Γ''

        hence l ≺t x
          using Γ-sorted ‹l |∈| trail-atms Γ'›
          unfolding ‹Γ = Γ'' @ Γ'› sorted-wrt-append fset-trail-atms by blast

        hence False
          using ‹x ≺t l› by order

        thus x |∈| trail-atms Γ' ..
      next
        assume x |∈| trail-atms Γ'
        thus x |∈| trail-atms Γ' .
      qed
    qed

    thus ?thesis
      using Γ-lower
      unfolding ‹atms-of-clss (N |∪| U_er) = atms-of-clss (N |∪| U_er')›
      by order
  qed
```

moreover have $\forall DE.\ Some\ (eres\ D\ E) = Some\ DE \longrightarrow (\forall A \mid\in\mid trail\text{-}atms\ \Gamma'.$
$\exists L.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ L\ DE \wedge A \preceq_t atm\text{-}of\ L)$
using *atms-in-*Γ'*-lt-atm-K eres-max-lit* **by** *blast*

moreover have $\forall Ln \in set\ \Gamma'.\ snd\ Ln = None \longleftrightarrow is\text{-}neg\ (fst\ Ln)$
using Γ*-deci-iff-neg* ‹*suffix* $\Gamma'\ \Gamma$›
by (*metis* (*no-types*, *opaque-lifting*) *in-set-conv-decomp suffixE suffix-appendD*)

moreover have *trail-true-cls* $\Gamma'\ C$
if $Ln \in set\ \Gamma'$ **and** $snd\ Ln = None$ **and** $C \mid\in\mid iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er}')$ **and** $C \prec_c \{\#fst\ Ln\#\}$
for *Ln C*
proof −
have $Ln \in set\ \Gamma$
using ‹$Ln \in set\ \Gamma'$› ‹*suffix* $\Gamma'\ \Gamma$› *set-mono-suffix* **by** *blast*

have $C = eres\ D\ E \vee C \mid\in\mid iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er})$
using ‹$C \mid\in\mid iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er}')$›
unfolding ‹$iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er})$›
by *simp*

thus *trail-true-cls* $\Gamma'\ C$
proof (*elim disjE*)
assume $C = eres\ D\ E$

hence $K \prec_l fst\ Ln$
using ‹$C \prec_c \{\#fst\ Ln\#\}$› *eres-max-lit*
by (*simp add*: *linorder-lit.is-maximal-in-mset-iff*)

hence $atm\text{-}of\ K \preceq_t atm\text{-}of\ (fst\ Ln)$
by (*cases K*; *cases fst Ln*) *simp-all*

moreover have $atm\text{-}of\ (fst\ Ln) \mid\in\mid trail\text{-}atms\ \Gamma'$
using ‹$Ln \in set\ \Gamma'$› **by** (*simp add*: *fset-trail-atms*)

moreover have $atm\text{-}of\ K \mid\in\mid atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}')$
by (*metis* ‹$iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ \mid`\mid\ (N \mid\cup\mid U_{er})$›
atm-of-in-atms-of-clssI eres-max-lit finsert-iff linorder-lit.is-maximal-in-mset-iff)

ultimately have $atm\text{-}of\ K \mid\in\mid trail\text{-}atms\ \Gamma'$
using Γ'*-lower*
unfolding *linorder-trm.is-lower-set-iff*
by *fastforce*

moreover have $atm\text{-}of\ K \mid\notin\mid trail\text{-}atms\ \Gamma'$
using *atms-in-*Γ'*-lt-atm-K* **by** *blast*

ultimately show *trail-true-cls* Γ' C
by *contradiction*
next
assume C $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})

hence *trail-true-cls* Γ C
using Γ*-deci-ball-lt-true* ‹*Ln* ∈ *set* Γ› ‹*snd Ln* = *None*› ‹$C \prec_c$ {#*fst Ln*#}›
by *metis*

then obtain L_C **where** $L_C \in\# C$ **and** *trail-true-lit* Γ L_C
unfolding *trail-true-cls-def* **by** *auto*

hence $\forall x \in\# C.\ x \prec_l$ *fst Ln*
using ‹$C \prec_c$ {#*fst Ln*#}›
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $L_C \prec_l$ *fst Ln*
using ‹$L_C \in\# C$› **by** *metis*

hence *atm-of* $L_C \preceq_t$ *atm-of* (*fst Ln*)
by (*cases* L_C; *cases fst Ln*) *simp-all*

moreover have *atm-of* (*fst Ln*) $\prec_t$ *atm-of K*
using *atms-in-*Γ'*-lt-atm-K*
by (*simp add*: *fset-trail-atms that*(*1*))

ultimately have *atm-of* $L_C \prec_t$ *atm-of K*
by *order*

have $L_C \in$ *fst* ‘ *set* Γ
using ‹*trail-true-lit* Γ L_C›
unfolding *trail-true-lit-def* **.**

hence $L_C \in$ *fst* ‘ *set* Γ'
using *mem-set-*Γ'*-iff*
using ‹*atm-of* $L_C \prec_t$ *atm-of K*› *linorder-trm.not-le* **by** *auto*

hence *trail-true-lit* Γ' L_C
unfolding *trail-true-lit-def* **.**

thus *trail-true-cls* Γ' C
using ‹$L_C \in\# C$›
unfolding *trail-true-cls-def* **by** *auto*
qed
qed

moreover have $\forall$ *Ln* ∈ *set* Γ'. $\forall C$. *snd Ln* = *Some* $C \longrightarrow C$ $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}')

using Γ-*prop-in* ‹*suffix* Γ' Γ› *set-mono-suffix* ‹$U_{er}' = \mathit{finsert}\ (\mathit{eres}\ D\ E)\ U_{er}$›
by *blast*

moreover have $\forall\, Ln \in \mathit{set}\ \Gamma'.\ \forall\, C.\ \mathit{snd}\ Ln = \mathit{Some}\ C \longrightarrow \mathit{linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset}\ C\ (\mathit{fst}\ Ln)$
using Γ-*prop-greatest* ‹*suffix* Γ' Γ› *set-mono-suffix* **by** *blast*

moreover have $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma' = \Gamma_1\ @\ (L,\ \mathit{Some}\ C)\ \#\ \Gamma_0 \longrightarrow \mathit{trail\text{-}false\text{-}cls}\ \Gamma_0\ \{\#K \in\# C.\ K \neq L\#\}$
using Γ-*prop-almost-false* ‹*suffix* Γ' Γ›
by (*metis* (*no-types*, *lifting*) *append.assoc suffix-def*)

moreover have $\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma' = \Gamma_1\ @\ (L,\ \mathit{Some}\ D)\ \#\ \Gamma_0 \longrightarrow$
$(\forall\, C \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}').\ C \prec_c D \longrightarrow \mathit{trail\text{-}true\text{-}cls}\ \Gamma_0\ C)$
proof (*intro allI impI ballI*)
fix
$\Gamma_1\ \Gamma_0 :: ('f\ \mathit{gliteral} \times 'f\ \mathit{gclause}\ \mathit{option})\ \mathit{list}$ **and**
$L :: 'f\ \mathit{gliteral}$ **and**
$C_0\ C_1 :: 'f\ \mathit{gclause}$
assume
Γ'-*eq*: $\Gamma' = \Gamma_1\ @\ (L,\ \mathit{Some}\ C_1)\ \#\ \Gamma_0$ **and**
C_0-*in*: $C_0 \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$ **and**
$C_0 \prec_c C_1$

have $C_0 = \mathit{eres}\ D\ E \vee C_0 \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using C_0-*in*
unfolding ‹$\mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}') = \mathit{finsert}\ (\mathit{eres}\ D\ E)\ (\mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))$›
by *simp*

thus *trail-true-cls* $\Gamma_0\ C_0$
proof (*elim disjE*)
assume $C_0 = \mathit{eres}\ D\ E$

have $\neg\ \mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ (\mathit{fst}\ (L,\ \mathit{Some}\ C_1))$ **and** $(L,\ \mathit{Some}\ C_1) \in \mathit{set}\ \Gamma$
unfolding *atomize-conj*
using Γ'-*eq* ‹$\Gamma' = \mathit{dropWhile}\ (\lambda Ln.\ \mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ (\mathit{fst}\ Ln))\ \Gamma$›
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*[*OF* Γ-*sorted mono-atms-lt*]
by (*metis in-set-conv-decomp*)

then have $\neg\ \mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ L$
by *simp*

hence $\mathit{atm\text{-}of}\ L \prec_t \mathit{atm\text{-}of}\ K$
by *order*

moreover have *linorder-lit.is-greatest-in-mset* $C_1\ L$
using Γ-*prop-greatest* ‹$(L,\ \mathit{Some}\ C_1) \in \mathit{set}\ \Gamma$› **by** *fastforce*

ultimately have *False*
using ‹$C_0 \prec_c C_1$›
by (*metis Neg-atm-of-iff* ‹$C_0 = eres\ D\ E$› *asympD eres-max-lit linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one linorder-lit.multp-if-maximal-less-that-maximal literal.collapse*(1) *ord-res.asymp-less-cls ord-res.less-lit-simps*(1) *ord-res.less-lit-simps*(4) *step-hyps*(11))

thus *trail-true-cls* Γ_0 C_0 **..**
next
assume C_0 $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
thus *trail-true-cls* Γ_0 C_0
using Γ-*prop-ball-lt-true* Γ'-*eq* ‹$C_0 \prec_c C_1$›
by (*metis* (*no-types*, *opaque-lifting*) ‹*suffix* Γ' Γ› *append-assoc suffixE*)
qed
qed

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}', \mathcal{F}, \Gamma', Some\ (eres\ D\ E))$›
proof (*intro ord-res-7-invars.intros*)
have *clause-true-in-*Γ'*-if*: *trail-true-cls* Γ' x **and**
$\bigwedge K_x$. *linorder-lit.is-maximal-in-mset* x $K_x \Longrightarrow$
$\neg$ ($\exists A$ $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}'). $A \prec_t$ *atm-of* $K_x \wedge A$ $|\notin|$ *trail-atms* Γ')
if *x-lt*: $\bigwedge DE$. $Some\ (eres\ D\ E) = Some\ DE \longrightarrow x \prec_c DE$ **and** *x-in*: x $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}')
for x
proof −
have $x \prec_c eres\ D\ E$
using *x-lt* **by** *metis*

hence $x \neq eres\ D\ E$
by *order*

hence *x-in'*: x $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
using *x-in* ‹*iefac* $\mathcal{F}$ (*eres D E*) = *eres D E*›
by (*simp add*: ‹$U_{er}' = finsert\ (eres\ D\ E)\ U_{er}$›)

have $x \prec_c E$
using ‹$x \prec_c eres\ D\ E$› ‹$eres\ D\ E \prec_c E$› **by** *order*

have *x-true*: *trail-true-cls* Γ x
using *clauses-lt-E-true x-in'* ‹$x \prec_c E$› **by** *metis*

have ($-$ K, *None*) $\in$ *set* Γ
using ‹*trail-false-lit* Γ K›
using ‹*is-pos* K›
using Γ-*deci-iff-neg*
by (*metis is-pos-neg-not-is-pos map-of-SomeD map-of-eq-None-iff not-Some-eq*

prod.collapse
prod.inject trail-false-lit-def)

obtain L_x **where** $L_x \in\# x$ **and** L_x*-true*: *trail-true-lit* Γ L_x
using *x-true* **unfolding** *trail-true-cls-def* **by** *metis*

moreover have $L_x \neq K$
using Γ*-consistent* ‹*trail-false-cls* Γ (*eres D E*)› L_x*-true eres-max-lit*
by (*metis linorder-lit.is-maximal-in-mset-iff not-trail-true-cls-and-trail-false-cls*
trail-true-cls-def)

moreover have $L_x \neq - K$
using *eres-max-lit* ‹$x \prec_c$ *eres D E*› ‹$L_x \neq K$› ‹$L_x \in\# x$›
by (*smt* (*verit*, *del-insts*) *empty-iff linorder-cls.less-not-sym*
linorder-lit.ex-maximal-in-mset linorder-lit.is-maximal-in-mset-iff
linorder-lit.less-trans linorder-lit.multp-if-maximal-less-that-maximal
linorder-lit.neqE
linorder-trm.not-less-iff-gr-or-eq literal.collapse(*1*) *ord-res.less-lit-simps*(*4*)
set-mset-empty step-hyps(*11*) *uminus-literal-def*)

ultimately have *atm-of* $L_x \neq$ *atm-of K*
by (*simp add*: *atm-of-eq-atm-of*)

moreover have $L_x \preceq_l K$
proof (*rule linorder-lit.less-than-maximal-if-multp*$_{HO}$[*OF eres-max-lit* - ‹$L_x \in\# x$›])
show *multp*$_{HO}$ ($\prec_l$) *x* (*eres D E*)
using ‹$x \prec_c$ *eres D E*›
by (*simp add*: *multp-imp-multp*$_{HO}$)
qed

ultimately have *atm-of* $L_x \prec_t$ *atm-of K*
by (*cases* L_x; *cases K*) *simp-all*

hence *trail-true-lit* Γ' L_x
unfolding ‹$\Gamma' =$ *dropWhile* (λ*Ln. atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)) Γ›
using L_x*-true*
unfolding *trail-true-lit-def*
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*[*OF* Γ*-sorted*
mono-atms-lt]
by (*metis* (*no-types*, *lifting*) *image-iff linorder-trm.not-le*)

thus *trail-true-cls* Γ' *x*
using ‹$L_x \in\# x$› **unfolding** *trail-true-cls-def* **by** *metis*

fix K_x
assume *x-max-lit*: *ord-res.is-maximal-lit* K_x *x*

hence $K_x \preceq_l K$
using ‹$x \prec_c$ *eres D E*› *eres-max-lit*
using *linorder-lit.multp-if-maximal-less-that-maximal* **by** *fastforce*

hence *atm-of* $K_x \preceq_t$ *atm-of* K
by (*cases* K_x; *cases* K) *simp-all*

have A |∈| *trail-atms* Γ'
if *A-lt*: $A \prec_t$ *atm-of* K_x **and** *A-in*: A |∈| *atms-of-clss* (N |∪| U_{er}) **for** A
unfolding ‹$\Gamma' =$ *dropWhile* (λ*Ln*. *atm-of* $K \preceq_t$ *atm-of* (*fst Ln*)) Γ›
proof (*rule in-trail-atms-dropWhileI*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**
next
show *monotone-on* (*set* Γ) (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) (λ*x y*. $y \leq x$) (λ*x*. *atm-of* $K \preceq_t$ *atm-of* (*fst x*))
using *mono-atms-lt* **.**
next
show ¬ *atm-of* $K \preceq_t A$
using *A-lt* ‹*atm-of* $K_x \preceq_t$ *atm-of* K› **by** *order*
next
have ¬ (∃ A|∈|*atms-of-clss* (N |∪| U_{er}). $A \prec_t$ *atm-of* $K_x \wedge A$ |∉| *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-E* ‹x |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})› ‹$x \prec_c$ *E*› *x-max-lit*
by *metis*

thus A |∈| *trail-atms* Γ
using *A-in A-lt* **by** *metis*
qed

thus ¬ (∃ A |∈| *atms-of-clss* (N |∪| U_{er}'). $A \prec_t$ *atm-of* $K_x \wedge A$ |∉| *trail-atms* Γ')
using ‹*atms-of-clss* (N |∪| U_{er}) = *atms-of-clss* (N |∪| U_{er}')› **by** *metis*
qed

thus
∀ C|∈|*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). (∀ *DE*. *Some* (*eres D E*) = *Some DE* ⟶ $C \prec_c$ *DE*) ⟶
trail-true-cls Γ' C
∀ C|∈|*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}'). (∀ *DE*. *Some* (*eres D E*) = *Some DE* ⟶ $C \prec_c$ *DE*) ⟶
(∀ K. *ord-res.is-maximal-lit* K C ⟶
¬ (∃ A|∈|*atms-of-clss* (N |∪| U_{er}'). $A \prec_t$ *atm-of* $K \wedge A$ |∉| *trail-atms* Γ'))
unfolding *atomize-conj* **by** *metis*

have *trail-true-cls* Γ' {#K ∈# C. $K \neq L_C$#} ∧ *is-pos* L_C
if *Some* (*eres D E*) = *Some DE* **and**
C-in: C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}') **and**

```
      C-lt: C ≺c DE and
      C-max-lit: linorder-lit.is-maximal-in-mset C L_C and
      L_C-undef: ¬ trail-defined-lit Γ' L_C
    for DE C L_C
  proof −
    have DE = eres D E
      using that by simp
    hence C ≺c eres D E
      using C-lt by order
    hence C ≠ eres D E
      by order
    hence C |∈| iefac F |‘| (N |∪| U_er)
      using C-in ‹iefac F |‘| (N |∪| U_er') = finsert (eres D E) (iefac F |‘| (N
|∪| U_er))›
      by simp
    moreover have C ≺c E
      using ‹C ≺c eres D E› ‹eres D E ≺c E› by order

    have L_C ⪯l K
      using ‹C ≺c eres D E› C-max-lit eres-max-lit
      by (meson linorder-cls.dual-order.asym linorder-lit.leI
          linorder-lit.multp-if-maximal-less-that-maximal)
    hence L_C ≺l K ∨ L_C = K
      by simp

    thus ?thesis
    proof (elim disjE)
      assume L_C ≺l K
      hence atm-of L_C ≺t atm-of K
        using ‹is-pos K›
        by (cases L_C; cases K) simp-all

      have trail-defined-lit Γ' L_C
        unfolding ‹Γ' = dropWhile (λLn. atm-of K ⪯t atm-of (fst Ln)) Γ›
      proof (intro trail-defined-lit-dropWhileI ballI)
        show sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ
          using Γ-sorted .
      next
        show monotone-on (set Γ) (λx y. atm-of (fst y) ≺t atm-of (fst x)) (λx y.
y ≤ x)
       (λx. atm-of K ⪯t atm-of (fst x))
          using mono-atms-lt .
      next
        have ¬ atm-of K ⪯t atm-of L_C
          using ‹atm-of L_C ≺t atm-of K› by order
        thus ¬ atm-of K ⪯t atm-of L_C ∧ ¬ atm-of K ⪯t atm-of (− L_C)
          by simp
      next
        have trail-defined-lit Γ K
```

```
        using ‹trail-false-lit Γ K›
        using trail-defined-lit-iff-true-or-false by blast
      moreover have atm-of K |∈| atms-of-clss (N |∪| U_er)
        by (metis ‹atms-of-clss (N |∪| U_er) = atms-of-clss (N |∪| U_er')›
            ‹iefac F |‘| (N |∪| U_er') = finsert (eres D E) (iefac F |‘| (N |∪| U_er))›
          atm-of-in-atms-of-clssI eres-max-lit finsertI1 linorder-lit.is-maximal-in-mset-iff)
      moreover have atm-of L_C |∈| atms-of-clss (N |∪| U_er)
        using ‹C |∈| iefac F |‘| (N |∪| U_er)› C-max-lit atm-of-in-atms-of-clssI
        unfolding linorder-lit.is-maximal-in-mset-iff
        by metis
      ultimately show trail-defined-lit Γ L_C
        using ‹atm-of L_C ≺_t atm-of K› Γ-lower
        unfolding trail-defined-lit-iff-trail-defined-atm
        by (meson linorder-trm.is-lower-set-iff)
    qed

    hence False
      using L_C-undef by contradiction

    thus ?thesis ..
  next
    have trail-true-cls Γ' C
      using clause-true-in-Γ'-if C-in ‹C ≺_c eres D E› by blast
    hence trail-true-cls Γ' {#K ∈# C. K ≠ L_C#}
      using L_C-undef
    by (smt (verit, best) mem-Collect-eq set-mset-filter trail-defined-lit-iff-true-or-false
        trail-true-cls-def)
    moreover assume L_C = K
    ultimately show trail-true-cls Γ' {#K ∈# C. K ≠ L_C#} ∧ is-pos L_C
      using ‹is-pos K› by metis
  qed
qed

thus ∀ Da. Some (eres D E) = Some Da ⟶
    (∀ C|∈|iefac F |‘| (N |∪| U_er'). C ≺_c Da ⟶ (∀ L_C. ord-res.is-maximal-lit
L_C C ⟶
      ¬ trail-defined-lit Γ' L_C ⟶ trail-true-cls Γ' {#K ∈# C. K ≠ L_C#} ∧
is-pos L_C))
  by metis
qed simp-all
next
case step-hyps: (resolution-neg Γ E L D U_er' U_er Γ' K C F)

note E-max-lit = ‹ord-res.is-maximal-lit L E›
note eres-max-lit = ‹ord-res.is-maximal-lit K (eres D E)›

have
  F-subset: F |⊆| N |∪| U_er and
```

E-in: $E \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
$\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall L_C.$ *linorder-lit.is-maximal-in-mset* $C\ L_C \longrightarrow$
$\neg$ *trail-defined-lit* $\Gamma\ L_C \longrightarrow$ (*trail-true-cls* $\Gamma\ \{\#K \in\# C.\ K \neq L_C\#\} \wedge$ *is-pos* L_C)) **and**
clauses-lt-E-true: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$ *trail-true-cls* Γ C **and**
no-undef-atm-lt-max-lit-if-lt-E: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$
$\neg\ (\exists A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \prec_t$ *atm-of* $K \wedge A \mid\notin\mid$ *trail-atms* $\Gamma))$ **and**
Γ-*sorted*: *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$) **and**
trail-atms-le0: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ \exists L.$ *ord-res.is-maximal-lit* $L\ E \wedge (A \preceq_t$ *atm-of* $L)$ **and**
Γ-*deci-iff-neg*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longleftrightarrow$ *is-neg* (*fst* Ln) **and**
Γ-*deci-ball-lt-true*: $\forall Ln \in$ *set* $\Gamma.$ *snd* $Ln =$ *None* $\longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c \{\#$*fst* $Ln\#\} \longrightarrow$ *trail-true-cls* $\Gamma\ C)$ **and**
Γ-*prop-in*: $\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
Γ-*prop-greatest*:
$\forall Ln \in$ *set* $\Gamma.\ \forall C.$ *snd* $Ln =$ *Some* $C \longrightarrow$ *linorder-lit.is-greatest-in-mset* C (*fst* Ln) **and**
Γ-*prop-almost-false*:
$\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* C) # $\Gamma_0 \longrightarrow$ *trail-false-cls* $\Gamma_0\ \{\#K \in\# C.\ K \neq L\#\}$ **and**
Γ-*prop-ball-lt-true*: $(\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* D) # $\Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* $\Gamma_0\ C))$ **and**
Γ-*prop-ball-lt-true*: $\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* D) # $\Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c D \longrightarrow$ *trail-true-cls* $\Gamma_0\ C)$
using *invar* **by** (*simp-all add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $E)$› *ord-res-7-invars-def*)

have *clauses-lt-E-almost-defined*: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ C \prec_c E \longrightarrow$
$(\forall K.$ *linorder-lit.is-maximal-in-mset* $C\ K \longrightarrow$ *trail-defined-cls* $\Gamma\ \{\#L \in\# C.\ L \neq K\#\})$
using *invar*[*unfolded* ‹$s = (U_{er}, \mathcal{F}, \Gamma,$ *Some* $E)$›] *clause-almost-defined-if-lt-next-clause*
by *simp*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* Γ-*sorted* **by** *metis*

have *trail-atms-le*: $\forall A \mid\in\mid$ *trail-atms* $\Gamma.\ A \preceq_t$ *atm-of* L
using *trail-atms-le0 E-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have $(-\ L,$ *Some* $D) \in$ *set* Γ

using ‹*map-of* Γ (− *L*) = *Some* (*Some D*)› **by** (*metis map-of-SomeD*)

hence *D-in*: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using Γ-*prop-in* **by** *simp*

have *D-max-lit*: *linorder-lit.is-greatest-in-mset D* (− *L*)
using Γ-*prop-greatest* ‹(− *L*, *Some D*) ∈ *set* Γ› **by** *fastforce*

have *D* ≺$_c$ *E*
using *clause-lt-clause-if-max-lit-comp*
using *E-max-lit* ‹*is-neg L*› *D-max-lit*
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*)

have *eres D E* ≺$_c$ *E*
using *eres-lt-if*
using *E-max-lit* ‹*is-neg L*› *D-max-lit* **by** *metis*

hence *eres D E* ≠ *E*
by *order*

have (− *K*, *Some C*) ∈ *set* Γ
using ‹*map-of* Γ (− *K*) = *Some* (*Some C*)› **by** (*metis map-of-SomeD*)

hence *C-in*: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using Γ-*prop-in* **by** *simp*

have *C-max-lit*: *linorder-lit.is-greatest-in-mset C* (− *K*)
using Γ-*prop-greatest* ‹(− *K*, *Some C*) ∈ *set* Γ› **by** *fastforce*

hence *C* ≺$_c$ *eres D E*
using ‹*ord-res.is-maximal-lit K* (*eres D E*)› ‹*is-neg L*›
by (*metis Neg-atm-of-iff Pos-atm-of-iff linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*
linorder-lit.multp-if-maximal-less-that-maximal linorder-lit.not-less-iff-gr-or-eq
linorder-trm.less-irrefl ord-res.less-lit-simps(*4*) *step-hyps*(*11*) *uminus-Neg*)

have *suffix* Γ′ Γ
using *step-hyps*(*9*) *suffix-dropWhile* **by** *metis*

hence *atms-of-cls* (*eres D E*) |⊆| *atms-of-cls D* |∪| *atms-of-cls E*
using *lit-in-one-of-resolvents-if-in-eres*
unfolding *atms-of-cls-def* **by** *fastforce*

moreover have *atms-of-cls D* |⊆| *atms-of-clss* (*N* |∪| U_{er})
using *D-in*
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb funion-upper1*)

moreover have *atms-of-cls E* |⊆| *atms-of-clss* (*N* |∪| U_{er})
using *E-in*
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb funion-upper1*)

ultimately have *atms-of-clss* $(N \mid\cup\mid U_{er}) = atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}')$
unfolding $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er}\rangle$ *atms-of-clss-def* **by** *auto*

have $L \in\# E$
using *E-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

hence $-L \notin\# E$
using *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*
using Γ-*consistent* $\langle trail\text{-}false\text{-}cls\ \Gamma\ E\rangle$ **by** *metis*

hence $\forall K \in\# eres\ D\ E.\ atm\text{-}of\ K \prec_t atm\text{-}of\ L$
using *lit-in-eres-lt-greatest-lit-in-grestest-resolvant*[*OF* $\langle eres\ D\ E \neq E\rangle$ *E-max-lit*]
by *metis*

hence $\forall K \in\# eres\ D\ E.\ K \neq L \wedge K \neq -L$
by *fastforce*

moreover have $\forall L \in\# eres\ D\ E.\ L \in\# D \vee L \in\# E$
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have *D-almost-false*: *trail-false-cls* Γ $\{\#K \in\# D.\ K \neq -L\#\}$
using *ord-res-7-invars-implies-propagated-clause-almost-false*
using $\langle(-L, Some\ D) \in set\ \Gamma\rangle$ *invar*[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, Some\ E)\rangle$]
by *metis*

ultimately have *trail-false-cls* Γ $(eres\ D\ E)$
using $\langle trail\text{-}false\text{-}cls\ \Gamma\ E\rangle$ **unfolding** *trail-false-cls-def* **by** *fastforce*

have *eres D E* $\mid\notin\mid$ *N* $\mid\cup\mid$ U_{er}
using *eres-not-in-known-clauses-if-trail-false-cls*
using Γ-*consistent clauses-lt-E-true* $\langle eres\ D\ E \prec_c E\rangle$ $\langle trail\text{-}false\text{-}cls\ \Gamma\ (eres\ D\ E)\rangle$ **by** *metis*

hence *eres D E* $\mid\notin\mid$ $\mathcal{F}$
using $\mathcal{F}$-*subset* **by** *blast*

hence *iefac* $\mathcal{F}$ $(eres\ D\ E) = eres\ D\ E$
by (*simp add*: *iefac-def*)

hence *iefac* $\mathcal{F}$ $\mid`\mid$ $(N \mid\cup\mid U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))$
by (*simp add*: $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er}\rangle$)

have *mem-set-*Γ'*-iff*: $(Ln \in set\ \Gamma') = (\neg\ atm\text{-}of\ K \preceq_t atm\text{-}of\ (fst\ Ln) \wedge Ln \in set\ \Gamma)$ **for** *Ln*
unfolding $\langle \Gamma' = dropWhile\ (\lambda Ln.\ atm\text{-}of\ K \preceq_t atm\text{-}of\ (fst\ Ln))\ \Gamma\rangle$
proof (*rule mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*)
show *sorted-wrt* $(\lambda x\ y.\ atm\text{-}of\ (fst\ y) \prec_t atm\text{-}of\ (fst\ x))\ \Gamma$
using Γ-*sorted* **.**

next
show *monotone-on* (*set* Γ) (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) (λ*x y*. *y* ≤ *x*)
(λ*Ln*. *atm-of K* ⪯$_t$ *atm-of* (*fst Ln*))
by (*rule monotone-onI*) *auto*
qed

have *atms-in-*Γ′*-lt-atm-K*: ∀ *A* |∈| *trail-atms* Γ′. *A* ≺$_t$ *atm-of K*
proof −
have ∃ *L*. *ord-res.is-maximal-lit L C* ∧ *x* ≺$_t$ *atm-of L* **if** *x* |∈| *trail-atms* Γ′ **for** *x*
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit* (− *K*) *C*
using *C-max-lit* **by** *blast*
next
show *x* ≺$_t$ *atm-of* (− *K*)
using ‹*x* |∈| *trail-atms* Γ′› *mem-set-*Γ′*-iff* **unfolding** *fset-trail-atms* **by** *fastforce*
qed

hence ∀ *A*|∈|*trail-atms* Γ′. *A* ≺$_t$ *atm-of* (− *K*)
using *C-max-lit*
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)

thus *?thesis*
by (*metis atm-of-uminus*)
qed

have *F* |⊆| *N* |∪| *U*$_{er}$′
unfolding ‹*U*$_{er}$′ = *finsert* (*eres D E*) *U*$_{er}$›
using *F-subset* **by** *blast*

moreover have ∀ *C*′. *Some C* = *Some C*′ ⟶ *C*′ |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$′)
using *C-in* **by** (*simp add*: ‹*U*$_{er}$′ = *finsert* (*eres D E*) *U*$_{er}$›)

moreover have
⋀*K*$_x$. *linorder-lit.is-maximal-in-mset x K*$_x$ ⟹ *trail-defined-cls* Γ′ {#*L* ∈# *x*. *L* ≠ *K*$_x$#} **and**
*clause-true-in-*Γ′*-if*: *trail-true-cls* Γ′ *x*
if *x-lt*: ⋀*y*. *Some C* = *Some y* ⟹ *x* ≺$_c$ *y* **and** *x-in*: *x* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$′) **for** *x*
proof −
have *x* ≺$_c$ *C*
using *x-lt* **by** *metis*

hence *x* ≺$_c$ *eres D E*
using ‹*C* ≺$_c$ *eres D E*› **by** *order*

hence $x \neq$ *eres D E*
by *order*

have $x \prec_c E$
using ‹$x \prec_c$ *eres D E*› ‹*eres D E* $\prec_c E$› **by** *order*

moreover have *x-in'*: $x \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *x-in* ‹$x \neq$ *eres D E*›
unfolding ‹*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}') =$ *finsert* (*eres D E*) (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)›
by *simp*

ultimately have *x-true*: *trail-true-cls* Γ x
using *clauses-lt-E-true* **by** *metis*

then obtain L_x **where** $L_x \in\# x$ **and** *trail-true-lit* Γ L_x
unfolding *trail-true-cls-def* **by** *metis*

have $L_x \preceq_l - K$
using *C-max-lit* ‹$x \prec_c C$› ‹$L_x \in\# x$›
by (*smt* (*verit*, *ccfv-threshold*) *asympD empty-not-add-mset ord-res.transp-less-lit insert-DiffM*
linorder-lit.is-greatest-in-set-iff linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.is-maximal-in-mset-iff linorder-lit.is-maximal-in-set-eq-is-greatest-in-set
linorder-lit.is-maximal-in-set-iff linorder-lit.leI ord-res.asymp-less-cls
ord-res.multp-if-all-left-smaller transpE)

have *mono-atms-lt*: *monotone-on* (*set* Γ) ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) ($\lambda x\ y.\ y \leq x$)
($\lambda x.$ *atm-of K* $\preceq_t$ *atm-of* (*fst x*))
proof (*intro monotone-onI*, *unfold le-bool-def*, *intro impI*)
fix *x y*
assume *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*) **and** *atm-of K* $\preceq_t$ *atm-of* (*fst y*)
thus *atm-of K* $\preceq_t$ *atm-of* (*fst x*)
by *order*
qed

obtain Γ_1 Γ_0 **where** Γ*-eq*: $\Gamma = \Gamma_1$ @ (− *K*, *Some C*) # Γ_0
using ‹(− *K*, *Some C*) ∈ *set* Γ› **by** (*metis split-list*)

hence *trail-true-cls* Γ_0 x
using Γ*-prop-ball-lt-true x-in'* ‹$x \prec_c C$› **by** *metis*

then obtain L_x **where** $L_x \in\# x$ **and** L_x*-true*: *trail-true-lit* Γ_0 L_x
unfolding *trail-true-cls-def* **by** *auto*

moreover have $\Gamma' = \Gamma_0$
proof −
have $\Gamma' =$ *dropWhile* ($\lambda Ln.$ *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)) ((Γ_1 @ [(− *K*,

```
Some C)]) @ Γ0)
      unfolding ‹Γ' = dropWhile (λLn. atm-of K ≼t atm-of (fst Ln)) Γ› Γ-eq
by simp

    also have ... = dropWhile (λLn. atm-of K ≼t atm-of (fst Ln)) Γ0
    proof (rule dropWhile-append2)
      fix Ln :: 'f gterm literal × 'f gclause option
      assume Ln ∈ set (Γ1 @ [(− K, Some C)])

      moreover have ∀x∈set Γ1. atm-of K ≺t atm-of (fst x)
        using Γ-sorted by (simp add: Γ-eq sorted-wrt-append)

      ultimately show atm-of K ≼t atm-of (fst Ln)
        using ‹is-neg K› by auto
    qed

    also have ... = Γ0
    proof (cases Γ0)
      case Nil
      thus ?thesis
        by (simp add: dropWhile-eq-self-iff)
    next
      case (Cons Ln Γ0')

      hence atm-of (fst Ln) ≺t atm-of K
        using Γ-sorted by (simp add: Γ-eq sorted-wrt-append)

      hence ¬ atm-of K ≼t atm-of (fst Ln)
        by order

      hence ¬ atm-of K ≼t atm-of (fst (hd Γ0))
        by (simp add: ‹Γ0 = Ln # Γ0'›)

      thus ?thesis
        by (simp add: dropWhile-eq-self-iff)
    qed

    finally show ?thesis .
  qed

  ultimately have trail-true-lit Γ' Lx
    by argo

  thus trail-true-cls Γ' x
    using ‹Lx ∈# x› unfolding trail-true-cls-def by metis

  fix Kx assume x-max-lit: ord-res.is-maximal-lit Kx x
  show trail-defined-cls Γ' {#L ∈# x. L ≠ Kx#}
    unfolding ‹Γ' = dropWhile (λLn. atm-of K ≼t atm-of (fst Ln)) Γ›
```

proof (*intro trail-defined-cls-dropWhileI ballI*)
show *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ*-sorted* **.**
next
show *monotone-on* (*set* Γ) ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) ($\lambda x\ y.\ y \leq x$)
($\lambda x.$ *atm-of K* $\preceq_t$ *atm-of* (*fst x*))
using *mono-atms-lt* **.**
next
fix L_x
assume $L_x \in\# \{\#L \in\# x.\ L \neq K_x\#\}$

hence $L_x \in\# x$ **and** $L_x \neq K_x$
by *simp-all*

hence $L_x \prec_l K_x$
using *x-max-lit* ‹$L_x \in\# x$› ‹$L_x \neq K_x$› **unfolding** *linorder-lit.is-maximal-in-mset-iff*
by *fastforce*

moreover have $K_x \preceq_l K$
using ‹$x \prec_c$ *eres D E*›
using *linorder-lit.multp-if-maximal-less-that-maximal*[*OF eres-max-lit x-max-lit*]
by *fastforce*

ultimately have *atm-of* $L_x \prec_t$ *atm-of K*
using ‹*is-neg K*›
by (*metis C-max-lit Pos-atm-of-iff* ‹$x \prec_c C$› *linorder-cls.dual-order.asym linorder-lit.dual-order.strict-trans1 linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.less-le-not-le linorder-lit.multp-if-maximal-less-that-maximal linorder-trm.linorder-cases literal.collapse*(*2*) *ord-res.less-lit-simps*(*3*) *ord-res.less-lit-simps*(*4*) *uminus-Neg x-max-lit*)

hence ¬ *atm-of K* $\preceq_t$ *atm-of* L_x
by *order*

thus ¬ *atm-of K* $\preceq_t$ *atm-of* $L_x \wedge$ ¬ *atm-of K* $\preceq_t$ *atm-of* $(-L_x)$
unfolding *atm-of-uminus conj-absorb* **.**
next
show *trail-defined-cls* Γ $\{\#L \in\# x.\ L \neq K_x\#\}$
using *clauses-lt-E-almost-defined x-in′* ‹$x \prec_c E$› *x-max-lit* **by** *metis*
qed
qed

moreover have *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ'
using Γ*-sorted step-hyps*(*9*) **by** (*metis sorted-wrt-dropWhile*)

moreover have Γ'*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss*

```
(N |∪| U_er'))
  proof −
    have linorder-trm.is-lower-fset (trail-atms Γ') (trail-atms Γ)
      unfolding linorder-trm.is-lower-set-iff
    proof (intro conjI ballI impI)
      show fset (trail-atms Γ') ⊆ fset (trail-atms Γ)
          unfolding fset-trail-atms using ‹suffix Γ' Γ› by (metis image-mono
set-mono-suffix)
    next
      obtain Γ'' where Γ = Γ'' @ Γ'
        using ‹suffix Γ' Γ› unfolding suffix-def by metis

      fix l x
      assume l |∈| trail-atms Γ' and x |∈| trail-atms Γ and x ≺t l

      have x |∈| trail-atms Γ'' ∨ x |∈| trail-atms Γ'
       using ‹x |∈| trail-atms Γ› unfolding ‹Γ = Γ'' @ Γ'› fset-trail-atms by auto

      thus x |∈| trail-atms Γ'
      proof (elim disjE)
        assume x |∈| trail-atms Γ''

        hence l ≺t x
          using Γ-sorted ‹l |∈| trail-atms Γ'›
          unfolding ‹Γ = Γ'' @ Γ'› sorted-wrt-append fset-trail-atms by blast

        hence False
          using ‹x ≺t l› by order

        thus x |∈| trail-atms Γ' ..
      next
        assume x |∈| trail-atms Γ'
        thus x |∈| trail-atms Γ' .
      qed
    qed

    thus ?thesis
      using Γ-lower
      unfolding ‹atms-of-clss (N |∪| U_er) = atms-of-clss (N |∪| U_er')›
      by order
  qed

  moreover have ∀ DE. Some C = Some DE ⟶ (∀ A |∈| trail-atms Γ'.
    ∃ L. ord-res.is-maximal-lit L DE ∧ A ⪯t atm-of L)
    using atms-in-Γ'-lt-atm-K C-max-lit by fastforce

  moreover have ∀ Ln ∈ set Γ'. snd Ln = None ⟷ is-neg (fst Ln)
    using Γ-deci-iff-neg ‹suffix Γ' Γ›
    by (metis (no-types, opaque-lifting) in-set-conv-decomp suffixE suffix-appendD)
```

```
  moreover have trail-true-cls Γ' C
    if Ln ∈ set Γ' and snd Ln = None and C |∈| iefac F |‘| (N |∪| U_er') and C
≺_c {#fst Ln#}
    for Ln C
  proof −
    have Ln ∈ set Γ
      using ‹Ln ∈ set Γ'› ‹suffix Γ' Γ› set-mono-suffix by blast

    have C = eres D E ∨ C |∈| iefac F |‘| (N |∪| U_er)
      using ‹C |∈| iefac F |‘| (N |∪| U_er')›
      unfolding ‹iefac F |‘| (N |∪| U_er') = finsert (eres D E) (iefac F |‘| (N |∪|
U_er))›
      by simp

    thus trail-true-cls Γ' C
    proof (elim disjE)
      assume C = eres D E

      hence K ≺_l fst Ln
        using ‹C ≺_c {#fst Ln#}› eres-max-lit
        by (simp add: linorder-lit.is-maximal-in-mset-iff)

      hence atm-of K ⪯_t atm-of (fst Ln)
        by (cases K; cases fst Ln) simp-all

      moreover have atm-of (fst Ln) |∈| trail-atms Γ'
        using ‹Ln ∈ set Γ'› by (simp add: fset-trail-atms)

      moreover have atm-of K |∈| atms-of-clss (N |∪| U_er')
        by (metis ‹iefac F |‘| (N |∪| U_er') = finsert (eres D E) (iefac F |‘| (N |∪|
U_er))›
          atm-of-in-atms-of-clssI eres-max-lit finsert-iff linorder-lit.is-maximal-in-mset-iff)

      ultimately have atm-of K |∈| trail-atms Γ'
        using Γ'-lower
        unfolding linorder-trm.is-lower-set-iff
        by fastforce

      moreover have atm-of K |∉| trail-atms Γ'
        using atms-in-Γ'-lt-atm-K by blast

      ultimately show trail-true-cls Γ' C
        by contradiction
    next
      assume C |∈| iefac F |‘| (N |∪| U_er)

      hence trail-true-cls Γ C
       using Γ-deci-ball-lt-true ‹Ln ∈ set Γ› ‹snd Ln = None› ‹C ≺_c {#fst Ln#}›
```

by *metis*

then obtain L_C **where** $L_C \in\# C$ **and** *trail-true-lit* Γ L_C
unfolding *trail-true-cls-def* **by** *auto*

hence $\forall x \in\# C.\ x \prec_l$ *fst Ln*
using ‹$C \prec_c \{\#$*fst Ln*$\#\}$›
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $L_C \prec_l$ *fst Ln*
using ‹$L_C \in\# C$› **by** *metis*

hence *atm-of* $L_C \preceq_t$ *atm-of* (*fst Ln*)
by (*cases* L_C; *cases fst Ln*) *simp-all*

moreover have *atm-of* (*fst Ln*) $\prec_t$ *atm-of K*
using *atms-in-*Γ'*-lt-atm-K*
by (*simp add*: *fset-trail-atms that*(*1*))

ultimately have *atm-of* $L_C \prec_t$ *atm-of K*
by *order*

have $L_C \in$ *fst* ‘ *set* Γ
using ‹*trail-true-lit* Γ L_C›
unfolding *trail-true-lit-def* **.**

hence $L_C \in$ *fst* ‘ *set* Γ'
using *mem-set-*Γ'*-iff*
using ‹*atm-of* $L_C \prec_t$ *atm-of K*› *linorder-trm.not-le* **by** *auto*

hence *trail-true-lit* Γ' L_C
unfolding *trail-true-lit-def* **.**

thus *trail-true-cls* Γ' *C*
using ‹$L_C \in\# C$›
unfolding *trail-true-cls-def* **by** *auto*
qed
qed

moreover have $\forall$ *Ln* $\in$ *set* Γ'. $\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ $U_{er}{}'$)
using Γ*-prop-in* ‹*suffix* Γ' Γ› *set-mono-suffix* ‹$U_{er}{}'$ = *finsert* (*eres D E*) U_{er}›
by *blast*

moreover have $\forall$ *Ln* $\in$ *set* Γ'. $\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *linorder-lit.is-greatest-in-mset C* (*fst Ln*)
using Γ*-prop-greatest* ‹*suffix* Γ' Γ› *set-mono-suffix* **by** *blast*

moreover have $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma' = \Gamma_1 \mathbin{@} (L,\ \mathit{Some}\ C) \#\ \Gamma_0 \longrightarrow \mathit{trail\text{-}false\text{-}cls}\ \Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$
using Γ-*prop-almost-false* ‹*suffix* $\Gamma'\ \Gamma$›
by (*metis* (*no-types*, *lifting*) *append.assoc suffix-def*)

moreover have $\forall \Gamma_1\ L\ D\ \Gamma_0.\ \Gamma' = \Gamma_1 \mathbin{@} (L,\ \mathit{Some}\ D) \#\ \Gamma_0 \longrightarrow$
$(\forall C \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}').\ C \prec_c D \longrightarrow \mathit{trail\text{-}true\text{-}cls}\ \Gamma_0\ C)$
proof (*intro allI impI ballI*)
fix
$\Gamma_1\ \Gamma_0$ :: (*'f gliteral* × *'f gclause option*) *list* **and**
L :: *'f gliteral* **and**
$C_0\ C_1$ :: *'f gclause*
assume
Γ'-*eq*: $\Gamma' = \Gamma_1 \mathbin{@} (L,\ \mathit{Some}\ C_1) \#\ \Gamma_0$ **and**
C_0-*in*: $C_0 \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$ **and**
$C_0 \prec_c C_1$

have $C_0 = \mathit{eres}\ D\ E \vee C_0 \mid\in\mid \mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using C_0-*in*
unfolding ‹$\mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}') = \mathit{finsert}\ (\mathit{eres}\ D\ E)\ (\mathit{iefac}\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))$›
by *simp*

thus *trail-true-cls* $\Gamma_0\ C_0$
proof (*elim disjE*)
assume $C_0 = \mathit{eres}\ D\ E$

have ¬ *atm-of* $K \preceq_t$ *atm-of* (*fst* (L, *Some* C_1)) **and** (L, *Some* C_1) ∈ *set* Γ
unfolding *atomize-conj*
using Γ'-*eq* ‹$\Gamma' = \mathit{dropWhile}\ (\lambda Ln.\ \mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ (\mathit{fst}\ Ln))\ \Gamma$›
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*[*OF* Γ-*sorted mono-atms-lt*]
by (*metis in-set-conv-decomp*)

then have ¬ *atm-of* $K \preceq_t$ *atm-of* L
by *simp*

hence *atm-of* $L \prec_t$ *atm-of* K
by *order*

moreover have *linorder-lit.is-greatest-in-mset* C_1 L
using Γ-*prop-greatest* ‹(L, *Some* C_1) ∈ *set* Γ› **by** *fastforce*

ultimately have *False*
using ‹$C_0 \prec_c C_1$›
by (*smt* (*verit*) ‹$C_0 = \mathit{eres}\ D\ E$› *eres-max-lit linorder-cls.dual-order.asym*
linorder-lit.dual-order.strict-trans
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.multp-if-maximal-less-that-maximal linorder-lit.not-less-iff-gr-or-eq

literal.collapse(*1*) *literal.collapse*(*2*) *ord-res.less-lit-simps*(*1*)
ord-res.less-lit-simps(*4*))

thus *trail-true-cls* Γ_0 C_0 **..**
next
assume C_0 $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
thus *trail-true-cls* Γ_0 C_0
using Γ-*prop-ball-lt-true* Γ'-*eq* ‹C_0 $\prec_c$ C_1›
by (*metis* (*no-types*, *opaque-lifting*) ‹*suffix* Γ' Γ› *append-assoc suffixE*)
qed
qed

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}', \mathcal{F}, \Gamma', \mathit{Some}\ C)$›
proof (*intro ord-res-7-invars.intros*)
have *clause-true-in-*Γ'*-if*: *trail-true-cls* Γ' x **and**
$\bigwedge K_x$. *linorder-lit.is-maximal-in-mset* x $K_x \Longrightarrow$
$\neg$ ($\exists A$ $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}'). $A \prec_t$ *atm-of* $K_x \wedge A$ $|\notin|$ *trail-atms* Γ')
if *x-lt*: $\bigwedge DE$. *Some* C = *Some* $DE \longrightarrow x \prec_c DE$ **and** *x-in*: x $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}')
for x
proof −
have $x \prec_c C$
using *x-lt* **by** *metis*

hence $x \prec_c$ *eres* D E
using ‹$C \prec_c$ *eres* D E› **by** *order*

hence $x \neq$ *eres* D E
by *order*

have $x \prec_c E$
using ‹$x \prec_c$ *eres* D E› ‹*eres* D $E \prec_c E$› **by** *order*

moreover have *x-in'*: x $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er})
using *x-in* ‹$x \neq$ *eres* D E›
unfolding ‹*iefac* $\mathcal{F}$ $|`|$ ($N$ $|\cup|$ $U_{er}'$) = *finsert* (*eres* $D$ $E$) (*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}))›
by *simp*

ultimately have *x-true*: *trail-true-cls* Γ x
using *clauses-lt-E-true* **by** *metis*

then obtain L_x **where** $L_x \in\#$ x **and** *trail-true-lit* Γ L_x
unfolding *trail-true-cls-def* **by** *metis*

have $L_x \preceq_l -K$
using *C-max-lit* ‹$x \prec_c C$› ‹$L_x \in\#$ x›
by (*smt* (*verit*, *ccfv-threshold*) *asympD empty-not-add-mset ord-res.transp-less-lit*

insert-DiffM
linorder-lit.is-greatest-in-set-iff linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset
linorder-lit.is-maximal-in-mset-iff linorder-lit.is-maximal-in-set-eq-is-greatest-in-set
linorder-lit.is-maximal-in-set-iff linorder-lit.leI ord-res.asymp-less-cls
ord-res.multp-if-all-left-smaller transpE)

have *mono-atms-lt*: *monotone-on* (*set* Γ) (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) (λ*x y*. *y* ≤ *x*)
(λ*x*. *atm-of K* ⪯$_t$ *atm-of* (*fst x*))
proof (*intro monotone-onI*, *unfold le-bool-def*, *intro impI*)
fix *x y*
assume *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*) **and** *atm-of K* ⪯$_t$ *atm-of* (*fst y*)
thus *atm-of K* ⪯$_t$ *atm-of* (*fst x*)
by *order*
qed

obtain Γ$_1$ Γ$_0$ **where** Γ-*eq*: Γ = Γ$_1$ @ (− *K*, *Some C*) # Γ$_0$
using ‹(− *K*, *Some C*) ∈ *set* Γ› **by** (*metis split-list*)

hence *trail-true-cls* Γ$_0$ *x*
using Γ-*prop-ball-lt-true x-in'* ‹*x* ≺$_c$ *C*› **by** *metis*

then obtain *L$_x$* **where** *L$_x$* ∈# *x* **and** *L$_x$-true*: *trail-true-lit* Γ$_0$ *L$_x$*
unfolding *trail-true-cls-def* **by** *auto*

moreover have Γ′ = Γ$_0$
proof −
have Γ′ = *dropWhile* (λ*Ln*. *atm-of K* ⪯$_t$ *atm-of* (*fst Ln*)) ((Γ$_1$ @ [(− *K*, *Some C*)]) @ Γ$_0$)
unfolding ‹Γ′ = *dropWhile* (λ*Ln*. *atm-of K* ⪯$_t$ *atm-of* (*fst Ln*)) Γ› Γ-*eq*
by *simp*

also have ... = *dropWhile* (λ*Ln*. *atm-of K* ⪯$_t$ *atm-of* (*fst Ln*)) Γ$_0$
proof (*rule dropWhile-append2*)
fix *Ln* :: *'f gterm literal* × *'f gclause option*
assume *Ln* ∈ *set* (Γ$_1$ @ [(− *K*, *Some C*)])

moreover have ∀ *x*∈*set* Γ$_1$. *atm-of K* ≺$_t$ *atm-of* (*fst x*)
using Γ-*sorted* **by** (*simp add*: Γ-*eq sorted-wrt-append*)

ultimately show *atm-of K* ⪯$_t$ *atm-of* (*fst Ln*)
using ‹*is-neg K*› **by** *auto*
qed

also have ... = Γ$_0$
proof (*cases* Γ$_0$)
case *Nil*
thus *?thesis*
by (*simp add*: *dropWhile-eq-self-iff*)

next
case (*Cons Ln* $\Gamma_0{}'$)

hence *atm-of* (*fst Ln*) $\prec_t$ *atm-of K*
using Γ*-sorted* **by** (*simp add*: Γ*-eq sorted-wrt-append*)

hence ¬ *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)
by *order*

hence ¬ *atm-of K* $\preceq_t$ *atm-of* (*fst* (*hd* Γ_0))
by (*simp add*: ‹Γ_0 = *Ln* # $\Gamma_0{}'$›)

thus *?thesis*
by (*simp add*: *dropWhile-eq-self-iff*)
qed

finally show *?thesis* **.**
qed

ultimately have *trail-true-lit* Γ' L_x
by *argo*

thus *trail-true-cls* Γ' *x*
using ‹L_x ∈# *x*› **unfolding** *trail-true-cls-def* **by** *metis*

fix K_x
assume *x-max-lit*: *ord-res.is-maximal-lit* K_x *x*

hence $K_x \preceq_l K$
using ‹*x* $\prec_c$ *eres D E*› *eres-max-lit*
using *linorder-lit.multp-if-maximal-less-that-maximal* **by** *fastforce*

hence *atm-of* K_x $\preceq_t$ *atm-of K*
by (*cases* K_x; *cases K*) *simp-all*

have *A* |∈| *trail-atms* Γ'
if *A-lt*: *A* $\prec_t$ *atm-of* K_x **and** *A-in*: *A* |∈| *atms-of-clss* (*N* |∪| U_{er}) **for** *A*
unfolding ‹Γ' = *dropWhile* (λ*Ln*. *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)) Γ›
proof (*rule in-trail-atms-dropWhileI*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ*-sorted* **.**
next
show *monotone-on* (*set* Γ) (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) (λ*x y*. *y* ≤ *x*) (λ*x*. *atm-of K* $\preceq_t$ *atm-of* (*fst x*))
using *mono-atms-lt* **.**
next
show ¬ *atm-of K* $\preceq_t$ *A*
using *A-lt* ‹*atm-of* K_x $\preceq_t$ *atm-of K*› **by** *order*

next
have $\neg$ ($\exists$ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* $\prec_t$ *atm-of* K_x $\wedge$ *A* |∉| *trail-atms* Γ)
using *no-undef-atm-lt-max-lit-if-lt-E* ‹*x* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› ‹*x* $\prec_c$ *E*› *x-max-lit*
by *metis*

thus *A* |∈| *trail-atms* Γ
using *A-in A-lt* **by** *metis*
qed

thus $\neg$ ($\exists$ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}'). *A* $\prec_t$ *atm-of* K_x $\wedge$ *A* |∉| *trail-atms* Γ')
using ‹*atms-of-clss* (*N* |∪| U_{er}) = *atms-of-clss* (*N* |∪| U_{er}')› **by** *metis*
qed

thus
$\forall$ *x*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}'). ($\forall$ *DE*. *Some C* = *Some DE* $\longrightarrow$ *x* $\prec_c$ *DE*) $\longrightarrow$
trail-true-cls Γ' *x*
$\forall$ *x*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}'). ($\forall$ *DE*. *Some C* = *Some DE* $\longrightarrow$ *x* $\prec_c$ *DE*) $\longrightarrow$
($\forall$ *K*. *ord-res.is-maximal-lit K x* $\longrightarrow$
$\neg$ ($\exists$ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}'). *A* $\prec_t$ *atm-of K* $\wedge$ *A* |∉| *trail-atms* Γ'))
unfolding *atomize-conj* **by** *metis*

have *trail-true-cls* Γ' {#*K* ∈# *B*. *K* ≠ L_B#} $\wedge$ *is-pos* L_B
if *Some C* = *Some C'* **and**
B-in: *B* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}') **and**
B-lt: *B* $\prec_c$ *C'* **and**
B-max-lit: *linorder-lit.is-maximal-in-mset B* L_B **and**
L_B*-undef*: $\neg$ *trail-defined-lit* Γ' L_B
for *C' B* L_B
proof −
have *C'* = *C*
using *that* **by** *simp*
hence *B* $\prec_c$ *C*
using *B-lt* **by** *order*
hence *B* $\prec_c$ *eres D E*
using ‹*C* $\prec_c$ *eres D E*› **by** *order*
hence *B* ≠ *eres D E*
by *order*
hence *B* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using *B-in* ‹*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))›
by *simp*
moreover have *B* $\prec_c$ *E*
using ‹*B* $\prec_c$ *eres D E*› ‹*eres D E* $\prec_c$ *E*› **by** *order*

have $L_B \preceq_l - K$
using ‹$B \prec_c C$› *B-max-lit C-max-lit*
by (*metis asympD linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.leI*
linorder-lit.multp-if-maximal-less-that-maximal ord-res.asymp-less-cls)
hence $L_B \prec_l - K \vee L_B = - K$
by *simp*

thus *?thesis*
proof (*elim disjE*)
assume $L_B \prec_l - K$
hence *atm-of* $L_B \prec_t$ *atm-of* K
using ‹*is-neg* K›
by (*cases* L_B; *cases* K) *simp-all*

have *trail-defined-lit* Γ' L_B
unfolding ‹$\Gamma' =$ *dropWhile* ($\lambda Ln.$ *atm-of* $K \preceq_t$ *atm-of* (*fst Ln*)) Γ›
proof (*intro trail-defined-lit-dropWhileI ballI*)
show *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**
next
show *monotone-on* (*set* Γ) ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) ($\lambda x\ y.$ $y \leq x$)
($\lambda x.$ *atm-of* $K \preceq_t$ *atm-of* (*fst x*))
using *mono-atms-lt* **.**
next
have $\neg$ *atm-of* $K \preceq_t$ *atm-of* L_B
using ‹*atm-of* $L_B \prec_t$ *atm-of* K› **by** *order*
thus $\neg$ *atm-of* $K \preceq_t$ *atm-of* $L_B \wedge \neg$ *atm-of* $K \preceq_t$ *atm-of* ($-$ L_B)
by *simp*
next
have *trail-defined-lit* Γ K
by (*metis map-of-eq-None-iff option.discI step-hyps*(*12*) *trail-defined-lit-def*)
moreover have *atm-of* K $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er})
by (*metis* ‹*atms-of-clss* (N $|\cup|$ U_{er}) = *atms-of-clss* (N $|\cup|$ U_{er}')›
‹*iefac* $\mathcal{F}$ |‘| (N $|\cup|$ U_{er}') = *finsert* (*eres D E*) (*iefac* $\mathcal{F}$ |‘| (N $|\cup|$ U_{er}))›
atm-of-in-atms-of-clssI eres-max-lit finsertI1 linorder-lit.is-maximal-in-mset-iff)
moreover have *atm-of* L_B $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er})
using ‹B $|\in|$ *iefac* $\mathcal{F}$ |‘| (N $|\cup|$ U_{er})› *B-max-lit atm-of-in-atms-of-clssI*
unfolding *linorder-lit.is-maximal-in-mset-iff*
by *metis*
ultimately show *trail-defined-lit* Γ L_B
using ‹*atm-of* $L_B \prec_t$ *atm-of* K› Γ-*lower*
unfolding *trail-defined-lit-iff-trail-defined-atm*
by (*meson linorder-trm.is-lower-set-iff*)
qed

hence *False*

using L_B*-undef* **by** *contradiction*

thus *?thesis* **..**
next
have *trail-true-cls* Γ' B
using *clause-true-in-*Γ'*-if B-in* ‹$B \prec_c C$› **by** *blast*
hence *trail-true-cls* Γ' $\{\#K \in\# B.\ K \neq L_B\#\}$
using L_B*-undef*
by (*smt* (*verit*, *best*) *mem-Collect-eq set-mset-filter trail-defined-lit-iff-true-or-false trail-true-cls-def*)
moreover assume $L_B = -K$
ultimately show *trail-true-cls* Γ' $\{\#K \in\# B.\ K \neq L_B\#\} \wedge$ *is-pos* L_B
using ‹*is-neg* K›
by (*cases* K) *simp-all*
qed
qed

thus $\forall$ *Da. Some* C = *Some Da* $\longrightarrow$
($\forall x|\in|$*iefac* $\mathcal{F}$ $|`|$ (N $|\cup|$ U_{er}'). $x \prec_c$ *Da* $\longrightarrow$ ($\forall L_C$. *ord-res.is-maximal-lit* L_C x $\longrightarrow$
$\neg$ *trail-defined-lit* Γ' L_C $\longrightarrow$ *trail-true-cls* Γ' $\{\#K \in\# x.\ K \neq L_C\#\} \wedge$ *is-pos* L_C))
by *metis*
qed *simp-all*
qed

lemma *rtranclp-ord-res-7-preserves-ord-res-7-invars*:
assumes
step: (*ord-res-7* N)** s s' **and**
invars: *ord-res-7-invars* N s
shows *ord-res-7-invars* N s'
using *step invars ord-res-7-preserves-invars*
by (*smt* (*verit*, *del-insts*) *rtranclp-induct*)

lemma *tranclp-ord-res-7-preserves-ord-res-7-invars*:
assumes
step: (*ord-res-7* N)$^{++}$ s s' **and**
invars: *ord-res-7-invars* N s
shows *ord-res-7-invars* N s'
using *step invars ord-res-7-preserves-invars*
by (*smt* (*verit*, *del-insts*) *tranclp-induct*)

lemma *propagating-clause-almost-false*:
assumes *invars*: *ord-res-7-invars* N (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) **and** (L, *Some* C) $\in$ *set* Γ
shows *trail-false-cls* Γ $\{\#K \in\# C.\ K \neq L\#\}$
proof $-$
have $\forall \Gamma_1\ L\ C\ \Gamma_0.\ \Gamma = \Gamma_1$ @ (L, *Some* C) # $\Gamma_0 \longrightarrow$ *trail-false-cls* Γ_0 $\{\#K \in\# C.\ K \neq L\#\}$
using *invars* **unfolding** *ord-res-7-invars-def* **by** *metis*

hence $\exists \Gamma_1\ \Gamma_0.\ \Gamma = \Gamma_1 \mathbin{@} (L,\ \mathit{Some}\ C) \#\ \Gamma_0 \wedge \mathit{trail\text{-}false\text{-}cls}\ \Gamma_0\ \{\#K \in\#\ C.\ K \neq L\#\}$
using ‹$(L,\ \mathit{Some}\ C) \in \mathit{set}\ \Gamma$› **unfolding** *in-set-conv-decomp* **by** *metis*

thus *trail-false-cls* $\Gamma\ \{\#K \in\#\ C.\ K \neq L\#\}$
by (*metis suffixI suffix-ConsD trail-false-cls-if-trail-false-suffix*)
qed

lemma *ex-ord-res-7-if-not-final*:
assumes
not-final: $\neg$ *ord-res-7-final* $(N,\ s)$ **and**
invars: *ord-res-7-invars N s*
shows $\exists s'.$ *ord-res-7 N s* s'
proof −
obtain $U_{er}\ \mathcal{F}\ \Gamma\ \mathcal{C}$ **where** $s = (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$
by (*metis surj-pair*)

note *invars′* = *invars*[*unfolded ord-res-7-invars-def*, *rule-format*, *OF* ‹$s = (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›]

have Γ-*sorted*: *sorted-wrt* $(\lambda x\ y.\ \mathit{atm\text{-}of}\ (\mathit{fst}\ y) \prec_t \mathit{atm\text{-}of}\ (\mathit{fst}\ x))\ \Gamma$
using *invars′* **by** *argo*

have Γ-*consistent*: *trail-consistent* Γ
using Γ-*sorted trail-consistent-if-sorted-wrt-atoms* **by** *metis*

have *distinct* (*map fst* Γ)
proof (*rule distinct-if-sorted-wrt-asymp*)
have *sorted-wrt* $(\lambda x\ y.\ \mathit{fst}\ y \prec_l \mathit{fst}\ x)\ \Gamma$
proof (*rule sorted-wrt-mono-rel*)
show *sorted-wrt* $(\lambda x\ y.\ \mathit{atm\text{-}of}\ (\mathit{fst}\ y) \prec_t \mathit{atm\text{-}of}\ (\mathit{fst}\ x))\ \Gamma$
using Γ-*sorted* **.**
next
fix $x\ y :: 'f\ \mathit{gterm\ literal} \times 'f\ \mathit{gterm\ literal\ multiset\ option}$
assume *atm-of* $(\mathit{fst}\ y) \prec_t \mathit{atm\text{-}of}\ (\mathit{fst}\ x)$
thus $\mathit{fst}\ y \prec_l \mathit{fst}\ x$
by (*cases fst x*; *cases fst y*) (*simp-all only*: *literal.sel ord-res.less-lit-simps*)
qed

thus *sorted-wrt* $(\lambda x\ y.\ y \prec_l x)$ (*map fst* Γ)
unfolding *sorted-wrt-map* **.**
next
show *asymp-on* (*set* (*map fst* Γ)) $(\lambda x\ y.\ y \prec_l x)$
using *linorder-lit.asymp-on-greater* **.**
qed

hence *map-of*-Γ-*eq-SomeI*: $\bigwedge x\ y.\ (x,\ y) \in \mathit{set}\ \Gamma \Longrightarrow \mathit{map\text{-}of}\ \Gamma\ x = \mathit{Some}\ y$
by (*metis map-of-is-SomeI*)

have Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using *invars′* **by** *argo*

obtain *E* **where** $\mathcal{C}$ = *Some E* **and** *E* ≠ {#}
using *not-final*[*unfolded* ‹*s* = (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)› *ord-res-7-final.simps*, *simplified*]
by *metis*

have *E-in*: *E* |∈| *iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er})
using *invars′* ‹$\mathcal{C}$ = *Some E*› **by** *metis*

obtain *L* **where** *E-max-lit*: *ord-res.is-maximal-lit L E*
using ‹*E* ≠ {#}› *linorder-lit.ex-maximal-in-mset* **by** *metis*

show *?thesis*
proof (*cases trail-false-cls* Γ *E*)
case *E-false*: *True*

hence *L-false*: *trail-false-lit* Γ *L*
using *E-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff trail-false-cls-def*
by *metis*

hence ∃ *opt*. (− *L*, *opt*) ∈ *set* Γ
by (*auto simp*: *trail-false-lit-def*)

have ¬ *is-pos L*
proof (*rule notI*)
assume *is-pos L*

hence *is-neg* (− *L*)
by (*metis* ‹*is-pos L*› *is-pos-neg-not-is-pos*)

hence (− *L*, *None*) ∈ *set* Γ
using *invars′* ‹∃ *opt*. (− *L*, *opt*) ∈ *set* Γ› **by** (*metis prod.sel*)

moreover have *E* ≺$_c$ {#− *L*#}
using *E-max-lit* ‹*is-pos L*›
by (*smt* (*verit*, *best*) *Neg-atm-of-iff add-mset-remove-trivial empty-iff*
ord-res.less-lit-simps(*4*) *linorder-cls.less-irrefl linorder-lit.is-greatest-in-mset-iff*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.less-linear
linorder-lit.multp-if-maximal-less-that-maximal literal.collapse(*1*)
ord-res.less-lit-simps(*1*) *set-mset-empty uminus-literal-def union-single-eq-member*)

ultimately have *trail-true-cls* Γ *E*
using *invars′ E-in* **by** *force*

hence ¬ *trail-false-cls* Γ *E*
by (*metis* Γ-*consistent not-trail-true-cls-and-trail-false-cls*)

thus *False*
using *E-false* **by** *contradiction*
qed

hence *L-neg*: *is-neg L*
by *argo*

then obtain *D* **where** $(-\ L,\ Some\ D) \in set\ \Gamma$
using *invars'* ‹*is-neg L*› ‹$\exists\, opt.\ (-\ L,\ opt) \in set\ \Gamma$›
by (*metis prod.sel is-pos-neg-not-is-pos not-Some-eq*)

hence *map-of* $\Gamma\ (-\ L) = Some\ (Some\ D)$
using *map-of*-Γ-*eq-SomeI* **by** *metis*

have $\exists \Gamma_1\ \Gamma_0.\ \Gamma = \Gamma_1\ @\ (-\ L,\ Some\ D)\ \#\ \Gamma_0 \wedge trail\text{-}false\text{-}cls\ \Gamma_0\ \{\#K \in\# D.\ K \neq -\ L\#\}$
using *invars'* ‹$(-\ L,\ Some\ D) \in set\ \Gamma$› *propagating-clause-almost-false*
unfolding *in-set-conv-decomp* **by** *metis*

have *D-almost-false*: *trail-false-cls* $\Gamma\ \{\#K \in\# D.\ K \neq -\ L\#\}$
using *invars* ‹$s = (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$› ‹$(-\ L,\ Some\ D) \in set\ \Gamma$› *propagating-clause-almost-false*
by *metis*

show *?thesis*
proof (*cases eres D E* $= \{\#\}$)
case *True*
thus *?thesis*
unfolding ‹$s = (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$› ‹$\mathcal{C} = Some\ E$›
using *E-false E-max-lit L-neg* ‹*map-of* $\Gamma\ (-\ L) = Some\ (Some\ D)$›
using *ord-res-7.resolution-bot* **by** *metis*
next
case *False*

then obtain *K* **where** *eres-max-lit*: *ord-res.is-maximal-lit K* (*eres D E*)
using *linorder-lit.ex-maximal-in-mset* **by** *metis*

hence $K \in\#$ *eres D E*
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence $K \in\#\ D \wedge K \neq -\ L \vee K \in\#\ E$
using *strong-lit-in-one-of-resolvents-if-in-eres*[*OF E-max-lit*] **by** *metis*

hence *K-false*: *trail-false-lit* $\Gamma\ K$
using *D-almost-false E-false* **unfolding** *trail-false-cls-def* **by** *auto*

hence $\exists\, opt.\ (-\ K,\ opt) \in set\ \Gamma$
by (*auto simp*: *trail-false-lit-def*)

```
      show ?thesis
      proof (cases K)
        case (Pos A_K)

        hence is-pos K
          by simp

        thus ?thesis
          unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
          using E-false E-max-lit L-neg ‹map-of Γ (− L) = Some (Some D)› False
eres-max-lit
          using ord-res-7.resolution-pos by metis
      next
        case (Neg A_K)

        hence is-neg K
          by simp

        then obtain C where (− K, Some C) ∈ set Γ
          using invars' ‹is-neg L› ‹∃ opt. (− K, opt) ∈ set Γ›
          by (metis prod.sel is-pos-neg-not-is-pos not-Some-eq)

        hence map-of Γ (− K) = Some (Some C)
          using map-of-Γ-eq-SomeI by metis

        thus ?thesis
          unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
          using E-false E-max-lit L-neg ‹map-of Γ (− L) = Some (Some D)› False
eres-max-lit
            ‹is-neg K›
          using ord-res-7.resolution-neg by metis
      qed
    qed
  next
    case E-not-false: False
    show ?thesis
    proof (cases ∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of L ∧ A |∉| trail-atms
Γ)
      case True

      then obtain A where A-least: linorder-trm.is-least-in-fset
        {|A |∈| atms-of-clss (N |∪| U_er). A ≺_t atm-of L ∧ A |∉| trail-atms Γ|} A
        by (metis (no-types, lifting) bot-fset.rep-eq empty-iff ffmember-filter
            linorder-trm.ex-least-in-fset)

      show ?thesis
        unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
        using ord-res-7.decide-neg[OF E-not-false E-max-lit A-least refl, of F]
        by metis
```

```
    next
      case nex-undef-atm-lt-L: False
      show ?thesis
      proof (cases trail-defined-lit Γ L)
        case True
        thus ?thesis
          unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
          using ord-res-7.skip-defined[OF E-not-false E-max-lit nex-undef-atm-lt-L]
          by metis
      next
        case L-undef: False
        show ?thesis
        proof (cases L)
          case L-eq: (Pos A_L)

          hence L-pos: is-pos L
            by simp

          show ?thesis
          proof (cases trail-false-cls Γ {#K ∈# E. K ≠ L#})
            case E-almost-false: True
            show ?thesis
            proof (cases ord-res.is-strictly-maximal-lit L E)
              case True
              thus ?thesis
                unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
             using ord-res-7.production[OF E-not-false E-max-lit nex-undef-atm-lt-L
L-undef L-pos
                    E-almost-false]
                by metis
            next
              case False
              thus ?thesis
                unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
               using ord-res-7.factoring[OF E-not-false E-max-lit nex-undef-atm-lt-L
L-undef L-pos
                    E-almost-false]
                by metis
            qed
          next
            case E-not-almost-false: False
            show ?thesis
            proof (cases ∃ D|∈|iefac F |`| (N |∪| U_er). E ≺_c D)
              case True

              then obtain F where linorder-cls.is-least-in-fset
                (ffilter ((≺_c) E) (iefac F |`| (N |∪| U_er))) F
            by (metis bot-fset.rep-eq empty-iff ffmember-filter linorder-cls.ex1-least-in-fset)
```

```
            thus ?thesis
              unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
           using ord-res-7.skip-undefined-pos[OF E-not-false E-max-lit nex-undef-atm-lt-L
L-undef
                  L-pos E-not-almost-false]
              by metis
            next
              case False
              thus ?thesis
                unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
                using ord-res-7.skip-undefined-pos-ultimate[OF E-not-false E-max-lit
                    nex-undef-atm-lt-L L-undef L-pos E-not-almost-false]
                by metis
            qed
          qed
        next
          case L-eq: (Neg A_L)

          hence is-neg L
            by simp

          thus ?thesis
            unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some E›
        using ord-res-7.skip-undefined-neg[OF E-not-false E-max-lit nex-undef-atm-lt-L
L-undef]
            by metis
        qed
      qed
    qed
  qed
qed

lemma ord-res-7-safe-state-if-invars:
  fixes N :: 'f gclause fset and s
  assumes invars: ord-res-7-invars N s
  shows safe-state (constant-context ord-res-7) ord-res-7-final (N, s)
  using safe-state-constant-context-if-invars[where
      R = ord-res-7 and F = ord-res-7-final and I = ord-res-7-invars]
  using ord-res-7-preserves-invars ex-ord-res-7-if-not-final invars by metis

end

end
theory Clause-Could-Propagate
  imports
    Background
    Implicit-Exhaustive-Factorization
begin
```

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

definition *clause-could-propagate* **where**
clause-could-propagate Γ *C L* ⟷ ¬ *trail-defined-lit* Γ *L* ∧
linorder-lit.is-maximal-in-mset C L ∧ *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*#}

lemma *trail-false-if-could-have-propagatated*:
clause-could-propagate Γ *C L* ⟹ *trail-false-cls* ((− *L*, *n*) # Γ) *C*
unfolding *clause-could-propagate-def trail-false-cls-def trail-false-lit-def* **by** *auto*

lemma *atoms-of-trail-lt-atom-of-propagatable-literal*:
assumes
Γ-*lower*: *linorder-trm.is-lower-set* (*fset* (*trail-atms* Γ)) 𝒜 **and**
C-prop: *clause-could-propagate* Γ *C L* **and**
atm-of L ∈ 𝒜
shows ∀ *A* |∈| *trail-atms* Γ. *A* ≺$_t$ *atm-of L*
proof −
have *atm-of L* |∉| *trail-atms* Γ
using *C-prop*
unfolding *clause-could-propagate-def trail-defined-lit-iff-trail-defined-atm*
by *argo*

then show *?thesis*
using Γ-*lower* ‹*atm-of L* ∈ 𝒜›
by (*metis linorder-trm.is-lower-set-iff linorder-trm.neqE*)
qed

lemma *trail-false-cls-filter-mset-iff*:
trail-false-cls Γ {#*Ka* ∈# *C*. *Ka* ≠ *K*#} ⟷ (∀ *L*∈#*C*. *L* ≠ *K* ⟶ *trail-false-lit* Γ *L*)
unfolding *trail-false-cls-def* **by** *auto*

lemma *clause-could-propagate-iff*: *clause-could-propagate* Γ *C K* ⟷
¬ *trail-defined-lit* Γ *K* ∧ *ord-res.is-maximal-lit K C* ∧ (∀ *L*∈#*C*. *L* ≠ *K* ⟶ *trail-false-lit* Γ *L*)
unfolding *clause-could-propagate-def trail-false-cls-filter-mset-iff* **..**

lemma *clause-could-propagate-efac*: *clause-could-propagate* Γ (*efac C*) = *clause-could-propagate* Γ *C*
proof (*rule ext*)
fix *L*
show *clause-could-propagate* Γ (*efac C*) *L* ⟷ *clause-could-propagate* Γ *C L*
unfolding *clause-could-propagate-def*
by (*metis* (*mono-tags*, *lifting*) *ex1-efac-eq-factoring-chain mem-Collect-eq*
ord-res.ground-factorings-preserves-maximal-literal set-mset-efac set-mset-filter
trail-false-cls-def)
qed

lemma *bex-clause-could-propagate-simp*:

fixes $\mathcal{F}$ N Γ L
shows *fBex* (*iefac* $\mathcal{F}$ |‘| N) (λC. *clause-could-propagate* Γ C L) $\longleftrightarrow$
fBex N (λC. *clause-could-propagate* Γ C L)
sketch (*rule iffI*; *elim bexE*)
proof (*rule iffI* ; *elim bexE*)
fix C :: *'f gclause*
assume C |∈| *iefac* $\mathcal{F}$ |‘| N **and** *clause-could-propagate* Γ C L
thus ∃ C |∈| N. *clause-could-propagate* Γ C L
by (*metis clause-could-propagate-efac fimageE iefac-def*)
next
fix C :: *'f gclause*
assume C |∈| N **and** *clause-could-propagate* Γ C L
thus ∃ C|∈|*iefac* $\mathcal{F}$ |‘| N. *clause-could-propagate* Γ C L
by (*metis clause-could-propagate-efac fimageI iefac-def*)
qed

end

end
theory *ORD-RES-8*
imports
Background
Implicit-Exhaustive-Factorization
Exhaustive-Resolution
Clause-Could-Propagate
begin

23 ORD-RES-8 (atom-guided literal trail construction)

type-synonym *'f ord-res-8-state* =
'f gclause fset ×*'f gclause fset* × *'f gclause fset* × (*'f gliteral* × *'f gclause option*) *list*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-8* **where**
decide-neg:
¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ C) ⟹
linorder-trm.is-least-in-fset {|A_2 |∈| *atms-of-clss* (N |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A_2$|} A ⟹
¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *clause-could-propagate* Γ C (*Pos* A)) ⟹
Γ' = (*Neg* A, *None*) # Γ ⟹
ord-res-8 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}$, Γ') |

propagate:
¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ C) ⟹
linorder-trm.is-least-in-fset {|A_2 |∈| *atms-of-clss* (N |∪| U_{er}).

$\forall A_1 \mid\in\mid$ *trail-atms* Γ. $A_1 \prec_t A_2 \mid\}$ $A \Longrightarrow$
linorder-cls.is-least-in-fset $\{\mid C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$.
clause-could-propagate Γ C (*Pos* A)$\mid\}$ $C \Longrightarrow$
linorder-lit.is-greatest-in-mset C (*Pos* A) $\Longrightarrow$
$\Gamma' =$ (*Pos* A, *Some* C) # $\Gamma \Longrightarrow$
ord-res-8 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}$, Γ') |

factorize:
$\neg$ ($\exists$ $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. *trail-false-cls* Γ C) $\Longrightarrow$
linorder-trm.is-least-in-fset $\{\mid A_2 \mid\in\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$).
$\forall A_1 \mid\in\mid$ *trail-atms* Γ. $A_1 \prec_t A_2 \mid\}$ $A \Longrightarrow$
linorder-cls.is-least-in-fset $\{\mid C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$.
clause-could-propagate Γ C (*Pos* A)$\mid\}$ $C \Longrightarrow$
$\neg$ *linorder-lit.is-greatest-in-mset* C (*Pos* A) $\Longrightarrow$
$\mathcal{F}' =$ *finsert* C $\mathcal{F} \Longrightarrow$
ord-res-8 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}'$, Γ) |

resolution:
linorder-cls.is-least-in-fset $\{\mid D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$. *trail-false-cls* Γ $D\mid\}$
$D \Longrightarrow$
linorder-lit.is-maximal-in-mset D (*Neg* A) $\Longrightarrow$
map-of Γ (*Pos* A) = *Some* (*Some* C) $\Longrightarrow$
$U_{er}' =$ *finsert* (*eres* C D) $U_{er} \Longrightarrow$
$\Gamma' =$ *dropWhile* (λLn. $\forall K$.
linorder-lit.is-maximal-in-mset (*eres* C D) $K \longrightarrow$ *atm-of* $K \preceq_t$ *atm-of* (*fst* Ln)) $\Gamma \Longrightarrow$
ord-res-8 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}', $\mathcal{F}$, Γ')

lemma *right-unique-ord-res-8*:
fixes N :: *'f gclause fset*
shows *right-unique* (*ord-res-8* N)
proof (*rule right-uniqueI*)
fix x y z
assume *step1*: *ord-res-8* N x y **and** *step2*: *ord-res-8* N x z
show $y = z$
using *step1*
proof (*cases* N x y *rule*: *ord-res-8.cases*)
case *step1-hyps*: (*decide-neg* $\mathcal{F}$ U_{er} Γ A Γ')
show *?thesis*
using *step2* **unfolding** ‹$x = (U_{er}, \mathcal{F}, \Gamma)$›
proof (*cases* N (U_{er}, $\mathcal{F}$, Γ) z *rule*: *ord-res-8.cases*)
case (*decide-neg* A Γ')
with *step1-hyps* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-trm.dual-order.asym linorder-trm.is-least-in-fset-iff*)
next
case (*propagate A2 C2* $\Gamma 2'$)
with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-fset-ffilterD*(*1*)
linorder-cls.is-least-in-fset-ffilterD(*2*) *linorder-trm.dual-order.asym*

```
      linorder-trm.is-least-in-fset-iff)
   thus ?thesis ..
  next
   case (factorize A2 C2 F2')
   with step1-hyps have False
    by (metis (no-types, lifting) linorder-cls.is-least-in-fset-ffilterD(1)
      linorder-cls.is-least-in-fset-ffilterD(2) linorder-trm.dual-order.asym
      linorder-trm.is-least-in-fset-iff)
   thus ?thesis ..
  next
   case (resolution D2 A2 C2 U_er2' Γ2')
   with step1-hyps have False
    by (metis linorder-cls.is-least-in-ffilter-iff)
   thus ?thesis ..
  qed
next
  case step1-hyps: (propagate F U_er Γ A C Γ')
  show ?thesis
   using step2 unfolding ‹x = (U_er, F, Γ)›
  proof (cases N (U_er, F, Γ) z rule: ord-res-8.cases)
   case (decide-neg A2 Γ2')
   with step1-hyps have False
    by (metis (no-types, lifting) linorder-cls.is-least-in-fset-ffilterD(1)
      linorder-cls.is-least-in-fset-ffilterD(2) linorder-trm.dual-order.asym
      linorder-trm.is-least-in-fset-iff)
   thus ?thesis ..
  next
   case (propagate A2 C2 Γ2')
   with step1-hyps show ?thesis
    by (metis (no-types, lifting) linorder-cls.Uniq-is-least-in-fset
     linorder-trm.dual-order.asym linorder-trm.is-least-in-fset-iff The-optional-eq-SomeI)
  next
   case (factorize A2 C2 F2')
   with step1-hyps have False
    by (metis (no-types, lifting) linorder-cls.Uniq-is-least-in-fset
      linorder-trm.Uniq-is-least-in-fset the1-equality')
   thus ?thesis ..
  next
   case (resolution D2 A2 C2 U_er2' Γ2')
   with step1-hyps have False
    by (metis linorder-cls.is-least-in-ffilter-iff)
   thus ?thesis ..
  qed
next
  case step1-hyps: (factorize F U_er Γ A C F')
  show ?thesis
   using step2 unfolding ‹x = (U_er, F, Γ)›
  proof (cases N (U_er, F, Γ) z rule: ord-res-8.cases)
   case (decide-neg A2 Γ2')
```

with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-fset-ffilterD*(*1*)
linorder-cls.is-least-in-fset-ffilterD(*2*) *linorder-trm.dual-order.asym*
linorder-trm.is-least-in-fset-iff)
thus *?thesis* **..**
next
case (*propagate A2 C2* Γ*2′*)
with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.Uniq-is-least-in-fset*
linorder-trm.Uniq-is-least-in-fset the1-equality′)
thus *?thesis* **..**
next
case (*factorize A2 C2* $\mathcal{F}$*2′*)
with *step1-hyps* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-cls.Uniq-is-least-in-fset*
linorder-trm.Uniq-is-least-in-fset the1-equality′)
next
case (*resolution D2 A2 C2* U_{er}*2′* Γ*2′*)
with *step1-hyps* **have** *False*
by (*metis linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
qed
next
case *step1-hyps*: (*resolution* Γ $\mathcal{F}$ U_{er} *D A C* $U_{er}{}'$ Γ′)
show *?thesis*
using *step2* **unfolding** ‹x = (U_{er}, $\mathcal{F}$, Γ)›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *z rule*: *ord-res-8.cases*)
case (*decide-neg A* Γ′)
with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *opaque-lifting*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*propagate A C* Γ′)
with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *opaque-lifting*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*factorize A C* $\mathcal{F}$′)
with *step1-hyps* **have** *False*
by (*metis* (*no-types*, *opaque-lifting*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case *step2-hyps*: (*resolution D2 A2 C2* U_{er}*2′* Γ*2′*)
have *D2* = *D*
using ‹*linorder-cls.is-least-in-fset* (*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*›
using ‹*linorder-cls.is-least-in-fset* (*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D2*›
by (*metis linorder-cls.Uniq-is-least-in-fset Uniq-D*)

```
      have A2 = A
        using ‹linorder-lit.is-maximal-in-mset D (Neg A)›
        using ‹linorder-lit.is-maximal-in-mset D2 (Neg A2)›
        unfolding ‹D2 = D›
        by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset literal.sel(2))

      have C2 = C
        using step1-hyps(5) step2-hyps(4)
        unfolding ‹A2 = A›
        by simp

      show ?thesis
        unfolding ‹y = (U_er', F, Γ')› ‹z = (U_er2', F, Γ2')› prod.inject
      proof (intro conjI)
        show U_er' = U_er2'
         unfolding ‹U_er' = finsert (eres C D) U_er› ‹U_er2' = finsert (eres C2 D2)
U_er›
          unfolding ‹D2 = D› ‹C2 = C› ..
      next
        show F = F ..
      next
        show Γ' = Γ2'
          using step1-hyps(7) step2-hyps(6)
          unfolding ‹D2 = D› ‹C2 = C›
          by argo
      qed
    qed
  qed
qed

inductive ord-res-8-final :: 'f ord-res-8-state ⇒ bool where
  model-found:
    ¬ (∃ A |∈| atms-of-clss (N |∪| U_er). A |∉| trail-atms Γ) ⟹
    ¬ (∃ C |∈| iefac F |`| (N |∪| U_er). trail-false-cls Γ C) ⟹
    ord-res-8-final (N, U_er, F, Γ) |

  contradiction-found:
    {#} |∈| iefac F |`| (N |∪| U_er) ⟹
    ord-res-8-final (N, U_er, F, Γ)

sublocale ord-res-8-semantics: semantics where
  step = constant-context ord-res-8 and
  final = ord-res-8-final
proof unfold-locales
  fix S :: 'f ord-res-8-state

  obtain
    N U_er F :: 'f gterm clause fset and
```

```
    Γ :: ('f gterm literal × 'f gterm literal multiset option) list where
    S-def: S = (N, U_er, F, Γ)
    by (metis prod.exhaust)

  assume ord-res-8-final S

  hence ∄x. ord-res-8 N (U_er, F, Γ) x
    unfolding S-def
  proof (cases (N, U_er, F, Γ) rule: ord-res-8-final.cases)
    case model-found

    have ∄A. linorder-trm.is-least-in-fset {|A₂ |∈| atms-of-clss (N |∪| U_er).
      ∀A₁ |∈| trail-atms Γ. A₁ ≺_t A₂|} A
      using ‹¬ (∃A|∈|atms-of-clss (N |∪| U_er). A |∉| trail-atms Γ)›
      using linorder-trm.is-least-in-ffilter-iff by fastforce

    moreover have ∄C. linorder-cls.is-least-in-fset
      (ffilter (trail-false-cls Γ) (iefac F |`| (N |∪| U_er))) C
      using ‹¬ fBex (iefac F |`| (N |∪| U_er)) (trail-false-cls Γ)›
      by (metis linorder-cls.is-least-in-ffilter-iff)

    ultimately show ?thesis
      unfolding ord-res-8.simps by blast
  next
    case contradiction-found

    hence ∃C |∈| iefac F |`| (N |∪| U_er). trail-false-cls Γ C
      using trail-false-cls-mempty by metis

    moreover have ∄D A. linorder-cls.is-least-in-fset (ffilter (trail-false-cls Γ)
      (iefac F |`| (N |∪| U_er))) D ∧ ord-res.is-maximal-lit (Neg A) D
      by (metis empty-iff linorder-cls.is-least-in-ffilter-iff linorder-cls.leD
       linorder-lit.is-maximal-in-mset-iff local.contradiction-found(1) mempty-lesseq-cls
          set-mset-empty trail-false-cls-mempty)

    ultimately show ?thesis
      unfolding ord-res-8.simps by blast
  qed

  thus finished (constant-context ord-res-8) S
    by (simp add: S-def finished-def constant-context.simps)
qed

definition trail-is-sorted where
  trail-is-sorted N s ⟷
    (∀U_er F Γ. s = (U_er, F, Γ) ⟶
      sorted-wrt (λx y. atm-of (fst y) ≺_t atm-of (fst x)) Γ)

lemma ord-res-8-preserves-trail-is-sorted:
```

assumes
step: *ord-res-8 N s s′* **and**
invar: *trail-is-sorted N s*
shows *trail-is-sorted N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-8.cases*)
case *step-hyps*: (*decide-neg* $\mathcal{F}$ U_{er} Γ *A* Γ′)

have ∀ *x* |∈| *trail-atms* Γ. *x* ≺$_t$ *A*
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *simp*

hence ∀ *x* ∈ *set* Γ. *atm-of* (*fst x*) ≺$_t$ *A*
by (*simp add*: *fset-trail-atms*)

moreover have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using *invar* **by** (*simp add*: ‹*s* = (*U*$_{er}$, $\mathcal{F}$, Γ)› *trail-is-sorted-def*)

ultimately have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ′
by (*simp add*: ‹Γ′ = (*Neg A*, *None*) # Γ›)

thus *?thesis*
by (*simp add*: ‹*s′* = (*U*$_{er}$, $\mathcal{F}$, Γ′)› *trail-is-sorted-def*)
next
case *step-hyps*: (*propagate* $\mathcal{F}$ U_{er} Γ *A C* Γ′)

have ∀ *x* |∈| *trail-atms* Γ. *x* ≺$_t$ *A*
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *simp*

hence ∀ *x* ∈ *set* Γ. *atm-of* (*fst x*) ≺$_t$ *A*
by (*simp add*: *fset-trail-atms*)

moreover have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using *invar* **by** (*simp add*: ‹*s* = (*U*$_{er}$, $\mathcal{F}$, Γ)› *trail-is-sorted-def*)

ultimately have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ′
by (*simp add*: ‹Γ′ = (*Pos A*, *Some C*) # Γ›)

thus *?thesis*
by (*simp add*: ‹*s′* = (*U*$_{er}$, $\mathcal{F}$, Γ′)› *trail-is-sorted-def*)
next
case (*factorize* $\mathcal{F}$ U_{er} Γ *A C* $\mathcal{F}$′)

have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using *invar* **by** (*simp add*: ‹*s* = (*U*$_{er}$, $\mathcal{F}$, Γ)› *trail-is-sorted-def*)

thus *?thesis*
by (*simp add*: ‹*s′* = (*U*$_{er}$, $\mathcal{F}$′, Γ)› *trail-is-sorted-def*)
next
case *step-hyps*: (*resolution* Γ $\mathcal{F}$ U_{er} *D A C* U_{er}′ Γ′)

```
  have sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ
    using invar by (simp add: ‹s = (Uer, F, Γ)› trail-is-sorted-def)

  hence sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ'
    unfolding step-hyps(7) by (rule sorted-wrt-dropWhile)

  thus ?thesis
    by (simp add: ‹s' = (Uer', F, Γ')› trail-is-sorted-def)
qed

inductive trail-annotations-invars
  for N :: 'f gterm literal multiset fset
  where
    Nil:
      trail-annotations-invars N (Uer, F, []) |
    Cons-None:
      trail-annotations-invars N (Uer, F, (L, None) # Γ)
      if trail-annotations-invars N (Uer, F, Γ) |
    Cons-Some:
      trail-annotations-invars N (Uer, F, (L, Some D) # Γ)
      if linorder-lit.is-greatest-in-mset D L and
        D |∈| iefac F |`| (N |∪| Uer) and
        trail-false-cls Γ {#K ∈# D. K ≠ L#} and
        linorder-cls.is-least-in-fset
          {|D |∈| iefac F |`| (N |∪| Uer). clause-could-propagate Γ D L|} D and
        trail-annotations-invars N (Uer, F, Γ)

lemma
  assumes
    linorder-lit.is-greatest-in-mset C L and
    trail-false-cls Γ {#K ∈# C. K ≠ L#} and
    ¬ trail-defined-cls Γ C
  shows clause-could-propagate Γ C L
  unfolding clause-could-propagate-iff
proof (intro conjI)
  show linorder-lit.is-maximal-in-mset C L
    using ‹linorder-lit.is-greatest-in-mset C L›
    by (metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)

  hence L ∈# C
    unfolding linorder-lit.is-maximal-in-mset-iff ..

  show ∀K ∈# C. K ≠ L ⟶ trail-false-lit Γ K
    using ‹trail-false-cls Γ {#K ∈# C. K ≠ L#}›
    unfolding trail-false-cls-def
    by simp

  hence ∀K ∈# C. K ≠ L ⟶ trail-defined-lit Γ K
```

using *trail-defined-lit-iff-true-or-false* **by** *metis*

thus ¬ *trail-defined-lit* Γ *L*
using ‹¬ *trail-defined-cls* Γ *C*› ‹*L* ∈# *C*›
unfolding *trail-defined-cls-def*
by *metis*
qed

lemma *propagating-clause-in-clauses*:
assumes *trail-annotations-invars N* (*U*$_{er}$, *F*, Γ) **and** *map-of* Γ *L* = *Some* (*Some C*)
shows *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$)
using *assms*
proof (*induction* (*U*$_{er}$, *F*, Γ) *arbitrary*: Γ *rule*: *trail-annotations-invars.induct*)
case *Nil*
hence *False*
by *simp*
thus *?case* **..**
next
case (*Cons-None* Γ *K*)
thus *?case*
by (*metis map-of-Cons-code*(*2*) *option.discI option.inject*)
next
case (*Cons-Some D K* Γ)
thus *?case*
by (*metis map-of-Cons-code*(*2*) *option.inject*)
qed

lemma *trail-annotations-invars-mono-wrt-trail-suffix*:
assumes *suffix* Γ′ Γ *trail-annotations-invars N* (*U*$_{er}$, *F*, Γ)
shows *trail-annotations-invars N* (*U*$_{er}$, *F*, Γ′)
using *assms*(*2*,*1*)
proof (*induction* (*U*$_{er}$, *F*, Γ) *arbitrary*: Γ Γ′ *rule*: *trail-annotations-invars.induct*)
case *Nil*
thus *?case*
by (*simp add*: *trail-annotations-invars.Nil*)
next
case (*Cons-None* Γ *L*)
have Γ′ = (*L*, *None*) # Γ ∨ *suffix* Γ′ Γ
using *Cons-None.prems* **unfolding** *suffix-Cons* **.**
thus *?case*
using *Cons-None.hyps*
by (*metis trail-annotations-invars.Cons-None*)
next
case (*Cons-Some C L* Γ)
have Γ′ = (*L*, *Some C*) # Γ ∨ *suffix* Γ′ Γ
using *Cons-Some.prems* **unfolding** *suffix-Cons* **.**
then show *?case*
using *Cons-Some.hyps*

```
    by (metis trail-annotations-invars.Cons-Some)
qed

lemma ord-res-8-preserves-trail-annotations-invars:
  assumes
    step: ord-res-8 N s s' and
    invars:
      trail-annotations-invars N s
      trail-is-sorted N s
  shows trail-annotations-invars N s'
  using step
proof (cases N s s' rule: ord-res-8.cases)
  case step-hyps: (decide-neg F U_er Γ A Γ')
  show ?thesis
    unfolding ‹s' = (U_er, F, Γ')› ‹Γ' = (Neg A, None) # Γ›
  proof (rule trail-annotations-invars.Cons-None)
    show trail-annotations-invars N (U_er, F, Γ)
      using invars(1) by (simp add: ‹s = (U_er, F, Γ)›)
  qed
next
  case step-hyps: (propagate F U_er Γ A C Γ')
  show ?thesis
    unfolding ‹s' = (U_er, F, Γ')› ‹Γ' = (Pos A, Some C) # Γ›
  proof (rule trail-annotations-invars.Cons-Some)
    show ord-res.is-strictly-maximal-lit (Pos A) C
      using step-hyps by argo
  next
    show C |∈| iefac F |`| (N |∪| U_er)
      using step-hyps(5)
      by (simp add: linorder-cls.is-least-in-fset-iff)
  next
    show trail-false-cls Γ {#K ∈# C. K ≠ Pos A#}
      using step-hyps(5)
      by (simp add: linorder-cls.is-least-in-fset-iff clause-could-propagate-def)
  next
    show linorder-cls.is-least-in-fset
      {|C |∈| iefac F |`| (N |∪| U_er). clause-could-propagate Γ C (Pos A)|} C
      using step-hyps by argo
  next
    show trail-annotations-invars N (U_er, F, Γ)
      using invars(1) by (simp add: ‹s = (U_er, F, Γ)›)
  qed
next
  case step-hyps: (factorize F U_er Γ A E F')

  hence efac E ≠ E
    by (metis (no-types, lifting) ex1-efac-eq-factoring-chain ex-ground-factoringI
        linorder-cls.is-least-in-ffilter-iff clause-could-propagate-iff)
```

moreover have *clause-could-propagate* Γ *E* (*Pos A*)
using *step-hyps* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *metis*

ultimately have *ord-res.is-strictly-maximal-lit* (*Pos A*) (*efac E*)
unfolding *clause-could-propagate-def*
using *ex1-efac-eq-factoring-chain ex-ground-factoringI*
ord-res.ground-factorings-preserves-maximal-literal **by** *blast*

have $\mathcal{F} \mid\subseteq\mid \mathcal{F}'$
unfolding ‹$\mathcal{F}' =$ *finsert E* $\mathcal{F}$› **by** *auto*

have *trail-annotations-invars N* (U_{er}, $\mathcal{F}$, Γ)
using *invars*(*1*) **by** (*simp add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma)$›)

moreover have ∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A$
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *blast*

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}, \mathcal{F}', \Gamma)$›
proof (*induction* (U_{er}, $\mathcal{F}$, Γ) *arbitrary*: Γ *rule*: *trail-annotations-invars.induct*)
case *Nil*
show *?case*
by (*simp add*: *trail-annotations-invars.Nil*)
next
case (*Cons-None* Γ *L*)
show *?case*
proof (*rule trail-annotations-invars.Cons-None*)
have ∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A$
using *Cons-None.prems* **by** (*simp add*: *fset-trail-atms*)

thus *trail-annotations-invars N* (U_{er}, $\mathcal{F}'$, Γ)
using *Cons-None.hyps* **by** *argo*
qed
next
case (*Cons-Some C L* Γ)

have
clause-could-propagate Γ *C L* **and**
C-least: ∀ *y*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *y* ≠ *C* ⟶ *clause-could-propagate* Γ *y* *L* ⟶ *C* $\prec_c$ *y*
using *Cons-Some.hyps*(*4*) **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *metis*+

hence *ord-res.is-maximal-lit L C*
unfolding *clause-could-propagate-def* **by** *argo*

show *?case*
proof (*rule trail-annotations-invars.Cons-Some*)
show *ord-res.is-strictly-maximal-lit L C*

using ‹*ord-res.is-strictly-maximal-lit L C*› **.**

have *efac C* = *C*
using *Cons-Some*
by (*metis* (*no-types*, *opaque-lifting*) *efac-spec is-pos-def linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one
nex-strictly-maximal-pos-lit-if-neq-efac the1-equality')

hence *C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
using ‹*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})› ‹$\mathcal{F}$ |⊆| $\mathcal{F}'$›
by (*smt* (*verit*, *best*) *assms fimage-iff fsubsetD iefac-def*)

show *C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
using ‹*C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})› **.**

show *trail-false-cls* Γ {#*x* ∈# *C*. *x* ≠ *L*#}
using ‹*trail-false-cls* Γ {#*x* ∈# *C*. *x* ≠ *L*#}› **.**

show *linorder-cls.is-least-in-fset*
{|*C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C L*|} *C*
unfolding *linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})
using ‹*C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})› **.**
next
show *clause-could-propagate* Γ *C L*
using ‹*clause-could-propagate* Γ *C L*› **.**
next
fix *D* :: *'f gterm literal multiset*
assume *D* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}) **and** *D* ≠ *C* **and** *clause-could-propagate*
Γ *D L*

have *atm-of L* ≺$_t$ *A*
using *Cons-Some.prems* **by** (*simp add*: *fset-trail-atms*)

hence *L* ≺$_l$ *Pos A*
by (*cases L*) *simp-all*

moreover have *ord-res.is-maximal-lit L D*
using ‹*clause-could-propagate* Γ *D L*› **unfolding** *clause-could-propagate-iff*
by *metis*

ultimately have *D* ≺$_c$ *efac E*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos A*) (*efac E*)›
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*
linorder-lit.multp-if-maximal-less-that-maximal)

hence *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using ‹*D* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})›

```
        unfolding ‹F' = finsert E F›
        using iefac-def by force

      thus C ≺c D
        using C-least ‹D ≠ C› ‹clause-could-propagate Γ D L› by metis
    qed

    have ∀ A1 |∈| trail-atms Γ. A1 ≺t A
      using Cons-Some.prems by (simp add: fset-trail-atms)

    thus trail-annotations-invars N (Uer, F', Γ)
      using Cons-Some.hyps by argo
  qed
 qed
next
 case step-hyps: (resolution Γ F Uer E A D Uer' Γ')

 show ?thesis
 proof (cases eres D E = {#})
  case True
  hence ∄K. ord-res.is-maximal-lit K (eres D E)
    by (simp add: linorder-lit.is-maximal-in-mset-iff)
  hence Γ' = []
    unfolding step-hyps by simp
  thus ?thesis
    unfolding ‹s' = (Uer', F, Γ')›
    using trail-annotations-invars.Nil by metis
 next
  case False

  then obtain K where eres-max-lit: linorder-lit.is-maximal-in-mset (eres D E)
K
    using linorder-lit.ex-maximal-in-mset by metis

  have Γ'-eq: Γ' = dropWhile (λLn. atm-of K ≼t atm-of (fst Ln)) Γ
    unfolding step-hyps(7)
  proof (rule dropWhile-cong)
    show Γ = Γ ..
  next
    fix x :: 'f gterm literal × 'f gterm literal multiset option
    assume x ∈ set Γ
    show (∀K. ord-res.is-maximal-lit K (eres D E) ⟶ atm-of K ≼t atm-of (fst
x)) =
      (atm-of K ≼t atm-of (fst x))
      unfolding case-prod-beta
      using eres-max-lit
      by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset)
  qed
```

have *trail-annotations-invars N* $(U_{er}, \mathcal{F}, \Gamma)$
using *invars*(*1*) **by** (*simp add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma)$›)

moreover have *suffix* $\Gamma' \Gamma$
unfolding *step-hyps*
using *suffix-dropWhile* **by** *metis*

moreover have $\forall$ *Ln* $\in$ *set* Γ'. $\neg$ (*atm-of K* $\preceq_t$ *atm-of* (*fst Ln*))
unfolding Γ'*-eq*
proof (*rule ball-set-dropWhile-if-sorted-wrt-and-monotone-on*)
have *sorted-wrt* (λ*x y. atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using *invars*(*2*)
by (*simp add*: ‹$s = (U_{er}, \mathcal{F}, \Gamma)$› *trail-is-sorted-def sorted-wrt-map*)

thus *sorted-wrt* (λ*x y. fst y* $\prec_l$ *fst x*) Γ
proof (*rule sorted-wrt-mono-rel*[*rotated*])
show $\bigwedge$*x y. atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*) $\Longrightarrow$ *fst y* $\prec_l$ *fst x*
by (*metis* (*no-types*, *lifting*) *linorder-lit.antisym-conv3 linorder-trm.dual-order.asym*
literal.exhaust-sel ord-res.less-lit-simps(*1*) *ord-res.less-lit-simps*(*3*)
ord-res.less-lit-simps(*4*))
qed
next
show *monotone-on* (*set* Γ) (λ*x y. fst y* $\prec_l$ *fst x*) (λ*Ln y. y* $\leq$ *Ln*)
(λ*Ln. atm-of K* $\preceq_t$ *atm-of* (*fst Ln*))
apply (*simp add*: *monotone-on-def*)
by (*smt* (*verit*, *best*) *Neg-atm-of-iff Pos-atm-of-iff linorder-lit.order.asym*
linorder-trm.less-linear linorder-trm.order.strict-trans ord-res.less-lit-simps(*1*)
ord-res.less-lit-simps(*3*) *ord-res.less-lit-simps*(*4*))
qed

ultimately show *?thesis*
unfolding ‹$s' = (U_{er}', \mathcal{F}, \Gamma')$›
proof (*induction* $(U_{er}, \mathcal{F}, \Gamma)$ *arbitrary*: Γ Γ' *rule*: *trail-annotations-invars.induct*)
case *Nil*
thus *?case*
by (*simp add*: *trail-annotations-invars.Nil*)
next
case (*Cons-None* Γ *L*)
thus *?case*
by (*metis insert-iff list.simps*(*15*) *suffix-Cons suffix-order.dual-order.refl*
trail-annotations-invars.Cons-None)
next
case (*Cons-Some C L* Γ)

have
clause-could-propagate Γ *C L* **and**
C-least: $\forall$ *y*$\in$|*iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er}). *y* $\neq$ *C* $\longrightarrow$ *clause-could-propagate* Γ
y L $\longrightarrow$ *C* $\prec_c$ *y*
using *Cons-Some.hyps*(*4*) **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by**

metis+

hence *C-max-lit*: *ord-res.is-maximal-lit L C*
unfolding *clause-could-propagate-def* **by** *argo*

obtain Γ'' **where** $(L, \mathit{Some}\ C) \#\ \Gamma = \Gamma'' @ \Gamma'$
using *Cons-Some.prems* **by** (*auto elim*: *suffixE*)

show *?case*
proof (*cases* Γ'')
case *Nil*
hence $\Gamma' = (L, \mathit{Some}\ C) \#\ \Gamma$
using ‹$(L, \mathit{Some}\ C) \#\ \Gamma = \Gamma'' @ \Gamma'$› **by** *simp*

show *?thesis*
unfolding ‹$\Gamma' = (L, \mathit{Some}\ C) \#\ \Gamma$›
proof (*rule trail-annotations-invars.Cons-Some*)
show *ord-res.is-strictly-maximal-lit L C*
using ‹*ord-res.is-strictly-maximal-lit L C*› **.**

show $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$
using ‹$C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$› ‹$U_{er}' = \mathit{finsert}\ (\mathit{eres}\ D\ E)\ U_{er}$›
by *simp*

show *trail-false-cls* Γ $\{\#K \in\#\ C.\ K \neq L\#\}$
using ‹*trail-false-cls* Γ $\{\#K \in\#\ C.\ K \neq L\#\}$› **.**

show *linorder-cls.is-least-in-fset* $\{|C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}').$ *clause-could-propagate* Γ C $L|\}$ C
unfolding *linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$
using ‹$C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$› **.**
next
show *clause-could-propagate* Γ C L
using *Cons-Some.hyps*(*4*) **unfolding** *linorder-cls.is-least-in-ffilter-iff*
by *metis*
next
fix x :: *'f gterm literal multiset*
assume $x \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$
and $x \neq C$
and *clause-could-propagate* Γ x L

have *linorder-lit.is-maximal-in-mset x L*
using ‹*clause-could-propagate* Γ x L› **unfolding** *clause-could-propagate-def*
by *argo*

moreover have $L \prec_l K$
using ‹$\forall\, Ln \in \mathit{set}\ \Gamma'.\ \neg\ (\mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ (\mathit{fst}\ Ln))$›

```
      unfolding ‹Γ′ = (L, Some C) # Γ›
      apply simp
    by (metis Neg-atm-of-iff linorder-lit.antisym-conv3 linorder-trm.less-linear
        literal.collapse(1) ord-res.less-lit-simps(1) ord-res.less-lit-simps(3)
        ord-res.less-lit-simps(4))

    ultimately have set-mset x ≠ set-mset (eres D E)
      using eres-max-lit
      by (metis linorder-lit.is-maximal-in-mset-iff linorder-lit.neq-iff)

    hence x ≠ iefac F (eres D E)
      using iefac-def by auto

    hence x |∈| iefac F |‘| (N |∪| U_er)
      using ‹x |∈| iefac F |‘| (N |∪| U_er′)›
      unfolding ‹U_er′ = finsert (eres D E) U_er›
      by simp

    then show C ≺_c x
      using C-least ‹x ≠ C› ‹clause-could-propagate Γ x L› by metis
  qed

  show trail-annotations-invars N (U_er′, F, Γ)
    using Cons-Some
    by (simp add: ‹Γ′ = (L, Some C) # Γ›)
 qed
next
  case (Cons a list)
  then show ?thesis
    by (metis Cons-Some.hyps(6) Cons-Some.prems ‹(L, Some C) # Γ = Γ′′
@ Γ′› empty-iff
        list.set(1) list.set-intros(1) self-append-conv2 suffix-Cons)
 qed
qed
qed

definition trail-is-lower-set where
  trail-is-lower-set N s ⟷
    (∀ U_er F Γ. s = (U_er, F, Γ) ⟶
      linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N |∪| U_er)))

lemma atoms-not-in-clause-set-undefined-if-trail-is-sorted-lower-set:
  assumes invar: trail-is-lower-set N (U_er, F, Γ)
  shows ∀ A. A |∉| atms-of-clss (N |∪| U_er) ⟶ A |∉| trail-atms Γ
  using invar[unfolded trail-is-lower-set-def, simplified]
  by (metis Un-iff linorder-trm.is-lower-set-iff sup.absorb2)

lemma ord-res-8-preserves-atoms-in-trail-lower-set:
```

assumes
step: *ord-res-8 N s s′* **and**
invars:
trail-is-lower-set N s
trail-annotations-invars N s
trail-is-sorted N s
shows *trail-is-lower-set N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-8.cases*)
case *step-hyps*: (*decide-neg* $\mathcal{F}$ U_{er} Γ *A* Γ′)

have
A-in: *A* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and**
A-gt: ∀ *x* |∈| *trail-atms* Γ. *x* ≺$_t$ *A* **and**
A-least: ∀ *x* |∈| *atms-of-clss* (*N* |∪| U_{er}). (∀ *w* |∈| *trail-atms* Γ. *w* ≺$_t$ *x*) ⟶ *x* ≠ *A* ⟶ *A* ≺$_t$ *x*
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *simp-all*

have *trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)
using *step-hyps* **by** *simp*

have
inv1: *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ **and**
inv2: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using *invars*(*1*,*3*)
by (*simp-all only*: ‹*s* = (*U*$_{er}$, $\mathcal{F}$, Γ)› *trail-is-lower-set-def trail-is-sorted-def*)

have *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ′
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ› *list.map sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *x*
assume *x* ∈ *set* Γ
hence *atm-of* (*fst x*) |∈| *trail-atms* Γ
by (*auto simp*: *fset-trail-atms*)
hence *atm-of* (*fst x*) ≺$_t$ *atm-of* (*Neg A*)
using *A-gt* **by** *simp*
thus *atm-of* (*fst x*) ≺$_t$ *atm-of* (*fst* (*Neg A*, *None*))
unfolding *comp-def prod.sel* **.**
next
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using *inv1* **.**
qed

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| U_{er}))
unfolding ‹*trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)› *finsert.rep-eq*
proof (*rule linorder-trm.is-lower-set-insertI*)
show *A* |∈| *atms-of-clss* (*N* |∪| U_{er})
using *A-in* **.**

next
show $\forall\, w \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$. $w \prec_t A \longrightarrow w \mid\in\mid$ *trail-atms* Γ
using *A-least inv2*
by (*metis linorder-trm.is-lower-set-iff linorder-trm.not-less-iff-gr-or-eq*)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ)
(*atms-of-clss* $(N \mid\cup\mid U_{er})$)
using *inv2* **.**
qed

ultimately show *?thesis*
by (*simp add*: ‹$s' = (U_{er}, \mathcal{F}, \Gamma')$› *trail-is-lower-set-def*)
next
case *step-hyps*: (*propagate* $\mathcal{F}$ U_{er} Γ A C Γ')

have
A-in: $A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$ **and**
A-gt: $\forall\, x \mid\in\mid$ *trail-atms* Γ. $x \prec_t A$ **and**
A-least: $\forall\, x \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$. $(\forall\, w \mid\in\mid$ *trail-atms* Γ. $w \prec_t x) \longrightarrow x \neq A \longrightarrow A \prec_t x$
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *simp-all*

have *trail-atms* Γ' = *finsert* A (*trail-atms* Γ)
using *step-hyps* **by** *simp*

have
inv1: *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ **and**
inv2: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid\cup\mid U_{er})$)
using *invars*(*1*,*3*)
by (*simp-all only*: ‹$s = (U_{er}, \mathcal{F}, \Gamma)$› *trail-is-lower-set-def trail-is-sorted-def*)

have *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ'
unfolding ‹$\Gamma' = (Pos\ A, Some\ C) \#\ \Gamma$› *list.map sorted-wrt.simps*
proof (*intro conjI ballI*)
fix x
assume $x \in$ *set* Γ
hence *atm-of* (*fst* x) $\mid\in\mid$ *trail-atms* Γ
by (*auto simp*: *fset-trail-atms*)
hence *atm-of* (*fst* x) $\prec_t$ *atm-of* (*Pos* A)
using *A-gt* **by** *simp*
thus *atm-of* (*fst* x) $\prec_t$ *atm-of* (*fst* (*Pos* A, *Some* C))
unfolding *comp-def prod.sel* **.**
next
show *sorted-wrt* $(\lambda x\ y.$ *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ
using *inv1* **.**
qed

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* $(N \mid\cup\mid U_{er})$)

```
    unfolding ‹trail-atms Γ' = finsert A (trail-atms Γ)› finsert.rep-eq
  proof (rule linorder-trm.is-lower-set-insertI)
    show A |∈| atms-of-clss (N |∪| U_er)
      using A-in .
  next
    show ∀ w|∈|atms-of-clss (N |∪| U_er). w ≺_t A ⟶ w |∈| trail-atms Γ
      using A-least inv2
      by (metis linorder-trm.is-lower-set-iff linorder-trm.not-less-iff-gr-or-eq)
  next
    show linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N |∪| U_er))
      using inv2 .
  qed

  ultimately show ?thesis
    by (simp add: ‹s' = (U_er, F, Γ')› trail-is-lower-set-def)
next
  case (factorize F U_er Γ A C F')
  thus ?thesis
    using invars(1) by (simp add: trail-is-lower-set-def fset-trail-atms)
next
  case step-hyps: (resolution Γ F U_er D A C U_er' Γ')

  have suffix Γ' Γ
    using step-hyps suffix-dropWhile by blast

  then obtain Γ'' where Γ = Γ'' @ Γ'
    unfolding suffix-def by metis

  have atms-of-clss (N |∪| U_er') = atms-of-clss (finsert (eres C D) (N |∪| U_er))
    unfolding ‹U_er' = finsert (eres C D) U_er› by simp
  also have ... = atms-of-cls (eres C D) |∪| atms-of-clss (N |∪| U_er)
    unfolding atms-of-clss-finsert ..
  also have ... = atms-of-clss (N |∪| U_er)
  proof −
    have ∀ A |∈| atms-of-cls (eres C D). A |∈| atms-of-cls C ∨ A |∈| atms-of-cls D
      using lit-in-one-of-resolvents-if-in-eres
      by (smt (verit, best) atms-of-cls-def fimage-iff fset-fset-mset)

    moreover have C |∈| iefac F |‘| (N |∪| U_er)
      using invars(2)[unfolded ‹s = (U_er, F, Γ)›] step-hyps(5)
      by (metis propagating-clause-in-clauses)

    moreover have D |∈| iefac F |‘| (N |∪| U_er)
      using linorder-cls.is-least-in-ffilter-iff step-hyps(3) by blast

    ultimately have ∀ A |∈| atms-of-cls (eres C D). A |∈| atms-of-clss (iefac F
|‘| (N |∪| U_er))
      by (metis atms-of-clss-finsert funion-iff mk-disjoint-finsert)
```

hence $\forall A \mid\in\mid$ *atms-of-cls* (*eres C D*). $A \mid\in\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$)
unfolding *atms-of-clss-fimage-iefac* **.**

thus *?thesis*
by *blast*
qed
finally have *atms-of-clss* ($N \mid\cup\mid U_{er}'$) = *atms-of-clss* ($N \mid\cup\mid U_{er}$) **.**

have
inv1: *sorted-wrt* ($\lambda x\ y$. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ **and**
inv2: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* ($N \mid\cup\mid U_{er}$))
using *invars*(*1*,*3*)
by (*simp-all only*: ‹$s = (U_{er}, \mathcal{F}, \Gamma)$› *trail-is-lower-set-def trail-is-sorted-def*)

have *sorted-wrt* ($\lambda x\ y$. $y \prec_t x$) (*map* (*atm-of* ∘ *fst*) Γ)
using *inv1* **by** (*simp add*: *sorted-wrt-map*)

hence *sorted-wrt* ($\lambda x\ y$. $y \prec_t x$) (*map* (*atm-of* ∘ *fst*) $\Gamma'' @$ *map* (*atm-of* ∘ *fst*) Γ')
by (*simp add*: ‹$\Gamma = \Gamma'' @ \Gamma'$›)

moreover have *linorder-trm.is-lower-set* (*set* (*map* (*atm-of* ∘ *fst*) $\Gamma'' @$ *map* (*atm-of* ∘ *fst*) Γ'))
(*fset* (*atms-of-clss* ($N \mid\cup\mid U_{er}$)))
using *inv2* ‹$\Gamma = \Gamma'' @ \Gamma'$›
by (*metis image-comp list.set-map map-append fset-trail-atms*)

ultimately have *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* ($N \mid\cup\mid U_{er}'$))
using *linorder-trm.sorted-and-lower-set-appendD-right*
using ‹*atms-of-clss* ($N \mid\cup\mid U_{er}'$) = *atms-of-clss* ($N \mid\cup\mid U_{er}$)›
by (*metis* (*no-types*, *lifting*) *image-comp list.set-map fset-trail-atms*)

thus *?thesis*
by (*simp add*: ‹$s' = (U_{er}', \mathcal{F}, \Gamma')$› *trail-is-lower-set-def*)
qed

definition *false-cls-is-mempty-or-has-neg-max-lit* **where**
false-cls-is-mempty-or-has-neg-max-lit N s ⟷
($\forall U_{er}\ \mathcal{F}\ \Gamma$. $s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow (\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$.
trail-false-cls $\Gamma\ C \longrightarrow C = \{\#\} \vee (\exists A.$ *linorder-lit.is-maximal-in-mset* C (*Neg A*))))

lemma *ord-res-8-preserves-false-cls-is-mempty-or-has-neg-max-lit*:
assumes
step: *ord-res-8 N s s'* **and**
invars:
false-cls-is-mempty-or-has-neg-max-lit N s
trail-is-lower-set N s

trail-is-sorted N s
shows *false-cls-is-mempty-or-has-neg-max-lit N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-8.cases*)
case *step-hyps*: (*decide-neg* $\mathcal{F}$ U_{er} Γ *A* Γ′)

have Γ-*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using ‹*trail-is-lower-set N s*›[*unfolded trail-is-lower-set-def*,
rule-format, *OF* ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›] .

have Γ-*sorted*: *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using ‹*trail-is-sorted N s*›[*unfolded trail-is-sorted-def*,
rule-format, *OF* ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›] .

have *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms*[*OF* Γ-*sorted*] .

hence *trail-consistent* Γ′
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ›
proof (*rule trail-consistent.Cons* [*rotated*])
show ¬ *trail-defined-lit* Γ (*Neg A*)
unfolding *trail-defined-lit-iff-trail-defined-atm*
using *linorder-trm.is-least-in-fset-ffilterD*(*2*) *linorder-trm.less-irrefl step-hyps*(*4*)
trail-defined-lit-iff-trail-defined-atm **by** *force*
qed

have *atm-defined-iff-lt-A*: *x* |∈| *trail-atms* Γ ⟷ *x* ≺$_t$ *A*
if *x-in*: *x* |∈| *atms-of-clss* (*N* |∪| U_{er}) **for** *x*
proof (*rule iffI*)
assume *x* |∈| *trail-atms* Γ
thus *x* ≺$_t$ *A*
using *step-hyps*(*4*)
unfolding *linorder-trm.is-least-in-ffilter-iff*
by *blast*
next
assume *x* ≺$_t$ *A*
thus *x* |∈| *trail-atms* Γ
using *step-hyps*(*4*)[*unfolded linorder-trm.is-least-in-ffilter-iff*]
using Γ-*lower*[*unfolded linorder-trm.is-lower-set-iff*]
by (*metis linorder-trm.dual-order.asym linorder-trm.neq-iff x-in*)
qed

have ¬ *trail-false-cls* Γ′ *C* **if** *C-in*: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **for** *C*
proof −
have ¬ *trail-false-cls* Γ *C*
using *C-in step-hyps*(*3*) **by** *metis*
hence *trail-true-cls* Γ *C* ∨ ¬ *trail-defined-cls* Γ *C*
using *trail-true-or-false-cls-if-defined* **by** *metis*
thus ¬ *trail-false-cls* Γ′ *C*

proof (*elim disjE*)
assume *trail-true-cls* Γ *C*
hence *trail-true-cls* Γ′ *C*
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ› *trail-true-cls-def*
by (*metis image-insert insert-iff list.set*(*2*) *trail-true-lit-def*)
thus ¬ *trail-false-cls* Γ′ *C*
using ‹*trail-consistent* Γ′›
by (*metis not-trail-true-cls-and-trail-false-cls*)
next
assume ¬ *trail-defined-cls* Γ *C*
then obtain *L* **where** *L-max*: *ord-res.is-maximal-lit L C*
by (*metis* ‹¬ *trail-false-cls* Γ *C*› *linorder-lit.ex-maximal-in-mset trail-false-cls-mempty*)

have *L* ∈# *C*
using *L-max linorder-lit.is-maximal-in-mset-iff* **by** *metis*

have *atm-of L* |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$)
using *C-in* ‹*L* ∈# *C*› **by** (*metis atm-of-in-atms-of-clssI*)

show ¬ *trail-false-cls* Γ′ *C*
proof (*cases atm-of L* = *A*)
case *True*

have ¬ *trail-defined-lit* Γ (*Pos A*)
using *step-hyps*(*4*)
unfolding *trail-defined-lit-iff-trail-defined-atm linorder-trm.is-least-in-ffilter-iff*
by *auto*

hence (∃ *K* ∈# *C*. *K* ≠ *Pos A* ∧ ¬ *trail-false-lit* Γ *K*) ∨
¬ *ord-res.is-maximal-lit* (*Pos A*) *C*
using *step-hyps*(*5*) *C-in*
unfolding *clause-could-propagate-iff*
unfolding *bex-simps de-Morgan-conj not-not* **by** *blast*

thus *?thesis*
proof (*elim disjE bexE conjE*)
fix *K* :: *'f gterm literal*
assume *K* ∈# *C* **and** *K* ≠ *Pos A* **and** ¬ *trail-false-lit* Γ *K*
thus ¬ *trail-false-cls* Γ′ *C*
by (*smt* (*verit*) *fst-conv image-iff insertE list.simps*(*15*) *step-hyps*(*6*)
trail-false-cls-def trail-false-lit-def uminus-Pos uminus-lit-swap)
next
assume ¬ *ord-res.is-maximal-lit* (*Pos A*) *C*

hence *L* = *Neg A*
using ‹*atm-of L* = *A*› *L-max* **by** (*metis literal.exhaust-sel*)

thus ¬ *trail-false-cls* Γ′ *C*
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ›

```
      unfolding trail-false-cls-def trail-false-lit-def
      using ‹L ∈# C›[unfolded ‹L = Neg A›]
        by (metis ‹¬ trail-defined-cls Γ C› fst-conv image-insert insertE
list.simps(15)
          trail-defined-cls-def trail-defined-lit-def uminus-lit-swap uminus-not-id')
    qed
  next
    case False

    moreover have ¬ atm-of L ≺t A
    proof (rule notI)
      assume atm-of L ≺t A
      moreover have ∀ K ∈# C. atm-of K ⪯t atm-of L
        using L-max linorder-lit.is-maximal-in-mset-iff
        by (metis Neg-atm-of-iff linorder-trm.le-less-linear linorder-trm.linear
          literal.collapse(1) ord-res.less-lit-simps(1) ord-res.less-lit-simps(2)
          ord-res.less-lit-simps(3) ord-res.less-lit-simps(4))
      ultimately have ∀ K ∈# C. atm-of K ≺t A
      by (metis linorder-trm.antisym-conv1 linorder-trm.dual-order.strict-trans)
      moreover have ∀ K ∈# C. atm-of K |∈| atms-of-clss (N |∪| Uer)
        using C-in by (metis atm-of-in-atms-of-clssI)
      ultimately have ∀ K ∈# C. atm-of K |∈| trail-atms Γ
        using atm-defined-iff-lt-A by metis
      hence ∀ K ∈# C. trail-defined-lit Γ K
        using trail-defined-lit-iff-trail-defined-atm by metis
      hence trail-defined-cls Γ C
        unfolding trail-defined-cls-def by argo
      thus False
        using ‹¬ trail-defined-cls Γ C› by contradiction
    qed

    ultimately have A ≺t atm-of L
      by order

    hence atm-of L |∉| trail-atms Γ'
      unfolding ‹Γ' = (Neg A, None) # Γ›
      unfolding trail-atms.simps prod.sel literal.sel
      using atm-defined-iff-lt-A[OF ‹atm-of L |∈| atms-of-clss (N |∪| Uer)›]
      using ‹¬ atm-of L ≺t A› by simp

    hence ¬ trail-defined-lit Γ' L
      using trail-defined-lit-iff-trail-defined-atm by blast

    hence ¬ trail-false-lit Γ' L
      using trail-defined-lit-iff-true-or-false by blast

    thus ¬ trail-false-cls Γ' C
      unfolding trail-false-cls-def
      using ‹L ∈# C› by metis
```

```
    qed
  qed
qed

hence ∀ C|∈|iefac F |‘| (N |∪| U_er). trail-false-cls Γ' C ⟶
  C = {#} ∨ (∃ A. ord-res.is-maximal-lit (Neg A) C)
  by metis

thus ?thesis
  unfolding false-cls-is-mempty-or-has-neg-max-lit-def ‹s' = (U_er, F, Γ')› by
simp
next
case step-hyps: (propagate F U_er Γ A C Γ')

have E = {#} ∨ (∃ A. ord-res.is-maximal-lit (Neg A) E)
  if E-in: E |∈| iefac F |‘| (N |∪| U_er) and E-false: trail-false-cls Γ' E for E
proof (cases A ∈ atm-of ‘ set-mset E)
  case True
  have ¬ trail-false-cls Γ E
    using ‹¬ fBex (iefac F |‘| (N |∪| U_er)) (trail-false-cls Γ)› E-in by metis

  hence Neg A ∈# E
    using E-false[unfolded ‹Γ' = (Pos A, Some C) # Γ›]
    by (metis subtrail-falseI uminus-Pos)

  have atm-of L |∈| trail-atms Γ' if L ∈# E for L
    using E-false ‹L ∈# E›
    by (simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set fset-trail-atms
        trail-false-cls-def trail-false-lit-def)

  moreover have x ≺_t A if x |∈| trail-atms Γ for x
    using step-hyps(4) that
    by (simp add: linorder-trm.is-least-in-ffilter-iff)

  ultimately have atm-of L ⪯_t A if L ∈# E for L
    unfolding ‹Γ' = (Pos A, Some C) # Γ› trail-atms.simps prod.sel literal.sel
    using ‹L ∈# E› by blast

  hence L ⪯_l Neg A if L ∈# E for L
    using ‹L ∈# E›
   by (metis linorder-lit.leI linorder-trm.leD literal.collapse(1) literal.collapse(2)
        ord-res.less-lit-simps(3) ord-res.less-lit-simps(4))

  hence ∃ A. ord-res.is-maximal-lit (Neg A) E
    using ‹Neg A ∈# E›
    by (metis ‹¬ trail-false-cls Γ E› linorder-lit.ex-maximal-in-mset
        linorder-lit.is-maximal-in-mset-iff reflclp-iff trail-false-cls-mempty)

  thus ?thesis ..
```

next
case *False*
hence *trail-false-cls* Γ *E*
using *E-false*[*unfolded* ‹Γ′ = (*Pos A*, *Some C*) # Γ›]
by (*metis atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel*(*1*) *sub-trail-falseI*)

moreover have ¬ *trail-false-cls* Γ *E*
using ‹¬ *fBex* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) (*trail-false-cls* Γ)› *E-in* **by** *metis*

ultimately have *False*
by *contradiction*

thus *?thesis* **..**
qed

thus *?thesis*
unfolding *false-cls-is-mempty-or-has-neg-max-lit-def* ‹*s*′ = (U_{er}, $\mathcal{F}$, Γ′)› **by** *simp*
next
case *step-hyps*: (*factorize* $\mathcal{F}$ U_{er} Γ *A C* $\mathcal{F}'$)

have *trail-false-cls* Γ (*iefac* $\mathcal{F}$ *C*) ⟷ *trail-false-cls* Γ *C* **if** *C* |∈| *N* |∪| U_{er} **for** $\mathcal{F}$ *C*
using *that* **by** (*simp add*: *iefac-def trail-false-cls-def*)

hence ∀ *C*|∈|*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}).
trail-false-cls Γ *C* ⟶ *C* = {#} ∨ (∃ *A*. *ord-res.is-maximal-lit* (*Neg A*) *C*)
using *step-hyps*(*3*) **by** *force*

thus *?thesis*
by (*simp add*: ‹*s*′ = (U_{er}, $\mathcal{F}'$, Γ)› *false-cls-is-mempty-or-has-neg-max-lit-def*)
next
case *step-hyps*: (*resolution* Γ $\mathcal{F}$ U_{er} *D A C* U_{er}' Γ′)

have *false-wrt-*Γ*-if-false-wrt-*Γ′: *trail-false-cls* Γ *E* **if** *trail-false-cls* Γ′ *E* **for** *E*
using *that*
unfolding *step-hyps*(*7*) *trail-false-cls-def trail-false-lit-def*
using *image-iff set-dropWhileD* **by** *fastforce*

have *iefac* $\mathcal{F}$ *E* = {#} ∨ (∃ *A*. *ord-res.is-maximal-lit* (*Neg A*) (*iefac* $\mathcal{F}$ *E*))
if *E* |∈| *N* |∪| U_{er}' *trail-false-cls* Γ′ (*iefac* $\mathcal{F}$ *E*) **for** *E*
proof (*cases E* = *eres C D*)
case *True*

show *?thesis*
proof (*cases eres C D* = {#})
case *True*
thus *?thesis*

```
      by (simp add: ‹E = eres C D›)
    next
      case False
      then obtain K where K-max: ord-res.is-maximal-lit K (eres C D)
        using linorder-lit.ex-maximal-in-mset by metis

      have Γ' = dropWhile (λx. atm-of K ≼t atm-of (fst x)) Γ
        unfolding step-hyps(7)
      proof (rule dropWhile-cong)
        show Γ = Γ ..
      next
        fix Ln :: 'f gterm literal × 'f gterm literal multiset option
        obtain L annot where Ln = (L, annot)
          by force
        have (∀ K. ord-res.is-maximal-lit K (eres C D) ⟶ atm-of K ≼t atm-of L)
⟷
          (atm-of K ≼t atm-of L)
          using K-max by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')
        thus (∀ K. ord-res.is-maximal-lit K (eres C D) ⟶ atm-of K ≼t atm-of (fst
Ln)) ⟷
          (atm-of K ≼t atm-of (fst Ln))
          unfolding ‹Ln = (L, annot)› prod.case prod.sel .
      qed

      have K ∈# eres C D
        using K-max linorder-lit.is-maximal-in-mset-iff by metis

      moreover have ¬ trail-defined-lit Γ' K
      proof −
        have sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ
          using invars[unfolded ‹s = (U_er, F, Γ)› trail-is-sorted-def]
          by (simp add: sorted-wrt-map)

        have ∀ Ln ∈ set Γ'. ¬ (atm-of K ≼t atm-of (fst Ln))
          unfolding ‹Γ' = dropWhile (λx. atm-of K ≼t atm-of (fst x)) Γ›
        proof (rule ball-set-dropWhile-if-sorted-wrt-and-monotone-on)
          show sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ
            using invars[unfolded ‹s = (U_er, F, Γ)› trail-is-sorted-def]
            by (simp add: sorted-wrt-map)
        next
          show monotone-on (set Γ) (λx y. atm-of (fst y) ≺t atm-of (fst x)) (≥)
            (λx. atm-of K ≼t atm-of (fst x))
            by (rule monotone-onI) auto
        qed

        hence ∀ Ln ∈ set Γ'. atm-of (fst Ln) ≺t atm-of K
          by auto

        hence atm-of K |∉| trail-atms Γ'
```

```
      by (smt (verit, best) fset-trail-atms image-iff linorder-trm.dual-order.asym)

    thus ?thesis
      using trail-defined-lit-iff-trail-defined-atm by metis
  qed

  ultimately have ¬ trail-false-cls Γ' (eres C D)
    using trail-defined-lit-iff-true-or-false trail-false-cls-def by metis

  hence ¬ trail-false-cls Γ' E
    unfolding ‹E = eres C D› .

  hence ¬ trail-false-cls Γ' (iefac F E)
    unfolding trail-false-cls-def by (metis iefac-def set-mset-efac)

  thus ?thesis
    using ‹trail-false-cls Γ' (iefac F E)›
    by contradiction
 qed
next
 case False
 hence E |∈| N |∪| U_er
  using step-hyps(6) that(1) by force

 moreover hence iefac F E ≠ {#}
  using step-hyps(3−)
  by (metis (no-types, opaque-lifting) empty-iff linorder-cls.is-least-in-ffilter-iff
      linorder-cls.not-less linorder-lit.is-maximal-in-mset-iff mempty-lesseq-cls
rev-fimage-eqI
      set-mset-empty trail-false-cls-mempty)

 moreover have trail-false-cls Γ (iefac F E)
  using ‹trail-false-cls Γ' (iefac F E)› false-wrt-Γ-if-false-wrt-Γ' by metis

 ultimately have ∃ A. ord-res.is-maximal-lit (Neg A) (iefac F E)
 using invars(1)[unfolded ‹s = (U_er, F, Γ)› false-cls-is-mempty-or-has-neg-max-lit-def]
  by auto

 thus ?thesis ..
qed

thus ?thesis
 by (simp add: ‹s' = (U_er', F, Γ')› false-cls-is-mempty-or-has-neg-max-lit-def)
qed


definition decided-literals-all-neg where
 decided-literals-all-neg N s ⟷
  (∀ U_er F Γ. s = (U_er, F, Γ) ⟶
```

$(\forall\, Ln \in set\ \Gamma.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L))$

lemma *ord-res-8-preserves-decided-literals-all-neg*:
assumes
step: *ord-res-8 N s s′* **and**
invar: *decided-literals-all-neg N s*
shows *decided-literals-all-neg N s′*
using *step*
proof (*cases N s s′ rule*: *ord-res-8.cases*)
case (*decide-neg* $\mathcal{F}$ U_{er} Γ A Γ')

have $\forall\, Ln{\in}set\ \Gamma.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
using *invar* **by** (*simp add*: ‹$s = (U_{er},\ \mathcal{F},\ \Gamma)$› *decided-literals-all-neg-def*)

hence $\forall\, Ln{\in}set\ \Gamma'.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
unfolding ‹$\Gamma' = (Neg\ A,\ None)\ \#\ \Gamma$› **by** *simp*

thus *?thesis*
by (*simp add*: ‹$s' = (U_{er},\ \mathcal{F},\ \Gamma')$› *decided-literals-all-neg-def*)
next
case (*propagate* $\mathcal{F}$ U_{er} Γ A C Γ')

have $\forall\, Ln{\in}set\ \Gamma.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
using *invar* **by** (*simp add*: ‹$s = (U_{er},\ \mathcal{F},\ \Gamma)$› *decided-literals-all-neg-def*)

hence $\forall\, Ln{\in}set\ \Gamma'.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
unfolding ‹$\Gamma' = (Pos\ A,\ Some\ C)\ \#\ \Gamma$› **by** *simp*

thus *?thesis*
by (*simp add*: ‹$s' = (U_{er},\ \mathcal{F},\ \Gamma')$› *decided-literals-all-neg-def*)
next
case (*factorize* $\mathcal{F}$ U_{er} Γ A C $\mathcal{F}'$)

have $\forall\, Ln{\in}set\ \Gamma.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
using *invar* **by** (*simp add*: ‹$s = (U_{er},\ \mathcal{F},\ \Gamma)$› *decided-literals-all-neg-def*)

thus *?thesis*
by (*simp add*: ‹$s' = (U_{er},\ \mathcal{F}',\ \Gamma)$› *decided-literals-all-neg-def*)
next
case *step-hyps*: (*resolution* Γ $\mathcal{F}$ U_{er} D A C U_{er}' Γ')

have $\forall\, Ln{\in}set\ \Gamma.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$
using *invar* **by** (*simp add*: ‹$s = (U_{er},\ \mathcal{F},\ \Gamma)$› *decided-literals-all-neg-def*)

moreover have *set* $\Gamma' \subseteq$ *set* Γ
unfolding ‹$\Gamma' = dropWhile$ - Γ›
by (*meson set-mono-suffix suffix-dropWhile*)

ultimately have $\forall\, Ln{\in}set\ \Gamma'.\ \forall\, L.\ Ln = (L,\ None) \longrightarrow is\text{-}neg\ L$

by *blast*

thus *?thesis*
by (*simp add*: ‹$s' = (U_{er}', \mathcal{F}, \Gamma')$› *decided-literals-all-neg-def*)
qed

definition *ord-res-8-invars* **where**
ord-res-8-invars N s ⟷
trail-is-sorted N s ∧
trail-is-lower-set N s ∧
false-cls-is-mempty-or-has-neg-max-lit N s ∧
trail-annotations-invars N s ∧
decided-literals-all-neg N s

lemma *ord-res-8-preserves-invars*:
assumes
step: *ord-res-8 N s s'* **and**
invars: *ord-res-8-invars N s*
shows *ord-res-8-invars N s'*
using *invars*
unfolding *ord-res-8-invars-def*
using
ord-res-8-preserves-trail-is-sorted[*OF step*]
ord-res-8-preserves-atoms-in-trail-lower-set[*OF step*]
ord-res-8-preserves-false-cls-is-mempty-or-has-neg-max-lit[*OF step*]
ord-res-8-preserves-trail-annotations-invars[*OF step*]
ord-res-8-preserves-decided-literals-all-neg[*OF step*]
by *metis*

lemma *rtranclp-ord-res-8-preserves-invars*:
assumes
step: (*ord-res-8 N*)** *s s'* **and**
invars: *ord-res-8-invars N s*
shows *ord-res-8-invars N s'*
using *step invars ord-res-8-preserves-invars*
by (*smt* (*verit*, *del-insts*) *rtranclp-induct*)

lemma *tranclp-ord-res-8-preserves-invars*:
assumes
step: (*ord-res-8 N*)$^{++}$ *s s'* **and**
invars: *ord-res-8-invars N s*
shows *ord-res-8-invars N s'*
using *step invars ord-res-8-preserves-invars*
by (*smt* (*verit*, *del-insts*) *tranclp-induct*)

lemma *ex-ord-res-8-if-not-final*:
assumes
not-final: ¬ *ord-res-8-final* (*N*, *s*) **and**

```
    invars: ord-res-8-invars N s
  shows ∃ s'. ord-res-8 N s s'
proof −
  obtain U_er F Γ where s = (U_er, F, Γ)
    by (metis surj-pair)

  note invars' = invars[unfolded ‹s = (U_er, F, Γ)› ord-res-8-invars-def]

  have
    undef-atm-or-false-cls:
      (∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ) ∧
        ¬ (∃ x |∈| iefac F |‘| (N |∪| U_er). trail-false-cls Γ x)∨
      (∃ x |∈| iefac F |‘| (N |∪| U_er). trail-false-cls Γ x) and
    {#} |∉| iefac F |‘| (N |∪| U_er)
    unfolding atomize-conj
    using not-final[unfolded ord-res-8-final.simps] ‹s = (U_er, F, Γ)› by metis

  show ?thesis
    using undef-atm-or-false-cls
  proof (elim disjE conjE)
    assume
      ex-undef-atm: ∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ and
      no-conflict: ¬ (∃ x|∈|iefac F |‘| (N |∪| U_er). trail-false-cls Γ x)

    moreover have {|A_2 |∈| atms-of-clss (N |∪| U_er). ∀ A_1 |∈| trail-atms Γ. A_1
≺_t A_2|} ≠ {||}
    proof −
      obtain A_2 :: 'f gterm where
        A_2-in: A_2 |∈| atms-of-clss (N |∪| U_er) and
        A_2-undef: A_2 |∉| trail-atms Γ
        using ex-undef-atm by metis

      have A_1 ≺_t A_2 if A_1-in: A_1 |∈| trail-atms Γ for A_1 :: 'f gterm
      proof −
        have A_1 ≠ A_2
          using A_1-in A_2-undef by metis

        moreover have linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N
|∪| U_er))
          using invars'[unfolded trail-is-lower-set-def, simplified] by argo

        ultimately show ?thesis
       by (meson A_2-in A_2-undef linorder-trm.is-lower-set-iff linorder-trm.linorder-cases
that)
      qed

      thus ?thesis
        using A_2-in
        by (smt (verit, ccfv-threshold) femptyE ffmember-filter)
```

qed

ultimately obtain $A :: 'f\ gterm$ **where**
A-least: *linorder-trm.is-least-in-fset* $\{|A_2 \mid\in\mid atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}).$
$\forall A_1 \mid\in\mid trail\text{-}atms\ \Gamma.\ A_1 \prec_t A_2|\}\ A$
using *ex-undef-atm linorder-trm.ex-least-in-fset* **by** *presburger*

show $\exists s'.$ *ord-res-8 N s s'*
proof (*cases* $\exists C|\in|ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos A*))
case *True*
hence $\{|C \mid\in\mid ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos A*)$|\} \neq \{||\}$
by *force*

then obtain C **where**
C-least: *linorder-cls.is-least-in-fset*
$\{|C \mid\in\mid ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos A*)$|\}\ C$
using *linorder-cls.ex1-least-in-fset* **by** *meson*

show *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma)\rangle$
using *ord-res-8.propagate*[*OF no-conflict A-least C-least*]
using *ord-res-8.factorize*[*OF no-conflict A-least C-least*]
by *metis*
next
case *False*
thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma)\rangle$
using *ord-res-8.decide-neg*[*OF no-conflict A-least*] **by** *metis*
qed
next
assume $\exists x|\in|ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *trail-false-cls* $\Gamma\ x$
then obtain $D :: 'f\ gclause$ **where**
D-least: *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*ief\!ac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}))$) D
by (*metis femptyE ffmember-filter linorder-cls.ex-least-in-fset*)

hence $D \mid\in\mid ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and** *trail-false-cls* $\Gamma\ D$
unfolding *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

moreover have $D \neq \{\#\}$
using $\langle D \mid\in\mid ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})\rangle$ $\langle\{\#\} \mid\notin\mid ief\!ac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})\rangle$ **by** *metis*

ultimately obtain A **where** *Neg-A-max*: *linorder-lit.is-maximal-in-mset D* (*Neg A*)
using *invars' false-cls-is-mempty-or-has-neg-max-lit-def* **by** *metis*

hence *trail-false-lit* Γ (*Neg A*)
using ‹*trail-false-cls* Γ *D*›
by (*metis linorder-lit.is-maximal-in-mset-iff trail-false-cls-def*)

hence *Pos A* ∈ *fst* ‘ *set* Γ
unfolding *trail-false-lit-def* **by** *simp*

hence ∃ *C*. (*Pos A*, *Some C*) ∈ *set* Γ
using *invars'*[*unfolded decided-literals-all-neg-def*, *simplified*]
by *fastforce*

then obtain *C* :: *'f gclause* **where**
map-of Γ (*Pos A*) = *Some* (*Some C*)
by (*metis invars' is-pos-def map-of-SomeD not-Some-eq decided-literals-all-neg-def*
weak-map-of-SomeI)

thus ∃ *s'*. *ord-res-8 N s s'*
unfolding ‹*s* = (*U*$_{er}$, *F*, Γ)›
using *ord-res-8.resolution D-least Neg-A-max* **by** *blast*
qed
qed

lemma *ord-res-8-safe-state-if-invars*:
fixes *N s*
assumes *invars*: *ord-res-8-invars N s*
shows *safe-state* (*constant-context ord-res-8*) *ord-res-8-final* (*N*, *s*)
using *safe-state-constant-context-if-invars*[**where**
R = *ord-res-8* **and** *F* = *ord-res-8-final* **and** *I* = *ord-res-8-invars*]
using *ord-res-8-preserves-invars ex-ord-res-8-if-not-final invars* **by** *metis*

end

end
theory *ORD-RES-9*
imports
ORD-RES-8
begin

24 ORD-RES-9 (factorize when propagating)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-9* **where**
decide-neg:
¬ (∃ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). *trail-false-cls* Γ *C*) ⟹
linorder-trm.is-least-in-fset {|*A*$_2$ |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$).
∀ *A*$_1$ |∈| *trail-atms* Γ. *A*$_1$ ≺$_t$ *A*$_2$|} *A* ⟹
¬ (∃ *C* |∈| *iefac F* |‘| (*N* |∪| *U*$_{er}$). *clause-could-propagate* Γ *C* (*Pos A*)) ⟹
Γ' = (*Neg A*, *None*) # Γ ⟹

ord-res-9 N $(U_{er}, \mathcal{F}, \Gamma)$ $(U_{er}, \mathcal{F}, \Gamma')$ |

propagate:
¬ (∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ *C*) ⟹
linorder-trm.is-least-in-fset {|A_2 |∈| *atms-of-clss* (*N* |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A_2$|} *A* ⟹
linorder-cls.is-least-in-fset {|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}).
clause-could-propagate Γ *C* (*Pos A*)|} *C* ⟹
Γ' = (*Pos A*, *Some* (*efac C*)) # Γ ⟹
$\mathcal{F}'$ = (*if linorder-lit.is-greatest-in-mset C* (*Pos A*) *then* $\mathcal{F}$ *else finsert C* $\mathcal{F}$) ⟹
ord-res-9 N $(U_{er}, \mathcal{F}, \Gamma)$ $(U_{er}, \mathcal{F}', \Gamma')$ |

resolution:
linorder-cls.is-least-in-fset {|*D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ *D*|} *D* ⟹
linorder-lit.is-maximal-in-mset D (*Neg A*) ⟹
map-of Γ (*Pos A*) = *Some* (*Some C*) ⟹
U_{er}' = *finsert* (*eres C D*) U_{er} ⟹
Γ' = *dropWhile* (λ*Ln*. ∀ *K*.
linorder-lit.is-maximal-in-mset (*eres C D*) *K* ⟶ *atm-of K* $\preceq_t$ *atm-of* (*fst Ln*)) Γ ⟹
ord-res-9 N $(U_{er}, \mathcal{F}, \Gamma)$ $(U_{er}', \mathcal{F}, \Gamma')$

lemma *right-unique-ord-res-9*:
fixes *N* :: *'f gclause fset*
shows *right-unique* (*ord-res-9 N*)
proof (*rule right-uniqueI*)
fix *x y z*
assume *step1*: *ord-res-9 N x y* **and** *step2*: *ord-res-9 N x z*
show *y* = *z*
using *step1*
proof (*cases N x y rule*: *ord-res-9.cases*)
case *hyps1*: (*decide-neg* $\mathcal{F}$ U_{er} Γ A_1 Γ_1')
show *?thesis*
using *step2* **unfolding** ‹*x* = (U_{er}, $\mathcal{F}$, Γ)›
proof (*cases N* $(U_{er}, \mathcal{F}, \Gamma)$ *z rule*: *ord-res-9.cases*)
case (*decide-neg A* Γ')
with *hyps1* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-trm.dual-order.asym*
linorder-trm.is-least-in-fset-iff)
next
case (*propagate A C* Γ' $\mathcal{F}'$)
with *hyps1* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-ffilter-iff*
linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
thus *?thesis* **..**
next
case (*resolution D A C* U_{er}' Γ')
with *hyps1* **have** *False*

by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
qed
next
case *hyps1*: (*propagate* $\mathcal{F}$ U_{er} Γ A_1 C_1 $\Gamma_1{}'$ $\mathcal{F}_1{}'$)
show *?thesis*
using *step2* **unfolding** ‹$x = (U_{er}, \mathcal{F}, \Gamma)$›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *z rule*: *ord-res-9.cases*)
case (*decide-neg A* Γ')
with *hyps1* **have** *False*
by (*metis* (*no-types*, *lifting*) *Uniq-D linorder-cls.is-least-in-ffilter-iff linorder-trm.Uniq-is-least-in-fset*)
thus *?thesis* **..**
next
case (*propagate A C* Γ' $\mathcal{F}'$)
with *hyps1* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-cls.dual-order.asym linorder-cls.is-least-in-ffilter-iff linorder-trm.dual-order.asym linorder-trm.is-least-in-fset-iff*)
next
case (*resolution D A C* $U_{er}{}'$ Γ')
with *hyps1* **have** *False*
by (*metis* (*no-types*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
qed
next
case *hyps1*: (*resolution* Γ $\mathcal{F}$ U_{er} D_1 A_1 C_1 $U_{er1}{}'$ $\Gamma_1{}'$)
show *?thesis*
using *step2* **unfolding** ‹$x = (U_{er}, \mathcal{F}, \Gamma)$›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *z rule*: *ord-res-9.cases*)
case (*decide-neg A* Γ')
with *hyps1* **have** *False*
by (*metis* (*no-types*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*propagate A C* Γ' $\mathcal{F}'$)
with *hyps1* **have** *False*
by (*metis* (*no-types*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case *hyps2*: (*resolution D A C* $U_{er}{}'$ Γ')
have $D_1 = D$
using *hyps1 hyps2*
by (*metis* (*no-types*) *Uniq-D linorder-cls.Uniq-is-least-in-fset*)
have $C_1 = C$
using *hyps1 hyps2*
unfolding ‹$D_1 = D$›
by (*metis* (*no-types*) *Uniq-D linorder-lit.Uniq-is-maximal-in-mset option.inject uminus-Neg*)

```
      show ?thesis
        using hyps1 hyps2
        unfolding ‹D₁ = D› ‹C₁ = C›
        by argo
    qed
  qed
qed

lemma ord-res-9-is-one-or-two-ord-res-9-steps:
  fixes N s s'
  assumes step: ord-res-9 N s s'
  shows ord-res-8 N s s' ∨ (ord-res-8 N OO ord-res-8 N) s s'
  using step
proof (cases N s s' rule: ord-res-9.cases)
  case step-hyps: (decide-neg F U_er Γ A Γ')
  show ?thesis
  proof (rule disjI1)
    show ord-res-8 N s s'
      using step-hyps ord-res-8.decide-neg by (simp only:)
  qed
next
  case step-hyps: (propagate F U_er Γ A C Γ' F')

  have
    C-in: C |∈| iefac F |`| (N |∪| U_er) and
    C-could-prop: clause-could-propagate Γ C (Pos A) and
    C-lt: ∀ D |∈| iefac F |`| (N |∪| U_er).
        D ≠ C ⟶ clause-could-propagate Γ D (Pos A) ⟶ C ≺_c D
     using step-hyps unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by
argo

  hence C-max-lit: ord-res.is-maximal-lit (Pos A) C
    unfolding clause-could-propagate-def by argo

  show ?thesis
  proof (cases ord-res.is-strictly-maximal-lit (Pos A) C)
    case True

    hence efac C = C
      using nex-strictly-maximal-pos-lit-if-neq-efac by force

    thus ?thesis
      using True step-hyps ord-res-8.propagate by simp
  next
    case False

    have F' = finsert C F
      using False step-hyps by simp
```

have *efac C* $\neq$ *C*
using *False C-max-lit* **by** (*metis greatest-literal-in-efacI literal.disc*(*1*))

hence *C-in'*: *C* $|\in|$ *N* $|\cup|$ U_{er}
using *C-in*
by (*smt* (*verit*, *ccfv-threshold*) *fimage-iff iefac-def ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)

have *C* $|\notin|$ $\mathcal{F}$
by (*smt* (*verit*, *ccfv-threshold*) *C-in* ‹*efac C* $\neq$ *C*› *factorizable-if-neq-efac fimage-iff ex1-efac-eq-factoring-chain iefac-def*)

have *fimage-iefac-*$\mathcal{F}'$*-eq*:
iefac $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ $U_{er}$) = *finsert* (*efac C*) ((*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) − {$|C|$})
proof (*intro fsubset-antisym fsubsetI*)
fix *x* :: *'f gclause*
assume *x* $|\in|$ *iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er})
then obtain *x'* **where** *x'* $|\in|$ *N* $|\cup|$ U_{er} **and** *x* = *iefac* $\mathcal{F}'$ *x'*
by *blast*
thus *x* $|\in|$ *finsert* (*efac C*) ((*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) $|-|$ {$|C|$})
proof (*cases x'* = *C*)
case *True*
hence *x* = *efac C*
using ‹*x* = *iefac* $\mathcal{F}'$ *x'*› **by** (*simp add*: ‹$\mathcal{F}'$ = *finsert C* $\mathcal{F}$› *iefac-def*)
thus *?thesis*
using ‹*efac C* $\neq$ *C*› **by** *simp*
next
case *False*
hence *x* = *iefac* $\mathcal{F}$ *x'*
using ‹*x* = *iefac* $\mathcal{F}'$ *x'*› **by** (*auto simp add*: ‹$\mathcal{F}'$ = *finsert C* $\mathcal{F}$› *iefac-def*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *False* ‹*x'* $|\in|$ *N* $|\cup|$ U_{er}› *ex1-efac-eq-factoring-chain factorizable-if-neq-efac fimage-eqI finsertCI fminus-iff fsingletonE iefac-def*)
qed
next
fix *x* :: *'f gclause*
assume *x-in*: *x* $|\in|$ *finsert* (*efac C*) (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) $|-|$ {$|C|$})
hence *x* = *efac C* ∨ *x* $|\in|$ (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) $|-|$ {$|C|$})
by *blast*
thus *x* $|\in|$ *iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er})
proof (*elim disjE*)
assume *x* = *efac C*
hence *x* = *iefac* $\mathcal{F}'$ *C*
by (*simp add*: ‹$\mathcal{F}'$ = *finsert C* $\mathcal{F}$› *iefac-def*)
thus *x* $|\in|$ *iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er})
using ‹*C* $|\in|$ *N* $|\cup|$ U_{er}› **by** *blast*
next

```
    assume x |∈| iefac F |‘| (N |∪| U_er) |−| {|C|}
    hence x |∈| iefac F |‘| (N |∪| U_er) and x ≠ C
      by simp-all
    then obtain x' where x' |∈| N |∪| U_er and x = iefac F x'
      by auto
    moreover have x' ≠ C
      using ‹x ≠ C› ‹x = iefac F x'›
      using ‹C |∉| F› iefac-def by force
    ultimately show x |∈| iefac F' |‘| (N |∪| U_er)
      by (simp add: ‹F' = finsert C F› iefac-def)
  qed
qed

have first-step8: ord-res-8 N (U_er, F, Γ) (U_er, F', Γ)
  using False step-hyps ord-res-8.factorize by simp

moreover have ord-res-8 N (U_er, F', Γ) (U_er, F', Γ')
proof (rule ord-res-8.propagate)
  have ¬ trail-false-cls Γ C
    using step-hyps using C-in by metis

  hence ¬ trail-false-cls Γ (efac C)
    by (simp add: trail-false-cls-def)

  thus ¬ (∃ C |∈| iefac F' |‘| (N |∪| U_er). trail-false-cls Γ C)
    using step-hyps unfolding fimage-iefac-F'-eq by blast
next
  show linorder-trm.is-least-in-fset {|A_2 |∈| atms-of-clss (N |∪| U_er).
      ∀ A_1|∈|trail-atms Γ. A_1 ≺_t A_2|} A
    using step-hyps by argo
next
  show linorder-cls.is-least-in-fset {|C |∈| iefac F' |‘| (N |∪| U_er).
        clause-could-propagate Γ C (Pos A)|} (efac C)
    unfolding linorder-cls.is-least-in-ffilter-iff
  proof (intro conjI ballI impI)
    show efac C |∈| iefac F' |‘| (N |∪| U_er)
      unfolding fimage-iefac-F'-eq by simp
  next
    show clause-could-propagate Γ (efac C) (Pos A)
      using C-could-prop
      unfolding clause-could-propagate-def
      by (simp add: trail-false-cls-def greatest-literal-in-efacI
          linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
  next
    fix D :: 'f gterm literal multiset
    assume
      D |∈| iefac F' |‘| (N |∪| U_er) and
      D ≠ efac C and
      clause-could-propagate Γ D (Pos A)
```

```
      thus efac C ≺c D
        using C-lt
        by (metis (no-types, opaque-lifting) C-in efac-properties-if-not-ident(1)
            fimage-iefac-F'-eq finsert-fminus finsert-iff linorder-cls.less-trans)
    qed
  next
    show ord-res.is-strictly-maximal-lit (Pos A) (efac C)
      using step-hyps
      by (simp add: clause-could-propagate-def greatest-literal-in-efacI
          linorder-cls.is-least-in-ffilter-iff)
  next
    show Γ' = (Pos A, Some (efac C)) # Γ
      using step-hyps by argo
  qed

  ultimately have (ord-res-8 N OO ord-res-8 N) s s'
    using step-hyps by blast

  thus ?thesis
    by argo
 qed
next
 case step-hyps: (resolution Γ F U_er D A C U_er' Γ')
 show ?thesis
 proof (rule disjI1)
   show ord-res-8 N s s'
     using step-hyps ord-res-8.resolution by (simp only:)
 qed
qed

lemma ord-res-9-preserves-invars:
 assumes
   step: ord-res-9 N s s' and
   invars: ord-res-8-invars N s
 shows ord-res-8-invars N s'
 using invars ord-res-9-is-one-or-two-ord-res-9-steps
 by (metis local.step ord-res-8-preserves-invars relcomppE)

lemma rtranclp-ord-res-9-preserves-ord-res-8-invars:
 assumes
   step: (ord-res-9 N)** s s' and
   invars: ord-res-8-invars N s
 shows ord-res-8-invars N s'
 using step invars ord-res-9-preserves-invars
 by (smt (verit, del-insts) rtranclp-induct)

lemma ex-ord-res-9-if-not-final:
 assumes
   not-final: ¬ ord-res-8-final (N, s) and
```

```
    invars: ord-res-8-invars N s
  shows ∃ s'. ord-res-9 N s s'
proof −
  obtain U_er F Γ where s = (U_er, F, Γ)
    by (metis surj-pair)

  note invars' = invars[unfolded ‹s = (U_er, F, Γ)› ord-res-8-invars-def]

  have
    undef-atm-or-false-cls:
      (∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ) ∧
        ¬ (∃ x |∈| iefac F |`| (N |∪| U_er). trail-false-cls Γ x)∨
      (∃ x |∈| iefac F |`| (N |∪| U_er). trail-false-cls Γ x) and
    {#} |∉| iefac F |`| (N |∪| U_er)
    unfolding atomize-conj
    using not-final[unfolded ord-res-8-final.simps] ‹s = (U_er, F, Γ)› by metis

  show ?thesis
    using undef-atm-or-false-cls
  proof (elim disjE conjE)
    assume
      ex-undef-atm: ∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ and
      no-conflict: ¬ (∃ x|∈|iefac F |`| (N |∪| U_er). trail-false-cls Γ x)

    moreover have {|A_2 |∈| atms-of-clss (N |∪| U_er). ∀ A_1 |∈| trail-atms Γ. A_1
≺_t A_2|} ≠ {||}
    proof −
      obtain A_2 :: 'f gterm where
        A_2-in: A_2 |∈| atms-of-clss (N |∪| U_er) and
        A_2-undef: A_2 |∉| trail-atms Γ
        using ex-undef-atm by metis

      have A_1 ≺_t A_2 if A_1-in: A_1 |∈| trail-atms Γ for A_1 :: 'f gterm
      proof −
        have A_1 ≠ A_2
          using A_1-in A_2-undef by metis

        moreover have linorder-trm.is-lower-fset (trail-atms Γ) (atms-of-clss (N
|∪| U_er))
          using invars'[unfolded trail-is-lower-set-def, simplified] by argo

        ultimately show ?thesis
       by (meson A_2-in A_2-undef linorder-trm.is-lower-set-iff linorder-trm.linorder-cases
that)
      qed

      thus ?thesis
        using A_2-in
        by (smt (verit, ccfv-threshold) femptyE ffmember-filter)
```

qed

ultimately obtain $A :: 'f\ gterm$ **where**
A-least: *linorder-trm.is-least-in-fset* $\{|A_2 \mid\in\mid atms\text{-}of\text{-}clss\ (N \mid\cup\mid U_{er}).$
$\forall A_1 \mid\in\mid trail\text{-}atms\ \Gamma.\ A_1 \prec_t A_2|\}\ A$
using *ex-undef-atm linorder-trm.ex-least-in-fset* **by** *presburger*

show $\exists s'.$ *ord-res-9* $N\ s\ s'$
proof (*cases* $\exists C|\in|$*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos* A))
case *True*
hence $\{|C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos* A)$|\} \neq \{||\}$
by *force*

then obtain C **where**
C-least: *linorder-cls.is-least-in-fset*
$\{|C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *clause-could-propagate* $\Gamma\ C$ (*Pos* A)$|\}\ C$
using *linorder-cls.ex1-least-in-fset* **by** *meson*

show *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma)\rangle$
using *ord-res-9.propagate*[*OF no-conflict A-least C-least*]
by *metis*
next
case *False*
thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma)\rangle$
using *ord-res-9.decide-neg*[*OF no-conflict A-least*] **by** *metis*
qed
next
assume $\exists x|\in|$*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *trail-false-cls* $\Gamma\ x$
then obtain $D :: 'f\ gclause$ **where**
D-least: *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$)) D
by (*metis femptyE ffmember-filter linorder-cls.ex-least-in-fset*)

hence $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and** *trail-false-cls* $\Gamma\ D$
unfolding *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

moreover have $D \neq \{\#\}$
using $\langle D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})\rangle$ $\langle\{\#\} \mid\notin\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})\rangle$ **by** *metis*

ultimately obtain A **where** *Neg-A-max*: *linorder-lit.is-maximal-in-mset* D (*Neg* A)
using *invars' false-cls-is-mempty-or-has-neg-max-lit-def* **by** *metis*

hence *trail-false-lit* Γ (*Neg* A)

```
      using ‹trail-false-cls Γ D›
      by (metis linorder-lit.is-maximal-in-mset-iff trail-false-cls-def)

    hence Pos A ∈ fst ‘ set Γ
      unfolding trail-false-lit-def by simp

    hence ∃ C. (Pos A, Some C) ∈ set Γ
      using invars'[unfolded decided-literals-all-neg-def, simplified]
      by fastforce

    then obtain C :: 'f gclause where
      map-of Γ (Pos A) = Some (Some C)
    by (metis invars' is-pos-def map-of-SomeD not-Some-eq decided-literals-all-neg-def
        weak-map-of-SomeI)

    thus ∃ s'. ord-res-9 N s s'
      unfolding ‹s = (U_er, F, Γ)›
      using ord-res-9.resolution D-least Neg-A-max by blast
  qed
qed

lemma ord-res-9-safe-state-if-invars:
  fixes N s
  assumes invars: ord-res-8-invars N s
  shows safe-state (constant-context ord-res-9) ord-res-8-final (N, s)
  using safe-state-constant-context-if-invars[where
    R = ord-res-9 and F = ord-res-8-final and I = ord-res-8-invars]
  using ord-res-9-preserves-invars ex-ord-res-9-if-not-final invars by metis

sublocale ord-res-9-semantics: semantics where
  step = constant-context ord-res-9 and
  final = ord-res-8-final
proof unfold-locales
  fix S :: 'f gclause fset ×'f gclause fset × 'f gclause fset ×
    ('f gliteral × 'f gclause option) list

  obtain
    N U_er F :: 'f gterm clause fset and
    Γ :: ('f gterm literal × 'f gterm literal multiset option) list where
    S-def: S = (N, U_er, F, Γ)
    by (metis prod.exhaust)

  assume ord-res-8-final S

  hence ∄ x. ord-res-9 N (U_er, F, Γ) x
    unfolding S-def
  proof (cases (N, U_er, F, Γ) rule: ord-res-8-final.cases)
    case model-found
```

have $\nexists A.$ *linorder-trm.is-least-in-fset* $\{|A_2 \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}).$
$\forall A_1 \mid\in\mid$ *trail-atms* $\Gamma.\ A_1 \prec_t A_2|\}\ A$
using ‹$\neg$ $(\exists A|\in|$*atms-of-clss* $(N \mid\cup\mid U_{er}).\ A \mid\notin\mid$ *trail-atms* $\Gamma)$›
using *linorder-trm.is-least-in-ffilter-iff* **by** *fastforce*

moreover have $\nexists C.$ *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| $(N \mid\cup\mid U_{er}))$) C
using ‹$\neg$ *fBex* (*iefac* $\mathcal{F}$ |‘| $(N \mid\cup\mid U_{er})$) (*trail-false-cls* $\Gamma)$›
by (*metis linorder-cls.is-least-in-ffilter-iff*)

ultimately show *?thesis*
unfolding *ord-res-9.simps* **by** *blast*
next
case *contradiction-found*

hence $\exists C \mid\in\mid$ *iefac* $\mathcal{F}$ |‘| $(N \mid\cup\mid U_{er}).$ *trail-false-cls* $\Gamma\ C$
using *trail-false-cls-mempty* **by** *metis*

moreover have $\nexists D\ A.$ *linorder-cls.is-least-in-fset* (*ffilter* (*trail-false-cls* Γ)
(*iefac* $\mathcal{F}$ |‘| $(N \mid\cup\mid U_{er}))$) $D \wedge$ *ord-res.is-maximal-lit* (*Neg* A) D
by (*metis empty-iff linorder-cls.is-least-in-ffilter-iff linorder-cls.leD*
linorder-lit.is-maximal-in-mset-iff local.contradiction-found(*1*) *mempty-lesseq-cls*
set-mset-empty trail-false-cls-mempty)

ultimately show *?thesis*
unfolding *ord-res-9.simps* **by** *blast*
qed

thus *finished* (*constant-context ord-res-9*) S
by (*simp add*: *S-def finished-def constant-context.simps*)
qed

end

end
theory *ORD-RES-10*
imports *ORD-RES-8*
begin

25 ORD-RES-10 (propagate iff a conflict is produced)

type-synonym *'f ord-res-10-state* =
'f gclause fset $\times$*'f gclause fset* $\times$ *'f gclause fset* $\times$ (*'f gliteral* $\times$ *'f gclause option*)
list

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-10* **where**
decide-neg:
$\neg$ ($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ C) $\Longrightarrow$
linorder-trm.is-least-in-fset {|A_2 |$\in$| *atms-of-clss* (N |$\cup$| U_{er}).
$\forall$ A_1 |$\in$| *trail-atms* Γ. A_1 $\prec_t$ A_2|} A $\Longrightarrow$
$\neg$ ($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *clause-could-propagate* Γ C (*Pos* A)) $\Longrightarrow$
Γ' = (*Neg* A, *None*) # Γ $\Longrightarrow$
ord-res-10 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}$, Γ') |

decide-pos:
$\neg$ ($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ C) $\Longrightarrow$
linorder-trm.is-least-in-fset {|A_2 |$\in$| *atms-of-clss* (N |$\cup$| U_{er}).
$\forall$ A_1 |$\in$| *trail-atms* Γ. A_1 $\prec_t$ A_2|} A $\Longrightarrow$
linorder-cls.is-least-in-fset {|C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}).
clause-could-propagate Γ C (*Pos* A)|} C $\Longrightarrow$
Γ' = (*Pos* A, *None*) # Γ $\Longrightarrow$
$\neg$ ($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ' C) $\Longrightarrow$
$\mathcal{F}'$ = (*if linorder-lit.is-greatest-in-mset* C (*Pos* A) *then* $\mathcal{F}$ *else finsert* C $\mathcal{F}$) $\Longrightarrow$
ord-res-10 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}'$, Γ') |

propagate:
$\neg$ ($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ C) $\Longrightarrow$
linorder-trm.is-least-in-fset {|A_2 |$\in$| *atms-of-clss* (N |$\cup$| U_{er}).
$\forall$ A_1 |$\in$| *trail-atms* Γ. A_1 $\prec_t$ A_2|} A $\Longrightarrow$
linorder-cls.is-least-in-fset {|C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}).
clause-could-propagate Γ C (*Pos* A)|} C $\Longrightarrow$
Γ' = (*Pos* A, *Some* (*efac* C)) # Γ $\Longrightarrow$
($\exists$ C |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ' C) $\Longrightarrow$
$\mathcal{F}'$ = (*if linorder-lit.is-greatest-in-mset* C (*Pos* A) *then* $\mathcal{F}$ *else finsert* C $\mathcal{F}$) $\Longrightarrow$
ord-res-10 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}, $\mathcal{F}'$, Γ') |

resolution:
linorder-cls.is-least-in-fset {|D |$\in$| *iefac* $\mathcal{F}$ |`| (N |$\cup$| U_{er}). *trail-false-cls* Γ D|}
D $\Longrightarrow$
linorder-lit.is-maximal-in-mset D (*Neg* A) $\Longrightarrow$
map-of Γ (*Pos* A) = *Some* (*Some* C) $\Longrightarrow$
U_{er}' = *finsert* (*eres* C D) U_{er} $\Longrightarrow$
Γ' = *dropWhile* (λ*Ln*. $\forall$ K.
linorder-lit.is-maximal-in-mset (*eres* C D) K $\longrightarrow$ *atm-of* K $\preceq_t$ *atm-of* (*fst*
Ln)) Γ $\Longrightarrow$
ord-res-10 N (U_{er}, $\mathcal{F}$, Γ) (U_{er}', $\mathcal{F}$, Γ')

lemma *right-unique-ord-res-10*:
fixes N :: *'f gclause fset*
shows *right-unique* (*ord-res-10* N)
proof (*rule right-uniqueI*)
fix x y z
assume *step1*: *ord-res-10* N x y **and** *step2*: *ord-res-10* N x z
show y = z

```
  using step1
proof (cases N x y rule: ord-res-10.cases)
  case hyps1: (decide-neg F U_er Γ A1 Γ1')
  show ?thesis
    using step2 unfolding ‹x = (U_er, F, Γ)›
  proof (cases N (U_er, F, Γ) z rule: ord-res-10.cases)
    case (decide-neg A Γ')
    with hyps1 show ?thesis
      by (metis (no-types, lifting) linorder-trm.dual-order.asym
          linorder-trm.is-least-in-ffilter-iff)
  next
    case (decide-pos A C Γ' F')
    with hyps1 have False
      by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff
          linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (propagate A C Γ' F')
    with hyps1 have False
      by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff
          linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (resolution D A C U_er' Γ')
    with hyps1 have False
      by (metis (no-types, opaque-lifting) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  qed
next
  case hyps1: (decide-pos F U_er Γ A1 C1 Γ1' F1')
  show ?thesis
    using step2 unfolding ‹x = (U_er, F, Γ)›
  proof (cases N (U_er, F, Γ) z rule: ord-res-10.cases)
    case (decide-neg A Γ')
    with hyps1 have False
      by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff
          linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (decide-pos A C Γ' F')
    with hyps1 show ?thesis
      by (metis (no-types, lifting) linorder-cls.Uniq-is-least-in-fset
          linorder-trm.Uniq-is-least-in-fset the1-equality')
  next
    case hyps2: (propagate A C Γ' F')
    have A1 = A
      using hyps1 hyps2
      by (metis (no-types, lifting) linorder-trm.dual-order.asym
          linorder-trm.is-least-in-ffilter-iff)
```

```
    hence trail-false-cls Γ1' = trail-false-cls Γ'
      using hyps1 hyps2
      unfolding trail-false-cls-def trail-false-lit-def
      by simp
    hence False
      using hyps1 hyps2 by argo
    thus ?thesis ..
  next
    case (resolution D A C U_er' Γ')
    with hyps1 have False
      by (meson linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  qed
next
  case hyps1: (propagate F U_er Γ A1 C1 Γ1' F1')
  show ?thesis
    using step2 unfolding ‹x = (U_er, F, Γ)›
  proof (cases N (U_er, F, Γ) z rule: ord-res-10.cases)
    case (decide-neg A Γ')
    with hyps1 have False
      by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff
          linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case hyps2: (decide-pos A C Γ' F')
    have A1 = A
      using hyps1 hyps2
      by (metis (no-types, lifting) linorder-trm.dual-order.asym
          linorder-trm.is-least-in-ffilter-iff)
    hence trail-false-cls Γ1' = trail-false-cls Γ'
      using hyps1 hyps2
      unfolding trail-false-cls-def trail-false-lit-def
      by simp
    hence False
      using hyps1 hyps2 by argo
    thus ?thesis ..
  next
    case (propagate A C Γ' F')
    with hyps1 show ?thesis
      by (metis (no-types, lifting) linorder-cls.Uniq-is-least-in-fset
          linorder-trm.Uniq-is-least-in-fset the1-equality')
  next
    case (resolution D A C U_er' Γ')
    with hyps1 have False
      by (meson linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  qed
next
  case hyps1: (resolution Γ F U_er D1 A1 C1 U_er1' Γ1')
```

```
    show ?thesis
      using step2 unfolding ‹x = (U_er, F, Γ)›
    proof (cases N (U_er, F, Γ) z rule: ord-res-10.cases)
      case (decide-neg A Γ')
      with hyps1 have False
        by (meson linorder-cls.is-least-in-ffilter-iff)
      thus ?thesis ..
    next
      case (decide-pos A C Γ' F')
      with hyps1 have False
        by (meson linorder-cls.is-least-in-ffilter-iff)
      thus ?thesis ..
    next
      case (propagate A C Γ' F')
      with hyps1 have False
        by (meson linorder-cls.is-least-in-ffilter-iff)
      thus ?thesis ..
    next
      case hyps2: (resolution D A C U_er' Γ')
      have D1 = D
        using hyps1 hyps2
       by (metis (no-types, opaque-lifting) linorder-cls.Uniq-is-least-in-fset Uniq-D)
      have A1 = A
        using hyps1 hyps2
        unfolding ‹D1 = D›
      by (metis (mono-tags, opaque-lifting) Uniq-D linorder-lit.Uniq-is-maximal-in-mset
            literal.sel(2))
      have C1 = C
        using hyps1 hyps2
        unfolding ‹A1 = A›
        by simp
      show ?thesis
        using hyps1 hyps2
        unfolding ‹D1 = D› ‹A1 = A› ‹C1 = C›
        by argo
    qed
  qed
qed

sublocale ord-res-10-semantics: semantics where
  step = constant-context ord-res-10 and
  final = ord-res-8-final
proof unfold-locales
  fix S :: 'f ord-res-10-state

  obtain
    N U_er F :: 'f gclause fset and
    Γ :: ('f gliteral × 'f gclause option) list where
    S-def: S = (N, U_er, F, Γ)
```

by (*metis prod.exhaust*)

assume *ord-res-8-final S*

hence ∄ *x*. *ord-res-10 N* (*U*$_{er}$, $\mathcal{F}$, Γ) *x*
unfolding *S-def*
proof (*cases* (*N*, *U*$_{er}$, $\mathcal{F}$, Γ) *rule*: *ord-res-8-final.cases*)
case *model-found*

have ∄ *A*. *linorder-trm.is-least-in-fset* {|*A*$_2$ |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$).
∀ *A*$_1$ |∈| *trail-atms* Γ. *A*$_1$ ≺$_t$ *A*$_2$|} *A*
using ‹¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| *U*$_{er}$). *A* |∉| *trail-atms* Γ)›
using *linorder-trm.is-least-in-ffilter-iff* **by** *fastforce*

moreover have ∄ *C*. *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$))) *C*
using ‹¬ *fBex* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$)) (*trail-false-cls* Γ)›
by (*metis linorder-cls.is-least-in-ffilter-iff*)

ultimately show *?thesis*
unfolding *ord-res-10.simps* **by** *blast*
next
case *contradiction-found*

hence ∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$). *trail-false-cls* Γ *C*
using *trail-false-cls-mempty* **by** *metis*

moreover have ∄ *D A*. *linorder-cls.is-least-in-fset* (*ffilter* (*trail-false-cls* Γ)
(*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$))) *D* ∧ *ord-res.is-maximal-lit* (*Neg A*) *D*
by (*metis empty-iff linorder-cls.is-least-in-ffilter-iff linorder-cls.leD*
linorder-lit.is-maximal-in-mset-iff local.contradiction-found(*1*) *mempty-lesseq-cls*
set-mset-empty trail-false-cls-mempty)

ultimately show *?thesis*
unfolding *ord-res-10.simps* **by** *blast*
qed

thus *finished* (*constant-context ord-res-10*) *S*
by (*simp add*: *S-def finished-def constant-context.simps*)
qed

inductive *ord-res-10-invars* **for** *N* **where**
ord-res-10-invars N (*U*$_{er}$, $\mathcal{F}$, Γ) **if**
sorted-wrt (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ **and**
∀ *Ln* ∈ *set* Γ. ∀ *C*. *snd Ln* = *Some C* ⟶ *linorder-lit.is-greatest-in-mset C* (*fst*
Ln) **and**
linorder-trm.is-lower-fset (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| *U*$_{er}$)) **and**
∀ *Ln* Γ′. Γ = *Ln* # Γ′ ⟶
(*snd Ln* ≠ *None* ⟷ (∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$). *trail-false-cls* Γ *C*))

$\wedge$
(*snd Ln* $\neq$ *None* $\longrightarrow$ *is-pos* (*fst Ln*)) $\wedge$
($\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) $\wedge$
($\forall$ *C*. *snd Ln* = *Some C* $\longrightarrow$ *clause-could-propagate* Γ' *C* (*fst Ln*)) $\wedge$
($\forall$ *x* $\in$ *set* Γ'. *snd x* = *None*) **and**
$\forall \Gamma_1$ *Ln* Γ_0. $\Gamma = \Gamma_1$ @ *Ln* # $\Gamma_0 \longrightarrow$
snd Ln = *None* $\longrightarrow \neg(\exists$ *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}). *trail-false-cls* (*Ln* # Γ_0) *C*)

lemma *ord-res-10-preserves-invars*:
assumes
step: *ord-res-10 N s s'* **and**
invars: *ord-res-10-invars N s*
shows *ord-res-10-invars N s'*
using *invars*
proof (*cases N s rule*: *ord-res-10-invars.cases*)
case *invars*: (*1* Γ U_{er} $\mathcal{F}$)

note Γ-*sorted* = ‹*sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ›
note Γ-*lower* = ‹*linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* $|\cup|$ U_{er}))›

have *trail-consistent* Γ
using Γ-*sorted trail-consistent-if-sorted-wrt-atoms* **by** *metis*

show *?thesis*
using *step* **unfolding** ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *s' rule*: *ord-res-10.cases*)
case *step-hyps*: (*decide-neg A* Γ')

have
A-in: *A* $|\in|$ *atms-of-clss* (*N* $|\cup|$ U_{er}) **and**
A-gt: $\forall$ $A_1|\in|$*trail-atms* Γ. $A_1 \prec_t A$ **and**
A-lt: $\forall$ *y*$|\in|$*atms-of-clss* (*N* $|\cup|$ U_{er}).
y $\neq$ *A* $\longrightarrow$ ($\forall$ $A_1|\in|$*trail-atms* Γ. $A_1 \prec_t y$) $\longrightarrow$ *A* $\prec_t$ *y*
using ‹*linorder-trm.is-least-in-fset* - *A*›
unfolding *atomize-conj linorder-trm.is-least-in-ffilter-iff*
by *argo*

have *trail-atms* Γ' = *finsert A* (*trail-atms* Γ)
unfolding ‹Γ' = (*Neg A*, -) # Γ› **by** *simp*

show *?thesis*
unfolding ‹*s'* = (-, -, -)›
proof (*intro ord-res-10-invars.intros allI impI conjI*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ'
unfolding ‹Γ' = (*Neg A*, *None*) # Γ› *sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *Ln* :: *'f gliteral* $\times$ *'f gclause option*

assume *Ln* ∈ *set* Γ

hence *atm-of* (*fst Ln*) |∈| *trail-atms* Γ
by (*simp add*: *fset-trail-atms*)

thus *atm-of* (*fst Ln*) ≺$_t$ *atm-of* (*fst* (*Neg A*, *None*))
unfolding *prod.sel literal.sel*
using *A-gt* **by** *metis*
next
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**
qed

show ∀ *Ln*∈*set* Γ′. ∀ *C*. *snd Ln* = *Some C* ⟶ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
unfolding ‹Γ′ = (-, *None*) # Γ›
using *invars* **by** *simp*

show *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| U_{er}))
unfolding ‹*trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show *A* |∈| *atms-of-clss* (*N* |∪| U_{er})
using *A-in* **.**
next
fix *w* :: *'f gterm*
assume *w* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and** *w* ≺$_t$ *A*
thus *w* |∈| *trail-atms* Γ
by (*metis A-lt* Γ-*lower linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI*)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using Γ-*lower* **.**
qed

{
fix
Ln :: *'f gliteral* × *'f gclause option* **and**
Γ′′ :: (*'f gliteral* × *'f gclause option*) *list*

assume Γ′ = *Ln* # Γ′′

have *snd Ln* = *None*
using ‹Γ′ = *Ln* # Γ′′› *step-hyps* **by** *auto*

moreover have ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* ((*Neg A*, *None*) # Γ) *C*)
proof (*rule notI* , *elim bexE*)
fix *C* :: *'f gclause*
assume

```
  C-in: C |∈| iefac 𝓕 |‘| (N |∪| U_er) and
  C-false: trail-false-cls ((Neg A, None) # Γ) C

have clause-could-propagate Γ C (Pos A)
  unfolding clause-could-propagate-def
proof (intro conjI)
  show ¬ trail-defined-lit Γ (Pos A)
    unfolding trail-defined-lit-iff-trail-defined-atm literal.sel
    by (metis A-gt linorder-trm.less-irrefl)
next
  show ord-res.is-maximal-lit (Pos A) C
    unfolding linorder-lit.is-maximal-in-mset-iff
  proof (intro conjI ballI impI)
    have ¬ trail-false-cls Γ C
      using step-hyps C-in by metis

    thus Pos A ∈# C
      using C-false by (metis subtrail-falseI uminus-Neg)
  next

    fix L :: 'f gliteral
    assume L-in: L ∈# C and L-neq: L ≠ Pos A

    have trail-false-lit ((Neg A, None) # Γ) L
      using C-false L-in unfolding trail-false-cls-def by metis

    hence − L ∈ fst ‘ set Γ
      unfolding trail-false-lit-def
      using L-neq
      by (cases L) simp-all

    hence trail-defined-lit Γ L
      unfolding trail-defined-lit-def by argo

    hence atm-of L |∈| trail-atms Γ
      unfolding trail-defined-lit-iff-trail-defined-atm .

    moreover have ∀ A₁|∈|trail-atms Γ. A₁ ≺_t A
      using step-hyps unfolding linorder-trm.is-least-in-ffilter-iff by argo

    ultimately have atm-of L ≺_t A
      by metis

    hence L ⪯_l Pos A
      by (cases L) simp-all

    thus ¬ Pos A ≺_l L
      by order
  qed
```

next
show *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *Pos A*#}
using *C-false*
unfolding *trail-false-cls-def trail-false-lit-def*
by (*smt* (*verit*, *ccfv-SIG*) *mem-Collect-eq set-mset-filter subtrail-falseI*
trail-false-cls-def trail-false-lit-def uminus-Neg)
qed

moreover have ¬ *clause-could-propagate* Γ *C* (*Pos A*)
using *C-in step-hyps* **by** *metis*

ultimately show *False*
by *contradiction*
qed

ultimately show (*snd Ln* ≠ *None*) = (∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}).
trail-false-cls Γ′ *C*)
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *argo*

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *argo*

have ¬ *fBex* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) (*trail-false-cls* Γ)
using *step-hyps* **by** *argo*

hence ∀ *x*∈*set* Γ. *snd x* = *None*
using *invars* **by** (*metis list.set-cases*)

thus ∀ *x* ∈ *set* Γ″. *snd x* = *None*
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *simp*

show ⋀*C*. *snd Ln* = *Some C* ⟹ *clause-could-propagate* Γ″ *C* (*fst Ln*)
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *force*

show ⋀*D*. *snd Ln* = *Some D* ⟹ *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *force*
}

have ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* ((*Neg A*, *None*) # Γ) *C*)
proof (*intro notI* , *elim bexE*)
fix *C* :: *'f gclause*
assume
C-in: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
C-false: *trail-false-cls* ((*Neg A*, *None*) # Γ) *C*

have *clause-could-propagate* Γ *C* (*Pos A*)
unfolding *clause-could-propagate-def*
proof (*intro conjI*)
show ¬ *trail-defined-lit* Γ (*Pos A*)

```
      unfolding trail-defined-lit-iff-trail-defined-atm
      by (metis A-gt linorder-trm.less-irrefl literal.sel(1))
    next
      have ¬ trail-false-cls Γ C
        using C-in step-hyps by metis

      thus trail-false-cls Γ {#K ∈# C. K ≠ Pos A#}
        by (smt (verit) C-false mem-Collect-eq set-mset-filter subtrail-falseI
            trail-false-cls-def uminus-Neg)

      show ord-res.is-maximal-lit (Pos A) C
        unfolding linorder-lit.is-maximal-in-mset-iff
      proof (intro conjI ballI impI)
        show Pos A ∈# C
          by (metis C-false ‹¬ trail-false-cls Γ C› subtrail-falseI uminus-Neg)
      next
        fix L
        assume L ∈# C and L ≠ Pos A
        hence atm-of L |∈| trail-atms Γ
          using ‹trail-false-cls Γ {#K ∈# C. K ≠ Pos A#}›
        using trail-defined-lit-iff-trail-defined-atm trail-defined-lit-iff-true-or-false
            trail-false-cls-filter-mset-iff by blast
        hence atm-of L ≺t A
          using A-gt by metis
        hence L ≺l Pos A
          by (cases L) simp-all
        thus ¬ Pos A ≺l L
          by order
      qed
    qed

    moreover have ¬ clause-could-propagate Γ C (Pos A)
      using C-in step-hyps by metis

    ultimately show False
      by contradiction
  qed

  thus ⋀Γ1 Ln Γ0. Γ' = Γ1 @ Ln # Γ0 ⟹ snd Ln = None ⟹
    ¬ (∃ C|∈|iefac F |`| (N |∪| Uer). trail-false-cls (Ln # Γ0) C)
    unfolding ‹Γ' = (-, None) # Γ›
    by (metis suffixI trail-false-cls-if-trail-false-suffix)
qed
next
  case step-hyps: (decide-pos A C Γ' F')

  have
    A-in: A |∈| atms-of-clss (N |∪| Uer) and
    A-gt: ∀ A1|∈|trail-atms Γ. A1 ≺t A and
```

A-lt: $\forall\, y|\in|$*atms-of-clss* (N $|\cup|$ U_{er}).
$y \neq A \longrightarrow (\forall\, A_1|\in|$*trail-atms* Γ. $A_1 \prec_t y) \longrightarrow A \prec_t y$
using ‹*linorder-trm.is-least-in-fset* - A›
unfolding *atomize-conj linorder-trm.is-least-in-ffilter-iff*
by *argo*

have *trail-atms* Γ' = *finsert* A (*trail-atms* Γ)
unfolding ‹Γ' = (*Pos* A, -) # Γ› **by** *simp*

show *?thesis*
unfolding ‹s' = (-, -, -)›
proof (*intro ord-res-10-invars.intros allI impI conjI*)
show *sorted-wrt* ($\lambda x\ y$. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ'
unfolding ‹Γ' = (*Pos* A, *None*) # Γ› *sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *Ln* :: *'f gliteral* × *'f gclause option*
assume *Ln* ∈ *set* Γ

hence *atm-of* (*fst Ln*) $|\in|$ *trail-atms* Γ
by (*simp add*: *fset-trail-atms*)

thus *atm-of* (*fst Ln*) $\prec_t$ *atm-of* (*fst* (*Pos A*, *None*))
unfolding *prod.sel literal.sel*
using *A-gt* **by** *metis*
next
show *sorted-wrt* ($\lambda x\ y$. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ
using Γ-*sorted* **.**
qed

show $\forall$ *Ln*∈*set* Γ'. $\forall$ *C*. *snd Ln* = *Some C* ⟶ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
unfolding ‹Γ' = (-, *None*) # Γ›
using *invars* **by** *simp*

show *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* (N $|\cup|$ U_{er}))
unfolding ‹*trail-atms* Γ' = *finsert* A (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show A $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er})
using *A-in* **.**
next
fix w :: *'f gterm*
assume w $|\in|$ *atms-of-clss* (N $|\cup|$ U_{er}) **and** $w \prec_t A$
thus w $|\in|$ *trail-atms* Γ
by (*metis A-lt* Γ-*lower linorder-trm.dual-order.asym linorder-trm.neq-iff*
linorder-trm.not-in-lower-setI)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $|\cup|$ U_{er}))
using Γ-*lower* **.**
qed

```
{
  fix Ln and Γ''
  assume Γ' = Ln # Γ''

  have snd Ln = None
    using ‹Γ' = Ln # Γ''› step-hyps by auto

 moreover have ¬ (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls ((Pos A,
None) # Γ) C)
  proof (rule notI , elim bexE)
    fix D :: 'f gclause
    assume D-in: D |∈| iefac F' |‘| (N |∪| U_er)

    hence D = efac C ∨ D |∈| iefac F |‘| (N |∪| U_er)
     unfolding ‹F' = (if ord-res.is-strictly-maximal-lit (Pos A) C then F else
finsert C F)›
      by (smt (z3) fimage-iff finsert-iff iefac-def)

    hence ¬ trail-false-cls Γ' D
    proof (elim disjE)
      assume D = efac C

      hence trail-false-cls Γ' D ⟷ trail-false-cls Γ' C
        by (simp add: trail-false-cls-def)

      moreover have ¬ trail-false-cls Γ' C
        using step-hyps unfolding linorder-cls.is-least-in-ffilter-iff by metis

      ultimately show ¬ trail-false-cls Γ' D
        by argo
    next
      assume D |∈| iefac F |‘| (N |∪| U_er)
      thus ¬ trail-false-cls Γ' D
        using step-hyps by metis
    qed

    moreover assume trail-false-cls ((Pos A, None) # Γ) D

    ultimately show False
      unfolding ‹Γ' = (Pos A, None) # Γ› by argo
  qed

  ultimately show snd Ln ≠ None ⟷
    (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls Γ' C)
    unfolding ‹Γ' = (Pos A, None) # Γ› by argo

  show snd Ln ≠ None ⟹ is-pos (fst Ln)
```

```
      using ‹snd Ln = None› by argo

    have ¬ fBex (iefac F |‘| (N |∪| U_er)) (trail-false-cls Γ)
      using step-hyps by argo

    hence ∀ x∈set Γ. snd x = None
      using invars by (metis list.set-cases)

    thus ∀ x∈set Γ''. snd x = None
      using ‹Γ' = Ln # Γ''› ‹Γ' = (Pos A, None) # Γ› by simp

    show ⋀C. snd Ln = Some C ⟹ clause-could-propagate Γ'' C (fst Ln)
      using ‹Γ' = Ln # Γ''› ‹Γ' = (Pos A, None) # Γ› by force

    show ⋀D. snd Ln = Some D ⟹ D |∈| iefac F' |‘| (N |∪| U_er)
      using ‹Γ' = Ln # Γ''› ‹Γ' = (Pos A, None) # Γ› by force
  }

  show ⋀Γ1 Ln Γ0. Γ' = Γ1 @ Ln # Γ0 ⟹ snd Ln = None ⟹
    ¬ (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls (Ln # Γ0) C)
    using step-hyps
    unfolding bex-trail-false-cls-simp
    by (meson suffixI trail-false-cls-if-trail-false-suffix)
qed
next
  case step-hyps: (propagate A C Γ' F')

  have
    A-in: A |∈| atms-of-clss (N |∪| U_er) and
    A-gt: ∀ A1|∈|trail-atms Γ. A1 ≺_t A and
    A-lt: ∀ y|∈|atms-of-clss (N |∪| U_er).
      y ≠ A ⟶ (∀ A1|∈|trail-atms Γ. A1 ≺_t y) ⟶ A ≺_t y
    using ‹linorder-trm.is-least-in-fset - A›
    unfolding atomize-conj linorder-trm.is-least-in-ffilter-iff
    by argo

  have
    C-in: C |∈| iefac F |‘| (N |∪| U_er) and
    C-prop: clause-could-propagate Γ C (Pos A) and
    C-lt: ∀ D |∈| iefac F |‘| (N |∪| U_er).
      D ≠ C ⟶ clause-could-propagate Γ D (Pos A) ⟶ C ≺_c D
    using ‹linorder-cls.is-least-in-fset - C›
    unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

  have trail-atms Γ' = finsert A (trail-atms Γ)
    unfolding ‹Γ' = (Pos A, -) # Γ› by simp

  show ?thesis
    unfolding ‹s' = (-, -, -)›
```

proof (*intro ord-res-10-invars.intros allI impI conjI*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ′
unfolding ‹Γ′ = (*Pos A*, -) # Γ› *sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *Ln* :: *'f gliteral* × *'f gclause option*
assume *Ln* ∈ *set* Γ

hence *atm-of* (*fst Ln*) |∈| *trail-atms* Γ
by (*simp add*: *fset-trail-atms*)

thus *atm-of* (*fst Ln*) ≺$_t$ *atm-of* (*fst* (*Pos A*, *Some* (*efac C*)))
unfolding *prod.sel literal.sel*
using *A-gt* **by** *metis*
next
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**
qed

show ∀ *Ln*∈*set* Γ′. ∀ *C*. *snd Ln* = *Some C* ⟶ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
unfolding ‹Γ′ = (*Pos A*, *Some* (*efac C*)) # Γ›
using *invars step-hyps*
by (*simp add*: *clause-could-propagate-def greatest-literal-in-efacI linorder-cls.is-least-in-ffilter-iff*)

show *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| *U*$_{er}$))
unfolding ‹*trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show *A* |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$)
using *A-in* **.**
next
fix *w* :: *'f gterm*
assume *w* |∈| *atms-of-clss* (*N* |∪| *U*$_{er}$) **and** *w* ≺$_t$ *A*
thus *w* |∈| *trail-atms* Γ
by (*metis A-lt* Γ-*lower linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI*)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| *U*$_{er}$))
using Γ-*lower* **.**
qed

{
fix *Ln* **and** Γ″
assume Γ′ = *Ln* # Γ″
hence *Ln* = (*Pos A*, *Some* (*efac C*)) **and** Γ″ = Γ
using ‹Γ′ = (*Pos A*, *Some* (*efac C*)) # Γ› **by** *simp-all*

obtain *D* **where** *D-in*: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$) **and** *D-false*: *trail-false-cls* Γ′ *D*

```
    using step-hyps by metis

  have (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls Γ' C)
  proof (rule bexI)
    show trail-false-cls Γ' D
      using D-false .
  next
    have ¬ trail-false-cls Γ' C
      by (metis clause-could-propagate-def linorder-cls.is-least-in-ffilter-iff
          linorder-lit.is-maximal-in-mset-iff ord-res.less-lit-simps(2) reflclp-iff
          step-hyps(2,4,5) subtrail-falseI uminus-Pos uminus-not-id')

    hence D ≠ C
      using D-false by metis

    thus D |∈| iefac F' |‘| (N |∪| U_er)
     unfolding ‹F' = (if ord-res.is-strictly-maximal-lit (Pos A) C then F else
finsert C F)›
      using D-in iefac-def by auto
  qed

  moreover have snd Ln ≠ None
    using ‹Γ' = Ln # Γ''› step-hyps by auto

  ultimately show snd Ln ≠ None ⟷
    (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls Γ' C)
    by argo

  show snd Ln ≠ None ⟹ is-pos (fst Ln)
    using ‹Ln = (Pos A, Some (efac C))› by auto

  show ∀ x∈set Γ''. snd x = None
    unfolding ‹Γ'' = Γ›
    using invars by (meson list.set-cases step-hyps(2))

  have clause-could-propagate Γ (efac C) (Pos A)
    using C-prop clause-could-propagate-efac by metis

  thus ⋀C. snd Ln = Some C ⟹ clause-could-propagate Γ'' C (fst Ln)
    using ‹Γ' = Ln # Γ''› ‹Γ' = (Pos A, Some (efac C)) # Γ›
    by force

  have efac C |∈| iefac F' |‘| (N |∪| U_er)
  proof (cases ord-res.is-strictly-maximal-lit (Pos A) C)
    case True
    thus ?thesis
      unfolding ‹F' = -›
      using C-in
      by (metis (mono-tags, opaque-lifting) literal.discI(1)
```

```
        nex-strictly-maximal-pos-lit-if-neq-efac)
    next
      case False
      then show ?thesis
        unfolding ‹F' = -›
        using C-in
      by (smt (z3) fimage-iff finsert-iff iefac-def nex-strictly-maximal-pos-lit-if-neq-efac
          obtains-positive-greatest-lit-if-efac-not-ident)
    qed

    thus ⋀D. snd Ln = Some D ⟹ D |∈| iefac F' |‘| (N |∪| U_er)
      using ‹Γ' = Ln # Γ''› ‹Γ' = (Pos A, Some (efac C)) # Γ› by force
  }

  show ⋀Γ1 Ln Γ0. Γ' = Γ1 @ Ln # Γ0 ⟹ snd Ln = None ⟹
    ¬ (∃ C|∈|iefac F' |‘| (N |∪| U_er). trail-false-cls (Ln # Γ0) C)
    unfolding ‹Γ' = (-, Some -) # Γ›
    using invars
    unfolding bex-trail-false-cls-simp
    by (metis list.inject not-None-eq split-pairs suffix-Cons suffix-def)
 qed
next
 case step-hyps: (resolution D A C U_er' Γ')

 note D-max-lit = ‹ord-res.is-maximal-lit (Neg A) D›
 have C-max-lit: ‹ord-res.is-strictly-maximal-lit (Pos A) C›
   using invars by (metis map-of-SomeD split-pairs step-hyps(4))

 have
   D-in: D |∈| iefac F |‘| (N |∪| U_er) and
   D-false: trail-false-cls Γ D and
   D-lt: ∀ E |∈| iefac F |‘| (N |∪| U_er). E ≠ D ⟶ trail-false-cls Γ E ⟶ D ≺_c
E
   using ‹linorder-cls.is-least-in-fset - D›
   unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

 have A |∈| atms-of-clss (N |∪| U_er)
   using D-in D-max-lit
 by (metis atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff literal.sel(2))

 have ord-res.ground-resolution D C ((D - {#Neg A#}) + (C - {#Pos A#}))
 proof (rule ord-res.ground-resolutionI)
   show D = add-mset (Neg A) (D - {#Neg A#})
     using D-max-lit unfolding linorder-lit.is-maximal-in-mset-iff by simp
 next
   show C = add-mset (Pos A) (C - {#Pos A#})
     using C-max-lit unfolding linorder-lit.is-greatest-in-mset-iff by simp
 next
   show C ≺_c D
```

using *C-max-lit D-max-lit*
by (*simp add*: *clause-lt-clause-if-max-lit-comp*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one)
next
show {#} = {#} ∧ *ord-res.is-maximal-lit* (*Neg A*) *D* ∨ *Neg A* ∈# {#}
using *D-max-lit* **by** *argo*
next
show {#} = {#}
by *argo*
next
show *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
using *C-max-lit* **.**
next
show *remove1-mset* (*Neg A*) *D* + *remove1-mset* (*Pos A*) *C* = (*D* − {#*Neg A*#}) + (*C* − {#*Pos A*#})
..
qed
hence *eres C D* ≠ *D*
unfolding *eres-ident-iff not-not ground-resolution-def* **by** *metis*

obtain Γ$_b$ **where** Γ = (*Pos A*, *Some C*) # Γ$_b$
using ‹*map-of* Γ (*Pos A*) = *Some* (*Some C*)› *invars*
by (*metis list.set-cases map-of-SomeD not-Some-eq snd-conv*)

have *A* |∈| *trail-atms* Γ
unfolding ‹Γ = (*Pos A*, *Some C*) # Γ$_b$› *trail-atms-def* **by** *simp*

moreover have *A* |∉| *trail-atms* Γ′
proof (*cases eres C D* = {#})
case *True*

hence ∄*K*. *ord-res.is-maximal-lit K* (*eres C D*)
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *simp*

hence Γ′ = *dropWhile* (λ*Ln*. *True*) Γ
using *step-hyps*(*6*) **by** *simp*

also have ... = []
by *simp*

finally show *?thesis*
using ‹Γ = (*Pos A*, *Some C*) # Γ$_b$› **by** *simp*
next
case *False*

then obtain *K* **where** *eres-max-lit*: *ord-res.is-maximal-lit K* (*eres C D*)
using *linorder-lit.ex-maximal-in-mset* **by** *presburger*

have *atm-of K* ≺$_t$ *atm-of* (*Neg A*)

proof (*rule lit-in-eres-lt-greatest-lit-in-grestest-resolvant* [*of C D*])
  show *eres C D* ≠ *D*
    using ‹*eres C D* ≠ *D*› **.**
next
  show *ord-res.is-maximal-lit* (*Neg A*) *D*
    using *D-max-lit* **.**
next
  show − *Neg A* ∉# *D*
    using *D-false* ‹*trail-consistent* Γ›
    by (*meson D-max-lit linorder-lit.is-maximal-in-mset-iff*
        not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent)
next
  show *K* ∈# *eres C D*
    using *eres-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *argo*
qed

hence *atm-of K* ≺*t A*
  unfolding *literal.sel* **.**

hence *atm-of K* ⪯*t A*
  by *order*

have Γ′ = *dropWhile* (λ*Ln. atm-of K* ⪯*t atm-of* (*fst Ln*)) Γ
  using *step-hyps*(6) *eres-max-lit*
by (*smt* (*verit*, *ccfv-threshold*) *Uniq-D dropWhile-cong linorder-lit.Uniq-is-maximal-in-mset*)

also have ... = *dropWhile* (λ*Ln. atm-of K* ⪯*t atm-of* (*fst Ln*)) Γ*b*
  unfolding ‹Γ = (*Pos A*, *Some C*) # Γ*b*›
  unfolding *dropWhile.simps prod.sel literal.sel*
  using ‹*atm-of K* ⪯*t A*› **by** *simp*

finally have Γ′ = *dropWhile* (λ*Ln. atm-of K* ⪯*t atm-of* (*fst Ln*)) Γ*b* **.**

hence *trail-atms* Γ′ |⊆| *trail-atms* Γ*b*
  by (*simp only*: *suffix-dropWhile trail-atms-subset-if-suffix*)

moreover have *A* |∉| *trail-atms* Γ*b*
proof (*rule notI*)
  assume *A* |∈| *trail-atms* Γ*b*
  then obtain *Ln* **where** *Ln* ∈ *set* Γ*b* **and** *atm-of* (*fst Ln*) = *A*
    unfolding *fset-trail-atms* **by** *blast*
  moreover have ∀ *y*∈*set* Γ*b*. *atm-of* (*fst y*) ≺*t A*
    using Γ-*sorted*[*unfolded* ‹Γ = (*Pos A*, *Some C*) # Γ*b*›] **by** *simp*
  ultimately have *A* ≺*t A*
    by *metis*
  thus *False*
    by *order*
qed

moreover have *trail-atms* Γ_b $|\subseteq|$ *trail-atms* Γ
proof (*rule trail-atms-subset-if-suffix*)
show *suffix* Γ_b Γ
by (*simp only*: ‹Γ = (*Pos A*, *Some C*) # Γ_b› *suffix-ConsI*)
qed

ultimately show *?thesis*
by *fast*
qed

ultimately have $\Gamma' \neq \Gamma$
by *metis*

have *C-in*: *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *invars*
by (*meson* ‹Γ = (*Pos A*, *Some C*) # Γ_b› *snd-conv*)

define *P* :: *'f gliteral* × *'f gclause option* ⇒ *bool* **where**
P ≡ λ*Ln*. ∀ *K*. *ord-res.is-maximal-lit K* (*eres C D*) ⟶ *atm-of K* ≼$_t$ *atm-of* (*fst Ln*)

have Γ = *takeWhile P* Γ @ Γ'
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* ‹Γ' = *dropWhile* - Γ› **by** *simp*

hence *suffix* Γ' Γ
unfolding *suffix-def* **by** *metis*

show *?thesis*
unfolding ‹*s'* = (-, -, -)›
proof (*intro ord-res-10-invars.intros allI impI conjI*)
show Γ'*-sorted*: *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ'
by (*metis* (*no-types*, *lifting*) Γ*-sorted* ‹*suffix* Γ' Γ› *sorted-wrt-append suffix-def*)

show ∀ *Ln*∈*set* Γ'. ∀ *C*. *snd Ln* = *Some C* ⟶ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
using ‹*suffix* Γ' Γ› *invars*(*3*) *set-mono-suffix* **by** *blast*

have ⋀*xs*. *fset* (*trail-atms xs*) = *atm-of* ‘ *fst* ‘ *set xs*
unfolding *fset-trail-atms* **..**
also have ⋀*xs*. *atm-of* ‘ *fst* ‘ *set xs* = *set* (*map* (*atm-of* ∘ *fst*) *xs*)
by (*simp add*: *image-comp*)
finally have ⋀*xs*. *fset* (*trail-atms xs*) = *set* (*map* (*atm-of* ∘ *fst*) *xs*) **.**

have *atms-of-clss* (*N* $|\cup|$ U_{er}') = *atms-of-cls* (*eres C D*) $|\cup|$ *atms-of-clss* (*N* $|\cup|$ U_{er})
unfolding ‹U_{er}' = *finsert* (*eres C D*) U_{er}› **by** *simp*

also have ... = *atms-of-clss* (*N* |∪| U_{er})
proof −
have *atms-of-cls* (*eres C D*) |⊆| *atms-of-cls C* |∪| *atms-of-cls D*
by (*smt* (*verit*, *best*) *atms-of-cls-def fimage-iff fset-fset-mset fsubsetI funionCI*
lit-in-one-of-resolvents-if-in-eres)

moreover have *atms-of-cls C* |⊆| *atms-of-clss* (*N* |∪| U_{er})
using *C-in*
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb fsubset-funion-eq*)

moreover have *atms-of-cls D* |⊆| *atms-of-clss* (*N* |∪| U_{er})
using *D-in*
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb fsubset-funion-eq*)

ultimately show *?thesis*
by *blast*
qed

finally have *atms-of-clss* (*N* |∪| U_{er}') = *atms-of-clss* (*N* |∪| U_{er}) **.**

show Γ′*-lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| U_{er}'))
unfolding ‹*fset* (*trail-atms* Γ′) = *set* (*map* (*atm-of o fst*) Γ′)›
proof (*rule linorder-trm.sorted-and-lower-set-appendD-right*(*2*))
have *sorted-wrt* (λ*x y*. *y* ≺$_t$ *x*) (*map* (*atm-of* ∘ *fst*) Γ)
using Γ*-sorted* **by** (*simp add*: *sorted-wrt-map*)

thus *sorted-wrt* (λ*x y*. *y* ≺$_t$ *x*) (*map* (*atm-of* ∘ *fst*) (*takeWhile P* Γ) @ *map* (*atm-of* ∘ *fst*) Γ′)
using ‹Γ = *takeWhile P* Γ @ Γ′›
by (*metis map-append*)
next
show *linorder-trm.is-lower-set*
(*set* (*map* (*atm-of* ∘ *fst*) (*takeWhile P* Γ) @ *map* (*atm-of* ∘ *fst*) Γ′))
(*fset* (*atms-of-clss* (*N* |∪| U_{er}')))
unfolding *map-append*[*symmetric*]
unfolding ‹Γ = *takeWhile P* Γ @ Γ′›[*symmetric*]
using Γ*-lower*
unfolding ‹*fset* (*trail-atms* Γ) = *set* (*map* (*atm-of o fst*) Γ)›
unfolding ‹*atms-of-clss* (*N* |∪| U_{er}') = *atms-of-clss* (*N* |∪| U_{er})›
.
qed

have *no-false-cls-in*-Γ′: ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}'). *trail-false-cls* Γ′ *C*)
if *eres-max-lit*: *ord-res.is-maximal-lit K* (*eres C D*)
for *K*

```
    proof −
      have FOO: ⋀Ln.
        (∀ K. ord-res.is-maximal-lit K (eres C D) ⟶ atm-of K ≼t atm-of (fst
Ln)) ⟷
        atm-of K ≼t atm-of (fst Ln)
        using eres-max-lit
        by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')

      have ∀ x ∈ set Γ'. atm-of (fst x) ≺t atm-of K
        using ‹Γ' = dropWhile - Γ›[unfolded FOO] Γ-sorted
      by (metis linorder-trm.not-le mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone
          mono-atms-lt)

      hence ∀ x ∈ set Γ'. atm-of (fst x) ≠ atm-of K
        by fastforce

      hence atm-of K |∉| trail-atms Γ'
        unfolding fset-trail-atms by auto

      hence ¬ trail-defined-lit Γ' K
        unfolding trail-defined-lit-iff-trail-defined-atm .

      hence ¬ trail-false-lit Γ' K
        by (metis trail-defined-lit-iff-true-or-false)

      hence ¬ trail-false-cls Γ' (eres C D)
        using eres-max-lit
        unfolding linorder-lit.is-maximal-in-mset-iff trail-false-cls-def
        by metis

      show ¬ (∃ C|∈|iefac F |‘| (N |∪| U_er'). trail-false-cls Γ' C)
        unfolding ‹U_er' = finsert (eres C D) U_er›
      proof (rule notI , elim bexE)
        fix E :: 'f gclause
        assume
          E-in: E |∈| iefac F |‘| (N |∪| finsert (eres C D) U_er) and
          E-false: trail-false-cls Γ' E

        have E = iefac F (eres C D) ∨ E |∈| iefac F |‘| (N |∪| U_er)
          using E-in by simp

        thus False
        proof (elim disjE)
          assume E = iefac F (eres C D)

          hence trail-false-cls Γ' (eres C D)
            using E-false
            by (simp add: iefac-def trail-false-cls-def)
```

```
      thus False
        using ‹¬ trail-false-cls Γ′ (eres C D)› by contradiction
      next
      assume E |∈| iefac F |‘| (N |∪| U_er)

      moreover have trail-false-cls Γ E
        using E-false ‹suffix Γ′ Γ› by (metis trail-false-cls-if-trail-false-suffix)

      ultimately have D ≼_c E
        using D-lt E-false by fast

      then obtain L where L ∈# E and Neg A ≼_l L
        using D-max-lit
        by (metis empty-iff linorder-cls.leD linorder-lit.is-maximal-in-mset-iff
            linorder-lit.leI ord-res.multp-if-all-left-smaller set-mset-empty)

      hence A ≼_t atm-of L
        by (cases L) simp-all

      moreover have A |∈| atms-of-clss (N |∪| U_er′)
        using ‹A |∈| atms-of-clss (N |∪| U_er)›
        using ‹atms-of-clss (N |∪| U_er′) = atms-of-clss (N |∪| U_er)›
        by (simp only:)

      ultimately have atm-of L |∉| trail-atms Γ′
       using linorder-trm.not-in-lower-setI[OF Γ′-lower] ‹A |∉| trail-atms Γ′›
by blast

      hence ¬ trail-defined-lit Γ′ L
        unfolding trail-defined-lit-iff-trail-defined-atm .

      hence ¬ trail-false-lit Γ′ L
        by (metis trail-defined-lit-iff-true-or-false)

      moreover have trail-false-lit Γ′ L
        using E-false ‹L ∈# E› unfolding trail-false-cls-def by metis

      ultimately show False
        by contradiction
    qed
  qed
qed

have ex-max-lit-in-eres-if: ∃ K. ord-res.is-maximal-lit K (eres C D)
  if Γ′ = Γ_1 @ Ln # Γ_0 for Ln and Γ_1 Γ_0
proof −
  have ∃ x xs. Γ′ = x # xs
    using that neq-Nil-conv by blast
```

hence $\exists x.\ \neg\ (\forall K.\ \textit{ord-res.is-maximal-lit}\ K\ (\textit{eres}\ C\ D) \longrightarrow \textit{atm-of}\ K \preceq_t \textit{atm-of}\ (\textit{fst}\ x))$
unfolding *step-hyps*(6) *dropWhile-eq-Cons-conv* **by** *auto*

thus *?thesis*
by *metis*
qed

{
fix *Ln* **and** Γ''
assume $\Gamma' = \textit{Ln}\ \#\ \Gamma''$

then obtain *K* **where**
eres-max-lit: *ord-res.is-maximal-lit K* (*eres C D*)
using *ex-max-lit-in-eres-if*[*of* [], *simplified*] **by** *metis*

have $\neg\ (\exists\, C|\in|\textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}{}').\ \textit{trail-false-cls}\ \Gamma'\ C)$
using *no-false-cls-in-*Γ' *eres-max-lit* **by** *metis*

moreover have *snd Ln* = *None*
using *invars* ‹$\Gamma' \neq \Gamma$› ‹*suffix* $\Gamma'\ \Gamma$›
by (*metis* ‹$\Gamma = (\textit{Pos}\ A, \textit{Some}\ C)\ \#\ \Gamma_b$› ‹$\Gamma' = \textit{Ln}\ \#\ \Gamma''$› *in-set-conv-decomp suffix-Cons*
suffix-def)

ultimately show *snd Ln* ≠ *None* ⟷ $(\exists\, C|\in|\textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}{}').\ \textit{trail-false-cls}\ \Gamma'\ C)$
by *argo*

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *argo*

show $\forall\, x{\in}\textit{set}\ \Gamma''.\ \textit{snd}\ x = \textit{None}$
using *invars* ‹$\Gamma' \neq \Gamma$› ‹*suffix* $\Gamma'\ \Gamma$›
by (*metis* ‹$\Gamma = (\textit{Pos}\ A, \textit{Some}\ C)\ \#\ \Gamma_b$› ‹$\Gamma' = \textit{Ln}\ \#\ \Gamma''$› *set-mono-suffix subsetD suffix-ConsD2*)

show $\bigwedge C.\ \textit{snd Ln} = \textit{Some}\ C \Longrightarrow \textit{clause-could-propagate}\ \Gamma''\ C\ (\textit{fst Ln})$
by (*simp add*: ‹*snd Ln* = *None*›)

show $\bigwedge E.\ \textit{snd Ln} = \textit{Some}\ E \Longrightarrow E\ |\in|\ \textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}{}')$
by (*simp add*: ‹*snd Ln* = *None*›)
}

show $\bigwedge \Gamma_1\ \textit{Ln}\ \Gamma_0.\ \Gamma' = \Gamma_1\ @\ \textit{Ln}\ \#\ \Gamma_0 \Longrightarrow \textit{snd Ln} = \textit{None} \Longrightarrow$
$\neg\ (\exists\, C|\in|\textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}{}').\ \textit{trail-false-cls}\ (\textit{Ln}\ \#\ \Gamma_0)\ C)$
using *ex-max-lit-in-eres-if no-false-cls-in-*Γ'

```
      by (metis suffixI trail-false-cls-if-trail-false-suffix)
    qed
  qed
qed

lemma rtranclp-ord-res-10-preserves-invars:
  assumes
    step: (ord-res-10 N)** s s' and
    invars: ord-res-10-invars N s
  shows ord-res-10-invars N s'
  using step invars ord-res-10-preserves-invars
  by (smt (verit, del-insts) rtranclp-induct)

lemma ex-ord-res-10-if-not-final:
  assumes
    not-final: ¬ ord-res-8-final (N, s) and
    invars: ord-res-10-invars N s
  shows ∃ s'. ord-res-10 N s s'
  using invars
proof (cases N s rule: ord-res-10-invars.cases)
  case invars': (1 Γ U_er ℱ)

  have
    undef-atm-or-false-cls:
      (∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ) ∧
        ¬ (∃ x |∈| iefac ℱ |‘| (N |∪| U_er). trail-false-cls Γ x)∨
      (∃ x |∈| iefac ℱ |‘| (N |∪| U_er). trail-false-cls Γ x) and
    {#} |∉| iefac ℱ |‘| (N |∪| U_er)
    unfolding atomize-conj
    using not-final[unfolded ord-res-8-final.simps] ‹s = (U_er, ℱ, Γ)› by metis

  show ?thesis
    using undef-atm-or-false-cls
  proof (elim disjE conjE)
    assume
      ex-undef-atm: ∃ x |∈| atms-of-clss (N |∪| U_er). x |∉| trail-atms Γ and
      no-conflict: ¬ (∃ x|∈|iefac ℱ |‘| (N |∪| U_er). trail-false-cls Γ x)

    moreover have {|A_2 |∈| atms-of-clss (N |∪| U_er). ∀ A_1 |∈| trail-atms Γ. A_1
≺_t A_2|} ≠ {||}
    proof −
      obtain A_2 :: 'f gterm where
        A_2-in: A_2 |∈| atms-of-clss (N |∪| U_er) and
        A_2-undef: A_2 |∉| trail-atms Γ
        using ex-undef-atm by metis

      have A_1 ≺_t A_2 if A_1-in: A_1 |∈| trail-atms Γ for A_1 :: 'f gterm
      proof −
        have A_1 ≠ A_2
```

using A_1*-in* A_2*-undef* **by** *metis*

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using *invars'*[*unfolded trail-is-lower-set-def*, *simplified*] **by** *argo*

ultimately show *?thesis*
by (*meson* A_2*-in* A_2*-undef linorder-trm.is-lower-set-iff linorder-trm.linorder-cases that*)
qed

thus *?thesis*
using A_2*-in*
by (*smt* (*verit*, *ccfv-threshold*) *femptyE ffmember-filter*)
qed

ultimately obtain *A* :: *'f gterm* **where**
A-least: *linorder-trm.is-least-in-fset* {|A_2 |∈| *atms-of-clss* (*N* |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. A_1 ≺$_t$ A_2|} *A*
using *ex-undef-atm linorder-trm.ex-least-in-fset* **by** *presburger*

show ∃ *s'*. *ord-res-10 N s s'*
proof (*cases* ∃ *C*|∈|*iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C* (*Pos A*))
case *True*
hence {|*C* |∈| *iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C* (*Pos A*)|} ≠ {||}
by *force*

then obtain *C* **where**
C-least: *linorder-cls.is-least-in-fset*
{|*C* |∈| *iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C* (*Pos A*)|} *C*
using *linorder-cls.ex1-least-in-fset* **by** *meson*

show *?thesis*
proof (*cases* ∃ *D*|∈|*iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}). *trail-false-cls* ((*Pos A*, *Some* (*efac C*)) # Γ) *D*)
case *True*
then show *?thesis*
unfolding ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›
using *ord-res-10.propagate*[*OF no-conflict A-least C-least*]
by *metis*
next
case *False*
hence ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |`| (*N* |∪| U_{er}). *trail-false-cls* ((*Pos A*, *None*) # Γ) *C*)
by (*simp add*: *trail-false-cls-def trail-false-lit-def*)
then show *?thesis*
unfolding ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›

```
      using ord-res-10.decide-pos[OF no-conflict A-least C-least]
      by metis
    qed
  next
    case False
    thus ?thesis
      unfolding ‹s = (U_er, F, Γ)›
      using ord-res-10.decide-neg[OF no-conflict A-least]
      by metis
  qed
next
  assume ∃ x|∈|iefac F |‘| (N |∪| U_er). trail-false-cls Γ x

  then obtain D :: 'f gclause where
    D-least: linorder-cls.is-least-in-fset
      (ffilter (trail-false-cls Γ) (iefac F |‘| (N |∪| U_er))) D
    by (metis femptyE ffmember-filter linorder-cls.ex-least-in-fset)

  hence D-in: D |∈| iefac F |‘| (N |∪| U_er) and D-false: trail-false-cls Γ D
    unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

  have D ≠ {#}
    using D-in ‹{#} |∉| iefac F |‘| (N |∪| U_er)› by metis

  hence Γ ≠ []
    using D-false
    by (auto simp add: trail-false-cls-def trail-false-lit-def)

  then obtain Ln Γ' where Γ = Ln # Γ'
    using neq-Nil-conv by metis

  hence snd Ln ≠ None
   using invars' ‹∃ x|∈|iefac F |‘| (N |∪| U_er). trail-false-cls Γ x› by presburger

  have ∀ x∈set Γ'. snd x = None
    using invars' ‹Γ = Ln # Γ'› by metis

  have ¬(∃ x|∈|iefac F |‘| (N |∪| U_er). trail-false-cls Γ' x)
  proof (cases Γ')
    case Nil
    thus ?thesis
      using ‹{#} |∉| iefac F |‘| (N |∪| U_er)›
      by (simp add: trail-false-cls-def trail-false-lit-def)
  next
    case (Cons x xs)
    hence snd x = None
      using ‹∀ x∈set Γ'. snd x = None› by simp
    then show ?thesis
      using ‹∀Γ_1 Ln Γ_0. Γ = Γ_1 @ Ln # Γ_0 ⟶ snd Ln = None ⟶
```

$\neg$ ($\exists$ *C* |$\in$| -. *trail-false-cls* (*Ln* # Γ_0) *C*)‹*rule-format*, *of* [*Ln*] *x xs*, *simplified*]
using *Cons* ‹Γ = *Ln* # Γ'› **by** *blast*
qed

hence $\neg$ *trail-false-cls* Γ' *D*
using *D-in* **by** *metis*

hence − *fst Ln* $\in$# *D*
using *D-false* ‹Γ = *Ln* # Γ'› **by** (*metis eq-fst-iff subtrail-falseI*)

moreover have *is-pos* (*fst Ln*)
using *invars'* ‹Γ = *Ln* # Γ'› ‹*snd Ln* ≠ *None*› **by** *metis*

moreover have $\forall$ *L* $\in$# *D*. *atm-of L* |$\in$| *trail-atms* Γ
using *D-false*
unfolding *trail-false-cls-def*
using *trail-defined-lit-iff-true-or-false*[*of* Γ]
using *trail-defined-lit-iff-trail-defined-atm*[*of* Γ]
by *metis*

moreover have $\forall$ *x* |$\in$| *trail-atms* Γ'. *x* $\prec_t$ *atm-of* (*fst Ln*)
using ‹*sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ›[
unfolded ‹Γ = *Ln* # Γ'› *sorted-wrt.simps*]
by (*simp add*: *fset-trail-atms*)

ultimately have *linorder-lit.is-maximal-in-mset D* (− *fst Ln*)
unfolding *linorder-lit.is-maximal-in-mset-iff*
by (*smt* (*verit*, *best*) ‹Γ = *Ln* # Γ'› *finsertE ord-res.less-lit-simps*(*4*)
linorder-lit.not-less-iff-gr-or-eq literal.exhaust-sel ord-res.less-lit-simps(*3*)
trail-atms.simps(*2*) *uminus-literal-def*)

moreover obtain *A* **where** *fst Ln* = *Pos A*
using ‹*is-pos* (*fst Ln*)› **by** (*cases fst Ln*) *simp-all*

ultimately have *eres-max-lit*: *ord-res.is-maximal-lit* (*Neg A*) *D*
by *simp*

obtain *C* :: *'f gclause* **where**
map-of Γ (*Pos A*) = *Some* (*Some C*)
unfolding ‹Γ = *Ln* # Γ'›
using ‹*fst Ln* = *Pos A*› ‹*snd Ln* ≠ *None*› **by** *force*

thus $\exists$ *s'*. *ord-res-10 N s s'*
unfolding ‹*s* = (U_{er}, $\mathcal{F}$, Γ)›
using *ord-res-10.resolution*[*OF D-least eres-max-lit*] **by** *blast*
qed
qed

lemma *ord-res-10-safe-state-if-invars*:
fixes N s
assumes *invars*: *ord-res-10-invars* N s
shows *safe-state* (*constant-context ord-res-10*) *ord-res-8-final* (N, s)
using *safe-state-constant-context-if-invars*[**where**
$\mathcal{R}$ = *ord-res-10* **and** $\mathcal{F}$ = *ord-res-8-final* **and** $\mathcal{I}$ = *ord-res-10-invars*]
using *ord-res-10-preserves-invars ex-ord-res-10-if-not-final invars* **by** *metis*

end

end
theory *ORD-RES-11*
imports *ORD-RES-10*
begin

26 ORD-RES-11 (SCL strategy)

type-synonym *'f ord-res-11-state* =
'f gclause fset ×*'f gclause fset* × *'f gclause fset* × (*'f gliteral* × *'f gclause option*) *list* ×
'f gclause option

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

lemma
fixes N U_{er} $\mathcal{F}$ Γ A
assumes
no-false-cls: ¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ C) **and**
A-least: *linorder-trm.is-least-in-fset* {|A_2 |∈| *atms-of-clss* (N |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A_2$|} A **and**
C-least: *linorder-cls.is-least-in-fset* {|C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}).
clause-could-propagate Γ C (*Pos* A)|} C
defines
$\Gamma' \equiv$ (*Pos* A, *None*) # Γ **and**
$\mathcal{F}' \equiv$ (*if linorder-lit.is-greatest-in-mset* C (*Pos* A) *then* $\mathcal{F}$ *else finsert* C $\mathcal{F}$)
shows
(∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ' C) ⟷
(∃ C |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ' C)
by (*meson bex-trail-false-cls-simp*)

inductive *ord-res-11* **where**
decide-neg:
¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ C) ⟹
linorder-trm.is-least-in-fset {|A_2 |∈| *atms-of-clss* (N |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. $A_1 \prec_t A_2$|} A ⟹
¬ (∃ C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *clause-could-propagate* Γ C (*Pos* A)) ⟹
Γ' = (*Neg* A, *None*) # Γ ⟹
ord-res-11 N (U_{er}, $\mathcal{F}$, Γ, *None*) (U_{er}, $\mathcal{F}$, Γ', *None*) |

decide-pos:
$\neg\ (\exists\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma\ C) \Longrightarrow$
$\mathit{linorder\text{-}trm.is\text{-}least\text{-}in\text{-}fset}\ \{|A_2\ |\in|\ \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).$
$\forall\, A_1\ |\in|\ \mathit{trail\text{-}atms}\ \Gamma.\ A_1 \prec_t A_2|\}\ A \Longrightarrow$
$\mathit{linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset}\ \{|C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).$
$\mathit{clause\text{-}could\text{-}propagate}\ \Gamma\ C\ (\mathit{Pos}\ A)|\}\ C \Longrightarrow$
$\Gamma' = (\mathit{Pos}\ A,\ \mathit{None}) \#\ \Gamma \Longrightarrow$
$\neg\ (\exists\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma'\ C) \Longrightarrow$
$\mathcal{F}' = (\mathit{if}\ \mathit{linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset}\ C\ (\mathit{Pos}\ A)\ \mathit{then}\ \mathcal{F}\ \mathit{else}\ \mathit{finsert}\ C\ \mathcal{F}) \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{None})\ (U_{er},\ \mathcal{F}',\ \Gamma',\ \mathit{None})\ |$

propagate:
$\neg\ (\exists\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma\ C) \Longrightarrow$
$\mathit{linorder\text{-}trm.is\text{-}least\text{-}in\text{-}fset}\ \{|A_2\ |\in|\ \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).$
$\forall\, A_1\ |\in|\ \mathit{trail\text{-}atms}\ \Gamma.\ A_1 \prec_t A_2|\}\ A \Longrightarrow$
$\mathit{linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset}\ \{|C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).$
$\mathit{clause\text{-}could\text{-}propagate}\ \Gamma\ C\ (\mathit{Pos}\ A)|\}\ C \Longrightarrow$
$\Gamma' = (\mathit{Pos}\ A,\ \mathit{Some}\ (\mathit{efac}\ C)) \#\ \Gamma \Longrightarrow$
$(\exists\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma'\ C) \Longrightarrow$
$\mathcal{F}' = (\mathit{if}\ \mathit{linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset}\ C\ (\mathit{Pos}\ A)\ \mathit{then}\ \mathcal{F}\ \mathit{else}\ \mathit{finsert}\ C\ \mathcal{F}) \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{None})\ (U_{er},\ \mathcal{F}',\ \Gamma',\ \mathit{None})\ |$

conflict:
$\mathit{linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset}\ \{|D\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma\ D|\}$
$D \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{None})\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{Some}\ D)\ |$

skip: $-\ L \notin\#\ C \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ (L,\ n) \#\ \Gamma,\ \mathit{Some}\ C)\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{Some}\ C)\ |$

resolution:
$\Gamma = (L,\ \mathit{Some}\ D) \#\ \Gamma' \Longrightarrow -\ L \in\#\ C \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{Some}\ C)\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{Some}\ ((C - \{\#-\ L\#\}) + (D$
$-\ \{\#L\#\}))\ |$

backtrack:
$\Gamma = (L,\ \mathit{None}) \#\ \Gamma' \Longrightarrow -\ L \in\#\ C \Longrightarrow$
$\mathit{ord\text{-}res\text{-}11}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathit{Some}\ C)\ (\mathit{finsert}\ C\ U_{er},\ \mathcal{F},\ \Gamma',\ \mathit{None})$

lemma *right-unique-ord-res-11*:
fixes $N :: 'f\ \mathit{gclause}\ \mathit{fset}$
shows *right-unique* (*ord-res-11 N*)
proof (*rule right-uniqueI*)
fix *x y z*
assume *step1*: *ord-res-11 N x y* **and** *step2*: *ord-res-11 N x z*
show $y = z$
using *step1*

proof (*cases N x y rule*: *ord-res-11.cases*)
case *hyps1*: (*decide-neg* $\mathcal{F}$ U_{er} Γ *A1* $\Gamma 1'$)
show *?thesis*
using *step2* **unfolding** ‹*x* = -›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, *None* :: *'f gclause option*) *z rule*: *ord-res-11.cases*)
case (*decide-neg A* Γ')
with *hyps1* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-trm.dual-order.asym linorder-trm.is-least-in-fset-iff*)
next
case (*decide-pos A C* Γ' $\mathcal{F}'$)
with *hyps1* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-ffilter-iff linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*propagate A C* Γ' $\mathcal{F}'$)
with *hyps1* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-ffilter-iff linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*conflict D*)
with *hyps1* **have** *False*
by (*metis* (*no-types*) *linorder-cls.is-least-in-ffilter-iff*)
thus *?thesis* **..**
qed
next
case *hyps1*: (*decide-pos* $\mathcal{F}$ U_{er} Γ *A1 C1* $\Gamma 1'$ $\mathcal{F}1'$)
show *?thesis*
using *step2* **unfolding** ‹*x* = -›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, *None* :: *'f gclause option*) *z rule*: *ord-res-11.cases*)
case (*decide-neg A* Γ')
with *hyps1* **have** *False*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-ffilter-iff linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff*)
thus *?thesis* **..**
next
case (*decide-pos A C* Γ' $\mathcal{F}'$)
with *hyps1* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *linorder-cls.Uniq-is-least-in-fset linorder-trm.Uniq-is-least-in-fset the1-equality'*)
next
case *hyps2*: (*propagate A C* Γ' $\mathcal{F}'$)
have *A1* = *A*
using ‹*linorder-trm.is-least-in-fset* - *A1*› ‹*linorder-trm.is-least-in-fset* - *A*›
by (*metis* (*no-types*) *linorder-trm.Uniq-is-least-in-fset Uniq-D*)
hence *trail-false-cls* $\Gamma 1'$ = *trail-false-cls* Γ'
unfolding ‹$\Gamma 1'$ = - # Γ› ‹Γ' = - # Γ›

```
    unfolding trail-false-cls-def trail-false-lit-def
    by simp
  hence False
    using hyps1 hyps2 by argo
  thus ?thesis ..
 next
  case (conflict D)
  with hyps1 have False
    by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
 qed
next
 case hyps1: (propagate \mathcal{F} U_er \Gamma A1 C1 \Gamma1' \mathcal{F}1')
 show ?thesis
  using step2 unfolding ‹x = -›
 proof (cases N (U_er, \mathcal{F}, \Gamma, None :: 'f gclause option) z rule: ord-res-11.cases)
  case (decide-neg A \Gamma')
  with hyps1 have False
    by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff
        linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff)
  thus ?thesis ..
 next
  case hyps2: (decide-pos A C \Gamma' \mathcal{F}')
  have A1 = A
    using ‹linorder-trm.is-least-in-fset - A1› ‹linorder-trm.is-least-in-fset - A›
    by (metis (no-types) linorder-trm.Uniq-is-least-in-fset Uniq-D)
  hence trail-false-cls \Gamma1' = trail-false-cls \Gamma'
    unfolding ‹\Gamma1' = - # \Gamma› ‹\Gamma' = - # \Gamma›
    unfolding trail-false-cls-def trail-false-lit-def
    by simp
  hence False
    using hyps1 hyps2 by argo
  thus ?thesis ..
 next
  case (propagate A C \Gamma' \mathcal{F}')
  with hyps1 show ?thesis
    by (metis (no-types, lifting) linorder-cls.Uniq-is-least-in-fset
        linorder-trm.Uniq-is-least-in-fset the1-equality')
 next
  case (conflict D)
  with hyps1 have False
    by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
 qed
next
 case hyps1: (conflict \Gamma \mathcal{F} U_er D1)
 show ?thesis
  using step2 unfolding ‹x = -›
 proof (cases N (U_er, \mathcal{F}, \Gamma, None :: 'f gclause option) z rule: ord-res-11.cases)
```

```
    case (decide-neg A Γ')
    with hyps1 have False
      by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (decide-pos A C Γ' F')
    with hyps1 have False
      by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (propagate A C Γ' F')
    with hyps1 have False
      by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
    case (conflict D)
    with hyps1 show ?thesis
      by (metis (no-types) linorder-cls.Uniq-is-least-in-fset Uniq-D)
  qed
next
  case hyps1: (skip L C U_er F n Γ)
  show ?thesis
    using step2 unfolding ‹x = -›
  proof (cases N (U_er, F, (L, n) # Γ, Some C) z rule: ord-res-11.cases)
    case skip
    with hyps1 show ?thesis
      by argo
  next
    case (resolution L' D' Γ')
    with hyps1 have False
      by simp
    thus ?thesis ..
  next
    case (backtrack K Γ')
    with hyps1 have False
      by simp
    thus ?thesis ..
  qed
next
  case hyps1: (resolution Γ L1 D1 Γ1' C U_er F)
  show ?thesis
    using step2 unfolding ‹x = -›
  proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-11.cases)
    case (skip L n Γ)
    with hyps1 have False
      by simp
    thus ?thesis ..
  next
    case (resolution L D Γ')
```

```
      with hyps1 show ?thesis
        by simp
    next
      case (backtrack K Γ')
      with hyps1 have False
        by simp
      thus ?thesis ..
    qed
  next
    case hyps1: (backtrack Γ L Γ' C U_er F)
    show ?thesis
      using step2 unfolding ‹x = -›
    proof (cases N (U_er, F, Γ, Some C) z rule: ord-res-11.cases)
      case (skip L n Γ)
      with hyps1 have False
        by simp
      thus ?thesis ..
    next
      case (resolution L D Γ')
      with hyps1 have False
        by simp
      thus ?thesis ..
    next
      case (backtrack K Γ')
      with hyps1 show ?thesis
        by simp
    qed
  qed
qed


inductive ord-res-11-final :: 'f ord-res-11-state ⇒ bool where
  model-found:
    ¬ (∃ A |∈| atms-of-clss (N |∪| U_er). A |∉| trail-atms Γ) ⟹
    ¬ (∃ C |∈| iefac F |`| (N |∪| U_er). trail-false-cls Γ C) ⟹
    ord-res-11-final (N, U_er, F, Γ, None) |

  contradiction-found:
    ord-res-11-final (N, U_er, F, [], Some {#})

sublocale ord-res-11-semantics: semantics where
  step = constant-context ord-res-11 and
  final = ord-res-11-final
proof unfold-locales
  fix S :: 'f ord-res-11-state

  assume ord-res-11-final S
  thus finished (constant-context ord-res-11) S
  proof (cases S rule: ord-res-11-final.cases)
```

case (*model-found N* U_{er} Γ $\mathcal{F}$)

have ∄ *A*. *linorder-trm.is-least-in-fset* {|A_2 |∈| *atms-of-clss* (*N* |∪| U_{er}).
∀ A_1 |∈| *trail-atms* Γ. A_1 ≺$_t$ A_2|} *A*
using ‹¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* |∉| *trail-atms* Γ)›
unfolding *linorder-trm.is-least-in-ffilter-iff*
by *blast*

moreover have ∄ *D*. *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *D*
using ‹¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ *C*)›
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *metis*

ultimately have ∄*x*. *ord-res-11 N* (U_{er}, $\mathcal{F}$, Γ, *None*) *x*
by (*auto elim*: *ord-res-11.cases*)

thus *?thesis*
by (*simp add*: ‹*S* = -› *finished-def constant-context.simps*)
next
case (*contradiction-found N* U_{er} $\mathcal{F}$)
hence ∄*x*. *ord-res-11 N* (U_{er}, $\mathcal{F}$, [], *Some* {#}) *x*
by (*auto elim*: *ord-res-11.cases*)
thus *?thesis*
by (*simp add*: ‹*S* = -› *finished-def constant-context.simps*)
qed
qed

inductive *ord-res-11-invars* **where**
ord-res-11-invars N (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) **if**
ord-res-10-invars N (U_{er}, $\mathcal{F}$, Γ) **and**
{#} |∈| *N* |∪| U_{er} ⟶ Γ = [] **and**
∀ *C*. $\mathcal{C}$ = *Some C* ⟶ *atms-of-cls C* |⊆| *atms-of-clss* (*N* |∪| U_{er}) **and**
∀ *C*. $\mathcal{C}$ = *Some C* ⟶ *trail-false-cls* Γ *C* **and**
atms-of-clss U_{er} |⊆| *atms-of-clss N* **and**
∀ *C* |∈| $\mathcal{F}$. ∃ *L*. *is-pos L* ∧ *linorder-lit.is-maximal-in-mset C L*

lemma *ord-res-11-invars-initial-state*: *ord-res-11-invars N* ({||}, {||}, [], *None*)
by (*intro ord-res-11-invars.intros ord-res-10-invars.intros*) *simp-all*

lemma *mempty-in-fimage-iefac*[*simp*]: {#} |∈| *iefac* $\mathcal{F}$ |‘| *N* ⟷ {#} |∈| *N*
using *iefac-def* **by** *auto*

lemma *ord-res-11-preserves-invars*:
assumes
step: *ord-res-11 N s s′* **and**
invars: *ord-res-11-invars N s*
shows *ord-res-11-invars N s′*
using *invars*

proof (*cases N s rule*: *ord-res-11-invars.cases*)
case *more-invars*: (*1* U_{er} $\mathcal{F}$ Γ $\mathcal{C}$)
show *?thesis*
using ‹*ord-res-10-invars N* (U_{er}, $\mathcal{F}$, Γ)›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *rule*: *ord-res-10-invars.cases*)
case *invars*: *1*

note Γ-*sorted* = ‹*sorted-wrt* (*λx y. atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ›
note Γ-*lower* = ‹*linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))›

have *trail-consistent* Γ
using Γ-*sorted trail-consistent-if-sorted-wrt-atoms* **by** *metis*

show *?thesis*
using *step* **unfolding** ‹s = (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) s' *rule*: *ord-res-11.cases*)
case *step-hyps*: (*decide-neg A* Γ')

have
A-in: *A* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and**
A-gt: ∀ A_1|∈|*trail-atms* Γ. A_1 $\prec_t$ *A* **and**
A-lt: ∀ *y*|∈|*atms-of-clss* (*N* |∪| U_{er}).
y ≠ *A* ⟶ (∀ A_1|∈|*trail-atms* Γ. A_1 $\prec_t$ *y*) ⟶ *A* $\prec_t$ *y*
using ‹*linorder-trm.is-least-in-fset* - *A*›
unfolding *atomize-conj linorder-trm.is-least-in-ffilter-iff*
by *argo*

have *trail-atms* Γ' = *finsert A* (*trail-atms* Γ)
unfolding ‹Γ' = (*Neg A*, -) # Γ› **by** *simp*

show *?thesis*
unfolding ‹s' = (-, -, -)›
proof (*intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI*)
show *sorted-wrt* (*λx y. atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ'
unfolding ‹Γ' = (*Neg A*, *None*) # Γ› *sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *Ln* :: *'f gliteral* × *'f gclause option*
assume *Ln* ∈ *set* Γ

hence *atm-of* (*fst Ln*) |∈| *trail-atms* Γ
by (*simp add*: *fset-trail-atms*)

thus *atm-of* (*fst Ln*) $\prec_t$ *atm-of* (*fst* (*Neg A*, *None*))
unfolding *prod.sel literal.sel*
using *A-gt* **by** *metis*
next
show *sorted-wrt* (*λx y. atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**

qed

show $\forall Ln{\in}set\ \Gamma'.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit\ (fst\ Ln)\ C$
unfolding ‹$\Gamma' = (\text{-}, None) \# \Gamma$›
using *invars* **by** *simp*

show *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* (N |∪| U_{er}))
unfolding ‹*trail-atms* Γ' = *finsert* A (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show A |∈| *atms-of-clss* (N |∪| U_{er})
using *A-in* **.**
next
fix w :: *'f gterm*
assume w |∈| *atms-of-clss* (N |∪| U_{er}) **and** $w \prec_t A$
thus w |∈| *trail-atms* Γ
by (*metis A-lt* Γ*-lower linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI*)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N |∪| U_{er}))
using Γ*-lower* **.**
qed

{
fix
Ln :: *'f gliteral* × *'f gclause option* **and**
Γ'' :: (*'f gliteral* × *'f gclause option*) *list*

assume $\Gamma' = Ln \# \Gamma''$

have *snd Ln* = *None*
using ‹$\Gamma' = Ln \# \Gamma''$› *step-hyps* **by** *auto*

moreover have ¬ (∃ C|∈|*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). *trail-false-cls* ((*Neg A*, *None*) # Γ) C)
proof (*rule notI* , *elim bexE*)
fix C :: *'f gclause*
assume
C-in: C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and**
C-false: *trail-false-cls* ((*Neg A*, *None*) # Γ) C

have *clause-could-propagate* Γ C (*Pos A*)
unfolding *clause-could-propagate-def*
proof (*intro conjI*)
show ¬ *trail-defined-lit* Γ (*Pos A*)
unfolding *trail-defined-lit-iff-trail-defined-atm literal.sel*
by (*metis A-gt linorder-trm.less-irrefl*)
next
show *ord-res.is-maximal-lit* (*Pos A*) C

```
    unfolding linorder-lit.is-maximal-in-mset-iff
  proof (intro conjI ballI impI)
    have ¬ trail-false-cls Γ C
      using step-hyps C-in by metis

    thus Pos A ∈# C
      using C-false by (metis subtrail-falseI uminus-Neg)
  next

    fix L :: 'f gliteral
    assume L-in: L ∈# C and L-neq: L ≠ Pos A

    have trail-false-lit ((Neg A, None) # Γ) L
      using C-false L-in unfolding trail-false-cls-def by metis

    hence − L ∈ fst ‘ set Γ
      unfolding trail-false-lit-def
      using L-neq
      by (cases L) simp-all

    hence trail-defined-lit Γ L
      unfolding trail-defined-lit-def by argo

    hence atm-of L |∈| trail-atms Γ
      unfolding trail-defined-lit-iff-trail-defined-atm .

    moreover have ∀ A1|∈|trail-atms Γ. A1 ≺t A
     using step-hyps unfolding linorder-trm.is-least-in-ffilter-iff by argo

    ultimately have atm-of L ≺t A
      by metis

    hence L ≼l Pos A
      by (cases L) simp-all

    thus ¬ Pos A ≺l L
      by order
  qed
next
  show trail-false-cls Γ {#K ∈# C. K ≠ Pos A#}
    using C-false
    unfolding trail-false-cls-def trail-false-lit-def
   by (smt (verit, ccfv-SIG) mem-Collect-eq set-mset-filter subtrail-falseI
       trail-false-cls-def trail-false-lit-def uminus-Neg)
qed

moreover have ¬ clause-could-propagate Γ C (Pos A)
  using C-in step-hyps by metis
```

ultimately show *False*
by *contradiction*
qed

ultimately show (*snd Ln* ≠ *None*) = (∃ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ′ *C*)
unfolding ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *argo*

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *argo*

have ¬ *fBex* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) (*trail-false-cls* Γ)
using *step-hyps* **by** *argo*

hence ∀ *x*∈*set* Γ. *snd x* = *None*
using *invars* **by** (*metis list.set-cases*)

thus ∀ *x* ∈ *set* Γ″. *snd x* = *None*
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *simp*

show ⋀*C*. *snd Ln* = *Some C* ⟹ *clause-could-propagate* Γ″ *C* (*fst Ln*)
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *force*

show ⋀*D*. *snd Ln* = *Some D* ⟹ *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using ‹Γ′ = *Ln* # Γ″› ‹Γ′ = (*Neg A*, *None*) # Γ› **by** *force*
}

have ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* ((*Neg A*, *None*) # Γ) *C*)
proof (*intro notI* , *elim bexE*)
fix *C* :: *'f gclause*
assume
C-in: *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
C-false: *trail-false-cls* ((*Neg A*, *None*) # Γ) *C*

have *clause-could-propagate* Γ *C* (*Pos A*)
unfolding *clause-could-propagate-def*
proof (*intro conjI*)
show ¬ *trail-defined-lit* Γ (*Pos A*)
unfolding *trail-defined-lit-iff-trail-defined-atm*
by (*metis A-gt linorder-trm.less-irrefl literal.sel*(*1*))
next
have ¬ *trail-false-cls* Γ *C*
using *C-in step-hyps* **by** *metis*

thus *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *Pos A*#}
by (*smt* (*verit*) *C-false mem-Collect-eq set-mset-filter subtrail-falseI*
trail-false-cls-def uminus-Neg)

show *ord-res.is-maximal-lit* (*Pos A*) *C*
unfolding *linorder-lit.is-maximal-in-mset-iff*
proof (*intro conjI ballI impI*)
show *Pos A* ∈# *C*
by (*metis C-false* ‹¬ *trail-false-cls* Γ *C*› *subtrail-falseI uminus-Neg*)
next
fix *L*
assume *L* ∈# *C* **and** *L* ≠ *Pos A*
hence *atm-of L* |∈| *trail-atms* Γ
using ‹*trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *Pos A*#}›
using *trail-defined-lit-iff-trail-defined-atm trail-defined-lit-iff-true-or-false*
trail-false-cls-filter-mset-iff **by** *blast*
hence *atm-of L* ≺$_t$ *A*
using *A-gt* **by** *metis*
hence *L* ≺$_l$ *Pos A*
by (*cases L*) *simp-all*
thus ¬ *Pos A* ≺$_l$ *L*
by *order*
qed
qed

moreover have ¬ *clause-could-propagate* Γ *C* (*Pos A*)
using *C-in step-hyps* **by** *metis*

ultimately show *False*
by *contradiction*
qed

thus ⋀Γ$_1$ *Ln* Γ$_0$. Γ′ = Γ$_1$ @ *Ln* # Γ$_0$ ⟹ *snd Ln* = *None* ⟹
¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$). *trail-false-cls* (*Ln* # Γ$_0$) *C*)
unfolding ‹Γ′ = (-, *None*) # Γ›
by (*metis suffixI trail-false-cls-if-trail-false-suffix*)

show {#} |∈| *N* |∪| *U*$_{er}$ ⟹ Γ′ = []
using *bex-trail-false-cls-simp more-invars*(3) *step-hyps*(3) *trail-false-cls-mempty*
by *blast*
next
show ⋀*C*. *None* = *Some C* ⟹ *atms-of-cls C* |⊆| *atms-of-clss* (*N* |∪| *U*$_{er}$)
by *simp*
next
show ⋀*C*. *None* = *Some C* ⟹ *trail-false-cls* Γ′ *C*
by *simp*
next
show *atms-of-clss U*$_{er}$ |⊆| *atms-of-clss N*
using *more-invars* **by** *argo*
next
show ∀ *C*|∈|$\mathcal{F}$. ∃ *L*. *is-pos L* ∧ *ord-res.is-maximal-lit L C*
using *more-invars* **by** *argo*
qed

next
case *step-hyps*: (*decide-pos A C* Γ' $\mathcal{F}'$)

have
A-in: *A* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and**
A-gt: ∀ A_1|∈|*trail-atms* Γ. A_1 ≺$_t$ *A* **and**
A-lt: ∀ *y*|∈|*atms-of-clss* (*N* |∪| U_{er}).
y ≠ *A* ⟶ (∀ A_1|∈|*trail-atms* Γ. A_1 ≺$_t$ *y*) ⟶ *A* ≺$_t$ *y*
using ‹*linorder-trm.is-least-in-fset* - *A*›
unfolding *atomize-conj linorder-trm.is-least-in-ffilter-iff*
by *argo*

have *trail-atms* Γ' = *finsert A* (*trail-atms* Γ)
unfolding ‹Γ' = (*Pos A*, -) # Γ› **by** *simp*

show *?thesis*
unfolding ‹*s*′ = (-, -, -)›
proof (*intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI*)
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ'
unfolding ‹Γ' = (*Pos A*, *None*) # Γ› *sorted-wrt.simps*
proof (*intro conjI ballI*)
fix *Ln* :: *'f gliteral* × *'f gclause option*
assume *Ln* ∈ *set* Γ

hence *atm-of* (*fst Ln*) |∈| *trail-atms* Γ
by (*simp add*: *fset-trail-atms*)

thus *atm-of* (*fst Ln*) ≺$_t$ *atm-of* (*fst* (*Pos A*, *None*))
unfolding *prod.sel literal.sel*
using *A-gt* **by** *metis*
next
show *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ
using Γ-*sorted* **.**
qed

show ∀ *Ln*∈*set* Γ'. ∀ *C*. *snd Ln* = *Some C* ⟶ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
unfolding ‹Γ' = (-, *None*) # Γ›
using *invars* **by** *simp*

show *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* (*N* |∪| U_{er}))
unfolding ‹*trail-atms* Γ' = *finsert A* (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show *A* |∈| *atms-of-clss* (*N* |∪| U_{er})
using *A-in* **.**
next
fix *w* :: *'f gterm*
assume *w* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and** *w* ≺$_t$ *A*
thus *w* |∈| *trail-atms* Γ

by (*metis A-lt* Γ*-lower linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI*)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using Γ*-lower* **.**
qed

{
fix *Ln* **and** Γ″
assume Γ′ = *Ln* # Γ″

have *snd Ln* = *None*
using ‹Γ′ = *Ln* # Γ″› *step-hyps* **by** *auto*

moreover have ¬ (∃ *C*|∈|*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *trail-false-cls* ((*Pos A*, *None*) # Γ) *C*)
proof (*rule notI* , *elim bexE*)
fix *D* :: ′*f gclause*
assume *D-in*: *D* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})

hence *D* = *efac C* ∨ *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
unfolding ‹$\mathcal{F}'$ = (*if ord-res.is-strictly-maximal-lit* (*Pos A*) *C then* $\mathcal{F}$ *else finsert C* $\mathcal{F}$)›
by (*smt* (*z3*) *fimage-iff finsert-iff iefac-def*)

hence ¬ *trail-false-cls* Γ′ *D*
proof (*elim disjE*)
assume *D* = *efac C*

hence *trail-false-cls* Γ′ *D* ⟷ *trail-false-cls* Γ′ *C*
by (*simp add*: *trail-false-cls-def*)

moreover have ¬ *trail-false-cls* Γ′ *C*
using *step-hyps* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *metis*

ultimately show ¬ *trail-false-cls* Γ′ *D*
by *argo*
next
assume *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
thus ¬ *trail-false-cls* Γ′ *D*
using *step-hyps* **by** *metis*
qed

moreover assume *trail-false-cls* ((*Pos A*, *None*) # Γ) *D*

ultimately show *False*
unfolding ‹Γ′ = (*Pos A*, *None*) # Γ›
by *contradiction*
qed

ultimately show *snd Ln* $\neq$ *None* $\longleftrightarrow$
($\exists$ *C*$|\in|$*iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er}). *trail-false-cls* Γ' *C*)
unfolding ‹Γ' = (*Pos A*, *None*) # Γ› **by** *argo*

show *snd Ln* $\neq$ *None* $\Longrightarrow$ *is-pos* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *argo*

have $\neg$ *fBex* (*iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})) (*trail-false-cls* Γ)
using *step-hyps* **by** *argo*

hence $\forall$ *x*$\in$*set* Γ. *snd x* = *None*
using *invars* **by** (*metis list.set-cases*)

thus $\forall$ *x*$\in$*set* Γ''. *snd x* = *None*
using ‹Γ' = *Ln* # Γ''› ‹Γ' = (*Pos A*, *None*) # Γ› **by** *simp*

show $\bigwedge$*C*. *snd Ln* = *Some C* $\Longrightarrow$ *clause-could-propagate* Γ'' *C* (*fst Ln*)
using ‹Γ' = *Ln* # Γ''› ‹Γ' = (*Pos A*, *None*) # Γ› **by** *force*

show $\bigwedge$*D*. *snd Ln* = *Some D* $\Longrightarrow$ *D* $|\in|$ *iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er})
using ‹Γ' = *Ln* # Γ''› ‹Γ' = (*Pos A*, *None*) # Γ› **by** *force*
}

show $\bigwedge\Gamma_1$ *Ln* Γ_0. Γ' = Γ_1 @ *Ln* # Γ_0 $\Longrightarrow$ *snd Ln* = *None* $\Longrightarrow$
$\neg$ ($\exists$ *C*$|\in|$*iefac* $\mathcal{F}'$ $|`|$ (*N* $|\cup|$ U_{er}). *trail-false-cls* (*Ln* # Γ_0) *C*)
using *step-hyps*
unfolding *bex-trail-false-cls-simp*
by (*meson suffixI trail-false-cls-if-trail-false-suffix*)

show {#} $|\in|$ *N* $|\cup|$ U_{er} $\Longrightarrow$ Γ' = []
by (*meson bex-trail-false-cls-simp step-hyps*(*3*) *trail-false-cls-mempty*)
next
show $\bigwedge$*C*. *None* = *Some C* $\Longrightarrow$ *atms-of-cls C* $|\subseteq|$ *atms-of-clss* (*N* $|\cup|$ U_{er})
by *simp*
next
show $\bigwedge$*C*. *None* = *Some C* $\Longrightarrow$ *trail-false-cls* Γ' *C*
by *simp*
next
show *atms-of-clss* U_{er} $|\subseteq|$ *atms-of-clss N*
using *more-invars* **by** *argo*
next
have $\exists$ *L*. *is-pos L* $\wedge$ *ord-res.is-maximal-lit L C*
using *step-hyps*
by (*metis* (*no-types*, *lifting*) *clause-could-propagate-def*
linorder-cls.is-least-in-ffilter-iff literal.discI(*1*))

thus $\forall$ *C*$|\in|\mathcal{F}'$. $\exists$ *L*. *is-pos L* $\wedge$ *ord-res.is-maximal-lit L C*
using *more-invars* ‹$\mathcal{F}'$ = (*if - then* $\mathcal{F}$ *else finsert C* $\mathcal{F}$)› **by** *simp*

```
  qed
next
  case step-hyps: (propagate A C Γ' F')

  have
    A-in: A |∈| atms-of-clss (N |∪| U_er) and
    A-gt: ∀ A₁|∈|trail-atms Γ. A₁ ≺_t A and
    A-lt: ∀ y|∈|atms-of-clss (N |∪| U_er).
    y ≠ A ⟶ (∀ A₁|∈|trail-atms Γ. A₁ ≺_t y) ⟶ A ≺_t y
    using ‹linorder-trm.is-least-in-fset - A›
    unfolding atomize-conj linorder-trm.is-least-in-ffilter-iff
    by argo

  have
    C-in: C |∈| iefac F |'| (N |∪| U_er) and
    C-prop: clause-could-propagate Γ C (Pos A) and
    C-lt: ∀ D |∈| iefac F |'| (N |∪| U_er).
    D ≠ C ⟶ clause-could-propagate Γ D (Pos A) ⟶ C ≺_c D
    using ‹linorder-cls.is-least-in-fset - C›
    unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

  have trail-atms Γ' = finsert A (trail-atms Γ)
    unfolding ‹Γ' = (Pos A, -) # Γ› by simp

  show ?thesis
    unfolding ‹s' = (-, -, -)›
  proof (intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI)
    show sorted-wrt (λx y. atm-of (fst y) ≺_t atm-of (fst x)) Γ'
      unfolding ‹Γ' = (Pos A, -) # Γ› sorted-wrt.simps
    proof (intro conjI ballI)
      fix Ln :: 'f gliteral × 'f gclause option
      assume Ln ∈ set Γ

      hence atm-of (fst Ln) |∈| trail-atms Γ
        by (simp add: fset-trail-atms)

      thus atm-of (fst Ln) ≺_t atm-of (fst (Pos A, Some (efac C)))
        unfolding prod.sel literal.sel
        using A-gt by metis
    next
      show sorted-wrt (λx y. atm-of (fst y) ≺_t atm-of (fst x)) Γ
        using Γ-sorted .
    qed

  show ∀ Ln∈set Γ'. ∀ C. snd Ln = Some C ⟶ ord-res.is-strictly-maximal-lit
(fst Ln) C
    unfolding ‹Γ' = (Pos A, Some (efac C)) # Γ›
    using invars step-hyps
    by (simp add: clause-could-propagate-def greatest-literal-in-efacI
```

linorder-cls.is-least-in-ffilter-iff)

show *linorder-trm.is-lower-fset* (*trail-atms* Γ′) (*atms-of-clss* (*N* |∪| U_{er}))
unfolding ‹*trail-atms* Γ′ = *finsert A* (*trail-atms* Γ)› *finsert.rep-eq*
proof (*intro linorder-trm.is-lower-set-insertI ballI impI*)
show *A* |∈| *atms-of-clss* (*N* |∪| U_{er})
using *A-in* **.**
next
fix *w* :: *'f gterm*
assume *w* |∈| *atms-of-clss* (*N* |∪| U_{er}) **and** *w* ≺$_t$ *A*
thus *w* |∈| *trail-atms* Γ
by (*metis A-lt* Γ*-lower linorder-trm.dual-order.asym linorder-trm.neq-iff*
linorder-trm.not-in-lower-setI)
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using Γ*-lower* **.**
qed

{
fix *Ln* **and** Γ″
assume Γ′ = *Ln* # Γ″
hence *Ln* = (*Pos A*, *Some* (*efac C*)) **and** Γ″ = Γ
using ‹Γ′ = (*Pos A*, *Some* (*efac C*)) # Γ› **by** *simp-all*

obtain *D* **where** *D-in*: *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and** *D-false*: *trail-false-cls* Γ′ *D*
using *step-hyps* **by** *metis*

have (∃ *C*|∈|*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ′ *C*)
proof (*rule bexI*)
show *trail-false-cls* Γ′ *D*
using *D-false* **.**
next
have ¬ *trail-false-lit* Γ′ (*Pos A*)
by (*metis C-prop map-of-Cons-code*(*2*) *map-of-eq-None-iff clause-could-propagate-def*
step-hyps(*6*) *trail-defined-lit-def trail-false-lit-def uminus-not-id*)

moreover have *Pos A* ∈# *C*
using *C-prop*
unfolding *clause-could-propagate-def linorder-lit.is-maximal-in-mset-iff*
by *argo*

ultimately have ¬ *trail-false-cls* Γ′ *C*
unfolding *trail-false-cls-def* **by** *metis*

hence *D* ≠ *C*
using *D-false* **by** *metis*

thus *D* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})

unfolding ‹$\mathcal{F}' = (\mathit{if}\ \mathit{ord\text{-}res.is\text{-}strictly\text{-}maximal\text{-}lit}\ (\mathit{Pos}\ A)\ C\ \mathit{then}\ \mathcal{F}\ \mathit{else}\ \mathit{finsert}\ C\ \mathcal{F})$›
using *D-in iefac-def* **by** *auto*
qed

moreover have *snd Ln* ≠ *None*
using ‹$\Gamma' = \mathit{Ln}\ \#\ \Gamma''$› *step-hyps* **by** *auto*

ultimately show *snd Ln* ≠ *None* ⟷
(∃ C|∈|*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}). *trail-false-cls* Γ' C)
by *argo*

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹$\mathit{Ln} = (\mathit{Pos}\ A, \mathit{Some}\ (\mathit{efac}\ C))$› **by** *auto*

show ∀ x∈*set* Γ''. *snd* x = *None*
unfolding ‹$\Gamma'' = \Gamma$›
using *invars* **by** (*meson list.set-cases step-hyps*)

have *clause-could-propagate* Γ (*efac* C) (*Pos* A)
using *C-prop clause-could-propagate-efac* **by** *metis*

thus ⋀C. *snd Ln* = *Some* C ⟹ *clause-could-propagate* Γ'' C (*fst Ln*)
using ‹$\Gamma' = \mathit{Ln}\ \#\ \Gamma''$› ‹$\Gamma' = (\mathit{Pos}\ A, \mathit{Some}\ (\mathit{efac}\ C))\ \#\ \Gamma$›
by *force*

have *efac* C |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er})
proof (*cases ord-res.is-strictly-maximal-lit* (*Pos* A) C)
case *True*
thus *?thesis*
unfolding ‹$\mathcal{F}' =$ -›
using *C-in*
by (*metis* (*mono-tags*, *opaque-lifting*) *literal.discI*(*1*)
nex-strictly-maximal-pos-lit-if-neq-efac)
next
case *False*
then show *?thesis*
unfolding ‹$\mathcal{F}' =$ -›
using *C-in*
by (*smt* (*z3*) *fimage-iff finsert-iff iefac-def nex-strictly-maximal-pos-lit-if-neq-efac*
obtains-positive-greatest-lit-if-efac-not-ident)
qed

thus ⋀D. *snd Ln* = *Some* D ⟹ D |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er})
using ‹$\Gamma' = \mathit{Ln}\ \#\ \Gamma''$› ‹$\Gamma' = (\mathit{Pos}\ A, \mathit{Some}\ (\mathit{efac}\ C))\ \#\ \Gamma$› **by** *force*
}

show ⋀Γ_1 *Ln* Γ_0. $\Gamma' = \Gamma_1$ @ *Ln* # Γ_0 ⟹ *snd Ln* = *None* ⟹
¬ (∃ C|∈|*iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er}). *trail-false-cls* (*Ln* # Γ_0) C)

unfolding ‹$\Gamma' = (\text{-}, \textit{Some}\ \text{-})\ \#\ \Gamma$›
using *invars*
unfolding *bex-trail-false-cls-simp*
by (*metis list.inject not-None-eq split-pairs suffix-Cons suffix-def*)

show $\{\#\} \mid\in\mid N \mid\cup\mid U_{er} \Longrightarrow \Gamma' = []$
by (*meson bex-trail-false-cls-simp step-hyps*(*3*) *trail-false-cls-mempty*)
next
show $\bigwedge C.\ \textit{None} = \textit{Some}\ C \Longrightarrow \textit{atms-of-cls}\ C \mid\subseteq\mid \textit{atms-of-clss}\ (N \mid\cup\mid U_{er})$
by *simp*
next
show $\bigwedge C.\ \textit{None} = \textit{Some}\ C \Longrightarrow \textit{trail-false-cls}\ \Gamma'\ C$
by *simp*
next
show $\textit{atms-of-clss}\ U_{er} \mid\subseteq\mid \textit{atms-of-clss}\ N$
using *more-invars* **by** *argo*
next
have $\exists L.\ \textit{is-pos}\ L \wedge \textit{ord-res.is-maximal-lit}\ L\ C$
using *step-hyps*
by (*metis* (*no-types*, *lifting*) *clause-could-propagate-def*
linorder-cls.is-least-in-ffilter-iff literal.discI(*1*))

thus $\forall\, C \mid\in\mid \mathcal{F}'.\ \exists L.\ \textit{is-pos}\ L \wedge \textit{ord-res.is-maximal-lit}\ L\ C$
using *more-invars* ‹$\mathcal{F}' = (\textit{if}\ \text{-}\ \textit{then}\ \mathcal{F}\ \textit{else}\ \textit{finsert}\ C\ \mathcal{F})$› **by** *simp*
qed
next
case *step-hyps*: (*conflict D*)
show *?thesis*
unfolding ‹$s' = \text{-}$›
proof (*intro ord-res-11-invars.intros allI impI*)
show $\textit{ord-res-10-invars}\ N\ (U_{er}, \mathcal{F}, \Gamma)$
using *more-invars* **by** *argo*
next
show $\{\#\} \mid\in\mid N \mid\cup\mid U_{er} \Longrightarrow \Gamma = []$
using *more-invars* **by** *argo*
next
show $\bigwedge C.\ \textit{Some}\ D = \textit{Some}\ C \Longrightarrow \textit{atms-of-cls}\ C \mid\subseteq\mid \textit{atms-of-clss}\ (N \mid\cup\mid U_{er})$
by (*metis atm-of-in-atms-of-clssI atms-of-cls-def fimage-fsubsetI fset-fset-mset*
linorder-cls.is-least-in-ffilter-iff option.inject step-hyps(*3*))
next
show $\bigwedge C.\ \textit{Some}\ D = \textit{Some}\ C \Longrightarrow \textit{trail-false-cls}\ \Gamma\ C$
using ‹*linorder-cls.is-least-in-fset* - *D*›
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *simp*
next
show $\textit{atms-of-clss}\ U_{er} \mid\subseteq\mid \textit{atms-of-clss}\ N$
using *more-invars* **by** *argo*
next

```
      show ∀ C|∈|F. ∃ L. is-pos L ∧ ord-res.is-maximal-lit L C
        using more-invars by argo
      qed
    next
      case step-hyps: (skip L C n Γ')

      have ⋀xs. fset (trail-atms xs) = atm-of ‘ fst ‘ set xs
        unfolding fset-trail-atms ..
      also have ⋀xs. atm-of ‘ fst ‘ set xs = set (map (atm-of o fst) xs)
        by (simp add: image-comp)
      finally have ⋀xs. fset (trail-atms xs) = set (map (atm-of o fst) xs) .

      show ?thesis
        unfolding ‹s' = (U_er, F, Γ', Some C)›
       proof (intro ord-res-11-invars.intros ord-res-10-invars.intros ballI allI impI
conjI)
        show sorted-wrt (λx y. atm-of (fst y) ≺_t atm-of (fst x)) Γ'
          using invars unfolding ‹Γ = (L, n) # Γ'› by simp
      next
        fix Ln :: 'f gliteral × 'f gclause option and C :: 'f gclause
        assume Ln ∈ set Γ' and snd Ln = Some C
        then show ord-res.is-strictly-maximal-lit (fst Ln) C
          using invars unfolding ‹Γ = (L, n) # Γ'› by simp
      next
        show linorder-trm.is-lower-fset (trail-atms Γ') (atms-of-clss (N |∪| U_er))
          unfolding ‹fset (trail-atms Γ') = set (map (atm-of o fst) Γ')›
        proof (rule linorder-trm.sorted-and-lower-set-appendD-right(2))
          have sorted-wrt (λx y. y ≺_t x) (map (atm-of ∘ fst) Γ)
            using Γ-sorted by (simp add: sorted-wrt-map)

          thus sorted-wrt (λx y. y ≺_t x) ([atm-of L] @ map (atm-of ∘ fst) Γ')
            unfolding ‹Γ = - # Γ'› by simp
        next
          have set ([atm-of L] @ map (atm-of ∘ fst) Γ') = fset (trail-atms Γ)
            unfolding append-Cons append-Nil list.set
           unfolding ‹fset (trail-atms Γ') = set (map (atm-of o fst) Γ')›[symmetric]
            unfolding ‹Γ = - # Γ'› trail-atms.simps prod.sel
            by simp

          thus linorder-trm.is-lower-set (set ([atm-of L] @ map (atm-of ∘ fst) Γ'))
            (fset (atms-of-clss (N |∪| U_er)))
            using Γ-lower
            by (simp only:)
        qed
      next
        fix
          Ln :: 'f gliteral × 'f gclause option and
          Γ'' :: ('f gliteral × 'f gclause option) list
        assume Γ' = Ln # Γ''
```

have *snd Ln = None*
by (*metis* ‹Γ′ = *Ln* # Γ′′› *in-set-conv-decomp invars*(*4*) *step-hyps*(*1*) *suffixE suffix-Cons*)

show (*snd Ln* ≠ *None*) = (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ′ *C*)
using ‹*snd Ln = None*›
by (*metis* ‹Γ′ = *Ln* # Γ′′› *invars*(*5*) *step-hyps*(*1*) *suffixE suffix-Cons*)

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹*snd Ln = None*› **by** *simp*

show ⋀*C*. *snd Ln = Some C* ⟹ *clause-could-propagate* Γ′′ *C* (*fst Ln*)
using ‹*snd Ln = None*› **by** *simp*

show ⋀*C*. *snd Ln = Some C* ⟹ *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
using ‹*snd Ln = None*› **by** *simp*

show ⋀*x*. *x* ∈ *set* Γ′′ ⟹ *snd x = None*
by (*simp add*: ‹Γ′ = *Ln* # Γ′′› *invars*(*4*) *step-hyps*(*1*))
next
fix
Ln :: *'f gliteral* × *'f gclause option* **and**
Γ_1 Γ_0 :: (*'f gliteral* × *'f gclause option*) *list*
assume Γ′ = Γ_1 @ *Ln* # Γ_0 **and** *snd Ln = None*
thus ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* (*Ln* # Γ_0) *C*)
by (*metis append-Cons invars*(*5*) *step-hyps*(*1*))
next
show {#} |∈| *N* |∪| U_{er} ⟹ Γ′ = []
using *step-hyps*(*1*) *more-invars*(*3*) **by** *fastforce*
next
show ⋀*D*. *Some C = Some D* ⟹ *atms-of-cls D* |⊆| *atms-of-clss* (*N* |∪| U_{er})
using *more-invars*(*4*) *step-hyps*(*2*) **by** *presburger*
next
show ⋀*D*. *Some C = Some D* ⟹ *trail-false-cls* Γ′ *D*
using *more-invars* ‹$\mathcal{C}$ = *Some C*› ‹Γ = (*L*, *n*) # Γ′› ‹− *L* ∉# *C*›
using *subtrail-falseI* **by** *auto*
next
show *atms-of-clss* U_{er} |⊆| *atms-of-clss N*
using *more-invars* **by** *argo*
next
show ⋀*C*. *C* |∈| $\mathcal{F}$ ⟹ ∃ *L*. *is-pos L* ∧ *ord-res.is-maximal-lit L C*
using *more-invars* **by** *metis*
qed
next
case *step-hyps*: (*resolution L D* Γ′ *C*)

have *D-max-lit*: *ord-res.is-strictly-maximal-lit L D*
using *invars* ‹$\Gamma = (L, \mathit{Some}\ D) \# \Gamma'$› **by** *simp*

show *?thesis*
unfolding ‹$s' = \text{-}$›
proof (*intro ord-res-11-invars.intros allI impI*)
show *ord-res-10-invars N* (U_{er}, $\mathcal{F}$, Γ)
using *more-invars* **by** *argo*
next
show $\{\#\} \mid\in\mid N \mid\cup\mid U_{er} \Longrightarrow \Gamma = []$
using *more-invars* **by** *argo*
next
have $D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *invars* ‹$\Gamma = (L, \mathit{Some}\ D) \# \Gamma'$›
by (*metis snd-conv*)

hence *atms-of-cls D* $\mid\subseteq\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$)
by (*metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb funion-upper1*)

moreover have *atms-of-cls C* $\mid\subseteq\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$)
by (*smt* (*verit*) *add-mset-add-single atms-of-cls-def dual-order.trans fimage-fsubsetI*
fimage-iff fset-fset-mset more-invars(*4*) *step-hyps*(*1*) *union-iff*)

ultimately show $\bigwedge E$. *Some* (*remove1-mset* ($-$ *L*) *C* + *remove1-mset L D*) = *Some E* $\Longrightarrow$
atms-of-cls E $\mid\subseteq\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er}$)
by (*smt* (*verit*, *ccfv-threshold*) *add-mset-add-single atms-of-cls-def diff-single-trivial*
fimage-iff fset-fset-mset fsubsetI fsubset-funion-eq funionI1 insert-DiffM
option.inject union-iff)
next
fix *E* :: *'f gclause*
assume *Some* (*remove1-mset* ($-$ *L*) *C* + *remove1-mset L D*) = *Some E*

hence *E* = *remove1-mset* ($-$ *L*) *C* + *remove1-mset L D*
by *simp*

show *trail-false-cls* Γ *E*
unfolding *trail-false-cls-def*
proof (*intro ballI*)
fix *K* :: *'f gliteral*
assume *K* $\in\#$ *E*

hence *K* $\in\#$ *remove1-mset* ($-$ *L*) *C* $\vee$ *K* $\in\#$ *remove1-mset L D*
unfolding ‹$E = \text{-}$› **by** *simp*

thus *trail-false-lit* Γ *K*
proof (*elim disjE*)

```
          assume K ∈# remove1-mset (− L) C

          moreover have trail-false-cls Γ C
            using more-invars ‹C = Some C› by metis

          ultimately show trail-false-lit Γ K
            unfolding trail-false-cls-def
            by (metis in-diffD)
        next
          assume K-in: K ∈# remove1-mset L D

          have clause-could-propagate Γ' D L
            using invars ‹Γ = (L, Some D) # Γ'› by simp

          hence trail-false-cls Γ' {#K ∈# D. K ≠ L#}
            by (simp only: clause-could-propagate-def)

          hence trail-false-cls Γ {#K ∈# D. K ≠ L#}
            unfolding ‹Γ = (L, Some D) # Γ'›
            by (simp add: trail-false-cls-def trail-false-lit-def)

          moreover have K ∈# {#K ∈# D. K ≠ L#}
              using D-max-lit K-in in-diffD linorder-lit.is-greatest-in-mset-iff by
fastforce

          ultimately show trail-false-lit Γ K
            unfolding trail-false-cls-def by metis
        qed
      qed
    next
      show atms-of-clss U_er |⊆| atms-of-clss N
        using more-invars by argo
    next
      show ∀ C |∈| F. ∃ L. is-pos L ∧ ord-res.is-maximal-lit L C
        using more-invars by metis
    qed
  next
    case step-hyps: (backtrack L Γ' C)

    have {#} |∉| N |∪| U_er
      using more-invars step-hyps(3) by fastforce

    hence {#} |∉| iefac F |`| (N |∪| U_er)
      unfolding mempty-in-fimage-iefac .

    hence ¬(∃ C|∈|iefac F |`| (N |∪| U_er). trail-false-cls Γ C)
      using invars ‹Γ = - # Γ'›
      by (metis (no-types, opaque-lifting) split-pairs)
```

hence $\neg(\exists\, C|\in|\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma'\ C)$
by (*metis append.simps*(*1*) *step-hyps*(*3*) *suffix-ConsD suffix-def trail-false-cls-if-trail-false-suffix*)

moreover have $\neg$ *trail-false-cls* Γ' *C*
by (*metis* ‹*trail-consistent* Γ› *fst-conv list.distinct*(*1*) *list.inject step-hyps*(*3*,*4*) *trail-consistent.simps trail-defined-lit-def trail-false-cls-def trail-false-lit-def uminus-lit-swap*)

ultimately have $\neg(\exists\, C|\in|\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ \mathit{finsert}\ C\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma'\ C)$
by (*simp add*: *iefac-def trail-false-cls-def*)

have $\bigwedge xs.\ \mathit{fset}\ (\mathit{trail\text{-}atms}\ xs) = \mathit{atm\text{-}of}\ `\ \mathit{fst}\ `\ \mathit{set}\ xs$
unfolding *fset-trail-atms* **..**
also have $\bigwedge xs.\ \mathit{atm\text{-}of}\ `\ \mathit{fst}\ `\ \mathit{set}\ xs = \mathit{set}\ (\mathit{map}\ (\mathit{atm\text{-}of}\ o\ \mathit{fst})\ xs)$
by (*simp add*: *image-comp*)
finally have $\bigwedge xs.\ \mathit{fset}\ (\mathit{trail\text{-}atms}\ xs) = \mathit{set}\ (\mathit{map}\ (\mathit{atm\text{-}of}\ o\ \mathit{fst})\ xs)$ **.**

have *atms-of-cls C* $|\subseteq|$ *atms-of-clss* $(N\ |\cup|\ U_{er})$
using *more-invars* ‹$\mathcal{C}$ = *Some C*› **by** *metis*

hence *atms-of-clss* $(N\ |\cup|\ \mathit{finsert}\ C\ U_{er}) = \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er})$
by *auto*

show *?thesis*
unfolding ‹s' = (*finsert C* U_{er}, $\mathcal{F}$, Γ', *None*)›
proof (*intro ord-res-11-invars.intros ord-res-10-invars.intros ballI allI impI conjI*)
show *sorted-wrt* $(\lambda x\ y.\ \mathit{atm\text{-}of}\ (\mathit{fst}\ y) \prec_t \mathit{atm\text{-}of}\ (\mathit{fst}\ x))\ \Gamma'$
using *invars* **unfolding** ‹Γ = - # Γ'› **by** *simp*
next
fix *Ln* :: *'f gliteral* × *'f gclause option* **and** *C* :: *'f gclause*
assume *Ln* ∈ *set* Γ' **and** *snd Ln* = *Some C*
then show *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
using *invars* **unfolding** ‹Γ = - # Γ'› **by** *simp*
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* $(N\ |\cup|\ \mathit{finsert}\ C\ U_{er})$)
unfolding ‹*fset* (*trail-atms* Γ') = *set* (*map* (*atm-of o fst*) Γ')›
unfolding ‹*atms-of-clss* $(N\ |\cup|\ \mathit{finsert}\ C\ U_{er})$ = *atms-of-clss* $(N\ |\cup|\ U_{er})$›
proof (*rule linorder-trm.sorted-and-lower-set-appendD-right*(*2*))
have *sorted-wrt* $(\lambda x\ y.\ y \prec_t x)$ (*map* (*atm-of* ∘ *fst*) Γ)
using Γ-*sorted* **by** (*simp add*: *sorted-wrt-map*)

thus *sorted-wrt* $(\lambda x\ y.\ y \prec_t x)$ ([*atm-of L*] @ *map* (*atm-of* ∘ *fst*) Γ')
unfolding ‹Γ = - # Γ'› **by** *simp*
next
have *set* ([*atm-of L*] @ *map* (*atm-of* ∘ *fst*) Γ') = *fset* (*trail-atms* Γ)

unfolding *append-Cons append-Nil list.set*
unfolding ‹*fset* (*trail-atms* Γ′) = *set* (*map* (*atm-of o fst*) Γ′)›[*symmetric*]
unfolding ‹Γ = - # Γ′› *trail-atms.simps prod.sel*
by *simp*

thus *linorder-trm.is-lower-set* (*set* ([*atm-of L*] @ *map* (*atm-of* ∘ *fst*) Γ′))
(*fset* (*atms-of-clss* (*N* |∪| U_{er})))
using Γ-*lower*
by (*simp only*:)
qed
next
fix
Ln :: *'f gliteral* × *'f gclause option* **and**
Γ′′ :: (*'f gliteral* × *'f gclause option*) *list*
assume Γ′ = *Ln* # Γ′′

have *snd Ln* = *None*
by (*simp add*: ‹Γ′ = *Ln* # Γ′′› *invars*(*4*) *step-hyps*(*3*))

thus (*snd Ln* ≠ *None*) ⟷ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| *finsert C* U_{er}). *trail-false-cls* Γ′ *C*)
using ‹¬(∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| *finsert C* U_{er}). *trail-false-cls* Γ′ *C*)›
by *argo*

show *snd Ln* ≠ *None* ⟹ *is-pos* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *simp*

show ⋀*C*. *snd Ln* = *Some C* ⟹ *clause-could-propagate* Γ′′ *C* (*fst Ln*)
using ‹*snd Ln* = *None*› **by** *simp*

show ⋀*D*. *snd Ln* = *Some D* ⟹ *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *finsert C* U_{er})
using ‹*snd Ln* = *None*› **by** *simp*

show ⋀*x*. *x* ∈ *set* Γ′′ ⟹ *snd x* = *None*
by (*simp add*: ‹Γ′ = *Ln* # Γ′′› *invars*(*4*) *step-hyps*(*3*))
next
fix
Ln :: *'f gliteral* × *'f gclause option* **and**
Γ$_1$ Γ$_0$ :: (*'f gliteral* × *'f gclause option*) *list*
assume Γ′ = Γ$_1$ @ *Ln* # Γ$_0$ **and** *snd Ln* = *None*
thus ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| *finsert C* U_{er}). *trail-false-cls* (*Ln* # Γ$_0$) *C*)
using ‹¬(∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| *finsert C* U_{er}). *trail-false-cls* Γ′ *C*)›
by (*metis suffixI trail-false-cls-if-trail-false-suffix*)
next
have *C* ≠ {#}
using ‹− *L* ∈# *C*› **by** *force*

hence {#} |∉| *N* |∪| *finsert C* U_{er}
using ‹{#} |∉| *N* |∪| U_{er}› **by** *simp*

thus $\{\#\} \mid\in\mid N \mid\cup\mid \mathit{finsert}\ C\ U_{er} \Longrightarrow \Gamma' = []$
by *contradiction*
next
show $\bigwedge D.\ \mathit{None} = \mathit{Some}\ D \Longrightarrow \mathit{atms\text{-}of\text{-}cls}\ D \mid\subseteq\mid \mathit{atms\text{-}of\text{-}clss}\ (N \mid\cup\mid \mathit{finsert}\ C\ U_{er})$
by *simp*
next
show $\bigwedge C.\ \mathit{None} = \mathit{Some}\ C \Longrightarrow \mathit{trail\text{-}false\text{-}cls}\ \Gamma'\ C$
by *simp*
next
have $\mathit{atms\text{-}of\text{-}cls}\ C \mid\subseteq\mid \mathit{atms\text{-}of\text{-}clss}\ N$
by (*smt* (*verit*, *ccfv-threshold*) ‹$\mathit{atms\text{-}of\text{-}cls}\ C \mid\subseteq\mid \mathit{atms\text{-}of\text{-}clss}\ (N \mid\cup\mid U_{er})$›
atms-of-clss-def fin-mono fmember-ffUnion-iff fsubsetI funion-iff more-invars(6))

thus $\mathit{atms\text{-}of\text{-}clss}\ (\mathit{finsert}\ C\ U_{er}) \mid\subseteq\mid \mathit{atms\text{-}of\text{-}clss}\ N$
using ‹$\mathit{atms\text{-}of\text{-}clss}\ U_{er} \mid\subseteq\mid \mathit{atms\text{-}of\text{-}clss}\ N$› **by** *simp*
next
show $\bigwedge C.\ C \mid\in\mid \mathcal{F} \Longrightarrow \exists L.\ \mathit{is\text{-}pos}\ L \wedge \mathit{ord\text{-}res.is\text{-}maximal\text{-}lit}\ L\ C$
using *more-invars* **by** *metis*
qed
qed
qed
qed

lemma *rtranclp-ord-res-11-preserves-invars*:
assumes
step: $(\mathit{ord\text{-}res\text{-}11}\ N)^{**}\ s\ s'$ **and**
invars: $\mathit{ord\text{-}res\text{-}11\text{-}invars}\ N\ s$
shows $\mathit{ord\text{-}res\text{-}11\text{-}invars}\ N\ s'$
using *step invars ord-res-11-preserves-invars*
by (*smt* (*verit*, *del-insts*) *rtranclp-induct*)

lemma *tranclp-ord-res-11-preserves-invars*:
assumes
step: $(\mathit{ord\text{-}res\text{-}11}\ N)^{++}\ s\ s'$ **and**
invars: $\mathit{ord\text{-}res\text{-}11\text{-}invars}\ N\ s$
shows $\mathit{ord\text{-}res\text{-}11\text{-}invars}\ N\ s'$
using *step invars ord-res-11-preserves-invars*
by (*smt* (*verit*, *del-insts*) *tranclp-induct*)

lemma *ex-ord-res-11-if-not-final*:
assumes
not-final: $\neg\ \mathit{ord\text{-}res\text{-}11\text{-}final}\ (N,\ s)$ **and**
invars: $\mathit{ord\text{-}res\text{-}11\text{-}invars}\ N\ s$
shows $\exists s'.\ \mathit{ord\text{-}res\text{-}11}\ N\ s\ s'$
using *invars*
proof (*cases N s rule*: *ord-res-11-invars.cases*)

case *more-invars*: (*1* U_{er} $\mathcal{F}$ Γ $\mathcal{C}$)
show *?thesis*
using ‹*ord-res-10-invars N* (U_{er}, $\mathcal{F}$, Γ)›
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ) *rule*: *ord-res-10-invars.cases*)
case *invars*: *1*

show *?thesis*
proof (*cases* $\mathcal{C}$)
case *None*
show *?thesis*
proof (*cases* $\exists\, C$ |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* Γ *C*)
case *True*

then obtain *C* **where**
linorder-cls.is-least-in-fset (*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))) *C*
using *linorder-cls.ex-is-least-in-ffilter-iff* **by** *metis*

thus *?thesis*
unfolding ‹$s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$› ‹$\mathcal{C}$ = *None*›
using *ord-res-11.conflict* **by** *metis*
next
case *no-false-cls*: *False*

hence $\exists\, A_2$ |∈| *atms-of-clss* (*N* |∪| U_{er}). $\forall\, A_1$ |∈| *trail-atms* Γ. $A_1 \prec_t A_2$
using *not-final*
unfolding ‹$s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$› ‹$\mathcal{C}$ = *None*›
by (*smt* (*verit*) *invars*(*3*) *linorder-trm.antisym-conv3 linorder-trm.not-in-lower-setI ord-res-11-final.model-found*)

then obtain *A* **where**
A-least: *linorder-trm.is-least-in-fset*
{|A_2 |∈| *atms-of-clss* (*N* |∪| U_{er}). $\forall\, A_1$ |∈| *trail-atms* Γ. $A_1 \prec_t A_2$|} *A*
using *linorder-trm.ex-is-least-in-ffilter-iff* **by** *presburger*

show *?thesis*
proof (*cases* $\exists$ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C* (*Pos A*))
case *True*

then obtain *C* **where**
C-least: *linorder-cls.is-least-in-fset*
{|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *clause-could-propagate* Γ *C* (*Pos A*)|} *C*
using *linorder-cls.ex-is-least-in-ffilter-iff* **by** *presburger*

show *?thesis*
proof (*cases* $\exists$ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *trail-false-cls* ((*Pos A*, *None*) # Γ) *C*)

```
      case True

      moreover have trail-false-cls ((Pos A, None) # Γ) =
        trail-false-cls ((Pos A, Some (efac C)) # Γ)
        unfolding trail-false-cls-def trail-false-lit-def by simp

      ultimately show ?thesis
        unfolding ‹s = (U_er, F, Γ, C)› ‹C = None›
        using ord-res-11.propagate[OF no-false-cls A-least C-least]
        by metis
    next
      case False
      thus ?thesis
        unfolding ‹s = (U_er, F, Γ, C)› ‹C = None›
        using ord-res-11.decide-pos[OF no-false-cls A-least C-least]
        by metis
    qed
  next
    case False
    thus ?thesis
      unfolding ‹s = (U_er, F, Γ, C)› ‹C = None›
      using ord-res-11.decide-neg[OF no-false-cls A-least] by metis
  qed
qed
next
  case (Some D)

  hence D-false: trail-false-cls Γ D
    using more-invars by metis

  show ?thesis
  proof (cases D = {#})
    case True

    hence Γ ≠ []
      using not-final
      unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some D›
      unfolding ord-res-11-final.simps by metis

    then obtain L n Γ' where Γ = (L, n) # Γ'
      by (metis list.exhaust prod.exhaust)

    moreover have - L ∉# D
      unfolding ‹D = {#}› by simp

    ultimately show ?thesis
      unfolding ‹s = (U_er, F, Γ, C)› ‹C = Some D›
      using ord-res-11.skip by metis
  next
```

```
      case False

      then obtain L n Γ' where Γ = (L, n) # Γ'
        using D-false[unfolded trail-false-cls-def trail-false-lit-def]
        by (metis eq-fst-iff image-iff list.set-cases multiset-nonemptyE)

      show ?thesis
      proof (cases − L ∈# D)
        case True
        show ?thesis
        proof (cases n)
          case None
          show ?thesis
          proof (cases − L ∈# D)
            case True
            thus ?thesis
              unfolding ‹s = (U_er, F, Γ, C)› ‹Γ = (L, n) # Γ'› ‹n = None› ‹C =
Some D›
               using ord-res-11.backtrack by metis
          next
            case False
            thus ?thesis
              using ord-res-11.skip
              unfolding ‹s = (U_er, F, Γ, C)› ‹Γ = (L, n) # Γ'› ‹n = None› ‹C =
Some D› by metis
          qed
        next
          case (Some C)
          thus ?thesis
            unfolding ‹s = (U_er, F, Γ, C)› ‹Γ = (L, n) # Γ'› ‹n = Some C› ‹C
= Some D›
            using ord-res-11.resolution[OF - ‹− L ∈# D›] by metis
        qed
      next
        case False
        thus ?thesis
          using ord-res-11.skip
          unfolding ‹s = (U_er, F, Γ, C)› ‹Γ = (L, n) # Γ'› ‹C = Some D› by
metis
      qed
    qed
  qed
qed

lemma ord-res-11-safe-state-if-invars:
  fixes N s
  assumes invars: ord-res-11-invars N s
  shows safe-state (constant-context ord-res-11) ord-res-11-final (N, s)
```

```
  using safe-state-constant-context-if-invars[where
    R = ord-res-11 and F = ord-res-11-final and I = ord-res-11-invars]
  using ord-res-11-preserves-invars ex-ord-res-11-if-not-final invars by metis

lemma rtrancl-ord-res-11-all-resolution-steps:
  assumes C-max-lit: ord-res.is-strictly-maximal-lit K C
  shows (ord-res-11 N)** (U, F, (K, Some C) # Γ, Some D) (U, F, (K, Some
C) # Γ, Some (eres C D))
proof −
  obtain CD where eres C D = CD and run: full-run (ground-resolution C) D
CD
    using ex1-eres-eq-full-run-ground-resolution by metis

  have (ord-res-11 N)** (U, F, (K, Some C) # Γ, Some D) (U, F, (K, Some C)
# Γ, Some CD)
  proof (rule full-run-preserves-invariant[OF run])
    show (ord-res-11 N)** (U, F, (K, Some C) # Γ, Some D) (U, F, (K, Some
C) # Γ, Some D)
      by simp
  next
    fix x y
    assume ground-resolution C x y

    hence ord-res-11 N (U, F, (K, Some C) # Γ, Some x) (U, F, (K, Some C)
# Γ, Some y)
      unfolding ground-resolution-def
    proof (cases x C y rule: ord-res.ground-resolution.cases)
      case res-hyps: (ground-resolutionI A P1' P2')

      have K = − Neg A
        using C-max-lit
      by (metis ord-res.Uniq-striclty-maximal-lit-in-ground-cls res-hyps(6) the1-equality'
          uminus-Neg)

      hence − K ∈# x
        by (simp add: ‹x = add-mset (Neg A) P1'›)

      moreover have y = remove1-mset (− K) x + remove1-mset K C
        using res-hyps ‹K = − Neg A› by force

      ultimately show ?thesis
        using ord-res-11.resolution[OF refl, of K x N U F C Γ]
        by metis
    qed

    moreover assume (ord-res-11 N)**
      (U, F, (K, Some C) # Γ, Some D)
      (U, F, (K, Some C) # Γ, Some x)
```

```
    ultimately show (ord-res-11 N)**
      (U, F, (K, Some C) # Γ, Some D)
      (U, F, (K, Some C) # Γ, Some y)
      by simp
  qed

  then show ?thesis
    unfolding ‹eres C D = CD›
    by argo
qed

lemma rtrancl-ord-res-11-all-skip-steps:
  (ord-res-11 N)** (U, F, Γ, Some C) (U, F, dropWhile (λLn. - fst Ln ∉# C)
Γ, Some C)
proof (induction Γ)
  case Nil
  show ?case by simp
next
  case (Cons Ln Γ)
  show ?case
  proof (cases - fst Ln ∈# C)
    case True
    hence dropWhile (λLn. - fst Ln ∉# C) (Ln # Γ) = Ln # Γ
      by simp
    thus ?thesis
      by simp
  next
    case False
    hence *: dropWhile (λLn. - fst Ln ∉# C) (Ln # Γ) = dropWhile (λLn. - fst
Ln ∉# C) Γ
      by simp

    obtain L C where Ln = (L, C)
      by (metis prod.exhaust)

    have ord-res-11 N (U, F, Ln # Γ, Some C) (U, F, Γ, Some C)
      unfolding ‹Ln = (L, C)›
    proof (rule ord-res-11.skip)
      show - L ∉# C
        using False unfolding ‹Ln = (L, C)› by simp
    qed

    thus ?thesis
      unfolding *
      using Cons.IH
      by simp
  qed
qed
```

end

end
theory *Simulation-SCLFOL-ORDRES*
imports
Background
ORD-RES
ORD-RES-1
ORD-RES-2
ORD-RES-3
ORD-RES-4
ORD-RES-5
ORD-RES-6
ORD-RES-7
ORD-RES-8
ORD-RES-9
ORD-RES-10
ORD-RES-11
Clause-Could-Propagate
begin

27 ORD-RES-1 (deterministic)

type-synonym *'f ord-res-1-state* = *'f gclause fset*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

sublocale *backward-simulation-with-measuring-function* **where**
step1 = *ord-res* **and**
step2 = *ord-res-1* **and**
final1 = *ord-res-final* **and**
final2 = *ord-res-1-final* **and**
order = λ- -. *False* **and**
match = (=) **and**
measure = λ-. ()
proof *unfold-locales*
show *wfP* (λ- -. *False*)
by *simp*
next
show ⋀*N1 N2*. *N1* = *N2* ⟹ *ord-res-1-final N2* ⟹ *ord-res-final N1*
unfolding *ord-res-1-final-def* **by** *metis*
next
fix *N1 N2 N2'* :: *'f ord-res-1-state*
assume *match*: *N1* = *N2* **and** *step2*: *ord-res-1 N2 N2'*
show (∃*N1'*. *ord-res*$^{++}$ *N1 N1'* ∧ *N1'* = *N2'*) ∨ *N1* = *N2'* ∧ *False*
proof (*intro disjI1 exI conjI*)

have *mempty-no-in*: {#} |∉| *N2*
if *C-least*: *linorder-cls.is-least-in-fset* {|*C* |∈| *N2*.

```
    ¬ ord-res.interp (fset N2) C ∪ ord-res.production (fset N2) C ⊨ C|} C and
      L-max: linorder-lit.is-maximal-in-mset C L
    for C L
  proof (rule notI)
    assume {#} |∈| N2
    moreover have ¬ ord-res.interp (fset N2) {#} ∪ ord-res.production (fset
N2) {#} ⊨ {#}
      by simp
    moreover have ⋀C. {#} ⪯c C
      using mempty-lesseq-cls by metis
    ultimately have C = {#}
      using C-least
      by (metis (no-types, lifting) ffmember-filter linorder-cls.is-least-in-fset-iff
        linorder-cls.less-le-not-le)
    moreover have L ∈# C
      using L-max by (simp add: linorder-lit.is-maximal-in-mset-iff)
    ultimately show False
      by simp
  qed

  have ord-res N2 N2'
    using step2
  proof (cases N2 N2' rule: ord-res-1.cases)
    case hyps: (factoring C L C')
    show ?thesis
    proof (rule ord-res.factoring)
      show {#} |∉| N2
        using hyps mempty-no-in is-least-false-clause-def by simp
    next
      show ex-false-clause (fset N2)
        unfolding ex-false-clause-def
        using hyps is-least-false-clause-def
        by (metis (no-types, lifting) linorder-cls.is-least-in-fset-ffilterD)
    next
      show C |∈| N2
        using hyps is-least-false-clause-def linorder-cls.is-least-in-fset-ffilterD(1)
by blast
    next
      show ord-res.ground-factoring C C'
        using hyps by argo
    next
      show N2' = finsert C' N2
        using hyps by argo
    qed
  next
    case hyps: (resolution C L D CD)
    show ?thesis
    proof (rule ord-res.resolution)
      show {#} |∉| N2
```

using *hyps mempty-no-in is-least-false-clause-def* **by** *simp*
next
show *ex-false-clause* (*fset N2*)
unfolding *ex-false-clause-def*
using *hyps is-least-false-clause-def*
by (*metis* (*no-types*, *lifting*) *linorder-cls.is-least-in-fset-ffilterD*)
next
show *C* |∈| *N2*
using *hyps is-least-false-clause-def linorder-cls.is-least-in-fset-ffilterD*(*1*)
by *blast*
next
show *D* |∈| *N2*
using *hyps* **by** *argo*
next
show *ord-res.ground-resolution C D CD*
using *hyps* **by** *argo*
next
show *N2'* = *finsert CD N2*
using *hyps* **by** *argo*
qed
qed
thus *ord-res*$^{++}$ *N1 N2'*
unfolding *match* **by** *simp*
next
show *N2'* = *N2'* **..**
qed
qed

end

28 ORD-RES-2 (full factorization)

type-synonym *'f ord-res-2-state* = *'f gclause fset* × *'f gclause fset* × *'f gclause fset*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

fun *ord-res-1-matches-ord-res-2*
:: *'f ord-res-1-state* ⇒ - ⇒ *bool* **where**
ord-res-1-matches-ord-res-2 S1 (*N*, (*U*$_r$, *U*$_{ef}$)) ⟷ (∃ *U*$_f$.
S1 = *N* |∪| *U*$_r$ |∪| *U*$_{ef}$ |∪| *U*$_f$ ∧
(∀ *C*$_f$ |∈| *U*$_f$. ∃ *C* |∈| *N* |∪| *U*$_r$ |∪| *U*$_{ef}$. *ord-res.ground-factoring*$^{++}$ *C C*$_f$ ∧
C$_f$ ≠ *efac C*$_f$ ∧
(*efac C*$_f$ |∈| *U*$_{ef}$ ∨ *is-least-false-clause* (*N* |∪| *U*$_r$ |∪| *U*$_{ef}$) *C*)))

lemma *ord-res-1-matches-ord-res-2-simps'*:
ord-res-1-matches-ord-res-2 S1 (*N*, (*U*$_r$, *U*$_{ef}$)) ⟷
(∃ *U*$_f$. *S1* = *N* |∪| *U*$_r$ |∪| *U*$_{ef}$ |∪| *U*$_f$ ∧
(∀ *C*$_f$ |∈| *U*$_f$. *C*$_f$ ≠ *efac C*$_f$ ∧ (∃ *C* |∈| *N* |∪| *U*$_r$ |∪| *U*$_{ef}$. *ord-res.ground-factoring*$^{++}$

$C\ C_f \wedge$
(*efac* C_f |∈| U_{ef} ∨ *is-least-false-clause* (N |∪| U_r |∪| U_{ef}) C))))
unfolding *ord-res-1-matches-ord-res-2.simps* **by** *metis*

lemma *ord-res-1-matches-ord-res-2-simps''*:
ord-res-1-matches-ord-res-2 S1 (N, (U_r, U_{ef})) ⟷
(∃ U_f. *S1* = N |∪| U_r |∪| U_{ef} |∪| U_f ∧
(∀ C_f |∈| U_f. $C_f \neq$ *efac* C_f ∧ (∃ C |∈| N |∪| U_r |∪| U_{ef}. *ord-res.ground-factoring*$^{++}$
$C\ C_f \wedge$
(*efac* C |∈| U_{ef} ∨ *is-least-false-clause* (N |∪| U_r |∪| U_{ef}) C))))
unfolding *ord-res-1-matches-ord-res-2-simps'*
by (*metis ground-factorings-preserves-efac tranclp-into-rtranclp*)

lemma *ord-res-1-final-iff-ord-res-2-final*:
assumes *match*: *ord-res-1-matches-ord-res-2* S_1 S_2
shows *ord-res-1-final* S_1 ⟷ *ord-res-2-final* S_2
proof −
obtain N U_r U_{ef} **where** S_2 = (N, (U_r, U_{ef}))
by (*metis prod.exhaust*)
with *match* **obtain** U_f **where**
S_1*-def*: S_1 = N |∪| U_r |∪| U_{ef} |∪| U_f **and**
U_f*-spec*: ∀ C_f |∈| U_f. ∃ C |∈| N |∪| U_r |∪| U_{ef}. *ord-res.ground-factoring*$^{++}$
$C\ C_f \wedge\ C_f \neq$ *efac* $C_f \wedge$
(*efac* C_f |∈| U_{ef} ∨ *is-least-false-clause* (N |∪| U_r |∪| U_{ef}) C)
by *auto*

have U_f*-unproductive*: ∀ C_f |∈| U_f. *ord-res.production* (*fset* (N |∪| U_r |∪| U_{ef}
|∪| U_f)) C_f = {}
proof (*intro ballI*)
fix C_f
assume C_f |∈| U_f
hence $C_f \neq$ *efac* C_f
using U_f*-spec* **by** *metis*
hence ∄ L. *is-pos* $L \wedge$ *ord-res.is-strictly-maximal-lit* L C_f
using *nex-strictly-maximal-pos-lit-if-neq-efac* **by** *metis*
thus *ord-res.production* (*fset* (N |∪| U_r |∪| U_{ef} |∪| U_f)) C_f = {}
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

have *Interp-eq*: ⋀C. *ord-res-Interp* (*fset* (N |∪| U_r |∪| U_{ef} |∪| U_f)) C =
ord-res-Interp (*fset* (N |∪| U_r |∪| U_{ef})) C
using *Interp-union-unproductive*[*of fset* (N |∪| U_r |∪| U_{ef}) *fset* U_f, *folded*
union-fset,
OF finite-fset finite-fset U_f*-unproductive*] **.**

have {#} |∈| N |∪| U_r |∪| U_{ef} |∪| U_f ⟷ {#} |∈| N |∪| U_r |∪| U_{ef}
proof (*rule iffI*)
assume {#} |∈| N |∪| U_r |∪| U_{ef} |∪| U_f
hence {#} |∈| N |∪| U_r |∪| U_{ef} ∨ {#} |∈| U_f

by *simp*
thus $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
proof (*elim disjE*)
assume $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
thus $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
by *assumption*
next
assume $\{\#\} \mid\in\mid U_f$
hence $\{\#\} \neq$ *efac* $\{\#\}$
using U_f-*spec*[*rule-format*, *of* $\{\#\}$] **by** *metis*
hence *False*
by *simp*
thus $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$ **..**
qed
next
assume $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
thus $\{\#\} \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f$
by *simp*
qed

moreover have $\neg$ *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)) \longleftrightarrow$
$\neg$ *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
proof (*rule iffI*; *erule contrapos-nn*)
assume *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
thus *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)$)
unfolding *ex-false-clause-def Interp-eq* **by** *auto*
next
assume *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)$)
then obtain C **where**
$C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f$ **and**
C-false: $\neg$ *ord-res-Interp* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$) $C \models C$
unfolding *ex-false-clause-def Interp-eq* **by** *metis*
hence $C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef} \vee C \mid\in\mid U_f$
by *simp*
thus *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
proof (*elim disjE*)
assume $C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
thus *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
unfolding *ex-false-clause-def* **using** *C-false* **by** *metis*
next
assume $C \mid\in\mid U_f$
then obtain C' **where** $C' \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}$ **and**
ord-res.ground-factoring^{++} C' C **and**
$C \neq$ *efac* C **and**
efac $C \mid\in\mid U_{ef} \vee$ *is-least-false-clause* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$ C'
using U_f-*spec*[*rule-format*, *of* C] **by** *metis*
thus *ex-false-clause* (*fset* $(N \mid\cup\mid U_r \mid\cup\mid U_{ef})$)
proof (*elim disjE exE conjE*)
assume *efac* $C \mid\in\mid U_{ef}$

show *ex-false-clause* (*fset* (*N* |∪| U_r |∪| U_{ef}))
proof (*cases ord-res-Interp* (*fset* (*N* |∪| U_r |∪| U_{ef})) (*efac C*) ⊨ *efac C*)
case *efac-C-true*: *True*
have *efac C* ⊆# *C*
using *efac-subset*[*of C*] **.**
hence *efac C* ≼$_c$ *C*
using *subset-implies-reflclp-multp* **by** *metis*
hence *ord-res-Interp* (*fset* (*N* |∪| U_r |∪| U_{ef})) *C* ⊨ *efac C*
using *efac-C-true ord-res.entailed-clause-stays-entailed* **by** *fastforce*
hence *ord-res-Interp* (*fset* (*N* |∪| U_r |∪| U_{ef})) *C* ⊨ *C*
using *efac-C-true* **by** (*simp add*: *true-cls-def*)
with *C-false* **have** *False*
by *contradiction*
thus *?thesis* **..**
next
case *False*

moreover have *efac C* |∈| *N* |∪| U_r |∪| U_{ef}
using ‹*efac C* |∈| U_{ef}› **by** *simp*

ultimately show *ex-false-clause* (*fset* (*N* |∪| U_r |∪| U_{ef}))
unfolding *ex-false-clause-def* **by** *metis*
qed
next
assume *is-least-false-clause* (*N* |∪| U_r |∪| U_{ef}) C'
hence C' |∈| *N* |∪| U_r |∪| U_{ef} **and** ¬ *ord-res-Interp* (*fset* (*N* |∪| U_r |∪| U_{ef})) C' ⊨ C'
using *linorder-cls.is-least-in-ffilter-iff is-least-false-clause-def* **by** *simp-all*
thus *ex-false-clause* (*fset* (*N* |∪| U_r |∪| U_{ef}))
unfolding *ex-false-clause-def* **by** *metis*
qed
qed
qed

ultimately show *?thesis*
by (*simp add*: S_1*-def* ‹S_2 = (*N*, U_r, U_{ef})› *ord-res-1-final-def ord-res-2-final.simps* *ord-res-final-def*)
qed

lemma *safe-states-if-ord-res-1-matches-ord-res-2*:
assumes *match*: *ord-res-1-matches-ord-res-2* S_1 S_2
shows *safe-state ord-res-1 ord-res-1-final* S_1 ∧ *safe-state ord-res-2-step ord-res-2-final* S_2
proof −
have *safe-state ord-res-1 ord-res-1-final* S_1
using *safe-state-if-all-states-safe ord-res-1-safe* **by** *metis*

moreover have *safe-state ord-res-2-step ord-res-2-final* S_2

using *safe-state-if-all-states-safe ord-res-2-step-safe* **by** *metis*

ultimately show *?thesis*
by *argo*
qed

definition *ord-res-1-measure* **where**
ord-res-1-measure s1 =
(*if* ∃ *C*. *is-least-false-clause s1 C then*
The (*is-least-false-clause s1*)
else
{#})

lemma *forward-simulation*:
assumes *match*: *ord-res-1-matches-ord-res-2 s1 s2* **and**
step1: *ord-res-1 s1 s1′*
shows (∃ *s2′*. *ord-res-2-step*$^{++}$ *s2 s2′* ∧ *ord-res-1-matches-ord-res-2 s1′ s2′*) ∨
ord-res-1-matches-ord-res-2 s1′ s2 ∧ *ord-res-1-measure s1′* ⊂# *ord-res-1-measure s1*
proof −
let
?match = *ord-res-1-matches-ord-res-2* **and**
?measure = *ord-res-1-measure* **and**
?order = (⊂#)

obtain *N* U_r U_{ef} :: *′f gterm clause fset* **where**
s2-def: *s2* = (*N*, (U_r, U_{ef}))
by (*metis prod.exhaust*)

from *match* **obtain** U_f **where**
s1-def: *s1* = *N* |∪| U_r |∪| U_{ef} |∪| U_f **and**
U_f*-spec*: ∀ C_f |∈| U_f. ∃ *C* |∈| *N* |∪| U_r |∪| U_{ef}. *ord-res.ground-factoring*$^{++}$ *C* C_f ∧ C_f ≠ *efac* C_f ∧
(*efac* C_f |∈| U_{ef} ∨ *is-least-false-clause* (*N* |∪| U_r |∪| U_{ef}) *C*)
unfolding *s2-def ord-res-1-matches-ord-res-2.simps* **by** *metis*

have U_f*-unproductive*: ∀ C_f |∈| U_f. *ord-res.production* (*fset* (*N* |∪| U_r |∪| U_{ef} |∪| U_f)) C_f = {}
proof (*intro ballI*)
fix C_f
assume C_f |∈| U_f
hence C_f ≠ *efac* C_f
using U_f*-spec* **by** *metis*
hence ∄ *L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L* C_f
using *nex-strictly-maximal-pos-lit-if-neq-efac* **by** *metis*
thus *ord-res.production* (*fset* (*N* |∪| U_r |∪| U_{ef} |∪| U_f)) C_f = {}
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

```
  have Interp-eq: ⋀C. ord-res-Interp (fset (N |∪| U_r |∪| U_ef |∪| U_f)) C =
    ord-res-Interp (fset (N |∪| U_r |∪| U_ef)) C
     using Interp-union-unproductive[of fset (N |∪| U_r |∪| U_ef) fset U_f, folded
union-fset,
        OF finite-fset finite-fset U_f-unproductive] .

  show (∃ s2'. ord-res-2-step⁺⁺ s2 s2' ∧ ?match s1' s2') ∨
    ?match s1' s2 ∧ ?order (?measure s1') (?measure s1)
    using step1
  proof (cases s1 s1' rule: ord-res-1.cases)
    case (factoring C L C')

    have C-least-false: is-least-false-clause (N |∪| U_r |∪| U_ef |∪| U_f) C
      using factoring
      unfolding is-least-false-clause-def s1-def by argo

    hence C-in: C |∈| N |∪| U_r |∪| U_ef |∪| U_f
      unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff s1-def by
argo
    hence C-in-disj: C |∈| N |∪| U_r |∪| U_ef ∨ C |∈| U_f
      by simp

    show ?thesis
    proof (cases C' = efac C')
      case True
      let ?s2' = (N, (U_r, finsert C' U_ef))

      have ord-res-2-step⁺⁺ s2 ?s2'
      proof (rule tranclp.r-into-trancl)
        show ord-res-2-step s2 (N, U_r, finsert C' U_ef)
          using C-in-disj
        proof (elim disjE)
          assume C |∈| N |∪| U_r |∪| U_ef
          show ?thesis
            unfolding s2-def
          proof (intro ord-res-2-step.intros ord-res-2.factoring)
            show is-least-false-clause (N |∪| U_r |∪| U_ef) C
              using is-least-false-clause-if-is-least-false-clause-in-union-unproductive[
                  OF U_f-unproductive ‹C |∈| N |∪| U_r |∪| U_ef› C-least-false]
              unfolding is-least-false-clause-def .
          next
            show ord-res.is-maximal-lit L C
              using ‹ord-res.is-maximal-lit L C› .
          next
            show is-pos L
              using ‹is-pos L› .
          next
            show finsert C' U_ef = finsert (efac C) U_ef
              using True factoring ground-factoring-preserves-efac by metis
```

```
      qed
    next
      assume C |∈| U_f
      then obtain x where
        x |∈| N |∪| U_r |∪| U_ef and
        ord-res.ground-factoring^++ x C and
        C ≠ efac C and
        efac C |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x
        using U_f-spec by metis

      show ?thesis
        unfolding s2-def
      proof (intro ord-res-2-step.intros ord-res-2.factoring)
        have ‹efac C |∉| U_ef›
        proof (rule notI)
          have efac C ⪯_c C
            using efac-subset[of C] subset-implies-reflclp-multp by metis
          hence efac C ≺_c C
            using ‹C ≠ efac C› by order

          moreover assume efac C |∈| U_ef

          ultimately show False
            using C-least-false[unfolded is-least-false-clause-def
                linorder-cls.is-least-in-ffilter-iff]
            by (metis ‹C ≠ efac C› funionCI linorder-cls.not-less-iff-gr-or-eq
                ord-res.entailed-clause-stays-entailed set-mset-efac true-cls-def)
        qed
        thus is-least-false-clause (N |∪| U_r |∪| U_ef) x
          using ‹efac C |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x›
by argo
      next
        show ord-res.is-maximal-lit L x
          using ‹ord-res.ground-factoring^++ x C› ‹ord-res.is-maximal-lit L C›
          using ord-res.ground-factorings-preserves-maximal-literal
          by (metis tranclp-into-rtranclp)
      next
        show is-pos L
          using ‹is-pos L› .
      next
        show finsert C' U_ef = finsert (efac x) U_ef
        using ‹ord-res.ground-factoring^++ x C› ‹ord-res.ground-factoring C C'›
      using True ground-factorings-preserves-efac ground-factoring-preserves-efac
          by (metis tranclp-into-rtranclp)
      qed
    qed
  qed

  moreover have ?match s1' ?s2'
```

```
    proof −
      have s1' = N |∪| U_r |∪| finsert C' U_ef |∪| U_f
        unfolding ‹s1' = finsert C' s1› s1-def by simp

      moreover have ∃ C |∈| N |∪| U_r |∪| finsert C' U_ef.
        ord-res.ground-factoring^++ C C_f ∧ C_f ≠ efac C_f ∧
       (efac C_f |∈| finsert C' U_ef ∨ is-least-false-clause (N |∪| U_r |∪| finsert C'
U_ef) C)
        if C_f |∈| U_f for C_f
      proof −
        obtain x where
          x |∈| N |∪| U_r |∪| U_ef and
          ord-res.ground-factoring^++ x C_f and
          C_f ≠ efac C_f and
          efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x
          using ‹C_f |∈| U_f› U_f-spec by metis

        show ?thesis
        proof (intro bexI conjI)
          show x |∈| N |∪| U_r |∪| finsert C' U_ef
            using ‹x |∈| N |∪| U_r |∪| U_ef› by simp
        next
          show ord-res.ground-factoring^++ x C_f
            using ‹ord-res.ground-factoring^++ x C_f› .
        next
          show C_f ≠ efac C_f
            using ‹C_f ≠ efac C_f› .
        next
           show efac C_f |∈| finsert C' U_ef ∨ is-least-false-clause (N |∪| U_r |∪|
finsert C' U_ef) x
            using ‹efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x›
          proof (elim disjE)
            assume efac C_f |∈| U_ef
            thus ?thesis
              by simp
          next
            assume is-least-false-clause (N |∪| U_r |∪| U_ef) x
            show ?thesis
            proof (cases C' = efac x)
              case True
              moreover have efac x = efac C_f
           using ‹ord-res.ground-factoring^++ x C_f› ground-factorings-preserves-efac
                by (metis tranclp-into-rtranclp)
              ultimately show ?thesis
                by simp
            next
              case False
              show ?thesis
                using C-in-disj
```

```
            proof (elim disjE)
              assume C |∈| N |∪| U_r |∪| U_ef
              then show ?thesis
                by (smt (verit) C-least-false True U_f-unproductive
               ‹is-least-false-clause (N |∪| U_r |∪| U_ef) x› ‹ord-res.ground-factoring^++
x C_f›
               finsert-iff ground-factoring-preserves-efac ground-factorings-preserves-efac
                     linorder-cls.Uniq-is-least-in-fset local.factoring(4)
                     is-least-false-clause-def
                    is-least-false-clause-if-is-least-false-clause-in-union-unproductive
                     the1-equality' tranclp-into-rtranclp)
            next
              assume C |∈| U_f
              then show ?thesis
                using C-least-false
          using is-least-false-clause-if-is-least-false-clause-in-union-unproductive[
                  OF U_f-unproductive]
                by (smt (z3) True U_f-spec ‹is-least-false-clause (N |∪| U_r |∪|
U_ef) x›
                  ‹ord-res.ground-factoring^++ x C_f› finsert-absorb finsert-iff
                 ground-factoring-preserves-efac ground-factorings-preserves-efac
                  linorder-cls.Uniq-is-least-in-fset local.factoring(4)
                  is-least-false-clause-def the1-equality' tranclp-into-rtranclp)
            qed
          qed
        qed
      qed
    qed

    ultimately show ?thesis
      by auto
  qed

  ultimately show ?thesis
    by metis
next
  case False
  let ?U_f' = finsert C' U_f

  have ?match s1' s2
  proof −
    have finsert C' s1 = N |∪| U_r |∪| U_ef |∪| ?U_f'
      unfolding s1-def by simp

    moreover have ∃ C |∈| N |∪| U_r |∪| U_ef.
      ord-res.ground-factoring^++ C C_f ∧ C_f ≠ efac C_f ∧
      (efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) C)
      if C_f |∈| ?U_f' for C_f
    proof −
```

```
      from ‹C_f |∈| ?U_f'› have C_f = C' ∨ C_f |∈| U_f
        by simp
      thus ?thesis
      proof (elim disjE)
        assume C_f = C'
        thus ?thesis
          using C-in-disj
        proof (elim disjE)
          assume C |∈| N |∪| U_r |∪| U_ef
          show ?thesis
          proof (intro bexI conjI)
            show C |∈| N |∪| U_r |∪| U_ef
              using ‹C |∈| N |∪| U_r |∪| U_ef› .
          next
            show ord-res.ground-factoring^++ C C_f
              using ‹ord-res.ground-factoring C C'› ‹C_f = C'› by simp
          next
            show C_f ≠ efac C_f
              using False ‹C_f = C'› by argo
          next
            have is-least-false-clause (N |∪| U_r |∪| U_ef) C
              using factoring
           using Interp-eq ‹C |∈| N |∪| U_r |∪| U_ef› linorder-cls.is-least-in-ffilter-iff
              by (simp add: s1-def is-least-false-clause-def)
            thus efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) C ..
          qed
        next
          assume C |∈| U_f
          then obtain x where
            x |∈| N |∪| U_r |∪| U_ef and
            ord-res.ground-factoring^++ x C and
            C ≠ efac C and
            efac C |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x
            using U_f-spec by metis

          show ?thesis
          proof (intro bexI conjI)
            show x |∈| N |∪| U_r |∪| U_ef
              using ‹x |∈| N |∪| U_r |∪| U_ef› .
          next
            show ord-res.ground-factoring^++ x C_f
             using ‹ord-res.ground-factoring^++ x C› ‹ord-res.ground-factoring C
C'› ‹C_f = C'›
              by simp
          next
            show C_f ≠ efac C_f
              using False ‹C_f = C'› by argo
          next
            show efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x
```

```
            using ‹efac C |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x›
          proof (elim disjE)
            assume efac C |∈| U_ef

            moreover have efac C = efac C_f
              unfolding ‹C_f = C'›
          using ‹ord-res.ground-factoring C C'› ground-factoring-preserves-efac
by metis

            ultimately show ?thesis
              by argo
          next
            assume is-least-false-clause (N |∪| U_r |∪| U_ef) x
            thus ?thesis
              by argo
          qed
        qed
      qed
    next
      assume C_f |∈| U_f
      thus ?thesis
        using U_f-spec by metis
    qed
  qed

 ultimately have ord-res-1-matches-ord-res-2 (finsert C' s1) (N, (U_r, U_ef))
    unfolding ord-res-1-matches-ord-res-2.simps by metis
  thus ?thesis
    unfolding s2-def ‹s1' = finsert C' s1› by simp
qed

moreover have ?order (?measure s1') (?measure s1)
proof −
  have ?measure s1 = C
    unfolding ord-res-1-measure-def
    using C-least-false[folded s1-def]
    by (metis (mono-tags, lifting) linorder-cls.Uniq-is-least-in-fset
        is-least-false-clause-def the1-equality' the-equality)

  moreover have ?measure s1' = C'
  proof −
    have C' ≺_c C
      using factoring ord-res.ground-factoring-smaller-conclusion by metis

    have unproductive: ∀x∈{C'}. ord-res.production (fset s1 ∪ {C'}) x = {}
      using ‹C' ≠ efac C'›
      by (simp add: nex-strictly-maximal-pos-lit-if-neq-efac
          unproductive-if-nex-strictly-maximal-pos-lit)
```

```
        have Interp-eq: ⋀D. ord-res-Interp (fset s1) D = ord-res-Interp (fset
(finsert C' s1)) D
          using Interp-union-unproductive[of fset s1 {C'}, folded union-fset,
              OF finite-fset - unproductive]
          by simp

        have is-least-false-clause (finsert C' s1) C'
          unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
        proof (intro conjI ballI impI)
          have ¬ ord-res-Interp (fset s1) C ⊨ C
            using C-least-false s1-def is-least-false-clause-def
              linorder-cls.is-least-in-ffilter-iff by simp
          thus ¬ ord-res-Interp (fset (finsert C' s1)) C' ⊨ C'
            by (metis Interp-eq ‹C' ≺c C› local.factoring(4)
                ord-res.entailed-clause-stays-entailed
             ord-res.set-mset-eq-set-mset-if-ground-factoring subset-refl true-cls-mono)
        next
          fix y
          assume y |∈| finsert C' s1 and y ≠ C' and
            y-false: ¬ ord-res-Interp (fset (finsert C' s1)) y ⊨ y
          hence y |∈| s1
            by simp

          moreover have ¬ ord-res-Interp (fset s1) y ⊨ y
            using y-false
            unfolding Interp-eq .

          ultimately have C ⪯c y
            using C-least-false[folded s1-def, unfolded is-least-false-clause-def]
            unfolding linorder-cls.is-least-in-ffilter-iff
            by force
          thus C' ≺c y
            using ‹C' ≺c C› by order
        qed simp
        thus ?thesis
          unfolding ord-res-1-measure-def ‹s1' = finsert C' s1›
          by (metis (mono-tags, lifting) linorder-cls.Uniq-is-least-in-fset
              is-least-false-clause-def the1-equality' the-equality)
      qed

      moreover have C' ⊂# C
        using factoring ord-res.strict-subset-mset-if-ground-factoring by metis

      ultimately show ?thesis
        unfolding s1-def by simp
    qed

    ultimately show ?thesis
      by argo
```

```
    qed
  next
    case (resolution C L D CD)

    have is-least-false-clause s1 C
      using resolution unfolding is-least-false-clause-def by argo
    hence
      C |∈| s1 and
      ¬ ord-res-Interp (fset s1) C ⊨ C and
      ∀ x |∈| s1. ¬ ord-res-Interp (fset s1) x ⊨ x ⟶ x ≠ C ⟶ C ≺_c x
     unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff by simp-all

    have C |∉| U_f
    proof (rule notI)
      assume C |∈| U_f
      then show False
        by (metis U_f-spec Uniq-D is-pos-def linorder-lit.Uniq-is-maximal-in-mset
local.resolution(2)
          local.resolution(3) efac-spec)
    qed
    hence C |∈| N |∪| U_r |∪| U_ef
      using ‹C |∈| s1› by (simp add: s1-def)

    have C-least-false: is-least-false-clause (N |∪| U_r |∪| U_ef |∪| U_f) C
      using resolution s1-def by metis
    hence C-least-false': is-least-false-clause (N |∪| U_r |∪| U_ef) C
      using is-least-false-clause-if-is-least-false-clause-in-union-unproductive[
        OF U_f-unproductive ‹C |∈| N |∪| U_r |∪| U_ef›] by argo

    define s2' where
      s2' = (N, (finsert CD U_r, U_ef))

    have ord-res-2-step^++ s2 s2'
    proof −
      have D |∉| U_f
      proof (rule notI)
        assume D |∈| U_f
        thus False
          using ‹ord-res.production (fset s1) D = {atm-of L}›
          using U_f-unproductive s1-def by simp
      qed
      hence D-in: D |∈| N |∪| U_r |∪| U_ef
        using ‹D |∈| s1›[unfolded s1-def] by simp

      have ord-res-2 N (U_r, U_ef) (finsert CD U_r, U_ef)
      proof (rule ord-res-2.resolution)
        show is-least-false-clause (N |∪| U_r |∪| U_ef) C
          using C-least-false' .
      next
```

show *ord-res.is-maximal-lit L C*
using *resolution* **by** *argo*
next
show *is-neg L*
using *resolution* **by** *argo*
next
show D $|\in|$ N $|\cup|$ U_r $|\cup|$ U_{ef}
using *D-in* **.**
next
show $D \prec_c C$
using *resolution* **by** *argo*
next
show *ord-res.production* (*fset* (N $|\cup|$ U_r $|\cup|$ U_{ef})) $D = \{$*atm-of L*$\}$
using *resolution*
unfolding *s1-def*
using *production-union-unproductive*[*OF finite-fset finite-fset - D-in*] U_f*-unproductive*
by (*metis* (*no-types*, *lifting*) *union-fset*)
next
show *ord-res.ground-resolution C D CD*
using *resolution* **by** *argo*
qed *simp-all*
thus *?thesis*
by (*auto simp*: *s2-def s2'-def ord-res-2-step.simps*)
qed

moreover have *?match s1' s2'*
proof −
have *finsert CD* (N $|\cup|$ U_r $|\cup|$ U_{ef} $|\cup|$ U_f) = N $|\cup|$ *finsert CD* U_r $|\cup|$ U_{ef} $|\cup|$ U_f
by *simp*

moreover have $\exists\, C$ $|\in|$ N $|\cup|$ *finsert CD* U_r $|\cup|$ U_{ef}.
ord-res.ground-factoring^{++} C C_f $\wedge$ $C_f \neq$ *efac* C_f $\wedge$
(*efac* C_f $|\in|$ U_{ef} $\vee$ *is-least-false-clause* (N $|\cup|$ *finsert CD* U_r $|\cup|$ U_{ef}) C)
if C_f $|\in|$ U_f **for** C_f
proof −
obtain x **where**
x $|\in|$ N $|\cup|$ U_r $|\cup|$ U_{ef} **and**
ord-res.ground-factoring^{++} x C_f **and**
$C_f \neq$ *efac* C_f **and**
efac C_f $|\in|$ U_{ef} $\vee$ *is-least-false-clause* (N $|\cup|$ U_r $|\cup|$ U_{ef}) x
using ‹C_f $|\in|$ U_f› U_f*-spec* **by** *metis*
show *?thesis*
proof (*intro bexI conjI*)
show x $|\in|$ N $|\cup|$ *finsert CD* U_r $|\cup|$ U_{ef}
using ‹x $|\in|$ N $|\cup|$ U_r $|\cup|$ U_{ef}› **by** *simp*
next
show *ord-res.ground-factoring*$^{++}$ x C_f

```
        using ‹ord-res.ground-factoring⁺⁺ x C_f› .
      next
        show C_f ≠ efac C_f
          using ‹C_f ≠ efac C_f› .
      next
        show ‹efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| finsert CD U_r |∪|
U_ef) x›
          using ‹efac C_f |∈| U_ef ∨ is-least-false-clause (N |∪| U_r |∪| U_ef) x› ‹x
|∈| N |∪| U_r |∪| U_ef›
        by (metis (no-types, lifting) C-least-false' Uniq-D ‹ord-res.ground-factoring⁺⁺
x C_f›
              is-least-false-clause-def is-pos-def linorder-cls.Uniq-is-least-in-fset
            linorder-lit.Uniq-is-maximal-in-mset local.resolution(2) local.resolution(3)
              ord-res.ground-factoring.cases tranclpD)
      qed
    qed

    ultimately show ?thesis
      unfolding s1-def resolution s2'-def by auto
  qed

  ultimately show ?thesis
    by metis
 qed
qed

theorem bisimulation-ord-res-1-ord-res-2:
 defines match ≡ λi s1 s2. i = ord-res-1-measure s1 ∧ ord-res-1-matches-ord-res-2
s1 s2
 shows ∃(MATCH :: nat × nat ⇒ 'f ord-res-1-state ⇒ 'f ord-res-2-state ⇒ bool)
\mathcal{R}_f \mathcal{R}_b.
   bisimulation ord-res-1 ord-res-1-final ord-res-2-step ord-res-2-final MATCH \mathcal{R}_f
\mathcal{R}_b
proof (rule ex-bisimulation-from-forward-simulation)
 show right-unique ord-res-1
   using right-unique-ord-res-1 .
next
 show right-unique ord-res-2-step
   using right-unique-ord-res-2-step .
next
 show ∀ s1. ord-res-1-final s1 ⟶ (∄ s1'. ord-res-1 s1 s1')
   using ord-res-1-semantics.final-finished
   by (simp add: finished-def)
next
 show ∀ s2. ord-res-2-final s2 ⟶ (∄ s2'. ord-res-2-step s2 s2')
   using ord-res-2-semantics.final-finished
   by (simp add: finished-def)
next
 show ∀ i s1 s2. match i s1 s2 ⟶ ord-res-1-final s1 = ord-res-2-final s2
```

using *ord-res-1-final-iff-ord-res-2-final*
by (*simp add*: *match-def*)
next
show $\forall$ *i s1 s2*. *match i s1 s2* $\longrightarrow$
safe-state ord-res-1 ord-res-1-final s1 $\wedge$
safe-state ord-res-2-step ord-res-2-final s2
proof (*intro allI impI*)
fix *i s1 S2*
assume *match i s1 S2*

then obtain *N s2* **where**
S2-def: *S2* = (*N*, *s2*) **and**
i = *ord-res-1-measure s1* **and**
match: *ord-res-1-matches-ord-res-2 s1 S2*
unfolding *match-def*
by (*metis prod.exhaust*)

show *safe-state ord-res-1 ord-res-1-final s1* $\wedge$ *safe-state ord-res-2-step ord-res-2-final S2*
using *safe-states-if-ord-res-1-matches-ord-res-2*[*OF match*] **.**
qed
next
show *wfP* ($\subset\#$)
using *wfp-subset-mset* **.**
next
show $\forall$ *i s1 s2 s1'*. *match i s1 s2* $\longrightarrow$ *ord-res-1 s1 s1'* $\longrightarrow$
($\exists$ *i' s2'*. *ord-res-2-step*$^{++}$ *s2 s2'* $\wedge$ *match i' s1' s2'*) $\vee$ ($\exists$ *i'*. *match i' s1' s2* $\wedge$ *i'* $\subset\#$ *i*)
proof (*intro allI impI*)
fix *i s1 S2 s1'*
assume *match i s1 S2*
then obtain *N s2* **where**
S2-def: *S2* = (*N*, *s2*) **and** *i* = *ord-res-1-measure s1* **and** *ord-res-1-matches-ord-res-2 s1 S2*
unfolding *match-def*
by (*metis prod.exhaust*)

moreover assume *ord-res-1 s1 s1'*

ultimately have ($\exists$ *S2'*. *ord-res-2-step*$^{++}$ *S2 S2'* $\wedge$ *ord-res-1-matches-ord-res-2 s1' S2'*) $\vee$
ord-res-1-matches-ord-res-2 s1' S2 $\wedge$ *ord-res-1-measure s1'* $\subset\#$ *ord-res-1-measure s1*
using *forward-simulation* **by** *metis*

thus ($\exists$ *i' S2'*. *ord-res-2-step*$^{++}$ *S2 S2'* $\wedge$ *match i' s1' S2'*) $\vee$ ($\exists$ *i'*. *match i' s1' S2* $\wedge$ *i'* $\subset\#$ *i*)
unfolding *S2-def prod.case*
using *lift-tranclp-to-pairs-with-constant-fst*[*of ord-res-2 N s2*]

by (*metis* (*mono-tags*, *lifting*) ‹*i* = *ord-res-1-measure s1*› *match-def*)
qed
qed

end

29 ORD-RES-3 (full resolve)

type-synonym *'f ord-res-3-state* = *'f gclause fset* × *'f gclause fset* × *'f gclause fset*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-2-matches-ord-res-3* :: - ⇒ *'f ord-res-3-state* ⇒ *bool* **where**
(∀ *C* |∈| U_{pr}. ∃ *D1* |∈| *N* |∪| U_{er} |∪| U_{ef}. ∃ *D2* |∈| *N* |∪| U_{er} |∪| U_{ef}.
(*ground-resolution D1*)$^{++}$ *D2 C* ∧ *C* ≠ *eres D1 D2* ∧ *eres D1 D2* |∈| U_{er})
⟹
ord-res-2-matches-ord-res-3 (*N*, (*U*$_{pr}$ |∪| U_{er}, U_{ef})) (*N*, (U_{er}, U_{ef}))

lemma *ord-res-2-final-iff-ord-res-3-final*:
assumes *match*: *ord-res-2-matches-ord-res-3* S_2 S_3
shows *ord-res-2-final* S_2 ⟷ *ord-res-3-final* S_3
using *match*
proof (*cases* S_2 S_3 *rule*: *ord-res-2-matches-ord-res-3.cases*)
case *match-hyps*: (*1* U_{pr} *N* U_{er} U_{ef})

note *invars* = *match-hyps*(*3*−)

have U_{pr}*-spec*: ∀ *C*|∈|U_{pr}. ∃ *D1*|∈|*N* |∪| U_{er} |∪| U_{ef}. ∃ *D2*|∈|*N* |∪| U_{er} |∪| U_{ef}.
(*ground-resolution D1*)$^{++}$ *D2 C* ∧ *C* ≠ *eres D1 D2* ∧ *eres D1 D2* |∈| U_{er}
using *invars* **by** *argo*

have *least-false-spec*: *is-least-false-clause* (*N* |∪| U_{pr} |∪| U_{er} |∪| U_{ef}) =
is-least-false-clause (*N* |∪| U_{er} |∪| U_{ef})
using *invars is-least-false-clause-conv-if-partial-resolution-invariant* **by** *metis*

have U_{pr}*-unproductive*: ∀ *C* |∈| U_{pr}. *ord-res.production* (*fset* (*N* |∪| U_{er} |∪| U_{ef} |∪| U_{pr})) *C* = {}
proof (*intro ballI*)
fix *C*
assume *C* |∈| U_{pr}
hence ∄ *L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L C*
using U_{pr}*-spec*
by (*metis eres-eq-after-tranclp-ground-resolution nex-strictly-maximal-pos-lit-if-neq-eres*)
thus *ord-res.production* (*fset* (*N* |∪| U_{er} |∪| U_{ef} |∪| U_{pr})) *C* = {}
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

hence *Interp-N-U_r-U_{ef}-eq-Interp-N-U_{er}-U_{ef}*: $\bigwedge C$.
ord-res-Interp (*fset* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef})) C =
ord-res-Interp (*fset* (N |∪| U_{er} |∪| U_{ef})) C
using *Interp-union-unproductive*[*OF finite-fset finite-fset*, *folded union-fset*,
of U_{pr} N |∪| U_{er} |∪| U_{ef}]
by (*simp add*: *funion-left-commute sup-commute*)

have *ex-false-clause* (*fset* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef})) ⟷
ex-false-clause (*fset* (N |∪| U_{er} |∪| U_{ef}))
proof (*rule iffI*)
assume *ex-false-clause* (*fset* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}))
then obtain C **where** *is-least-false-clause* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}) C
using *obtains-least-false-clause-if-ex-false-clause* **by** *metis*
thus *ex-false-clause* (*fset* (N |∪| U_{er} |∪| U_{ef}))
using *least-false-spec ex-false-clause-iff* **by** *metis*
next
assume *ex-false-clause* (*fset* (N |∪| U_{er} |∪| U_{ef}))
thus *ex-false-clause* (*fset* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}))
unfolding *ex-false-clause-def*
unfolding *Interp-N-U_r-U_{ef}-eq-Interp-N-U_{er}-U_{ef}*
by *auto*
qed

moreover have {#} |∈| N |∪| U_{pr} |∪| U_{er} |∪| U_{ef} ⟷ {#} |∈| N |∪| U_{er} |∪| U_{ef}
proof (*rule iffI*)
assume {#} |∈| N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}
hence {#} |∈| N |∪| U_{ef} ∨ {#} |∈| U_{pr} |∪| U_{er}
by *auto*
thus {#} |∈| N |∪| U_{er} |∪| U_{ef}
proof (*elim disjE*)
assume {#} |∈| N |∪| U_{ef}
thus *?thesis*
by *auto*
next
have {#} |∉| U_{pr}
using *U_{pr}-spec*[*rule-format*, *of* {#}]
by (*metis eres-eq-after-tranclp-ground-resolution eres-mempty-right*)
moreover assume {#} |∈| U_{pr} |∪| U_{er}
ultimately show *?thesis*
by *simp*
qed
next
assume {#} |∈| N |∪| U_{er} |∪| U_{ef}
then show {#} |∈| N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}
by *auto*
qed

ultimately have *ord-res-final* (N |∪| U_{pr} |∪| U_{er} |∪| U_{ef}) = *ord-res-final* (N

$|\cup|\ U_{er}\ |\cup|\ U_{ef})$
unfolding *ord-res-final-def* **by** *argo*

thus *ord-res-2-final* $S_2 \longleftrightarrow$ *ord-res-3-final* S_3
unfolding *match-hyps*(*1*,*2*)
by (*simp add*: *ord-res-2-final.simps ord-res-3-final.simps sup-assoc*)
qed

definition *ord-res-2-measure* **where**
ord-res-2-measure S1 =
(*let* (*N*, (U_r, U_{ef})) = *S1 in*
(*if* ∃ *C*. *is-least-false-clause* (*N* $|\cup|$ U_r $|\cup|$ U_{ef}) *C then*
The (*is-least-false-clause* (*N* $|\cup|$ U_r $|\cup|$ U_{ef}))
else
{#}))

definition *resolvent-at* **where**
resolvent-at C D i = (*THE CD*. (*ground-resolution C* ^^ *i*) *D CD*)

lemma *resolvent-at-0*[*simp*]: *resolvent-at C D 0* = *D*
by (*simp add*: *resolvent-at-def*)

lemma *resolvent-at-less-cls-resolvent-at*:
assumes *reso-at*: (*ground-resolution C* ^^ *n*) *D CD*
assumes $i < j$ **and** $j \leq n$
shows *resolvent-at C D j* $\prec_c$ *resolvent-at C D i*
proof −
obtain j' **where**
$j = i + Suc\ j'$
using ‹$i < j$› **by** (*metis less-iff-Suc-add nat-arith.suc1*)

obtain n' **where**
$n = j + n'$
using ‹$j \leq n$› **by** (*metis le-add-diff-inverse*)

obtain CD_i CD_j CD_n **where**
(*ground-resolution C* ^^ *i*) *D* CD_i **and**
(*ground-resolution C* ^^ *Suc j'*) CD_i CD_j
(*ground-resolution C* ^^ n') CD_j CD_n
using *reso-at* ‹$n = j + n'$› ‹$j = i + Suc\ j'$› **by** (*metis relpowp-plusD*)

have ∗: *resolvent-at C D i* = CD_i
unfolding *resolvent-at-def*
using ‹(*ground-resolution C* ^^ *i*) *D* CD_i›
by (*simp add*: *Uniq-ground-resolution Uniq-relpowp the1-equality'*)

have (*ground-resolution C* ^^ *j*) *D* CD_j
unfolding ‹$j = i + Suc\ j'$›
using ‹(*ground-resolution C* ^^ *i*) *D* CD_i› ‹(*ground-resolution C* ^^ *Suc j'*)

CD_i CD_j›
  by (*metis relpowp-trans*)
 hence ∗∗: *resolvent-at C D j* = CD_j
  unfolding *resolvent-at-def*
  by (*simp add*: *Uniq-ground-resolution Uniq-relpowp the1-equality′*)

 have (*ground-resolution C*)$^{++}$ CD_i CD_j
  using ‹(*ground-resolution C* ^^ *Suc j′*) CD_i CD_j›
  by (*metis Zero-not-Suc tranclp-if-relpowp*)
 hence $CD_j \prec_c CD_i$
  using *resolvent-lt-right-premise-if-tranclp-ground-resolution* **by** *metis*
 thus *?thesis*
  unfolding ∗ ∗∗ **.**
qed

lemma
 assumes *reso-at*: (*ground-resolution C* ^^ *n*) *D CD* **and** *i* < *n*
 shows
  left-premisse-lt-resolvent-at: *C* ≺$_c$ *resolvent-at C D i* **and**
  max-lit-resolvent-at:
    ord-res.is-maximal-lit L D ⟹ *ord-res.is-maximal-lit L* (*resolvent-at C D i*)
and
  nex-pos-strictly-max-lit-in-resolvent-at:
   ∄*L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L* (*resolvent-at C D i*) **and**
  ground-resolution-resolvent-at-resolvent-at-Suc:
   ground-resolution C (*resolvent-at C D i*) (*resolvent-at C D* (*Suc i*)) **and**
  relpowp-to-resolvent-at: (*ground-resolution C* ^^ *i*) *D* (*resolvent-at C D i*)
proof −
 obtain *j* **where** *n-def*: *n* = *i* + *Suc j*
  using ‹*i* < *n*› *less-natE* **by** *auto*

 obtain *CD′* **where** (*ground-resolution C* ^^ *i*) *D CD′* **and** (*ground-resolution C* ^^ *Suc j*) *CD′ CD*
  using *reso-at n-def* **by** (*metis relpowp-plusD*)

 have *resolvent-at C D i* = *CD′*
  unfolding *resolvent-at-def*
  using ‹(*ground-resolution C* ^^ *i*) *D CD′*›
  by (*simp add*: *Uniq-ground-resolution Uniq-relpowp the1-equality′*)

 have *C* ≺$_c$ *CD′*
 proof (*rule left-premise-lt-right-premise-if-tranclp-ground-resolution*)
  show (*ground-resolution C*)$^{++}$ *CD′ CD*
   using ‹(*ground-resolution C* ^^ *Suc j*) *CD′ CD*›
   by (*metis Zero-not-Suc tranclp-if-relpowp*)
 qed
 thus *C* ≺$_c$ *resolvent-at C D i*
  unfolding ‹*resolvent-at C D i* = *CD′*› **by** *argo*

show *ord-res.is-maximal-lit L* (*resolvent-at C D i*) **if** *ord-res.is-maximal-lit L D*
unfolding ‹*resolvent-at C D i = CD'*›
using *that*
using ‹(*ground-resolution C* ^^ *i*) *D CD'*›
by (*smt* (*verit*, *ccfv-SIG*) *Uniq-ground-resolution Uniq-relpowp Zero-not-Suc*
‹⋀*thesis*. (⋀*CD'*. ⟦(*ground-resolution C* ^^ *i*) *D CD'*; (*ground-resolution C* ^^ *Suc j*) *CD' CD*⟧ ⟹ *thesis*) ⟹ *thesis*›
linorder-lit.Uniq-is-greatest-in-mset linorder-lit.Uniq-is-maximal-in-mset literal.sel(*1*)
n-def relpowp-ground-resolutionD reso-at the1-equality' zero-eq-add-iff-both-eq-0)

show ∄*L*. *is-pos L* ∧ *ord-res.is-strictly-maximal-lit L* (*resolvent-at C D i*)
unfolding ‹*resolvent-at C D i = CD'*›
by (*metis Zero-not-Suc* ‹(*ground-resolution C* ^^ *Suc j*) *CD' CD*›
nex-strictly-maximal-pos-lit-if-resolvable tranclpD tranclp-if-relpowp)

show *ground-resolution C* (*resolvent-at C D i*) (*resolvent-at C D* (*Suc i*))
proof −
obtain *CD''* **where** *ground-resolution C CD' CD''* **and** (*ground-resolution C* ^^ *j*) *CD'' CD*
using ‹(*ground-resolution C* ^^ *Suc j*) *CD' CD*› **by** (*metis relpowp-Suc-D2*)
hence (*ground-resolution C* ^^ *Suc i*) *D CD''*
using ‹(*ground-resolution C* ^^ *i*) *D CD'*› **by** *auto*
hence *resolvent-at C D* (*Suc i*) = *CD''*
unfolding *resolvent-at-def*
by (*meson Uniq-ground-resolution Uniq-relpowp the1-equality'*)

show *?thesis*
unfolding ‹*resolvent-at C D i = CD'*› ‹*resolvent-at C D* (*Suc i*) = *CD''*›
using ‹*ground-resolution C CD' CD''*› **.**
qed

show (*ground-resolution C* ^^ *i*) *D* (*resolvent-at C D i*)
using ‹(*ground-resolution C* ^^ *i*) *D CD'*› ‹*resolvent-at C D i = CD'*› **by** *argo*
qed

definition *resolvents-upto* **where**
resolvents-upto C D n = *resolvent-at C D* |‘| *fset-upto* (*Suc 0*) *n*

lemma *resolvents-upto-0*[*simp*]:
resolvents-upto C D 0 = {||}
by (*simp add*: *resolvents-upto-def*)

lemma *resolvents-upto-Suc*[*simp*]:
resolvents-upto C D (*Suc n*) = *finsert* (*resolvent-at C D* (*Suc n*)) (*resolvents-upto C D n*)
by (*simp add*: *resolvents-upto-def*)

lemma *resolvent-at-fmember-resolvents-upto*:

assumes $k \neq 0$
shows *resolvent-at C D k* $|\in|$ *resolvents-upto C D k*
unfolding *resolvents-upto-def*
proof (*rule fimageI*)
show k $|\in|$ *fset-upto* (*Suc 0*) k
using *assms* **by** *simp*
qed

lemma *backward-simulation-2-to-3*:
fixes *match measure less*
defines *match* $\equiv$ *ord-res-2-matches-ord-res-3*
assumes
match: *match S2 S3* **and**
step2: *ord-res-3-step S3 S3'*
shows ($\exists$ *S2'*. *ord-res-2-step*$^{++}$ *S2 S2'* $\wedge$ *match S2' S3'*)
using *match*[*unfolded match-def*]
proof (*cases S2 S3 rule*: *ord-res-2-matches-ord-res-3.cases*)
case *match-hyps*: (*1* U_{pr} *N* U_{er} U_{ef})
note *invars* = *match-hyps*(*3*$-$)

have U_{pr}*-spec*: $\forall$ *C*$|\in|$$U_{pr}$. $\exists$ *D1*$|\in|$*N* $|\cup|$ U_{er} $|\cup|$ U_{ef}. $\exists$ *D2*$|\in|$*N* $|\cup|$ U_{er} $|\cup|$ U_{ef}.
(*ground-resolution D1*)$^{++}$ *D2 C* $\wedge$ *C* $\neq$ *eres D1 D2* $\wedge$ *eres D1 D2* $|\in|$ U_{er}
using *invars* **by** *argo*

hence *C-not-least-with-partial*: $\neg$ *is-least-false-clause* (*N* $|\cup|$ U_{pr} $|\cup|$ U_{er} $|\cup|$ U_{ef}) *C*
if *C-in*: *C* $|\in|$ U_{pr} **for** *C*
proof $-$
obtain *D1 D2* **where**
D1 $|\in|$ *N* $|\cup|$ U_{er} $|\cup|$ U_{ef} **and**
D2 $|\in|$ *N* $|\cup|$ U_{er} $|\cup|$ U_{ef} **and**
(*ground-resolution D1*)$^{++}$ *D2 C* **and**
C $\neq$ *eres D1 D2* **and**
eres D1 D2 $|\in|$ U_{er}
using U_{pr}*-spec C-in* **by** *metis*

have *eres D1 C* = *eres D1 D2*
using ‹(*ground-resolution D1*)$^{++}$ *D2 C*› *eres-eq-after-tranclp-ground-resolution* **by** *metis*
hence *eres D1 C* $\prec_c$ *C*
using *eres-le*[*of D1 C*] ‹*C* $\neq$ *eres D1 D2*› **by** *order*

show *?thesis*
proof (*cases ord-res-Interp* (*fset* (*N* $|\cup|$ U_{pr} $|\cup|$ U_{er} $|\cup|$ U_{ef})) (*eres D1 D2*) $\models$ *eres D1 D2*)
case *True*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) ‹(*ground-resolution D1*)$^{++}$ *D2 C*› ‹*eres D1 C*

$= \mathit{eres\ D1\ D2}\rangle$
clause-true-if-eres-true is-least-false-clause-def
linorder-cls.is-least-in-fset-ffilterD(2))
next
case *False*
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *Un-iff* ‹*eres D1 C = eres D1 D2*› ‹*eres D1 C* $\prec_c$ *C*›
‹*eres D1 D2* $|\in|$ U_{er}› *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff linorder-cls.not-less-iff-gr-or-eq sup-fset.rep-eq*)
qed
qed

have *least-false-conv*: *is-least-false-clause* (N $|\cup|$ U_{pr} $|\cup|$ U_{er} $|\cup|$ U_{ef}) =
is-least-false-clause (N $|\cup|$ U_{er} $|\cup|$ U_{ef})
using *invars is-least-false-clause-conv-if-partial-resolution-invariant* **by** *metis*

have U_{pr}*-unproductive*: $\bigwedge N.\ \forall\, C$ $|\in|$ U_{pr}. *ord-res.production N C* = {}
proof (*intro ballI*)
fix *C*
assume *C* $|\in|$ U_{pr}
hence $\exists\, D$ $|\in|$ N $|\cup|$ U_{er} $|\cup|$ U_{ef}. ($\exists\, C'$. *ground-resolution D C C'*)
using U_{pr}*-spec* **by** (*metis eres-eq-after-tranclp-ground-resolution resolvable-if-neq-eres*)
hence $\nexists\, L$. *is-pos L* $\wedge$ *ord-res.is-strictly-maximal-lit L C*
using *nex-strictly-maximal-pos-lit-if-resolvable* **by** *metis*
thus $\bigwedge N$. *ord-res.production N C* = {}
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

hence *Interp-N-*U_r*-*U_{ef}*-eq-Interp-N-*U_{er}*-*U_{ef}:
$\bigwedge C$. *ord-res-Interp* (*fset* (N $|\cup|$ U_{pr} $|\cup|$ U_{er} $|\cup|$ U_{ef})) *C* = *ord-res-Interp* (*fset* (N $|\cup|$ U_{er} $|\cup|$ U_{ef})) *C*
using *Interp-union-unproductive*[*OF finite-fset finite-fset*, *folded union-fset*, *of* U_{pr} N $|\cup|$ U_{er} $|\cup|$ U_{ef}]
by (*simp add*: *funion-left-commute sup-commute*)

have U_{pr}*-have-generalization*: $\forall$ *Ca* $|\in|$ U_{pr}. $\exists\, D$ $|\in|$ U_{er}. $D \prec_c Ca \wedge \{D\} \models e$ {*Ca*}
proof (*intro ballI*)
fix *Ca*
assume *Ca* $|\in|$ U_{pr}
then obtain *D1 D2* **where**
D1$|\in|$$N$ $|\cup|$ U_{er} $|\cup|$ U_{ef} **and**
D2$|\in|$$N$ $|\cup|$ U_{er} $|\cup|$ U_{ef} **and**
(*ground-resolution D1*)$^{++}$ *D2 Ca* **and**
Ca $\neq$ *eres D1 D2* **and**
eres D1 D2 $|\in|$ U_{er}
using U_{pr}*-spec* **by** *metis*

have *eres D1 D2 = eres D1 Ca*
using ‹(*ground-resolution D1*)$^{++}$ *D2 Ca*› *eres-eq-after-tranclp-ground-resolution*
by *metis*

show $\exists\, D \mid\in\mid U_{er}.\ D \prec_c Ca \wedge \{D\} \models e \{Ca\}$
proof (*intro bexI conjI*)
have *eres D1 Ca* $\preceq_c$ *Ca*
using *eres-le* **.**
thus *eres D1 D2* $\prec_c$ *Ca*
using ‹*Ca* $\neq$ *eres D1 D2*› ‹*eres D1 D2 = eres D1 Ca*› **by** *order*
next
show {*eres D1 D2*} $\models e$ {*Ca*}
using ‹(*ground-resolution D1*)$^{++}$ *D2 Ca*› *eres-entails-resolvent* **by** *metis*
next
show *eres D1 D2* $\mid\in\mid$ U_{er}
using ‹*eres D1 D2* $\mid\in\mid$ U_{er}› **by** *simp*
qed
qed

from *step2* **obtain** *s3′* **where** *S3′-def*: *S3′* = (*N*, *s3′*) **and** *ord-res-3 N* (U_{er}, U_{ef}) *s3′*
by (*auto simp*: *match-hyps*(*1*,*2*) *elim*: *ord-res-3-step.cases*)

show *?thesis*
using ‹*ord-res-3 N* (U_{er}, U_{ef}) *s3′*›
proof (*cases N* (U_{er}, U_{ef}) *s3′ rule*: *ord-res-3.cases*)
case (*factoring C L* U_{ef}')

define *S2′* **where**
S2′ = (*N*, (U_{pr} $\mid\cup\mid$ U_{er}, *finsert* (*efac C*) U_{ef}))

have *ord-res-2-step*$^{++}$ *S2 S2′*
unfolding *match-hyps*(*1*,*2*) *S2′-def*
proof (*intro tranclp.r-into-trancl ord-res-2-step.intros ord-res-2.factoring*)
show *is-least-false-clause* (*N* $\mid\cup\mid$ (U_{pr} $\mid\cup\mid$ U_{er}) $\mid\cup\mid$ U_{ef}) *C*
using ‹*is-least-false-clause* (*N* $\mid\cup\mid$ U_{er} $\mid\cup\mid$ U_{ef}) *C*›
using *least-false-conv*
by (*metis sup-assoc*)
next
show *ord-res.is-maximal-lit L C*
using *factoring* **by** *metis*
next
show *is-pos L*
using *factoring* **by** *metis*
next
show *finsert* (*efac C*) U_{ef} = *finsert* (*efac C*) U_{ef}
by *argo*
qed

moreover have *match S2′ S3′*
unfolding *S2′-def S3′-def*
unfolding *factoring*
unfolding *match-def*
proof (*rule ord-res-2-matches-ord-res-3.intros*)
show $\forall\, Ca|\in|U_{pr}.$
$\exists\, D1|\in|N\ |\cup|\ U_{er}\ |\cup|\ \mathit{finsert}\ (\mathit{efac}\ C)\ U_{ef}.\ \exists\, D2|\in|N\ |\cup|\ U_{er}\ |\cup|\ \mathit{finsert}\ (\mathit{efac}\ C)\ U_{ef}.$
$(\mathit{ground\text{-}resolution}\ D1)^{++}\ D2\ Ca \wedge Ca \neq \mathit{eres}\ D1\ D2 \wedge \mathit{eres}\ D1\ D2\ |\in|\ U_{er}$
using U_{pr}*-spec* **by** *auto*
qed

ultimately show *?thesis*
by *metis*
next
case (*resolution C L D* U_{rr}')

have $(\mathit{ground\text{-}resolution}\ D)^{**}\ C\ (\mathit{eres}\ D\ C) \nexists x.\ \mathit{ground\text{-}resolution}\ D\ (\mathit{eres}\ D\ C)\ x$
unfolding *atomize-conj*
by (*metis ex1-eres-eq-full-run-ground-resolution full-run-def*)

moreover have $\exists\, x.\ \mathit{ground\text{-}resolution}\ D\ C\ x$
unfolding *ground-resolution-def*
using *resolution*
by (*metis Neg-atm-of-iff ex-ground-resolutionI ord-res.mem-productionE singletonI*)

ultimately have $(\mathit{ground\text{-}resolution}\ D)^{++}\ C\ (\mathit{eres}\ D\ C)$
by (*metis rtranclpD*)

then obtain n **where** $(\mathit{ground\text{-}resolution}\ D \mathbin{\frown\frown} \mathit{Suc}\ n)\ C\ (\mathit{eres}\ D\ C)$
by (*metis not0-implies-Suc not-gr-zero tranclp-power*)

hence $\mathit{resolvent\text{-}at}\ D\ C\ (\mathit{Suc}\ n) = \mathit{eres}\ D\ C$
by (*metis Uniq-ground-resolution Uniq-relpowp resolvent-at-def the1-equality′*)

have *steps*: $k \leq \mathit{Suc}\ n \Longrightarrow (\mathit{ord\text{-}res\text{-}2\text{-}step} \mathbin{\frown\frown} k)$
$(N,\ U_{pr}\ |\cup|\ U_{er},\ U_{ef})\ (N,\ U_{pr}\ |\cup|\ U_{er}\ |\cup|\ \mathit{resolvents\text{-}upto}\ D\ C\ k,\ U_{ef})$ **for** k
proof (*induction k*)
case *0*
show *?case*
by *simp*
next
case (*Suc k*)
have $k < \mathit{Suc}\ n$
using ‹$\mathit{Suc}\ k \leq \mathit{Suc}\ n$› **by** *presburger*
hence $k \leq \mathit{Suc}\ n$

by *presburger*
hence (*ord-res-2-step* ^^ *k*) (*N*, U_{pr} |∪| U_{er}, U_{ef})
(*N*, U_{pr} |∪| U_{er} |∪| *resolvents-upto D C k*, U_{ef})
using *Suc.IH* **by** *metis*

moreover have *ord-res-2-step*
(*N*, U_{pr} |∪| U_{er} |∪| *resolvents-upto D C k*, U_{ef})
(*N*, U_{pr} |∪| U_{er} |∪| *resolvents-upto D C* (*Suc k*), U_{ef})
unfolding *resolvents-upto-Suc*
proof (*intro ord-res-2-step.intros ord-res-2.resolution*)
show *is-least-false-clause* (*N* |∪| (U_{pr} |∪| U_{er} |∪| *resolvents-upto D C k*) |∪| U_{ef})
(*resolvent-at D C k*)
using ‹*k* < *Suc n*›
proof (*induction k*)
case *0*
have *is-least-false-clause* (*N* |∪| U_{pr} |∪| U_{er} |∪| U_{ef}) *C*
using ‹*is-least-false-clause* (*N* |∪| U_{er} |∪| U_{ef}) *C*›
unfolding *least-false-conv* **.**
thus *?case*
unfolding *funion-fempty-right funion-assoc*[*symmetric*]
by *simp*
next
case (*Suc k'*)

have ⋀*x*. *ord-res-Interp* (*fset* (*N* |∪| (U_{pr} |∪| U_{er} |∪| *resolvents-upto D C* (*Suc k'*)) |∪| U_{ef})) *x* =
ord-res-Interp (*fset* (*N* |∪| U_{er} |∪| U_{ef}) ∪ *fset* (U_{pr} |∪| *resolvents-upto D C* (*Suc k'*))) *x*
by (*simp add*: *funion-left-commute sup-assoc sup-commute*)
also have ⋀*x*. *ord-res-Interp* (*fset* (*N* |∪| U_{er} |∪| U_{ef}) ∪ *fset* (U_{pr} |∪| *resolvents-upto D C* (*Suc k'*))) *x* =
ord-res-Interp (*fset* (*N* |∪| U_{er} |∪| U_{ef})) *x*
proof (*intro Interp-union-unproductive ballI*)
fix *x y* **assume** *y* |∈| U_{pr} |∪| *resolvents-upto D C* (*Suc k'*)
hence *y* |∈| U_{pr} ∨ *y* |∈| *resolvents-upto D C* (*Suc k'*)
by *blast*
thus *ord-res.production* (*fset* (*N* |∪| U_{er} |∪| U_{ef}) ∪ *fset* (U_{pr} |∪| *resolvents-upto D C* (*Suc k'*))) *y* = {}
proof (*elim disjE*)
assume *y* |∈| U_{pr}
thus *?thesis*
using U_{pr}-*unproductive* **by** *metis*
next
assume *y* |∈| *resolvents-upto D C* (*Suc k'*)
then obtain *i* **where** *i* |∈| *fset-upto* (*Suc 0*) (*Suc k'*) **and** *y* = *resolvent-at D C i*
unfolding *resolvents-upto-def* **by** *blast*

```
          have ∄L. is-pos L ∧ ord-res.is-strictly-maximal-lit L (resolvent-at D C
i)
          proof (rule nex-pos-strictly-max-lit-in-resolvent-at)
            show (ground-resolution D ^^ Suc n) C (eres D C)
              using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
          next
            have i ≤ Suc k'
              using ‹i |∈| fset-upto (Suc 0) (Suc k')› by auto
            thus i < Suc n
              using ‹Suc k' < Suc n› by presburger
          qed

          then show ?thesis
        using ‹y = resolvent-at D C i› unproductive-if-nex-strictly-maximal-pos-lit
            by metis
        qed
      qed simp-all
      finally have Interp-simp: ⋀x.
          ord-res-Interp (fset (N |∪| (U_pr |∪| U_er |∪| resolvents-upto D C (Suc
k')) |∪| U_ef)) x =
          ord-res-Interp (fset (N |∪| U_er |∪| U_ef)) x .

      show ?case
        unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
      proof (intro conjI ballI impI)
        have resolvent-at D C (Suc k') |∈| resolvents-upto D C (Suc k')
          using resolvent-at-fmember-resolvents-upto by simp
       thus resolvent-at D C (Suc k') |∈| N |∪| (U_pr |∪| U_er |∪| resolvents-upto
D C (Suc k')) |∪| U_ef
          by simp
      next

        show ¬ ord-res-Interp (fset (N |∪| (U_pr |∪| U_er |∪| resolvents-upto D
C (Suc k')) |∪| U_ef))
          (resolvent-at D C (Suc k')) ⟙ resolvent-at D C (Suc k')
          unfolding Interp-simp
          by (metis (no-types, lifting) Suc.prems Zero-not-Suc
          ‹(ground-resolution D ^^ Suc n) C (eres D C)› clause-true-if-resolved-true
              insert-not-empty is-least-false-clause-def
                    linorder-cls.is-least-in-fset-ffilterD(2) local.resolution(2) lo-
cal.resolution(7)
              relpowp-to-resolvent-at tranclp-if-relpowp)
      next
        fix y
        assume y ≠ resolvent-at D C (Suc k')
         assume ¬ ord-res-Interp (fset (N |∪| (U_pr |∪| U_er |∪| resolvents-upto
D C (Suc k')) |∪| U_ef)) y ⟙ y
        hence ¬ ord-res-Interp (fset (N |∪| U_er |∪| U_ef)) y ⟙ y
          unfolding Interp-simp .
```

```
      hence ¬ ord-res-Interp (fset (N |∪| U_pr |∪| U_er |∪| U_ef)) y ⊨ y
        using Interp-N-U_r-U_ef-eq-Interp-N-U_er-U_ef by metis

      assume y |∈| N |∪| (U_pr |∪| U_er |∪| resolvents-upto D C (Suc k')) |∪| U_ef
      hence y |∈| N |∪| U_pr |∪| U_er |∪| U_ef ∨ y |∈| resolvents-upto D C (Suc k')
        by auto
      thus resolvent-at D C (Suc k') ≺_c y
      proof (elim disjE)
        assume y |∈| N |∪| U_pr |∪| U_er |∪| U_ef
        have C ⪯_c y
        proof (cases y = C)
          case True
          thus ?thesis
            by order
        next
          case False
          thus ?thesis
            using ‹y |∈| N |∪| U_pr |∪| U_er |∪| U_ef›
            using ‹¬ ord-res-Interp (fset (N |∪| U_pr |∪| U_er |∪| U_ef)) y ⊨ y›
            using ‹is-least-false-clause (N |∪| U_er |∪| U_ef) C›
            unfolding least-false-conv[symmetric]
            unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
            by simp
        qed

        moreover have resolvent-at D C (Suc k') ≺_c C
          by (metis Suc.prems ‹(ground-resolution D ^^ Suc n) C (eres D C)›
              less-or-eq-imp-le
                resolvent-at-less-cls-resolvent-at resolvent-at-0 zero-less-Suc)

        ultimately show resolvent-at D C (Suc k') ≺_c y
          by order
      next
        assume y |∈| resolvents-upto D C (Suc k')
        then obtain i where
          i-in: i |∈| fset-upto (Suc 0) (Suc k') and y-def: y = resolvent-at D C i
          unfolding resolvents-upto-def by blast

        hence i < Suc k'
          using ‹y ≠ resolvent-at D C (Suc k')›
          by auto

        show resolvent-at D C (Suc k') ≺_c y
          unfolding y-def
        proof (rule resolvent-at-less-cls-resolvent-at)
          show (ground-resolution D ^^ Suc n) C (eres D C)
            using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
```

```
          next
            show i < Suc k'
              using ‹y ≠ resolvent-at D C (Suc k')› i-in y-def by auto
          next
            show Suc k' ≤ Suc n
              using ‹Suc k' < Suc n› by presburger
          qed
        qed
      qed
    qed
  next
    show ord-res.is-maximal-lit L (resolvent-at D C k)
    proof (rule max-lit-resolvent-at)
      show (ground-resolution D ^^ Suc n) C (eres D C)
        using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
    next
      show k < Suc n
        using ‹k < Suc n› .
    next
      show ord-res.is-maximal-lit L C
      using ‹ord-res.is-maximal-lit L C› .
    qed
  next
    show is-neg L
      using ‹is-neg L› .
  next
    show D |∈| N |∪| (U_pr |∪| U_er |∪| resolvents-upto D C k) |∪| U_ef
      using ‹D |∈| N |∪| U_er |∪| U_ef› by auto
  next
    show D ≺_c resolvent-at D C k
    proof (rule left-premisse-lt-resolvent-at)
      show (ground-resolution D ^^ Suc n) C (eres D C)
        using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
    next
      show k < Suc n
        using ‹k < Suc n› .
    qed
  next
    have ord-res.production (fset (N |∪| (U_pr |∪| U_er |∪| resolvents-upto D C
k) |∪| U_ef)) D =
      ord-res.production (fset (N |∪| U_er |∪| U_ef) ∪ fset (U_pr |∪| resolvents-upto
D C k)) D
      by (simp add: funion-left-commute sup-assoc sup-commute)
    also have ... = ord-res.production (fset (N |∪| U_er |∪| U_ef)) D
    proof (intro production-union-unproductive ballI)
      fix x
      assume x |∈| U_pr |∪| resolvents-upto D C k
      hence ∄ L. is-pos L ∧ ord-res.is-strictly-maximal-lit L x
        unfolding funion-iff
```

```
        proof (elim disjE)
          assume x |∈| U_pr
          thus ?thesis
            using U_pr-spec
        by (metis eres-eq-after-tranclp-ground-resolution nex-strictly-maximal-pos-lit-if-neq-eres)
        next
          assume x |∈| resolvents-upto D C k
        then obtain i where i |∈| fset-upto (Suc 0) k and x-def: x = resolvent-at
D C i
            unfolding resolvents-upto-def by auto

          have 0 < i and i ≤ k
            using ‹i |∈| fset-upto (Suc 0) k› by simp-all

          show ?thesis
            unfolding x-def
          proof (rule nex-pos-strictly-max-lit-in-resolvent-at)
            show (ground-resolution D ^^ Suc n) C (eres D C)
              using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
          next
            show i < Suc n
              using ‹i ≤ k› ‹k < Suc n› by presburger
          qed
        qed
        thus ord-res.production (fset (N |∪| U_er |∪| U_ef) ∪
          fset (U_pr |∪| resolvents-upto D C k)) x = {}
          using unproductive-if-nex-strictly-maximal-pos-lit by metis
      next
        show D |∈| N |∪| U_er |∪| U_ef
          using ‹D |∈| N |∪| U_er |∪| U_ef› .
      qed simp-all
    finally show ord-res.production (fset (N |∪| (U_pr |∪| U_er |∪| resolvents-upto
D C k) |∪| U_ef)) D =
        {atm-of L}
      using ‹ord-res.production (fset (N |∪| U_er |∪| U_ef)) D = {atm-of L}› by
argo
  next
    show ord-res.ground-resolution (resolvent-at D C k) D (resolvent-at D C
(Suc k))
      unfolding ground-resolution-def[symmetric]
    proof (rule ground-resolution-resolvent-at-resolvent-at-Suc)
      show (ground-resolution D ^^ Suc n) C (eres D C)
        using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
    next
      show k < Suc n
        using ‹k < Suc n› .
    qed
  next
    show U_pr |∪| U_er |∪| finsert (resolvent-at D C (Suc k)) (resolvents-upto D
```

$C\ k) =$
finsert (*resolvent-at D C* (*Suc k*)) (U_{pr} |∪| U_{er} |∪| *resolvents-upto D C k*)
by *simp*
qed

ultimately show *?case*
by (*meson relpowp-Suc-I*)
qed

hence (*ord-res-2-step* ^^ *Suc n*) *S2* (*N*, U_{pr} |∪| U_{er} |∪| *resolvents-upto D C* (*Suc n*), U_{ef})
unfolding *match-hyps*(*1*,*2*) **by** *blast*

moreover have *match* (*N*, U_{pr} |∪| U_{er} |∪| *resolvents-upto D C* (*Suc n*), U_{ef}) *S3'*
proof −
have *1*: *S3'* = (*N*, *finsert* (*eres D C*) U_{er}, U_{ef})
unfolding *S3'-def* ‹*s3'* = (U_{rr}', U_{ef})› ‹U_{rr}' = *finsert* (*eres D C*) U_{er}› **..**

have *2*: U_{pr} |∪| U_{er} |∪| *resolvents-upto D C* (*Suc n*) =
U_{pr} |∪| *resolvents-upto D C n* |∪| *finsert* (*eres D C*) U_{er}
by (*auto simp*: ‹*resolvent-at D C* (*Suc n*) = *eres D C*›)

show *?thesis*
unfolding *match-def 1 2*
proof (*rule ord-res-2-matches-ord-res-3.intros*)
show ∀ *E*|∈|U_{pr} |∪| *resolvents-upto D C n*.
∃ *D1*|∈|*N* |∪| *finsert* (*eres D C*) U_{er} |∪| U_{ef}. ∃ *D2*|∈|*N* |∪| *finsert* (*eres D C*) U_{er} |∪| U_{ef}.
(*ground-resolution D1*)$^{++}$ *D2 E* ∧ *E* ≠ *eres D1 D2* ∧ *eres D1 D2* |∈| *finsert* (*eres D C*) U_{er}
proof (*intro ballI*)
fix *Ca*
assume *Ca* |∈| U_{pr} |∪| *resolvents-upto D C n*
hence *Ca* |∈| U_{pr} ∨ *Ca* |∈| *resolvents-upto D C n*
by *simp*
thus ∃ *D1*|∈|*N* |∪| *finsert* (*eres D C*) U_{er} |∪| U_{ef}. ∃ *D2*|∈|*N* |∪| *finsert* (*eres D C*) U_{er} |∪| U_{ef}.
(*ground-resolution D1*)$^{++}$ *D2 Ca* ∧ *Ca* ≠ *eres D1 D2* ∧ *eres D1 D2* |∈| *finsert* (*eres D C*) U_{er}
proof (*elim disjE*)
show *Ca* |∈| U_{pr} ⟹ *?thesis*
using U_{pr}*-spec* **by** *auto*
next
assume *Ca* |∈| *resolvents-upto D C n*
then obtain *i* **where** *i-in*: *i* |∈| *fset-upto* (*Suc 0*) *n* **and** *Ca-def*:*Ca* = *resolvent-at D C i*
unfolding *resolvents-upto-def* **by** *auto*

```
          from i-in have 0 < i i ≤ n
            by simp-all

          show ?thesis
          proof (intro bexI conjI)
            have (ground-resolution D ^^ i) C Ca
              unfolding ‹Ca = resolvent-at D C i›
            proof (rule relpowp-to-resolvent-at)
              show (ground-resolution D ^^ Suc n) C (eres D C)
                using ‹(ground-resolution D ^^ Suc n) C (eres D C)› .
            next
              show i < Suc n
                using ‹i ≤ n› by presburger
            qed
            thus (ground-resolution D)^++ C Ca
              using ‹0 < i› by (simp add: tranclp-if-relpowp)
          next
            show Ca ≠ eres D C
              by (metis Ca-def ‹(ground-resolution D ^^ Suc n) C (eres D C)›
                ‹∄x. ground-resolution D (eres D C) x› ‹i ≤ n›
                ground-resolution-resolvent-at-resolvent-at-Suc less-Suc-eq-le)
          next
            show eres D C |∈| finsert (eres D C) U_er
              by simp
          next
            show D |∈| N |∪| finsert (eres D C) U_er |∪| U_ef
              using resolution by simp
          next
            have C |∈| N |∪| U_er |∪| U_ef
              using resolution
             by (simp add: is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff)
            thus C |∈| N |∪| finsert (eres D C) U_er |∪| U_ef
              by simp
          qed
        qed
      qed
    qed

    ultimately have ∃S2'. (ord-res-2-step ^^ Suc n) S2 S2' ∧ match S2' S3'
      by metis

    thus ∃S2'. ord-res-2-step^++ S2 S2' ∧ match S2' S3'
      by (metis Zero-neq-Suc tranclp-if-relpowp)
  qed
qed

lemma safe-states-if-ord-res-2-matches-ord-res-3:
  assumes match: ord-res-2-matches-ord-res-3 S_2 S_3
```

shows
safe-state ord-res-2-step ord-res-2-final S_2
safe-state ord-res-3-step ord-res-3-final S_3
proof −
show *safe-state ord-res-2-step ord-res-2-final* S_2
using *safe-state-if-all-states-safe ord-res-2-step-safe* **by** *metis*

show *safe-state ord-res-3-step ord-res-3-final* S_3
using *safe-state-if-all-states-safe ord-res-3-step-safe* **by** *metis*
qed

theorem *bisimulation-ord-res-2-ord-res-3*:
defines *match* ≡ λ- *S2 S3*. *ord-res-2-matches-ord-res-3 S2 S3*
shows ∃(*MATCH* :: *nat* × *nat* ⇒ *'f ord-res-2-state* ⇒ *'f ord-res-3-state* ⇒ *bool*) $\mathcal{R}_f$ $\mathcal{R}_b$.
bisimulation ord-res-2-step ord-res-2-final ord-res-3-step ord-res-3-final MATCH $\mathcal{R}_f$ $\mathcal{R}_b$
proof (*rule ex-bisimulation-from-backward-simulation*)
show *right-unique ord-res-2-step*
using *right-unique-ord-res-2-step* **.**
next
show *right-unique ord-res-3-step*
using *right-unique-ord-res-3-step* **.**
next
show ∀ *s1*. *ord-res-2-final s1* ⟶ (∄ *s1'*. *ord-res-2-step s1 s1'*)
by (*metis finished-def ord-res-2-semantics.final-finished*)
next
show ∀ *s2*. *ord-res-3-final s2* ⟶ (∄ *s2'*. *ord-res-3-step s2 s2'*)
by (*metis finished-def ord-res-3-semantics.final-finished*)
next
show ∀ *i s1 s2*. *match i s1 s2* ⟶ *ord-res-2-final s1* = *ord-res-3-final s2*
unfolding *match-def*
using *ord-res-2-final-iff-ord-res-3-final* **by** *metis*
next
show ∀ *i s1 s2*. *match i s1 s2* ⟶
safe-state ord-res-2-step ord-res-2-final s1 ∧ *safe-state ord-res-3-step ord-res-3-final s2*
unfolding *match-def*
using *safe-states-if-ord-res-2-matches-ord-res-3* **by** *metis*
next
show *wfP* (λ- -. *False*)
by *simp*
next
show ∀ *i s1 s2 s2'*.
match i s1 s2 ⟶
ord-res-3-step s2 s2' ⟶
(∃ *i' s1'*. *ord-res-2-step*$^{++}$ *s1 s1'* ∧ *match i' s1' s2'*) ∨ (∃ *i'*. *match i' s1 s2'* ∧ *False*)
unfolding *match-def*

using *backward-simulation-2-to-3* **by** *metis*
qed

end

30 ORD-RES-4 (implicit factorization)

type-synonym *'f ord-res-4-state* = *'f gclause fset* × *'f gclause fset* × *'f gclause fset*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-3-matches-ord-res-4* :: *'f ord-res-3-state* ⇒ *'f ord-res-4-state* ⇒ *bool* **where**
$\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er} \Longrightarrow U_{ef} = iefac\ \mathcal{F} \mid`\mid \{\mid C \mid\in\mid N \mid\cup\mid U_{er}.\ iefac\ \mathcal{F}\ C \neq C\mid\} \Longrightarrow$
ord-res-3-matches-ord-res-4 $(N,\ (U_{er},\ U_{ef}))\ (N,\ U_{er},\ \mathcal{F})$

lemma *ord-res-3-final-iff-ord-res-4-final*:
assumes *match*: *ord-res-3-matches-ord-res-4 S3 S4*
shows *ord-res-3-final S3* ⟷ *ord-res-4-final S4*
using *match*
proof (*cases S3 S4 rule*: *ord-res-3-matches-ord-res-4.cases*)
case *match-hyps*: (*1* $\mathcal{F}\ N\ U_{er}\ U_{ef}$)
note *invars* = *match-hyps*(*3*−)

have $\{\#\} \mid\in\mid N \mid\cup\mid U_{er} \mid\cup\mid U_{ef} \longleftrightarrow \{\#\} \mid\in\mid iefac\ \mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$
using *invars* **by** (*auto simp*: *iefac-def*)

moreover have *ex-false-clause* (*fset* ($N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}$)) ⟷
ex-false-clause (*fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$))
unfolding *ex-false-clause-iff*
unfolding *funion-funion-eq-funion-funion-fimage-iefac-if*[*OF invars*(*2*)]
unfolding *is-least-false-clause-with-iefac-conv* **..**

ultimately have *ord-res-final* ($N \mid\cup\mid U_{er} \mid\cup\mid U_{ef}$) ⟷ *ord-res-final* (*iefac* $\mathcal{F}$ $\mid`\mid (N \mid\cup\mid U_{er})$)
unfolding *ord-res-final-def* **by** *argo*

thus *?thesis*
unfolding *match-hyps*(*1*,*2*)
by (*simp add*: *ord-res-3-final.simps ord-res-4-final.simps*)
qed

lemma *forward-simulation-between-3-and-4*:
assumes
match: *ord-res-3-matches-ord-res-4 S3 S4* **and**
step: *ord-res-3-step S3 S3'*
shows (∃ *S4'*. *ord-res-4-step*$^{++}$ *S4 S4'* ∧ *ord-res-3-matches-ord-res-4 S3' S4'*)
using *match*

proof (*cases S3 S4 rule*: *ord-res-3-matches-ord-res-4.cases*)
case *match-hyps*: (*1* $\mathcal{F}$ *N* U_{er} U_{ef})
note *match-invars* = *match-hyps*(*3*−)

from *step* **obtain** *s3′* **where** *step′*: *ord-res-3 N* (U_{er}, U_{ef}) *s3′* **and** *S3′* = (*N*, *s3′*)
unfolding *match-hyps*(*1*,*2*)
by (*auto elim*: *ord-res-3-step.cases*)

from *step′* **show** *?thesis*
proof (*cases N* (U_{er}, U_{ef}) *s3′ rule*: *ord-res-3.cases*)
case (*factoring C L* U_{ef}′)

have ¬ *ord-res.is-strictly-maximal-lit L C*
using ‹*is-least-false-clause* (*N* |∪| U_{er} |∪| U_{ef}) *C*› ‹*ord-res.is-maximal-lit L C*› ‹*is-pos L*›
by (*metis* (*no-types*, *lifting*) *is-least-false-clause-def is-pos-def*
pos-lit-not-greatest-if-maximal-in-least-false-clause)

have *C* |∈| *N* |∪| U_{er}
proof −
have *C* |∈| *N* |∪| U_{er} |∪| U_{ef}
using ‹*is-least-false-clause* (*N* |∪| U_{er} |∪| U_{ef}) *C*›
by (*simp add*: *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*)
moreover have *C* |∉| U_{ef}
proof (*rule notI*)
assume *C* |∈| U_{ef}
then obtain C_0 **where** *C* = *iefac* $\mathcal{F}$ C_0 **and** C_0 |∈| *N* |∪| U_{er} **and** *iefac* $\mathcal{F}$ C_0 ≠ C_0
using *match-invars*(*2*) **by** *force*
then show *False*
by (*metis Uniq-D* ‹¬ *ord-res.is-strictly-maximal-lit L C*› *iefac-def*
linorder-lit.Uniq-is-maximal-in-mset
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset local.factoring(*3*)
obtains-positive-greatest-lit-if-efac-not-ident)
qed
ultimately show *?thesis*
by *simp*
qed

show *?thesis*
proof (*intro exI conjI*)
show *ord-res-4-step*$^{++}$ *S4* (*N*, U_{er}, *finsert C* $\mathcal{F}$)
unfolding *match-hyps*(*1*,*2*)
proof (*intro tranclp.r-into-trancl ord-res-4-step.intros ord-res-4.factoring*)
have *is-least-false-clause* (*N* |∪| U_{er} |∪| U_{ef}) *C*
using *factoring* **by** *argo*
hence *is-least-false-clause* (*N* |∪| U_{er} |∪| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *C*
unfolding *funion-funion-eq-funion-funion-fimage-iefac-if*[*OF match-invars*(*2*)]

.
thus *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *C*
unfolding *is-least-false-clause-with-iefac-conv* .
next
show *ord-res.is-maximal-lit L C*
using ‹*ord-res.is-maximal-lit L C*› .
next
show *is-pos L*
using ‹*is-pos L*› .
qed (*rule refl*)+
next
show *ord-res-3-matches-ord-res-4 S3'* (*N*, U_{er}, *finsert C* $\mathcal{F}$)
unfolding ‹*S3'* = (*N*, *s3'*)› ‹*s3'* = (U_{er}, U_{ef}')› ‹U_{ef}' = *finsert* (*efac C*) U_{ef}›
proof (*rule ord-res-3-matches-ord-res-4.intros*)
show *finsert C* $\mathcal{F}$ |⊆| *N* |∪| U_{er}
using *match-invars* ‹*C* |∈| *N* |∪| U_{er}› **by** *simp*
next
have ∃ *C'*. *ord-res.ground-factoring C C'*
using ‹*ord-res.is-maximal-lit L C*› ‹*is-pos L*›
by (*metis* ‹¬ *ord-res.is-strictly-maximal-lit L C*› *ex-ground-factoringI is-pos-def*)
hence *efac C* ≠ *C*
by (*metis ex1-efac-eq-factoring-chain*)
hence *iefac* (*finsert C* $\mathcal{F}$) *C* ≠ *C*
by (*simp add*: *iefac-def*)

have {|*Ca* |∈| *N* |∪| U_{er}. *iefac* (*finsert C* $\mathcal{F}$) *Ca* ≠ *Ca*|} =
finsert C {|*Ca* |∈| *N* |∪| U_{er}. *iefac* $\mathcal{F}$ *Ca* ≠ *Ca*|}
proof (*intro fsubset-antisym fsubsetI*)
fix *x*
assume *x* |∈| {|*Ca* |∈| *N* |∪| U_{er}. *iefac* (*finsert C* $\mathcal{F}$) *Ca* ≠ *Ca*|}
hence *x* |∈| *N* |∪| U_{er} **and** *iefac* (*finsert C* $\mathcal{F}$) *x* ≠ *x*
by *simp-all*
then show *x* |∈| *finsert C* {|*Ca* |∈| *N* |∪| U_{er}. *iefac* $\mathcal{F}$ *Ca* ≠ *Ca*|}
by (*smt* (*verit*, *best*) *ffmember-filter finsert-iff iefac-def*)
next
fix *x*
assume *x* |∈| *finsert C* {|*Ca* |∈| *N* |∪| U_{er}. *iefac* $\mathcal{F}$ *Ca* ≠ *Ca*|}
hence *x* = *C* ∨ *x* |∈| *N* |∪| U_{er} ∧ *iefac* $\mathcal{F}$ *x* ≠ *x*
by *auto*
thus *x* |∈| {|*Ca* |∈| *N* |∪| U_{er}. *iefac* (*finsert C* $\mathcal{F}$) *Ca* ≠ *Ca*|}
proof (*elim disjE conjE*)
assume *x* = *C*
thus *?thesis*
using ‹*C* |∈| *N* |∪| U_{er}› ‹*iefac* (*finsert C* $\mathcal{F}$) *C* ≠ *C*› **by** *auto*
next
assume *x* |∈| *N* |∪| U_{er} **and** *iefac* $\mathcal{F}$ *x* ≠ *x*
thus *?thesis*

```
          by (smt (verit, best) ffmember-filter finsertCI iefac-def)
        qed
      qed
      thus finsert (efac C) U_ef =
        iefac (finsert C F) |`| {|Ca |∈| N |∪| U_er. iefac (finsert C F) Ca ≠ Ca|}
        using iefac-def match-invars(2) by auto
    qed
  qed
next
  case (resolution C L D U_rr')

  have D |∈| iefac F |`| (N |∪| U_er)
  proof -
    have D |∈| N |∪| U_er |∪| U_ef
      using resolution by argo
    hence D |∈| N |∪| U_er |∪| iefac F |`| (N |∪| U_er)
    unfolding funion-funion-eq-funion-funion-fimage-iefac-if[OF match-invars(2)]
.
    moreover have D |∉| N |∪| U_er - iefac F |`| (N |∪| U_er)
    by (metis clauses-for-iefac-are-unproductive insert-not-empty local.resolution(7))
    ultimately show ?thesis
      by blast
  qed

  show ?thesis
  proof (intro exI conjI)
    show ord-res-4-step^++ S4 (N, finsert (eres D C) U_er, F)
      unfolding match-hyps(1,2)
      proof (intro tranclp.r-into-trancl ord-res-4-step.intros ord-res-4.resolution)
        have is-least-false-clause (N |∪| U_er |∪| U_ef) C
          using resolution by argo
        hence is-least-false-clause (N |∪| U_er |∪| iefac F |`| (N |∪| U_er)) C
       unfolding funion-funion-eq-funion-funion-fimage-iefac-if[OF match-invars(2)]
.
        thus is-least-false-clause (iefac F |`| (N |∪| U_er)) C
          unfolding is-least-false-clause-with-iefac-conv .
      next
        show ord-res.is-maximal-lit L C
          using resolution by argo
      next
        show is-neg L
          using resolution by argo
      next
        show D |∈| iefac F |`| (N |∪| U_er)
          using ‹D |∈| iefac F |`| (N |∪| U_er)› .
      next
        show D ≺_c C
          using resolution by argo
      next
```

```
        have ord-res.production (fset (N |∪| U_er |∪| U_ef)) D =
          ord-res.production (fset (N |∪| U_er |∪| iefac F |`| (N |∪| U_er))) D
        unfolding funion-funion-eq-funion-funion-fimage-iefac-if[OF match-invars(2)]
..
        also have ... = ord-res.production (fset (iefac F |`| (N |∪| U_er)) ∪ fset
(N |∪| U_er)) D
          by (simp add: sup.commute)
        also have ... = ord-res.production (fset (iefac F |`| (N |∪| U_er))) D
        proof (rule production-union-unproductive-strong)
          show ∀ x ∈ fset (N |∪| U_er) − fset (iefac F |`| (N |∪| U_er)).
            ord-res.production (fset (iefac F |`| (N |∪| U_er)) ∪ fset (N |∪| U_er)) x
= {}
            using clauses-for-iefac-are-unproductive[of N |∪| U_er F] by simp
        next
          show D |∈| iefac F |`| (N |∪| U_er)
            using ‹D |∈| iefac F |`| (N |∪| U_er)› .
        qed (rule finite-fset)+

        finally show ord-res.production (fset (iefac F |`| (N |∪| U_er))) D =
{atm-of L}
          using resolution by argo
      qed (rule refl)+
    next
      show ord-res-3-matches-ord-res-4 S3' (N, finsert (eres D C) U_er, F)
        unfolding ‹S3' = (N, s3')› ‹s3' = (U_rr', U_ef)› ‹U_rr' = finsert (eres D
C) U_er›
      proof (rule ord-res-3-matches-ord-res-4.intros)
        show F |⊆| N |∪| finsert (eres D C) U_er
          using match-invars by auto
      next
        show U_ef = iefac F |`| {|C |∈| N |∪| finsert (eres D C) U_er. iefac F C ≠
C|}
        proof (cases eres D C |∈| F)
          case True
          then show ?thesis
            using ‹F |⊆| N |∪| U_er›
            using match-invars by force
        next
          case False
          hence iefac F (eres D C) = eres D C
            by (simp add: iefac-def)
          hence {|C |∈| N |∪| finsert (eres D C) U_er. iefac F C ≠ C|} = {|C |∈|
N |∪| U_er. iefac F C ≠ C|}
            using ffilter-eq-ffilter-minus-singleton by auto
          thus ?thesis
            using match-invars by argo
        qed
      qed
    qed
```

qed
qed

theorem *bisimulation-ord-res-3-ord-res-4*:
defines *match* ≡ λ- *S3 S4*. *ord-res-3-matches-ord-res-4 S3 S4*
shows ∃(*MATCH* :: *nat* × *nat* ⇒ *'f ord-res-3-state* ⇒ *'f ord-res-4-state* ⇒ *bool*) $\mathcal{R}_f$ $\mathcal{R}_b$.
bisimulation ord-res-3-step ord-res-3-final ord-res-4-step ord-res-4-final MATCH $\mathcal{R}_f$ $\mathcal{R}_b$
proof (*rule ex-bisimulation-from-forward-simulation*)
show *right-unique ord-res-3-step*
using *right-unique-ord-res-3-step* **.**
next
show *right-unique ord-res-4-step*
using *right-unique-ord-res-4-step* **.**
next
show ∀ *s1*. *ord-res-3-final s1* ⟶ (∄ *s1'*. *ord-res-3-step s1 s1'*)
by (*metis finished-def ord-res-3-semantics.final-finished*)
next
show ∀ *s2*. *ord-res-4-final s2* ⟶ (∄ *s2'*. *ord-res-4-step s2 s2'*)
by (*metis finished-def ord-res-4-semantics.final-finished*)
next
show ∀ *i s1 s2*. *match i s1 s2* ⟶ *ord-res-3-final s1* ⟷ *ord-res-4-final s2*
unfolding *match-def*
using *ord-res-3-final-iff-ord-res-4-final* **by** *metis*
next
show ∀ *i s1 s2*. *match i s1 s2* ⟶
safe-state ord-res-3-step ord-res-3-final s1 ∧ *safe-state ord-res-4-step ord-res-4-final s2*
using *ord-res-3-step-safe ord-res-4-step-safe*
by (*simp add*: *safe-state-if-all-states-safe*)
next
show *wfP* (λ*i' i*. *False*)
by *simp*
next
show ∀ *i s1 s2 s1'*. *match i s1 s2* ⟶ *ord-res-3-step s1 s1'* ⟶
(∃ *i' s2'*. *ord-res-4-step*$^{++}$ *s2 s2'* ∧ *match i' s1' s2'*) ∨ (∃ *i'*. *match i' s1' s2* ∧ *False*)
unfolding *match-def*
using *forward-simulation-between-3-and-4* **by** *metis*
qed

end

31 ORD-RES-5 (explicit model construction)

type-synonym *'f ord-res-5-state* = *'f gclause fset* × *'f gclause fset* × *'f gclause fset* ×
(*'f gterm* ⇒ *'f gclause option*) × *'f gclause option*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-4-matches-ord-res-5* :: *'f ord-res-4-state* ⇒ *'f ord-res-5-state* ⇒ *bool* **where**
ord-res-5-invars N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) ⟹
(∀ *C*. $\mathcal{C}$ = *Some C* ⟷ *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *C*) ⟹
ord-res-4-matches-ord-res-5 (*N*, U_{er}, $\mathcal{F}$) (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)

lemma *ord-res-4-final-iff-ord-res-5-final*:
assumes *match*: *ord-res-4-matches-ord-res-5 S4 S5*
shows *ord-res-4-final S4* ⟷ *ord-res-5-final S5*
using *match*
proof (*cases S4 S5 rule*: *ord-res-4-matches-ord-res-5.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$)

show *?thesis*
unfolding *match-hyps*(*1*,*2*,*3*)
proof (*intro iffI ord-res-5-final.intros*)
assume *ord-res-4-final* (*N*, U_{er}, $\mathcal{F}$)
hence {#} |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) ∨ ¬ *ex-false-clause* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})))
by (*simp add*: *ord-res-4-final.simps ord-res-final-def*)
thus *ord-res-5-final* (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)
proof (*elim disjE*)
assume {#} |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})
hence *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) {#}
using *is-least-false-clause-mempty* **by** *metis*
hence $\mathcal{C}$ = *Some* {#}
by (*smt* (*verit*) *all-smaller-clauses-true-wrt-respective-Interp-def is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff linorder-cls.le-imp-less-or-eq match-hyps*(*3*) *mempty-lesseq-cls ord-res-5-invars-def*)
thus *?thesis*
using *ord-res-5-final.contradiction-found* **by** *metis*
next
assume ¬ *ex-false-clause* (*fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})))
hence $\mathcal{C}$ = *None*
using *match-hyps*(*2*−)
by (*metis ex-false-clause-if-least-false-clause option.exhaust*)
thus *?thesis*
using *ord-res-5-final.model-found* **by** *metis*
qed
next
assume *ord-res-5-final* (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)
thus *ord-res-4-final* (*N*, U_{er}, $\mathcal{F}$)
proof (*cases* (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) *rule*: *ord-res-5-final.cases*)
case *model-found*
have *all-smaller-clauses-true-wrt-respective-Interp N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)
using ‹*ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›

```
      unfolding ord-res-5-invars-def by metis
    hence ∀ C |∈| iefac F |‘| (N |∪| U_er). ord-res-Interp (iefac F ‘ (fset N ∪ fset
U_er)) C ⟙ C
     by (simp add: model-found all-smaller-clauses-true-wrt-respective-Interp-def)
    hence ¬ ex-false-clause (fset (iefac F |‘| (N |∪| U_er)))
      by (simp add: ex-false-clause-def)
    then show ?thesis
      by (metis ord-res-4-final.intros ord-res-final-def)
  next
    case contradiction-found
    hence {#} |∈| iefac F |‘| (N |∪| U_er)
      using ‹ord-res-5-invars N (U_er, F, M, C)›
      by (metis next-clause-in-factorized-clause-def ord-res-5-invars-def)
    then show ?thesis
      by (metis ord-res-4-final.intros ord-res-final-def)
  qed
 qed
qed

lemma forward-simulation-between-4-and-5:
 fixes S4 S4' S5
 assumes match: ord-res-4-matches-ord-res-5 S4 S5 and step: ord-res-4-step S4
S4'
 shows ∃ S5'. ord-res-5-step^++ S5 S5' ∧ ord-res-4-matches-ord-res-5 S4' S5'
 using match
proof (cases S4 S5 rule: ord-res-4-matches-ord-res-5.cases)
 case match-hyps: (1 N U_er F M C)
 hence
   S4-def: S4 = (N, U_er, F) and
   S5-def: S5 = (N, U_er, F, M, C)
   unfolding atomize-conj by metis

 have dom-M-eq: ⋀C. C = Some C ⟹ dom M = ord-res.interp (fset (iefac F
|‘| (N |∪| U_er))) C
  using match-hyps unfolding ord-res-5-invars-def model-eq-interp-upto-next-clause-def
by simp

 obtain s4' where S4'-def: S4' = (N, s4') and step': ord-res-4 N (U_er, F) s4'
   using step unfolding S4-def by (auto simp: ord-res-4-step.simps)

 show ?thesis
   using step'
 proof (cases N (U_er, F) s4' rule: ord-res-4.cases)
   case step-hyps: (factoring NN C L F')
   have C = Some C
     using match-hyps(3−) step-hyps by metis

   define M' :: 'f gterm ⇒ 'f gterm literal multiset option where
     M' = (λ-. None)
```

define $\mathcal{C}' :: 'f\ gclause\ option$ **where**
$\mathcal{C}' = The\text{-}optional\ (linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset\ (iefac\ \mathcal{F}'\ |`|\ (N\ |\cup|\ U_{er})))$

have *ord-res-5-step*: *ord-res-5* $N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}', \mathcal{M}', \mathcal{C}')$
proof (*rule ord-res-5.factoring*)
have *is-least-false-clause* $(iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\ C$
using *step-hyps* **by** *argo*
then show $\neg\ dom\ \mathcal{M} \models C$
using *dom-*$\mathcal{M}$*-eq*[*OF* ‹$\mathcal{C} = Some\ C$›]
by (*metis* (*mono-tags*, *lifting*) *is-least-false-clause-def*
linorder-cls.is-least-in-ffilter-iff ord-res-Interp-entails-if-greatest-lit-is-pos
unproductive-if-nex-strictly-maximal-pos-lit sup-bot.right-neutral)
next
show *ord-res.is-maximal-lit L C*
using *step-hyps* **by** *metis*
next
show *is-pos L*
using *step-hyps* **by** *metis*
next
show $\neg$ *ord-res.is-strictly-maximal-lit L C*
using *step-hyps*
by (*metis* (*no-types*, *lifting*) *is-least-false-clause-def literal.collapse*(*1*)
pos-lit-not-greatest-if-maximal-in-least-false-clause)
next
show $\mathcal{F}' = finsert\ C\ \mathcal{F}$
using *step-hyps* **by** *metis*
qed (*simp-all add*: $\mathcal{M}'$*-def* $\mathcal{C}'$*-def*)

moreover have $\exists \mathcal{M}''\ \mathcal{C}''.$
$(ord\text{-}res\text{-}5\ N)^{**}\ (U_{er}, \mathcal{F}', \mathcal{M}', \mathcal{C}')\ (U_{er}, \mathcal{F}', \mathcal{M}'', \mathcal{C}'') \wedge$
$(\forall C.\ (\mathcal{C}'' = Some\ C) \longleftrightarrow is\text{-}least\text{-}false\text{-}clause\ (iefac\ \mathcal{F}'\ |`|\ (N\ |\cup|\ U_{er}))\ C)$
proof (*rule ord-res-5-construct-model-upto-least-false-clause*)
show *ord-res-5-invars* $N\ (U_{er}, \mathcal{F}', \mathcal{M}', \mathcal{C}')$
using *ord-res-5-step* ‹*ord-res-5-invars* $N\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$› ‹$\mathcal{C} = Some\ C$›
by (*metis ord-res-5-preserves-invars*)
qed

ultimately obtain $\mathcal{M}''\ \mathcal{C}''$ **where**
s5-steps: $(ord\text{-}res\text{-}5\ N)^{++}\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}', \mathcal{M}'', \mathcal{C}'')$ **and**
next-clause-least-false:
$(\forall C.\ (\mathcal{C}'' = Some\ C) \longleftrightarrow is\text{-}least\text{-}false\text{-}clause\ (iefac\ \mathcal{F}'\ |`|\ (N\ |\cup|\ U_{er}))\ C)$
by (*meson rtranclp-into-tranclp2*)

have *ord-res-5-step*$^{++}$ *S5* $(N, U_{er}, \mathcal{F}', \mathcal{M}'', \mathcal{C}'')$
unfolding *S5-def* ‹$\mathcal{C} = Some\ C$›
using *s5-steps* **by** (*metis tranclp-ord-res-5-step-if-tranclp-ord-res-5*)

moreover have *ord-res-4-matches-ord-res-5 S4* $'\ (N, U_{er}, \mathcal{F}', \mathcal{M}'', \mathcal{C}'')$

```
    unfolding S4'-def ‹s4' = (U_er, F')›
  proof (intro ord-res-4-matches-ord-res-5.intros)
    show ord-res-5-invars N (U_er, F', M'', C'')
      using s5-steps ‹C = Some C› ‹ord-res-5-invars N (U_er, F, M, C)›
      by (smt (verit, best) ord-res-5-preserves-invars tranclp-induct)
  next
   show ∀ C. (C'' = Some C) = is-least-false-clause (iefac F' |`| (N |∪| U_er)) C
      using next-clause-least-false .
  qed

  ultimately show ?thesis
    by metis
next
  case step-hyps: (resolution NN C L D U_er')
  have C = Some C
    using match-hyps(3−) step-hyps by metis

  define M' :: 'f gterm ⇒ 'f gterm literal multiset option where
    M' = (λ-. None)

  define C' :: 'f gclause option where
    C' = The-optional (linorder-cls.is-least-in-fset (iefac F |`| (N |∪| U_er')))

  have ord-res-5-step: ord-res-5 N (U_er, F, M, Some C) (U_er', F, M', C')
  proof (rule ord-res-5.resolution)
    have is-least-false-clause (iefac F |`| (N |∪| U_er)) C
      using step-hyps by argo
    then show ¬ dom M ⊨ C
      using dom-M-eq[OF ‹C = Some C›]
      by (metis (mono-tags, lifting) is-least-false-clause-def
          linorder-cls.is-least-in-ffilter-iff ord-res-Interp-entails-if-greatest-lit-is-pos
          unproductive-if-nex-strictly-maximal-pos-lit sup-bot.right-neutral)
  next
    show ord-res.is-maximal-lit L C
      using step-hyps by metis
  next
    show is-neg L
      using step-hyps by metis
  next
    show M (atm-of L) = Some D
      using step-hyps
      by (smt (verit) ‹C = Some C› all-produced-atoms-in-model-def insertI1
match-hyps(3)
          ord-res-5-invars-def)
  next
    show U_er' = finsert (eres D C) U_er
      using step-hyps by metis
  qed (simp-all add: M'-def C'-def)
```

moreover have $\exists \mathcal{M}'' \mathcal{C}''.$
(*ord-res-5 N*)** (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, $\mathcal{C}'$) (U_{er}', $\mathcal{F}$, $\mathcal{M}''$, $\mathcal{C}''$) $\wedge$
($\forall$ *C*. ($\mathcal{C}''$ = *Some C*) $\longleftrightarrow$ *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *C*)
proof (*rule ord-res-5-construct-model-upto-least-false-clause*)
show *ord-res-5-invars N* (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, $\mathcal{C}'$)
using *ord-res-5-step* ‹*ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)› ‹$\mathcal{C}$ = *Some C*›
by (*metis ord-res-5-preserves-invars*)
qed

ultimately obtain $\mathcal{M}''$ $\mathcal{C}''$ **where**
s5-steps: (*ord-res-5 N*)$^{++}$ (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*) (U_{er}', $\mathcal{F}$, $\mathcal{M}''$, $\mathcal{C}''$) **and**
next-clause-least-false:
($\forall$ *C*. ($\mathcal{C}''$ = *Some C*) $\longleftrightarrow$ *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *C*)
by (*meson rtranclp-into-tranclp2*)

have *ord-res-5-step*$^{++}$ *S5* (*N*, U_{er}', $\mathcal{F}$, $\mathcal{M}''$, $\mathcal{C}''$)
unfolding *S5-def* ‹$\mathcal{C}$ = *Some C*›
using *s5-steps* **by** (*metis tranclp-ord-res-5-step-if-tranclp-ord-res-5*)

moreover have *ord-res-4-matches-ord-res-5 S4'* (*N*, U_{er}', $\mathcal{F}$, $\mathcal{M}''$, $\mathcal{C}''$)
unfolding *S4'-def* ‹*s4'* = (U_{er}', $\mathcal{F}$)›
proof (*intro ord-res-4-matches-ord-res-5.intros*)
show *ord-res-5-invars N* (U_{er}', $\mathcal{F}$, $\mathcal{M}''$, $\mathcal{C}''$)
using *s5-steps* ‹$\mathcal{C}$ = *Some C*› ‹*ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
by (*smt* (*verit*, *best*) *ord-res-5-preserves-invars tranclp-induct*)
next
show $\forall$ *C*. ($\mathcal{C}''$ = *Some C*) = *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *C*
using *next-clause-least-false* **.**
qed

ultimately show *?thesis*
by *metis*
qed
qed

theorem *bisimulation-ord-res-4-ord-res-5*:
defines *match* ≡ λ-. *ord-res-4-matches-ord-res-5*
shows ∃(*MATCH* :: *nat* × *nat* ⇒ *'f ord-res-4-state* ⇒ *'f ord-res-5-state* ⇒ *bool*) $\mathcal{R}_f$ $\mathcal{R}_b$.
bisimulation ord-res-4-step ord-res-4-final ord-res-5-step ord-res-5-final MATCH $\mathcal{R}_f$ $\mathcal{R}_b$
proof (*rule ex-bisimulation-from-forward-simulation*)
show *right-unique ord-res-4-step*
using *right-unique-ord-res-4-step* **.**
next
show *right-unique ord-res-5-step*
using *right-unique-ord-res-5-step* **.**
next
show $\forall$ *s*. *ord-res-4-final s* $\longrightarrow$ ($\nexists$ *s'*. *ord-res-4-step s s'*)

by (*metis finished-def ord-res-4-semantics.final-finished*)
next
show $\forall s.$ *ord-res-5-final* $s \longrightarrow (\nexists s'.$ *ord-res-5-step* $s\ s')$
by (*metis finished-def ord-res-5-semantics.final-finished*)
next
show $\forall i$ *S4 S5*. *match* i *S4 S5* $\longrightarrow$ *ord-res-4-final S4* $\longleftrightarrow$ *ord-res-5-final S5*
unfolding *match-def*
using *ord-res-4-final-iff-ord-res-5-final* **by** *metis*
next
show $\forall i$ *S4 S5*. *match* i *S4 S5* $\longrightarrow$
safe-state ord-res-4-step ord-res-4-final S4 $\wedge$ *safe-state ord-res-5-step ord-res-5-final S5*
proof (*intro allI impI conjI*)
fix i *S4 S5*
show *safe-state ord-res-4-step ord-res-4-final S4*
using *ord-res-4-step-safe safe-state-if-all-states-safe* **by** *metis*

assume *match* i *S4 S5*
thus *safe-state ord-res-5-step ord-res-5-final S5*
using ‹*match* i *S4 S5*›
using *ord-res-5-safe-state-if-invars*
using *match-def ord-res-4-matches-ord-res-5.cases* **by** *metis*
qed
next
show *wfp* (λ- -. *False*)
by *simp*
next
show $\forall i$ *s1 s2 s1'*.
match i *s1 s2* $\longrightarrow$
ord-res-4-step s1 s1' $\longrightarrow$
$(\exists i'$ *s2'*. *ord-res-5-step*$^{++}$ *s2 s2'* $\wedge$ *match* i' *s1' s2'*$) \vee (\exists i'$. *match* i' *s1' s2* $\wedge$ *False*)
unfolding *match-def*
using *forward-simulation-between-4-and-5* **by** *metis*
qed

end

32 ORD-RES-6 (model backjump)

type-synonym *'f ord-res-6-state* = *'f gclause fset* $\times$ *'f gclause fset* $\times$ *'f gclause fset* $\times$
(*'f gterm* $\Rightarrow$ *'f gclause option*) $\times$ *'f gclause option*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-5-matches-ord-res-6* :: *'f ord-res-5-state* $\Rightarrow$ *'f ord-res-6-state* $\Rightarrow$ *bool* **where**
ord-res-5-invars N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) $\Longrightarrow$

ord-res-5-matches-ord-res-6 $(N,\ U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$ $(N,\ U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$

lemma *ord-res-5-final-iff-ord-res-6-final*:
fixes *i S5 S6*
assumes *match*: *ord-res-5-matches-ord-res-6 S5 S6*
shows *ord-res-5-final S5* $\longleftrightarrow$ *ord-res-6-final S6*
using *match*
proof (*cases S5 S6 rule*: *ord-res-5-matches-ord-res-6.cases*)
case (*1 N* U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$)
thus *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *ord-res-5-final.simps ord-res-6-final.cases ord-res-6-final.contradiction-found ord-res-6-final.model-found*)
qed

lemma *backward-simulation-between-5-and-6*:
fixes *S5 S6 S6′*
assumes *match*: *ord-res-5-matches-ord-res-6 S5 S6* **and** *step*: *ord-res-6-step S6 S6′*
shows $\exists$ *S5′*. *ord-res-5-step*$^{++}$ *S5 S5′* $\wedge$ *ord-res-5-matches-ord-res-6 S5′ S6′*
using *match*
proof (*cases S5 S6 rule*: *ord-res-5-matches-ord-res-6.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$)
hence *S5-def*: *S5* $= (N,\ U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$ **and** *S6-def*: *S6* $= (N,\ U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$
by *metis+*

obtain *s6′* **where** *S6′-def*: *S6′* $= (N,\ s6')$ **and** *step′*: *ord-res-6 N* $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$ *s6′*
using *step* **unfolding** *S6-def*
using *ord-res-6-step.simps* **by** *auto*

show *?thesis*
using *step′*
proof (*cases N* $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C})$ *s6′ rule*: *ord-res-6.cases*)
case *step-hyps*: (*skip C* $\mathcal{C}'$)

define *S5′* **where**
S5′ $= (N,\ U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C}')$

show *?thesis*
proof (*intro exI conjI*)
have *step5*: *ord-res-5 N* $(U_{er},\ \mathcal{F},\ \mathcal{M},$ *Some C*$)$ $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C}')$
using *ord-res-5.skip step-hyps* **by** *metis*
hence *ord-res-5-step S5 S5′*
unfolding *S5-def S5′-def*
by (*metis ord-res-5-step.simps step-hyps*(*1*))
thus *ord-res-5-step*$^{++}$ *S5 S5′*
by *simp*

have *ord-res-5-invars N* $(U_{er},\ \mathcal{F},\ \mathcal{M},\ \mathcal{C}')$

```
      using step5 match-hyps(3) ord-res-5-preserves-invars step-hyps(1) by metis
    thus ord-res-5-matches-ord-res-6 S5' S6'
      unfolding S5'-def S6'-def ‹s6' = (U_er, F, M, C')›
      using ord-res-5-matches-ord-res-6.intros by metis
  qed
next
  case step-hyps: (production C L M' C')

  define S5' where
    S5' = (N, U_er, F, M', C')

  show ?thesis
  proof (intro exI conjI)
    have step5: ord-res-5 N (U_er, F, M, Some C) (U_er, F, M', C')
      using ord-res-5.production step-hyps by metis
    hence ord-res-5-step S5 S5'
      unfolding S5-def S5'-def
      by (metis ord-res-5-step.simps step-hyps(1))
    thus ord-res-5-step^++ S5 S5'
      by simp

    have ord-res-5-invars N (U_er, F, M', C')
      using step5 match-hyps(3) ord-res-5-preserves-invars step-hyps(1) by metis
    thus ord-res-5-matches-ord-res-6 S5' S6'
      unfolding S5'-def S6'-def ‹s6' = (U_er, F, M', C')›
      using ord-res-5-matches-ord-res-6.intros by metis
  qed
next
  case step-hyps: (factoring D K F')

  define S5' where
    S5' = (N, U_er, F', M, Some (efac D))

  have D |∈| iefac F |'| (N |∪| U_er)
  by (metis match-hyps(3) next-clause-in-factorized-clause-def ord-res-5-invars-def
step-hyps(1))
  hence iefac F' |'| (N |∪| U_er) ≠ {||}
    by blast
  then obtain C where C-least: linorder-cls.is-least-in-fset (iefac F' |'| (N |∪|
U_er)) C
    by (metis linorder-cls.ex1-least-in-fset)

  have efac D ≠ D
  by (metis ex1-efac-eq-factoring-chain is-pos-def ex-ground-factoringI step-hyps(4,5,6))

  show ?thesis
  proof (intro exI conjI)
    have The-optional (linorder-cls.is-least-in-fset (iefac F' |'| (N |∪| U_er))) =
Some C
```

proof (*rule The-optional-eq-SomeI*)
show $\exists_{\leq 1}$ *x*. *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})) *x*
by *blast*
next
show *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er})) *C*
using *C-least* **.**
qed
hence *step5*: *ord-res-5 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some D*) (U_{er}, $\mathcal{F}'$, *Map.empty*, *Some C*)
using *ord-res-5.factoring step-hyps* **by** *metis*
moreover have (*ord-res-5 N*)** ... (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac D*))
proof (*rule full-rtranclp-ord-res-5-run-upto*)
show *ord-res-6 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some D*) (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac D*))
using *step' S6-def S6'-def* ‹*s6'* = (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac D*))› ‹$\mathcal{C}$ = *Some D*› **by** *argo*
next
show *ord-res-5-invars N* (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac D*))
using *match-hyps*(*3*) *ord-res-6-preserves-invars step' step-hyps*(*2*) **by** *blast*
next
have *iefac* $\mathcal{F}$ *D* = *D* **and** *D* |∈| *N* |∪| U_{er}
unfolding *atomize-conj*
using ‹*efac D* ≠ *D*› ‹*D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})›[*unfolded fimage-iff*]
unfolding *iefac-def*
by (*metis ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)

have *iefac-*$\mathcal{F}'$*-eq*: *iefac* $\mathcal{F}'$ = (*iefac* $\mathcal{F}$)(*D* := *efac D*)
unfolding ‹$\mathcal{F}'$ = *finsert D* $\mathcal{F}$› *iefac-def* **by** *auto*

have *fimage-iefac-*$\mathcal{F}'$*-eq*:
iefac $\mathcal{F}'$ |‘| (*N* |∪| U_{er}) = *finsert* (*efac D*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er} − {|*D*|}))
unfolding *iefac-*$\mathcal{F}'$*-eq*
unfolding *fun-upd-fimage*[*of iefac* $\mathcal{F}$ *D efac D*] ‹*D* |∈| *N* |∪| U_{er}›
using ‹*D* |∈| *N* |∪| U_{er}› **by** *argo*

have {|*C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *efac D*|} =
{|*C* |∈| *finsert* (*efac D*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er} − {|*D*|})). *C* ≺$_c$ *efac D*|}
unfolding *fimage-iefac-*$\mathcal{F}'$*-eq* **..**

also have ... = {|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er} − {|*D*|}). *C* ≺$_c$ *efac D*|}
by *auto*

also have ... = {|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *efac D*|}
by (*smt* (*verit*, *ccfv-SIG*) ‹*iefac* $\mathcal{F}$ *D* = *D*› *efac-properties-if-not-ident*(*1*) *ffilter-eq-ffilter-minus-singleton fimage-finsert finsertI1 finsert-fminus1 finsert-fminus-single linorder-cls.less-imp-not-less*)

finally have {|*C* |∈| *iefac* $\mathcal{F}'$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *efac D*|} =
{|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *efac D*|} **.**

```
    next
      have dom-M-eq: dom M = ord-res.interp (fset (iefac F |'| (N |∪| U_er)))
D
        using ‹ord-res-5-invars N (U_er, F, M, C)› ‹C = Some D›
        unfolding ord-res-5-invars-def model-eq-interp-upto-next-clause-def
        by metis

      have atm-of K ∉ dom M
        by (metis linorder-lit.is-maximal-in-mset-iff literal.collapse(1)
            pos-literal-in-imp-true-cls step-hyps(3) step-hyps(4) step-hyps(5))

      have A ≺_t atm-of K if A ∈ dom M for A
      proof −
        obtain C where
          C |∈| iefac F |'| (N |∪| U_er) and
          C ≺_c D and
          A ∈ ord-res.production (fset (iefac F |'| (N |∪| U_er))) C
          using ‹A ∈ dom M› unfolding dom-M-eq
          unfolding ord-res.interp-def UN-iff
          by blast

        hence ord-res.is-strictly-maximal-lit (Pos A) C
          using ord-res.mem-productionE by metis

        hence Pos A ≼_l K
          using ‹ord-res.is-maximal-lit K D› ‹C ≺_c D›
        by (metis ord-res.asymp-less-lit ord-res.transp-less-lit linorder-cls.less-asym
              linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.leI
              linorder-lit.multp_HO-if-maximal-less-that-maximal multp-eq-multp_HO)

        hence A ≼_t atm-of K
        by (metis literal.collapse(1) literal.sel(1) ord-res.less-lit-simps(1) reflclp-iff
              step-hyps(5))

        moreover have A ≠ atm-of K
          using ‹atm-of K ∉ dom M› ‹A ∈ dom M› by metis

        ultimately show ?thesis
          by order
      qed
      hence dom M ⊆ {A. ∃ K. ord-res.is-maximal-lit K (efac D) ∧ A ≺_t atm-of
K}
        using linorder-lit.is-maximal-in-mset-iff step-hyps(4) by auto
      thus M = restrict-map M {A. ∃ K. ord-res.is-maximal-lit K (efac D) ∧ A
≺_t atm-of K}
        using restrict-map-ident-if-dom-subset by fastforce
    next
      show linorder-cls.is-least-in-fset (iefac F' |'| (N |∪| U_er)) C
        using C-least .
```

qed
ultimately have *steps5*: $(\textit{ord-res-5}\ N)^{++}\ (U_{er}, \mathcal{F}, \mathcal{M}, \textit{Some}\ D)\ (U_{er}, \mathcal{F}', \mathcal{M}, \textit{Some}\ (\textit{efac}\ D))$
by *simp*
thus $\textit{ord-res-5-step}^{++}$ *S5 S5'*
using *S5'-def S5-def step-hyps*(1) *tranclp-ord-res-5-step-if-tranclp-ord-res-5*
by *metis*

have $\textit{ord-res-5-invars}\ N\ (U_{er}, \mathcal{F}', \mathcal{M}, \textit{Some}\ (\textit{efac}\ D))$
using *steps5 match-hyps*(3) *tranclp-ord-res-5-preserves-invars step-hyps*(1)
by *metis*
thus *ord-res-5-matches-ord-res-6 S5' S6'*
unfolding *S5'-def S6'-def* ‹$s6' = (U_{er}, \mathcal{F}', \mathcal{M}, \textit{Some}\ (\textit{efac}\ D))$›
using *ord-res-5-matches-ord-res-6.intros* **by** *metis*
qed
next
case *step-hyps*: $(\textit{resolution-bot}\ C\ L\ D\ U_{er}'\ \mathcal{M}')$

define *S5'* :: - × - × - × (*'f gterm* ⇒ *'f gclause option*) × *'f gclause option*
where
S5' = $(N, U_{er}', \mathcal{F}, \mathcal{M}', \textit{Some}\ \{\#\})$

show *?thesis*
proof (*intro exI conjI*)
have $\{\#\}\ |\in|\ \textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}')$
using ‹$U_{er}' = \textit{finsert}\ (\textit{eres}\ D\ C)\ U_{er}$› ‹$\textit{eres}\ D\ C = \{\#\}$›
using *iefac-def* **by** *simp*

hence $\textit{linorder-cls.is-least-in-fset}\ (\textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}'))\ \{\#\}$
by (*metis linorder-cls.is-minimal-in-fset-eq-is-least-in-fset*
linorder-cls.is-minimal-in-fset-iff linorder-cls.leD mempty-lesseq-cls)

hence $\textit{The-optional}\ (\textit{linorder-cls.is-least-in-fset}\ (\textit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}'))) = \textit{Some}\ \{\#\}$
by (*metis linorder-cls.Uniq-is-least-in-fset The-optional-eq-SomeI*)

hence *step5*: $\textit{ord-res-5}\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, \textit{Some}\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', \textit{Some}\ \{\#\})$
using *ord-res-5.resolution step-hyps* **by** *metis*

thus $\textit{ord-res-5-step}^{++}$ *S5 S5'*
unfolding *S5-def* ‹$\mathcal{C} = \textit{Some}\ C$› *S5'-def*
by (*simp only*: *ord-res-5-step.intros tranclp.r-into-trancl*)

show *ord-res-5-matches-ord-res-6 S5' S6'*
using *step5*
by (*metis S5'-def S6'-def match-hyps*(3) *ord-res-5-matches-ord-res-6.intros*
ord-res-5-preserves-invars step-hyps(1) *step-hyps*(2))
qed
next

case *step-hyps*: (*resolution-pos E L D* U_{er}' $\mathcal{M}'$ *K*)

define $S5'$ :: - × - × - × ('*f gterm* ⇒ '*f gclause option*) × '*f gclause option* **where**
$S5' = (N,\ U_{er}',\ \mathcal{F},\ \mathcal{M}',\ \mathit{Some}\ (\mathit{eres}\ D\ E))$

hence *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}') ≠ {||}
using ‹$U_{er}' = \mathit{finsert}\ (\mathit{eres}\ D\ E)\ U_{er}$› **by** *simp*
then obtain *C* **where** *C-least*: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *C*
by (*metis linorder-cls.ex1-least-in-fset*)

show *?thesis*
proof (*intro exI conjI*)
have *The-optional* (*linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}'))) = *Some C*
proof (*rule The-optional-eq-SomeI*)
show $\exists_{\leq 1}$ *x*. *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *x*
by *blast*
next
show *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *C*
using *C-least* **.**
qed

hence *step5*: *ord-res-5 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (U_{er}', $\mathcal{F}$, *Map.empty*, *Some C*)
using *ord-res-5.resolution step-hyps* **by** *metis*

moreover have (*ord-res-5 N*)** ... (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
proof (*rule full-rtranclp-ord-res-5-run-upto*)
show *ord-res-6 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
using *step'* ‹$\mathcal{C} = \mathit{Some}\ E$› ‹$s6' = (U_{er}', \mathcal{F}, \mathcal{M}', \mathit{Some}\ (\mathit{eres}\ D\ E))$› **by** *argo*
next
show *ord-res-5-invars N* (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some* (*eres D E*))
using *match-hyps*(*3*) *ord-res-6-preserves-invars step' step-hyps*(*2*) **by** *blast*
next
have *eres D E* ≠ *E*
using *step-hyps* **by** (*metis linorder-lit.Uniq-is-maximal-in-mset the1-equality'*)

moreover have *eres D E* $\preceq_c$ *E*
using *eres-le* **.**

ultimately have *eres D E* $\prec_c$ *E*
by *order*

have ∀ *F*. *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er})) *F* ⟶ *E* $\preceq_c$ *F*
using ‹*ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
unfolding *ord-res-5-invars-def* ‹$\mathcal{C} = \mathit{Some}\ E$›

using *next-clause-lt-least-false-clause*[*of* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* E)]
by *simp*

have *E-least-false*: *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) E
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show E |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
using ‹*ord-res-5-invars* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
unfolding *ord-res-5-invars-def* ‹$\mathcal{C}$ = *Some* E›
by (*metis next-clause-in-factorized-clause-def*)
next
have ¬ *ord-res.interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E ⊫ E
using ‹*ord-res-5-invars* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
unfolding *ord-res-5-invars-def* ‹$\mathcal{C}$ = *Some* E›
using ‹¬ *dom* $\mathcal{M}$ ⊫ E› **by** (*metis model-eq-interp-upto-next-clause-def*)
moreover have *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E = {}
proof −
have ∄L. *is-pos* L ∧ *ord-res.is-strictly-maximal-lit* L E
using ‹*ord-res.is-maximal-lit* L E› ‹*is-neg* L›
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset)
thus *?thesis*
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed
ultimately show ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) E ⊫ E
by *simp*
next
fix F
assume *F-in*: F |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}) **and** $F \neq E$ **and**
F-false: ¬ *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) F ⊫ F
have ¬ $F \prec_c E$
using ‹*ord-res-5-invars* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
unfolding *ord-res-5-invars-def* ‹$\mathcal{C}$ = *Some* E›
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def*
using *F-in F-false*
by (*metis option.inject*)
thus $E \prec_c F$
using ‹$F \neq E$› **by** *order*
qed

have *D-prod*: *ord-res.production* (*fset* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))) $D \neq$ {}
using ‹*ord-res-5-invars* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
unfolding *ord-res-5-invars-def* ‹$\mathcal{C}$ = *Some* E›
by (*metis atoms-in-model-were-produced-def empty-iff step-hyps*(6))

have *iefac* $\mathcal{F}$ (*eres* D E) = *eres* D E
using *E-least-false D-prod*
by (*smt* (*verit*, *ccfv-threshold*)
‹∀F. *is-least-false-clause* (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})) F ⟶ $(\prec_c)^{==}$ E F›

```
      ‹eres D E ≺c E› clause-true-if-resolved-true ex1-eres-eq-full-run-ground-resolution
        fimage-finsert finsert-absorb finsert-iff full-run-def funion-finsert-right
        is-least-false-clause-def is-least-false-clause-finsert-smaller-false-clause
        linorder-cls.is-least-in-fset-ffilterD(2) linorder-cls.leD match-hyps(3)
      next-clause-in-factorized-clause-def ord-res-5-invars-def ord-res-6-preserves-invars
        rtranclpD step' step-hyps(2) step-hyps(7))

    hence {|C |∈| iefac F |‘| (N |∪| U_er'). C ≺c eres D E|} =
      {|C |∈| iefac F |‘| (N |∪| U_er). C ≺c eres D E|}
      unfolding ‹U_er' = finsert (eres D E) U_er› by auto
  next
    show M' = restrict-map M {A. ∃K. ord-res.is-maximal-lit K (eres D E)
∧ A ≺t atm-of K}
      using ‹M' = restrict-map M {A. A ≺t atm-of K}›
    by (smt (verit, ccfv-SIG) Collect-cong linorder-lit.Uniq-is-maximal-in-mset
step-hyps(10)
        the1-equality')
  next
    show linorder-cls.is-least-in-fset (iefac F |‘| (N |∪| U_er')) C
      using C-least .
  qed

  ultimately have steps5: (ord-res-5 N)++ (U_er, F, M, Some E) (U_er', F,
M', Some (eres D E))
    by simp

  thus ord-res-5-step++ S5 S5'
    unfolding S5-def ‹C = Some E› S5'-def
    by (metis tranclp-ord-res-5-step-if-tranclp-ord-res-5)

  show ord-res-5-matches-ord-res-6 S5' S6'
    unfolding S5'-def S6'-def ‹s6' = (U_er', F, M', Some (eres D E))›
    using steps5
  using match-hyps(3) ord-res-5-matches-ord-res-6.intros ord-res-6-preserves-invars
step'
      step-hyps(2) by metis
 qed
next
 case step-hyps: (resolution-neg E L D U_er' M' K C)

 define S5' :: - × - × - × ('f gterm ⇒ 'f gclause option) × 'f gclause option
where
  S5' = (N, U_er', F, M', Some C)

 hence iefac F |‘| (N |∪| U_er') ≠ {||}
  using ‹U_er' = finsert (eres D E) U_er› by simp
 then obtain B where B-least: linorder-cls.is-least-in-fset (iefac F |‘| (N |∪|
U_er')) B
  by (metis linorder-cls.ex1-least-in-fset)
```

show *?thesis*
proof (*intro exI conjI*)
have *The-optional* (*linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}'))) = *Some B*
proof (*rule The-optional-eq-SomeI*)
show $\exists_{\leq 1}$ *x*. *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *x*
by *blast*
next
show *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}')) *B*
using *B-least* **.**
qed

hence *step5*: *ord-res-5 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (U_{er}', $\mathcal{F}$, *Map.empty*, *Some B*)
using *ord-res-5.resolution step-hyps* **by** *metis*

moreover have (*ord-res-5 N*)** ... (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)
proof (*rule full-rtranclp-ord-res-5-run-upto*)
show *ord-res-6 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)
using *step'* ‹$\mathcal{C}$ = *Some E*› ‹*s6'* = (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)› **by** *argo*
next
show *ord-res-5-invars N* (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)
using *match-hyps*(*3*) *ord-res-6-preserves-invars step' step-hyps*(*2*) **by** *blast*
next
have *ord-res.is-strictly-maximal-lit* (*Pos* (*atm-of K*)) *C*
using ‹$\mathcal{M}$ (*atm-of K*) = *Some C*›
‹*ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›[*unfolded ord-res-5-invars-def atoms-in-model-were-produced-def*, *simplified*]
using *ord-res.mem-productionE* **by** *blast*

moreover have *Pos* (*atm-of K*) $\prec_l$ *K*
using ‹*is-neg K*› **by** (*cases K*) *simp-all*

ultimately have *C* $\prec_c$ *eres D E*
using ‹*ord-res.is-maximal-lit K* (*eres D E*)›
by (*metis ord-res.asymp-less-lit ord-res.transp-less-lit linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.multp$_{HO}$-if-maximal-less-that-maximal multp-eq-multp$_{HO}$*)

hence *C* $\prec_c$ *E*
using *eres-le*[*of D E*] **by** *order*

have *C* $\prec_c$ *efac* (*eres D E*)
by (*metis Uniq-D* ‹*C* $\prec_c$ *eres D E*› *efac-spec is-pos-def linorder-lit.Uniq-is-maximal-in-mset step-hyps*(*10*) *step-hyps*(*11*))

moreover have *efac* (*eres D E*) $\preceq_c$ *eres D E*
by (*metis efac-subset subset-implies-reflclp-multp*)

ultimately have $C \prec_c$ *iefac* $\mathcal{F}$ (*eres D E*)
unfolding *iefac-def* **by** *auto*

hence $\{|Ca \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}'). Ca \prec_c C|\} =$
$\{|Ca \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}). Ca \prec_c C|\}$
unfolding ‹$U_{er}' =$ *finsert* (*eres D E*) U_{er}› **by** *auto*

have $(\exists K.$ *ord-res.is-maximal-lit K C* $\wedge$ $A \prec_t$ *atm-of K*$) \longleftrightarrow A \prec_t$ *atm-of K* **for** *A*
using ‹*ord-res.is-strictly-maximal-lit* (*Pos* (*atm-of K*)) *C*›
by (*metis Uniq-def linorder-lit.Uniq-is-maximal-in-mset*
linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.sel(*1*))

hence $\{A. \exists K.$ *ord-res.is-maximal-lit K C* $\wedge$ $A \prec_t$ *atm-of K*$\} = \{A. A \prec_t$ *atm-of K*$\}$
by *metis*

thus $\mathcal{M}' =$ *restrict-map* $\mathcal{M}$ $\{A. \exists K.$ *ord-res.is-maximal-lit K C* $\wedge$ $A \prec_t$ *atm-of K*$\}$
using ‹$\mathcal{M}' =$ *restrict-map* $\mathcal{M}$ $\{A. A \prec_t$ *atm-of K*$\}$› **by** *argo*
next
show *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}')$) *B*
using *B-least* **.**
qed

ultimately have *steps5*: $(ord\text{-}res\text{-}5\ N)^{++}$ (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)
by *simp*

thus $ord\text{-}res\text{-}5\text{-}step^{++}$ *S5 S5'*
unfolding *S5-def* ‹$\mathcal{C} =$ *Some E*› *S5'-def*
by (*metis tranclp-ord-res-5-step-if-tranclp-ord-res-5*)

show *ord-res-5-matches-ord-res-6 S5' S6'*
unfolding *S5'-def S6'-def* ‹*s6'* = (U_{er}', $\mathcal{F}$, $\mathcal{M}'$, *Some C*)›
using *steps5*
using *match-hyps*(*3*) *ord-res-5-matches-ord-res-6.intros ord-res-6-preserves-invars step'*
step-hyps(*2*) **by** *metis*
qed
qed
qed

theorem *bisimulation-ord-res-5-ord-res-6*:
defines *match* $\equiv \lambda$*-. ord-res-5-matches-ord-res-6*
shows $\exists$(*MATCH* :: *nat* $\times$ *nat* $\Rightarrow$ *'f ord-res-5-state* $\Rightarrow$ *'f ord-res-6-state* $\Rightarrow$ *bool*) $\mathcal{R}_f$ $\mathcal{R}_b$.
bisimulation ord-res-5-step ord-res-5-final ord-res-6-step ord-res-6-final MATCH

$\mathcal{R}_f$ $\mathcal{R}_b$
proof (*rule ex-bisimulation-from-backward-simulation*)
show *right-unique ord-res-5-step*
using *right-unique-ord-res-5-step* **.**
next
show *right-unique ord-res-6-step*
using *right-unique-ord-res-6-step* **.**
next
show $\forall$ *s. ord-res-5-final s* $\longrightarrow$ ($\nexists$ *s'. ord-res-5-step s s'*)
by (*metis finished-def ord-res-5-semantics.final-finished*)
next
show $\forall$ *s. ord-res-6-final s* $\longrightarrow$ ($\nexists$ *s'. ord-res-6-step s s'*)
by (*metis finished-def ord-res-6-semantics.final-finished*)
next
show $\forall$ *i S5 S6. match i S5 S6* $\longrightarrow$ *ord-res-5-final S5* $\longleftrightarrow$ *ord-res-6-final S6*
unfolding *match-def*
using *ord-res-5-final-iff-ord-res-6-final* **by** *metis*
next
show $\forall$ *i S5 S6.*
match i S5 S6 $\longrightarrow$
safe-state ord-res-5-step ord-res-5-final S5 $\wedge$ *safe-state ord-res-6-step ord-res-6-final S6*
proof (*intro allI impI conjI*)
fix *i S5 S6*
assume *match i S5 S6*
show *safe-state ord-res-5-step ord-res-5-final S5*
using ‹*match i S5 S6*›
using *ord-res-5-safe-state-if-invars*
using *match-def ord-res-5-matches-ord-res-6.cases* **by** *metis*
show *safe-state ord-res-6-step ord-res-6-final S6*
using ‹*match i S5 S6*›
using *ord-res-6-safe-state-if-invars*
using *match-def ord-res-5-matches-ord-res-6.cases* **by** *metis*
qed
next
show *wfp* (λ- -. *False*)
by *simp*
next
show $\forall$ *i S5 S6 S6'.*
match i S5 S6 $\longrightarrow$
ord-res-6-step S6 S6' $\longrightarrow$
($\exists$ *i' S5'. ord-res-5-step*$^{++}$ *S5 S5'* $\wedge$ *match i' S5' S6'*) $\vee$ ($\exists$ *i'. match i' S5 S6'* $\wedge$ *False*)
unfolding *match-def*
using *backward-simulation-between-5-and-6* **by** *metis*
qed

end

33 ORD-RES-7 (clause-guided literal trail construction)

type-synonym *'f ord-res-7-state* =
'f gclause fset ×*'f gclause fset* × *'f gclause fset* × (*'f gliteral* × *'f gclause option*) *list* ×
'f gclause option

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-6-matches-ord-res-7* ::
'f gterm fset ⇒ *'f ord-res-6-state* ⇒ *'f ord-res-7-state* ⇒ *bool* **where**
ord-res-5-invars N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) ⟹
ord-res-7-invars N (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) ⟹
(∀ *A C*. $\mathcal{M}$ *A* = *Some C* ⟷ *map-of* Γ (*Pos A*) = *Some* (*Some C*)) ⟹
(∀ *A*. $\mathcal{M}$ *A* = *None* ⟷ *map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ) ⟹
i = *atms-of-clss* (*N* |∪| U_{er}) − *trail-atms* Γ ⟹
ord-res-6-matches-ord-res-7 i (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) (*N*, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)

lemma *ord-res-6-final-iff-ord-res-7-final*:
fixes *i S6 S7*
assumes *match*: *ord-res-6-matches-ord-res-7 i S6 S7*
shows *ord-res-6-final S6* ⟷ *ord-res-7-final S7*
using *match*
proof (*cases i S6 S7 rule*: *ord-res-6-matches-ord-res-7.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ $\mathcal{M}$ $\mathcal{C}$ Γ)

show *ord-res-6-final S6* ⟷ *ord-res-7-final S7*
proof (*rule iffI*)
assume *ord-res-6-final S6*
thus *ord-res-7-final S7*
unfolding ‹*S6* = (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$)›
proof (*cases* (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$) *rule*: *ord-res-6-final.cases*)
case *model-found*
thus *ord-res-7-final S7*
unfolding ‹*S7* = (*N*, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)›
using *ord-res-7-final.model-found*
by *metis*
next
case *contradiction-found*
thus *ord-res-7-final S7*
unfolding ‹*S7* = (*N*, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)›
using *ord-res-7-final.contradiction-found*
by *metis*
qed
next
assume *ord-res-7-final S7*
thus *ord-res-6-final S6*
unfolding ‹*S7* = (*N*, U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)›

```
    proof (cases (N, U_er, F, Γ, C) rule: ord-res-7-final.cases)
      case model-found
      thus ord-res-6-final S6
        unfolding ‹S6 = (N, U_er, F, M, C)›
        using ord-res-6-final.model-found
        by metis
    next
      case contradiction-found
      thus ord-res-6-final S6
        unfolding ‹S6 = (N, U_er, F, M, C)›
        using ord-res-6-final.contradiction-found
        by metis
    qed
  qed
qed

lemma backward-simulation-between-6-and-7:
  fixes i S6 S7 S7'
  assumes match: ord-res-6-matches-ord-res-7 i S6 S7 and step: constant-context
ord-res-7 S7 S7'
  shows
    (∃ i' S6'. ord-res-6-step⁺⁺ S6 S6' ∧ ord-res-6-matches-ord-res-7 i' S6' S7') ∨
    (∃ i'. ord-res-6-matches-ord-res-7 i' S6 S7' ∧ i' |⊂| i)
  using match
proof (cases i S6 S7 rule: ord-res-6-matches-ord-res-7.cases)
  case match-hyps: (1 N U_er F M C Γ)

  note S6-def = ‹S6 = (N, U_er, F, M, C)›
  note invars-6 = ‹ord-res-5-invars N (U_er, F, M, C)›
  note invars-7 = ‹ord-res-7-invars N (U_er, F, Γ, C)›[
      unfolded ord-res-7-invars-def, rule-format, OF refl]

  have Γ-sorted: sorted-wrt (λx y. atm-of (fst y) ≺ₜ atm-of (fst x)) Γ
    using invars-7 by argo

  have Γ-consistent: trail-consistent Γ
    using invars-7 by (metis trail-consistent-if-sorted-wrt-atoms)

  hence Γ-distinct-atoms: distinct (map fst Γ)
    using distinct-lits-if-trail-consistent by iprover

  have clause-true-wrt-model-if-true-wrt-Γ: dom M ⊨ D
    if D-true: trail-true-cls Γ D for D
  proof −
    obtain L where L ∈# D and L-true: trail-true-lit Γ L
      using D-true unfolding trail-true-cls-def by auto

    have ∃C. (L, C) ∈ set Γ
      using L-true unfolding trail-true-lit-def by auto
```

```
    show ?thesis
    proof (cases L)
      case (Pos A)

      then obtain C where (Pos A, Some C) ∈ set Γ
        using invars-7 ‹∃C. (L, C) ∈ set Γ›
        by (metis fst-conv literal.disc(1) not-None-eq snd-conv)

      hence map-of Γ (Pos A) = Some (Some C)
        using Γ-distinct-atoms by (metis map-of-is-SomeI)

      hence M A = Some C
       using ‹∀A C. (M A = Some C) = (map-of Γ (Pos A) = Some (Some C))›
by metis

      hence A ∈ dom M
        by blast

      then show ?thesis
        using ‹L ∈# D› ‹L = Pos A› by blast
    next
      case (Neg A)

      hence (Neg A, None) ∈ set Γ
        using invars-7 ‹∃C. (L, C) ∈ set Γ›
        by (metis fst-conv literal.disc(2) snd-conv)

      hence map-of Γ (Neg A) ≠ None
        by (simp add: weak-map-of-SomeI)

      hence M A = None
       using ‹∀A. (M A = None) = (map-of Γ (Neg A) ≠ None ∨ A |∉| trail-atms
Γ)› by metis

      hence A ∉ dom M
        by blast

      then show ?thesis
        using ‹L ∈# D› ‹L = Neg A› by blast
    qed
  qed

  have clause-false-wrt-model-if-false-wrt-Γ: ¬ dom M ⊨ D
    if D-false: trail-false-cls Γ D for D
    unfolding true-cls-def
  proof (intro notI , elim bexE)
    fix L :: 'f gterm literal
    assume L ∈# D and dom M ⊨l L
```

have *trail-false-lit* Γ *L*
using ‹*L* ∈# *D*› *D-false* **unfolding** *trail-false-cls-def* **by** *metis*

hence ¬ *trail-true-lit* Γ *L* **and** *trail-defined-lit* Γ *L*
unfolding *atomize-conj*
using Γ*-consistent* ‹*L* ∈# *D*› *not-trail-true-cls-and-trail-false-cls that*
trail-defined-lit-iff-true-or-false trail-true-cls-def **by** *blast*

show *False*
proof (*cases L*)
case (*Pos A*)

hence *M A* ≠ *None*
using ‹*dom M* ⊨*l L*› **by** *blast*

hence *map-of* Γ (*Pos A*) ≠ *None*
using ‹∀ *A C*. (*M A* = *Some C*) = (*map-of* Γ (*Pos A*) = *Some* (*Some C*))›
by *blast*

hence *Pos A* ∈ *fst* ‘ *set* Γ
by (*simp add*: *map-of-eq-None-iff*)

hence *trail-true-lit* Γ (*Pos A*)
unfolding *trail-true-lit-def* **.**

moreover have ¬ *trail-true-lit* Γ (*Pos A*)
using ‹¬ *trail-true-lit* Γ *L*› ‹*L* = *Pos A*› **by** *argo*

ultimately show *False*
by *contradiction*
next
case (*Neg A*)

hence *M A* = *None*
using ‹*dom M* ⊨*l L*› **by** *blast*

hence *map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ
using ‹∀ *A*. (*M A* = *None*) = (*map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms*
Γ)› **by** *blast*

hence *trail-true-lit* Γ (*Neg A*) ∨ ¬ *trail-defined-lit* Γ (*Neg A*)
unfolding *map-of-eq-None-iff not-not*
unfolding *trail-true-lit-def trail-defined-lit-iff-trail-defined-atm literal.sel*
.

then show *?thesis*
using ‹¬ *trail-true-lit* Γ *L*› ‹*trail-defined-lit* Γ *L*› ‹*L* = *Neg A*› **by** *argo*
qed

qed

obtain $s\mathcal{7}'$ **where**
$S\mathcal{7}' = (N,\ s\mathcal{7}')$ **and**
step': *ord-res-7* N $(U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$ $s\mathcal{7}'$
using *step* **unfolding** ‹$S\mathcal{7} = (N,\ U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›
by (*auto elim*: *constant-context.cases*)

have *invars-s7'*: *ord-res-7-invars* N $s\mathcal{7}'$
using *ord-res-7-preserves-invars*[*OF step'* ‹*ord-res-7-invars* N $(U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›]
.

show *?thesis*
using *step'*
proof (*cases* N $(U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$ $s\mathcal{7}'$ *rule*: *ord-res-7.cases*)
case *step-hyps*: (*decide-neg* C L A Γ')

define i' **where**
$i' =$ *atms-of-clss* $(N\ |\cup|\ U_{er})\ |-|$ *trail-atms* Γ'

have $A\ |\in|$ *atms-of-clss* $(N\ |\cup|\ U_{er})$ **and** $A \prec_t$ *atm-of* L **and** $A\ |\notin|$ *trail-atms* Γ
using *step-hyps* **unfolding** *atomize-conj linorder-trm.is-least-in-ffilter-iff* **by** *argo*

have *ord-res-6-matches-ord-res-7* i' *S6* $S\mathcal{7}'$
unfolding *S6-def* ‹$\mathcal{C} =$ *Some* C› ‹$S\mathcal{7}' = (N,\ s\mathcal{7}')$› ‹$s\mathcal{7}' = (U_{er},\ \mathcal{F},\ \Gamma',$ *Some* $C)$›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars* N $(U_{er},\ \mathcal{F},\ \mathcal{M},$ *Some* $C)$
using *invars-6* **unfolding** ‹$\mathcal{C} =$ *Some* C› **.**
next
show *ord-res-7-invars* N $(U_{er},\ \mathcal{F},\ \Gamma',$ *Some* $C)$
using *invars-s7'* **unfolding** ‹$s\mathcal{7}' = (U_{er},\ \mathcal{F},\ \Gamma',$ *Some* $C)$› **.**
next
show $\forall\, A\ C.\ (\mathcal{M}\ A =$ *Some* $C) = (map\text{-}of\ \Gamma'\ (Pos\ A) =$ *Some* (*Some* $C))$
using *match-hyps* **unfolding** ‹$\Gamma' = (Neg\ A,\ None)\ \#\ \Gamma$› **by** *simp*
next
show $\forall\, A.\ (\mathcal{M}\ A = None) = (map\text{-}of\ \Gamma'\ (Neg\ A) \neq None \vee A\ |\notin|$ *trail-atms* $\Gamma')$
unfolding ‹$\Gamma' = (Neg\ A,\ None)\ \#\ \Gamma$›
using *match-hyps* ‹$A\ |\notin|$ *trail-atms* Γ› **by** *force*
next
show $i' =$ *atms-of-clss* $(N\ |\cup|\ U_{er})\ |-|$ *trail-atms* Γ'
unfolding *i'-def* **..**
qed

moreover have $i'\ |\subset|\ i$
proof −

have $i = \mathit{finsert}\ A\ i'$
unfolding *match-hyps i'-def*
using ‹A |∈| *atms-of-clss* (N |∪| U_{er})› ‹A |∉| *trail-atms* Γ› *step-hyps*(6) **by** *force*

moreover have A |∉| i'
unfolding *i'-def*
using *step-hyps*(6) **by** *fastforce*

ultimately show *?thesis*
by *auto*
qed

ultimately show *?thesis*
by *metis*
next
case *step-hyps*: (*skip-defined C L* $\mathcal{C}'$)

define *S6'* **where**
S6' = (N, U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}'$)

have *C-almost-defined*: *trail-defined-cls* Γ {#x ∈# C. $x \neq L$#}
using *step-hyps* **by** (*metis clause-almost-definedI invars-7*)

hence *C-defined*: *trail-defined-cls* Γ C
using *step-hyps* **unfolding** *trail-defined-cls-def* **by** *auto*

hence *C-true*: *trail-true-cls* Γ C
using *step-hyps* **by** (*metis trail-true-or-false-cls-if-defined*)

have *step6*: *ord-res-6* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some* C) (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}'$)
proof (*rule ord-res-6.skip*)
show *dom* $\mathcal{M}$ ⊨ C
using *clause-true-wrt-model-if-true-wrt-*Γ[*OF C-true*] **.**
next
show $\mathcal{C}'$ = *The-optional* (*linorder-cls.is-least-in-fset*
(*ffilter* ((≺$_c$) C) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}))))
using *step-hyps* **by** *argo*
qed

hence *ord-res-6-step*$^{++}$ *S6 S6'*
using *S6-def* ‹$\mathcal{C}$ = *Some C*› *S6'-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i S6' S7'*
unfolding *S6'-def* ‹*S7'* = (N, *s7'*)› ‹*s7'* = (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}'$)›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars* N (U_{er}, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}'$)
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*

```
    next
      show ord-res-7-invars N (U_er, F, Γ, C')
        using invars-s7' unfolding ‹s7' = (U_er, F, Γ, C')› .
    next
      show ∀A C. (M A = Some C) = (map-of Γ (Pos A) = Some (Some C))
        using match-hyps by argo
    next
      show ∀A. (M A = None) = (map-of Γ (Neg A) ≠ None ∨ A |∉| trail-atms
Γ)
        using match-hyps by argo
    next
      show i = atms-of-clss (N |∪| U_er) |−| trail-atms Γ
        using match-hyps by argo
    qed

    ultimately show ?thesis
      by metis
  next
    case step-hyps: (skip-undefined-neg C L Γ' C')

    define S6' where
      S6' = (N, U_er, F, M, C')

    define i' where
      i' = atms-of-clss (N |∪| U_er) |−| trail-atms Γ'

    have trail-true-lit Γ' L
      unfolding ‹Γ' = (L, None) # Γ› by (simp add: trail-true-lit-def)

    hence C-true: trail-true-cls Γ' C
     using step-hyps unfolding linorder-lit.is-maximal-in-mset-iff trail-true-cls-def
by metis

    have step6: ord-res-6 N (U_er, F, M, Some C) (U_er, F, M, C')
    proof (rule ord-res-6.skip)
      show dom M ⊨ C
        using C-true
      by (metis domIff linorder-lit.is-maximal-in-mset-iff literal.collapse(2) match-hyps(6)
          step-hyps(4) step-hyps(6) step-hyps(7) trail-defined-lit-iff-trail-defined-atm
            true-cls-def true-lit-simps(2))
    next
      show C' = The-optional (linorder-cls.is-least-in-fset
        (ffilter ((≺c) C) (iefac F |`| (N |∪| U_er))))
        using step-hyps by argo
    qed

    hence ord-res-6-step++ S6 S6'
      using S6-def ‹C = Some C› S6'-def ord-res-6-step.intros by blast
```

moreover have *ord-res-6-matches-ord-res-7 i' S6' S7'*
unfolding *S6'-def* ‹*S7'* = (*N*, *s7'*)› ‹*s7'* = (*U*$_{er}$, $\mathcal{F}$, Γ', $\mathcal{C}$')›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{C}$')
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (*U*$_{er}$, $\mathcal{F}$, Γ', $\mathcal{C}$')
using *invars-s7'* **unfolding** ‹*s7'* = (*U*$_{er}$, $\mathcal{F}$, Γ', $\mathcal{C}$')› **.**
next
show ∀ *A C*. ($\mathcal{M}$ *A* = *Some C*) = (*map-of* Γ' (*Pos A*) = *Some* (*Some C*))
using *match-hyps*
unfolding ‹Γ' = (*L*, *None*) # Γ›
by (*metis literal.disc*(*1*) *map-of-Cons-code*(*2*) *step-hyps*(*7*))
next
show ∀ *A*. ($\mathcal{M}$ *A* = *None*) = (*map-of* Γ' (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ')
using *match-hyps*
unfolding ‹Γ' = (*L*, *None*) # Γ›
by (*metis finsert-iff literal.collapse*(*2*) *literal.sel*(*2*) *map-of-Cons-code*(*2*) *option.discI*
prod.sel(*1*) *step-hyps*(*6*) *step-hyps*(*7*) *trail-atms.simps*(*2*)
trail-defined-lit-iff-trail-defined-atm)
next
show *i'* = *atms-of-clss* (*N* |∪| *U*$_{er}$) |−| *trail-atms* Γ'
using *i'-def* **.**
qed

ultimately show *?thesis*
by *metis*
next
case *step-hyps*: (*skip-undefined-pos C L D*)

define *S6'* **where**
S6' = (*N*, *U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some D*)

have *trail-defined-cls* Γ {#*x* ∈# *C*. *x* ≠ *L* ∧ *x* ≠ − *L*#}
proof (*rule clause-almost-almost-definedI*)
show *C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$)
using *invars-7 step-hyps* **by** *metis*
next
show *ord-res.is-maximal-lit L C*
using *step-hyps* **by** *argo*
next
show ¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| *U*$_{er}$). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ)
using *step-hyps* **by** *argo*
qed

moreover have − *L* ∉# *C*

by (*metis atm-of-uminus is-pos-def linorder-lit.is-maximal-in-mset-iff linorder-lit.neqE linorder-trm.less-irrefl literal.collapse*(2) *literal.sel*(1) *ord-res.less-lit-simps*(4) *step-hyps*(4) *step-hyps*(7) *uminus-not-id'*)

ultimately have *trail-defined-cls* Γ {#*x* ∈# *C*. *x* ≠ *L*#}
unfolding *trail-defined-cls-def* **by** *auto*

hence *trail-true-cls* Γ {#*x* ∈# *C*. *x* ≠ *L*#}
using ‹¬ *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*#}› **by** (*metis trail-true-or-false-cls-if-defined*)

hence *C-true*: *trail-true-cls* Γ *C*
by (*auto simp*: *trail-true-cls-def*)

have *step6*: *ord-res-6 N* (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some C*) (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some D*)
proof (*rule ord-res-6.skip*)
show *dom* $\mathcal{M}$ ⊨ *C*
using *clause-true-wrt-model-if-true-wrt-*Γ[*OF C-true*] **.**
next
show *Some D* = *The-optional* (*linorder-cls.is-least-in-fset*
(*ffilter* ((≺$_c$) *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| *U*$_{er}$))))
using *linorder-cls.Uniq-is-least-in-fset step-hyps*(9) *The-optional-eq-SomeI*
by *fastforce*
qed

hence *ord-res-6-step*$^{++}$ *S6 S6'*
using *S6-def* ‹$\mathcal{C}$ = *Some C*› *S6'-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i S6' S7'*
unfolding *S6'-def* ‹*S7'* = (*N*, *s7'*)› ‹*s7'* = (*U*$_{er}$, $\mathcal{F}$, Γ, *Some D*)›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some D*)
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (*U*$_{er}$, $\mathcal{F}$, Γ, *Some D*)
using *invars-s7'* **unfolding** ‹*s7'* = (*U*$_{er}$, $\mathcal{F}$, Γ, *Some D*)› **.**
next
show ∀ *A C*. ($\mathcal{M}$ *A* = *Some C*) = (*map-of* Γ (*Pos A*) = *Some* (*Some C*))
using *match-hyps* **by** *argo*
next
show ∀ *A*. ($\mathcal{M}$ *A* = *None*) = (*map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ)
using *match-hyps* **by** *argo*
next
show *i* = *atms-of-clss* (*N* |∪| *U*$_{er}$) |−| *trail-atms* Γ
using *match-hyps* **by** *argo*
qed

ultimately show *?thesis*

```
    by metis
  next
    case step-hyps: (skip-undefined-pos-ultimate C L Γ')

    define S6' where
      S6' = (N, U_er, F, M, None :: 'f gclause option)

    define i' where
      i' = atms-of-clss (N |∪| U_er) |-| trail-atms Γ'

    have trail-defined-cls Γ {#x ∈# C. x ≠ L ∧ x ≠ - L#}
    proof (rule clause-almost-almost-definedI)
      show C |∈| iefac F |`| (N |∪| U_er)
        using invars-7 step-hyps by metis
    next
      show ord-res.is-maximal-lit L C
        using step-hyps by argo
    next
     show ¬ (∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of L ∧ A |∉| trail-atms Γ)
        using step-hyps by argo
    qed

    moreover have - L ∉# C
    by (metis atm-of-uminus is-pos-def linorder-lit.is-maximal-in-mset-iff linorder-lit.neqE
       linorder-trm.less-irrefl literal.collapse(2) literal.sel(1) ord-res.less-lit-simps(4)
         step-hyps(4) step-hyps(7) uminus-not-id')

    ultimately have trail-defined-cls Γ {#x ∈# C. x ≠ L#}
      unfolding trail-defined-cls-def by auto

    hence trail-true-cls Γ {#x ∈# C. x ≠ L#}
    using ‹¬ trail-false-cls Γ {#K ∈# C. K ≠ L#}› by (metis trail-true-or-false-cls-if-defined)

    hence C-true: trail-true-cls Γ C
      by (auto simp: trail-true-cls-def)

    have step6: ord-res-6 N (U_er, F, M, Some C) (U_er, F, M, None)
    proof (rule ord-res-6.skip)
      show dom M ⊨ C
        using clause-true-wrt-model-if-true-wrt-Γ[OF C-true] .
    next
      have ¬ (∃ D. linorder-cls.is-least-in-fset (ffilter ((≺_c) C) (iefac F |`| (N |∪|
U_er))) D)
        using ‹¬ fBex (iefac F |`| (N |∪| U_er)) ((≺_c) C)›
        by (meson linorder-cls.is-least-in-ffilter-iff)

      thus None = The-optional (linorder-cls.is-least-in-fset
        (ffilter ((≺_c) C) (iefac F |`| (N |∪| U_er))))
        unfolding The-optional-def by metis
```

qed

hence *ord-res-6-step*$^{++}$ *S6 S6′*
using *S6-def* ‹$\mathcal{C}$ = *Some C*› *S6′-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i′ S6′ S7′*
unfolding *S6′-def* ‹*S7′* = (*N*, *s7′*)› ‹*s7′* = (U_{er}, $\mathcal{F}$, Γ′, *None*)›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *None*)
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (U_{er}, $\mathcal{F}$, Γ′, *None*)
using *invars-s7′* **unfolding** ‹*s7′* = (U_{er}, $\mathcal{F}$, Γ′, *None*)› **.**
next
show ∀ *A C*. ($\mathcal{M}$ *A* = *Some C*) = (*map-of* Γ′ (*Pos A*) = *Some* (*Some C*))
using *match-hyps*(*3*−)
unfolding ‹Γ′ = (− *L*, *None*) # Γ›
by (*metis is-pos-neg-not-is-pos literal.disc*(*1*) *map-of-Cons-code*(*2*) *step-hyps*(*7*))
next
show ∀ *A*. ($\mathcal{M}$ *A* = *None*) = (*map-of* Γ′ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ′)
using *match-hyps*(*3*−)
unfolding ‹Γ′ = (− *L*, *None*) # Γ›
by (*metis* (*no-types*, *opaque-lifting*) *atm-of-eq-atm-of eq-fst-iff fset-simps*(*2*) *insertCI*
insertE literal.discI(*2*) *literal.sel*(*2*) *map-of-Cons-code*(*2*) *option.distinct*(*1*)
trail-defined-lit-iff-trail-defined-atm step-hyps(*6*) *step-hyps*(*7*) *trail-atms.simps*(*2*))
next
show *i′* = *atms-of-clss* (*N* |∪| U_{er}) |−| *trail-atms* Γ′
using *i′-def* **.**
qed

ultimately show *?thesis*
by *metis*
next
case *step-hyps*: (*production C L* Γ′ $\mathcal{C}$′)

define *S6′* **where**
S6′ = (*N*, U_{er}, $\mathcal{F}$, $\mathcal{M}$(*atm-of L* ↦ *C*), $\mathcal{C}$′)

define *i′* **where**
i′ = *atms-of-clss* (*N* |∪| U_{er}) |−| *trail-atms* Γ′

have *L* ∈# *C*
using *step-hyps* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

have *step6*: *ord-res-6 N* (U_{er}, $\mathcal{F}$, $\mathcal{M}$, *Some C*) (U_{er}, $\mathcal{F}$, $\mathcal{M}$(*atm-of L* ↦ *C*), $\mathcal{C}$′)

proof (*rule ord-res-6.production*)
have *atm-of L* |∉| *trail-atms* Γ
using ‹¬ *trail-defined-lit* Γ *L*›
unfolding *trail-defined-lit-iff-trail-defined-atm* **.**

hence *M* (*atm-of L*) = *None*
using *match-hyps*(*3*−) **by** *metis*

hence *atm-of L* ∉ *dom M*
unfolding *dom-def* **by** *simp*

hence ¬ *dom M* ⟨=*l L*
using ‹*is-pos L*› **unfolding** *true-lit-def* **by** *metis*

moreover have ¬ *dom M* ⟨= {#*K* ∈# *C*. *K* ≠ *L*#}
using *clause-false-wrt-model-if-false-wrt-*Γ[*OF* ‹*trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*#}›] **.**

ultimately show ¬ *dom M* ⟨= *C*
using ‹*L* ∈# *C*›
unfolding *true-cls-def* **by** *auto*
next
show *ord-res.is-maximal-lit L C*
using *step-hyps* **by** *argo*
next
show *is-pos L*
using *step-hyps* **by** *argo*
next
show *ord-res.is-strictly-maximal-lit L C*
using *step-hyps* **by** *argo*
next
show *M*(*atm-of L* ↦ *C*) = *M*(*atm-of L* ↦ *C*) **..**
next
show *C′* = *The-optional* (*linorder-cls.is-least-in-fset*
(*ffilter* ((≺*c*) *C*) (*iefac F* |‘| (*N* |∪| *Ue r*))))
using *step-hyps* **by** *argo*
qed

hence *ord-res-6-step*++ *S6 S6′*
using *S6-def* ‹*C* = *Some C*› *S6′-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i′ S6′ S7′*
unfolding *S6′-def* ‹*S7′* = (*N*, *s7′*)› ‹*s7′* = (*Ue r*, *F*, Γ′, *C′*)›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (*Ue r*, *F*, *M*(*atm-of L* ↦ *C*), *C′*)
using *invars-6* **unfolding** ‹*C* = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (*Ue r*, *F*, Γ′, *C′*)

using *invars-s7′* **unfolding** ‹$s7' = (U_{er}, \mathcal{F}, \Gamma', \mathcal{C}')$› **.**
next
show $\forall A\ D.\ ((\mathcal{M}(\textit{atm-of}\ L \mapsto C))\ A = \textit{Some}\ D) = (\textit{map-of}\ \Gamma'\ (\textit{Pos}\ A) = \textit{Some}\ (\textit{Some}\ D))$
using *match-hyps*(3−)
unfolding ‹$\Gamma' = (L, \textit{Some}\ C) \#\ \Gamma$›
using *step-hyps*(7) **by** *auto*
next
show $\forall A.\ ((\mathcal{M}(\textit{atm-of}\ L \mapsto C))\ A = \textit{None}) = (\textit{map-of}\ \Gamma'\ (\textit{Neg}\ A) \neq \textit{None} \vee A\ |\notin|\ \textit{trail-atms}\ \Gamma')$
using *match-hyps*(3−)
unfolding ‹$\Gamma' = (L, \textit{Some}\ C) \#\ \Gamma$›
by (*metis* (*no-types*, *opaque-lifting*) *domI domIff finsert-iff fun-upd-apply literal.collapse*(1) *literal.discI*(2) *map-of-Cons-code*(2) *map-of-eq-None-iff prod.sel*(1) *step-hyps*(6) *step-hyps*(7) *trail-atms.simps*(2) *trail-defined-lit-def uminus-Pos*)
next
show $i' = \textit{atms-of-clss}\ (N\ |\cup|\ U_{er})\ |-|\ \textit{trail-atms}\ \Gamma'$
using *i′-def* **.**
qed

ultimately show *?thesis*
by *metis*
next
case *step-hyps*: (*factoring C L* $\mathcal{F}'$)

define *S6′* **where**
S6′ = $(N, U_{er}, \mathcal{F}', \mathcal{M}, \textit{Some}\ (\textit{efac}\ C))$

have $L \in\# C$
using *step-hyps* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

have *step6*: *ord-res-6 N* $(U_{er}, \mathcal{F}, \mathcal{M}, \textit{Some}\ C)$ $(U_{er}, \mathcal{F}', \mathcal{M}, \textit{Some}\ (\textit{efac}\ C))$
proof (*rule ord-res-6.factoring*)
have *atm-of L* $|\notin|$ *trail-atms* Γ
using ‹¬ *trail-defined-lit* Γ *L*›
unfolding *trail-defined-lit-iff-trail-defined-atm* **.**

hence $\mathcal{M}\ (\textit{atm-of}\ L) = \textit{None}$
using *match-hyps*(3−) **by** *metis*

hence $\textit{atm-of}\ L \notin \textit{dom}\ \mathcal{M}$
unfolding *dom-def* **by** *simp*

hence ¬ *dom* $\mathcal{M} \models l\ L$
using ‹*is-pos L*› **unfolding** *true-lit-def* **by** *metis*

moreover have ¬ *dom* $\mathcal{M} \models \{\#K \in\# C.\ K \neq L\#\}$

using *clause-false-wrt-model-if-false-wrt-*Γ[*OF* ‹*trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*#}›] **.**

ultimately show ¬ *dom* $\mathcal{M}$ ⊨ *C*
using ‹*L* ∈# *C*›
unfolding *true-cls-def* **by** *auto*
next
show *ord-res.is-maximal-lit L C*
using *step-hyps* **by** *argo*
next
show *is-pos L*
using *step-hyps* **by** *argo*
next
show ¬ *ord-res.is-strictly-maximal-lit L C*
using *step-hyps* **by** *argo*
next
show $\mathcal{F}'$ = *finsert C* $\mathcal{F}$
using *step-hyps* **by** *argo*
qed

hence *ord-res-6-step*$^{++}$ *S6 S6'*
using *S6-def* ‹$\mathcal{C}$ = *Some C*› *S6'-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i S6' S7'*
unfolding *S6'-def* ‹*S7'* = (*N*, *s7'*)› ‹*s7'* = (U_{er}, $\mathcal{F}'$, Γ, *Some* (*efac C*))›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (U_{er}, $\mathcal{F}'$, $\mathcal{M}$, *Some* (*efac C*))
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some C*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (U_{er}, $\mathcal{F}'$, Γ, *Some* (*efac C*))
using *invars-s7'* **unfolding** ‹*s7'* = (U_{er}, $\mathcal{F}'$, Γ, *Some* (*efac C*))› **.**
next
show ∀ *A C*. ($\mathcal{M}$ *A* = *Some C*) = (*map-of* Γ (*Pos A*) = *Some* (*Some C*))
using *match-hyps*(3−) **by** *argo*
next
show ∀ *A*. ($\mathcal{M}$ *A* = *None*) = (*map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ)
using *match-hyps*(3−) **by** *argo*
next
show *i* = *atms-of-clss* (*N* |∪| U_{er}) |−| *trail-atms* Γ
using *match-hyps*(3−) **by** *argo*
qed

ultimately show *?thesis*
by *metis*
next
case *step-hyps*: (*resolution-bot E L D* U_{er}' Γ')

define *S6′* **where**
S6′ = (*N*, *U*$_{er}$′, $\mathcal{F}$, (λ-. *None*) :: *'f gterm* ⇒ *'f gclause option*, *Some* ({#} :: *'f gclause*))

define *i′* **where**
i′ = *atms-of-clss* (*N* |∪| *U*$_{er}$′) |−| *trail-atms* Γ′

have *step6*: *ord-res-6 N* (*U*$_{er}$, $\mathcal{F}$, $\mathcal{M}$, *Some E*) (*U*$_{er}$′, $\mathcal{F}$, λ-. *None*, *Some* {#})
proof (*rule ord-res-6.resolution-bot*)
show ¬ *dom* $\mathcal{M}$ ⊨ *E*
using *clause-false-wrt-model-if-false-wrt-*Γ[*OF* ‹*trail-false-cls* Γ *E*›] .
next
show *ord-res.is-maximal-lit L E*
using *step-hyps* **by** *argo*
next
show *is-neg L*
using *step-hyps* **by** *argo*
next
show $\mathcal{M}$ (*atm-of L*) = *Some D*
by (*metis literal.collapse*(2) *match-hyps*(5) *step-hyps*(5) *step-hyps*(6) *uminus-Neg*)
next
show *U*$_{er}$′ = *finsert* (*eres D E*) *U*$_{er}$
using *step-hyps* **by** *argo*
next
show *eres D E* = {#}
using *step-hyps* **by** *argo*
next
show ((λ-. *None*)) = (λ-. *None*) **..**
qed

hence *ord-res-6-step*$^{++}$ *S6 S6′*
using *S6-def* ‹$\mathcal{C}$ = *Some E*› *S6′-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i′ S6′ S7′*
unfolding *S6′-def* ‹*S7′* = (*N*, *s7′*)› ‹*s7′* = (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* {#})›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (*U*$_{er}$′, $\mathcal{F}$, λ-. *None*, *Some* {#})
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some E*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* {#})
using *invars-s7′* **unfolding** ‹*s7′* = (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* {#})› .
next
show ∀ *A C*. (*None* = *Some C*) = (*map-of* Γ′ (*Pos A*) = *Some* (*Some C*))
unfolding ‹Γ′ = []› **by** *simp*
next
show ∀ *A*. (*None* = *None*) = (*map-of* Γ′ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ′)

```
      unfolding ‹Γ' = []› by simp
    next
      show i' = atms-of-clss (N |∪| U_er') |−| trail-atms Γ'
        using i'-def .
    qed

    ultimately show ?thesis
      by metis
  next
    case step-hyps: (resolution-pos E L D U_er' Γ' K)

    define S6' where
      S6' = (N, U_er', F, restrict-map M {A. A ≺_t atm-of K}, Some (eres D E))

    define i' where
      i' = atms-of-clss (N |∪| U_er') |−| trail-atms Γ'

    have mem-set-Γ'-iff: ⋀x. (x ∈ set Γ') = (atm-of (fst x) ≺_t atm-of K ∧ x ∈
set Γ)
      unfolding ‹Γ' = dropWhile (λLn. atm-of K ⪯_t atm-of (fst Ln)) Γ›
    unfolding mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone[OF Γ-sorted
mono-atms-lt]
      by auto

    have step6: ord-res-6 N (U_er, F, M, Some E) (U_er', F, restrict-map M {A.
A ≺_t atm-of K}, Some (eres D E))
    proof (rule ord-res-6.resolution-pos)
      show ¬ dom M ⊨ E
        using clause-false-wrt-model-if-false-wrt-Γ[OF ‹trail-false-cls Γ E›] .
    next
      show ord-res.is-maximal-lit L E
        using step-hyps by argo
    next
      show is-neg L
        using step-hyps by argo
    next
      show M (atm-of L) = Some D
        by (metis literal.collapse(2) match-hyps(5) step-hyps(5) step-hyps(6) umi-
nus-Neg)
    next
      show U_er' = finsert (eres D E) U_er
        using step-hyps by argo
    next
      show eres D E ≠ {#}
        using step-hyps by argo
    next
     show restrict-map M {A. A ≺_t atm-of K} = restrict-map M {A. A ≺_t atm-of
K}
       ..
```

next
show *ord-res.is-maximal-lit K* (*eres D E*)
using *step-hyps* **by** *argo*
next
show *is-pos K*
using *step-hyps* **by** *argo*
qed

hence *ord-res-6-step*$^{++}$ *S6 S6′*
using *S6-def* ‹$\mathcal{C}$ = *Some E*› *S6′-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i′ S6′ S7′*
unfolding *S6′-def* ‹*S7′* = (*N*, *s7′*)› ‹*s7′* = (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* (*eres D E*))›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (*U*$_{er}$′, $\mathcal{F}$, *restrict-map* $\mathcal{M}$ {*A*. *A* ≺$_t$ *atm-of K*}, *Some* (*eres D E*))
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some E*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* (*eres D E*))
using *invars-s7′* **unfolding** ‹*s7′* = (*U*$_{er}$′, $\mathcal{F}$, Γ′, *Some* (*eres D E*))› **.**
next
show ∀ *A C*. (*restrict-map* $\mathcal{M}$ {*A*. *A* ≺$_t$ *atm-of K*} *A* = *Some C*) = (*map-of* Γ′ (*Pos A*) = *Some* (*Some C*))
proof (*intro allI*)
fix *A* :: *'f gterm* **and** *C* :: *'f gclause*
show *restrict-map* $\mathcal{M}$ {*A*. *A* ≺$_t$ *atm-of K*} *A* = *Some C* ⟷ *map-of* Γ′ (*Pos A*) = *Some* (*Some C*)
proof (*cases A* ∈ *dom* $\mathcal{M}$ ∧ *A* ≺$_t$ *atm-of K*)
case *True*
have *restrict-map* $\mathcal{M}$ {*A*. *A* ≺$_t$ *atm-of K*} *A* = *Some C* ⟷ $\mathcal{M}$ *A* = *Some C*
using *True* **by** *simp*

also have ... ⟷ *map-of* Γ (*Pos A*) = *Some* (*Some C*)
using *match-hyps*(*3*−) **by** *metis*

also have ... ⟷ *map-of* Γ′ (*Pos A*) = *Some* (*Some C*)
proof −
have *Pos A* ∈ *fst* ‘ *set* Γ
using *True*
by (*metis domIff map-of-eq-None-iff match-hyps*(*5*) *not-None-eq*)

hence ∃$\mathcal{C}$. (*Pos A*, $\mathcal{C}$) ∈ *set* Γ
by *fastforce*

hence ∃$\mathcal{C}$. (*Pos A*, $\mathcal{C}$) ∈ *set* Γ ∧ (*Pos A*, $\mathcal{C}$) ∈ *set* Γ′
using *True* **unfolding** *mem-set-Γ′-iff prod.sel literal.sel* **by** *metis*

```
      moreover have distinct (map fst Γ′)
        using Γ-distinct-atoms
      proof (rule distinct-suffix)
        show suffix (map fst Γ′) (map fst Γ)
          using map-mono-suffix step-hyps(9) suffix-dropWhile by blast
      qed

      ultimately have map-of Γ (Pos A) = map-of Γ′ (Pos A)
        using Γ-distinct-atoms by (auto dest: map-of-is-SomeI)

      thus ?thesis
        by argo
    qed

    finally show ?thesis .
  next
    case False
    have restrict-map M {A. A ≺t atm-of K} A = None
      using False unfolding restrict-map-def by auto

    moreover have map-of Γ′ (Pos A) ≠ Some (Some C)
      using False unfolding de-Morgan-conj
    proof (elim disjE)
      assume A ∉ dom M

      hence ⋀C. (Pos A, C) ∉ set Γ
        using match-hyps(5)
        by (metis (no-types, opaque-lifting) domIff fst-eqD invars-7 is-pos-def
map-of-SomeD
            not-None-eq snd-conv weak-map-of-SomeI)

      hence ⋀C. (Pos A, C) ∉ set Γ′
        unfolding mem-set-Γ′-iff by simp

      then show map-of Γ′ (Pos A) ≠ Some (Some C)
        by (meson map-of-SomeD)
    next
      assume ¬ A ≺t atm-of K

      hence ⋀C. (Pos A, C) ∉ set Γ′
        unfolding mem-set-Γ′-iff by simp

      then show map-of Γ′ (Pos A) ≠ Some (Some C)
        by (meson map-of-SomeD)
    qed

    ultimately show ?thesis
      by simp
  qed
```

```
    qed
  next
    show ∀A. (restrict-map M {A. A ≺t atm-of K} A = None) =
      (map-of Γ′ (Neg A) ≠ None ∨ A |∉| trail-atms Γ′)
    proof (intro allI)
      fix A :: 'f gterm
      show (restrict-map M {A. A ≺t atm-of K} A = None) =
        (map-of Γ′ (Neg A) ≠ None ∨ A |∉| trail-atms Γ′)
      proof (cases A ≺t atm-of K)
        case True

        have restrict-map M {A. A ≺t atm-of K} A = None ⟷ M A = None
          using True by simp

        also have ... ⟷ map-of Γ (Neg A) ≠ None ∨ A |∉| trail-atms Γ
          using match-hyps(6) by metis

        also have ... ⟷ map-of Γ′ (Neg A) ≠ None ∨ A |∉| trail-atms Γ
          using True mem-set-Γ′-iff
       by (metis eq-fst-iff literal.sel(2) map-of-SomeD not-None-eq weak-map-of-SomeI)

        also have ... ⟷ map-of Γ′ (Neg A) ≠ None ∨ A |∉| trail-atms Γ′
          using True mem-set-Γ′-iff
          by (smt (verit, best) fset-trail-atms image-iff)

        finally show ?thesis .
      next
        case False

        have restrict-map M {A. A ≺t atm-of K} A = None
          using False by simp

        moreover have A |∉| trail-atms Γ′
          using False mem-set-Γ′-iff
          by (smt (verit, ccfv-threshold) fset-trail-atms image-iff)

        ultimately show ?thesis
          by metis
      qed
    qed
  next
    show i′ = atms-of-clss (N |∪| U_er′) |−| trail-atms Γ′
      using i′-def .
  qed

  ultimately show ?thesis
    by metis
next
  case step-hyps: (resolution-neg E L D U_er′ Γ′ K C)
```

define $S6'$ **where**
$S6' = (N,\ U_{er}',\ \mathcal{F},\ \textit{restrict-map}\ \mathcal{M}\ \{A.\ A \prec_t \textit{atm-of}\ K\},\ \textit{Some}\ C)$

define i' **where**
$i' = \textit{atms-of-clss}\ (N\ |\cup|\ U_{er}')\ |-|\ \textit{trail-atms}\ \Gamma'$

have *mem-set-*Γ'*-iff*: $\bigwedge x.\ (x \in \textit{set}\ \Gamma') = (\textit{atm-of}\ (\textit{fst}\ x) \prec_t \textit{atm-of}\ K \wedge x \in \textit{set}\ \Gamma)$
unfolding $\langle \Gamma' = \textit{dropWhile}\ (\lambda Ln.\ \textit{atm-of}\ K \preceq_t \textit{atm-of}\ (\textit{fst}\ Ln))\ \Gamma \rangle$
unfolding *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*[*OF* Γ*-sorted mono-atms-lt*]
by *auto*

have *step6*: *ord-res-6* $N\ (U_{er}, \mathcal{F}, \mathcal{M}, \textit{Some}\ E)\ (U_{er}', \mathcal{F}, \textit{restrict-map}\ \mathcal{M}\ \{A.\ A \prec_t \textit{atm-of}\ K\}, \textit{Some}\ C)$
proof (*rule ord-res-6.resolution-neg*)
show $\neg\ \textit{dom}\ \mathcal{M} \models E$
using *clause-false-wrt-model-if-false-wrt-*Γ[*OF* $\langle$*trail-false-cls* Γ $E\rangle$] **.**
next
show *ord-res.is-maximal-lit* L E
using *step-hyps* **by** *argo*
next
show *is-neg* L
using *step-hyps* **by** *argo*
next
show $\mathcal{M}\ (\textit{atm-of}\ L) = \textit{Some}\ D$
by (*metis literal.collapse*(2) *match-hyps*(5) *step-hyps*(5) *step-hyps*(6) *uminus-Neg*)
next
show $U_{er}' = \textit{finsert}\ (\textit{eres}\ D\ E)\ U_{er}$
using *step-hyps* **by** *argo*
next
show $\textit{eres}\ D\ E \neq \{\#\}$
using *step-hyps* **by** *argo*
next
show *restrict-map* $\mathcal{M}\ \{A.\ A \prec_t \textit{atm-of}\ K\} = \textit{restrict-map}\ \mathcal{M}\ \{A.\ A \prec_t \textit{atm-of}\ K\}$
..
next
show *ord-res.is-maximal-lit* $K\ (\textit{eres}\ D\ E)$
using *step-hyps* **by** *argo*
next
show *is-neg* K
using *step-hyps* **by** *argo*
next
show $\mathcal{M}\ (\textit{atm-of}\ K) = \textit{Some}\ C$
using *match-hyps*(3−)
by (*metis* (*mono-tags*, *lifting*) *step-hyps*(11) *step-hyps*(12) *uminus-literal-def*)

qed

hence *ord-res-6-step*$^{++}$ *S6 S6'*
using *S6-def* ‹$\mathcal{C}$ = *Some E*› *S6'-def ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7 i' S6' S7'*
unfolding *S6'-def* ‹*S7'* = (*N*, *s7'*)› ‹*s7'* = (U_{er}', $\mathcal{F}$, Γ', *Some C*)›
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars N* (U_{er}', $\mathcal{F}$, *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*}, *Some C*)
using *invars-6* **unfolding** ‹$\mathcal{C}$ = *Some E*›
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*
next
show *ord-res-7-invars N* (U_{er}', $\mathcal{F}$, Γ', *Some C*)
using *invars-s7'* **unfolding** ‹*s7'* = (U_{er}', $\mathcal{F}$, Γ', *Some C*)› **.**
next
show ∀ *A C*. (*restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *Some C*) = (*map-of* Γ' (*Pos A*) = *Some* (*Some C*))
proof (*intro allI*)
fix *A* :: *'f gterm* **and** *C* :: *'f gclause*
show *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *Some C* ⟷ *map-of* Γ' (*Pos A*) = *Some* (*Some C*)
proof (*cases A* ∈ *dom* $\mathcal{M}$ ∧ *A* $\prec_t$ *atm-of K*)
case *True*
have *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *Some C* ⟷ $\mathcal{M}$ *A* = *Some C*
using *True* **by** *simp*

also have ... ⟷ *map-of* Γ (*Pos A*) = *Some* (*Some C*)
using *match-hyps*(*3*−) **by** *metis*

also have ... ⟷ *map-of* Γ' (*Pos A*) = *Some* (*Some C*)
proof −
have *Pos A* ∈ *fst* ‘ *set* Γ
using *True*
by (*metis domIff map-of-eq-None-iff match-hyps*(*5*) *not-None-eq*)

hence ∃$\mathcal{C}$. (*Pos A*, $\mathcal{C}$) ∈ *set* Γ
by *fastforce*

hence ∃$\mathcal{C}$. (*Pos A*, $\mathcal{C}$) ∈ *set* Γ ∧ (*Pos A*, $\mathcal{C}$) ∈ *set* Γ'
using *True* **unfolding** *mem-set-*Γ'*-iff prod.sel literal.sel* **by** *metis*

moreover have *distinct* (*map fst* Γ')
using Γ*-distinct-atoms*
proof (*rule distinct-suffix*)
show *suffix* (*map fst* Γ') (*map fst* Γ)
using *map-mono-suffix step-hyps*(*9*) *suffix-dropWhile* **by** *blast*
qed

```
        ultimately have map-of Γ (Pos A) = map-of Γ' (Pos A)
          using Γ-distinct-atoms by (auto dest: map-of-is-SomeI)

        thus ?thesis
          by argo
      qed

      finally show ?thesis .
    next
      case False
      have restrict-map M {A. A ≺t atm-of K} A = None
        using False unfolding restrict-map-def by auto

      moreover have map-of Γ' (Pos A) ≠ Some (Some C)
        using False unfolding de-Morgan-conj
      proof (elim disjE)
        assume A ∉ dom M

        hence ⋀C. (Pos A, C) ∉ set Γ
          using match-hyps(5)
           by (metis (no-types, opaque-lifting) domIff fst-eqD invars-7 is-pos-def
map-of-SomeD
              not-None-eq snd-conv weak-map-of-SomeI)

        hence ⋀C. (Pos A, C) ∉ set Γ'
          unfolding mem-set-Γ'-iff by simp

        then show map-of Γ' (Pos A) ≠ Some (Some C)
          by (meson map-of-SomeD)
      next
        assume ¬ A ≺t atm-of K

        hence ⋀C. (Pos A, C) ∉ set Γ'
          unfolding mem-set-Γ'-iff by simp

        then show map-of Γ' (Pos A) ≠ Some (Some C)
          by (meson map-of-SomeD)
      qed

      ultimately show ?thesis
        by simp
    qed
  qed
next
  show ∀ A. (restrict-map M {A. A ≺t atm-of K} A = None) =
    (map-of Γ' (Neg A) ≠ None ∨ A |∉| trail-atms Γ')
  proof (intro allI)
    fix A :: 'f gterm
```

show (*restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *None*) =
(*map-of* Γ' (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ')
proof (*cases A* $\prec_t$ *atm-of K*)
case *True*

have *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *None* ⟷ $\mathcal{M}$ *A* = *None*
using *True* **by** *simp*

also have ... ⟷ *map-of* Γ (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ
using *match-hyps*(*6*) **by** *metis*

also have ... ⟷ *map-of* Γ' (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ
using *True mem-set-*Γ'*-iff*
by (*metis eq-fst-iff literal.sel*(*2*) *map-of-SomeD not-None-eq weak-map-of-SomeI*)

also have ... ⟷ *map-of* Γ' (*Neg A*) ≠ *None* ∨ *A* |∉| *trail-atms* Γ'
using *True mem-set-*Γ'*-iff*
by (*smt* (*verit*, *best*) *fset-trail-atms image-iff*)

finally show *?thesis* **.**
next
case *False*

have *restrict-map* $\mathcal{M}$ {*A*. *A* $\prec_t$ *atm-of K*} *A* = *None*
using *False* **by** *simp*

moreover have *A* |∉| *trail-atms* Γ'
using *False mem-set-*Γ'*-iff*
by (*smt* (*verit*, *ccfv-threshold*) *fset-trail-atms image-iff*)

ultimately show *?thesis*
by *metis*
qed
qed
next
show i' = *atms-of-clss* (*N* |∪| U_{er}') |−| *trail-atms* Γ'
using i'*-def* **.**
qed

ultimately show *?thesis*
by *metis*
qed
qed

theorem *bisimulation-ord-res-6-ord-res-7*:
defines *match* ≡ *ord-res-6-matches-ord-res-7*
shows ∃(*MATCH* :: *nat* × *nat* ⇒ *'f ord-res-6-state* ⇒ *'f ord-res-7-state* ⇒ *bool*)
$\mathcal{R}_f$ $\mathcal{R}_b$.
bisimulation ord-res-6-step ord-res-6-final (*constant-context ord-res-7*) *ord-res-7-final*

```
    MATCH R_f R_b
proof (rule ex-bisimulation-from-backward-simulation)
  show right-unique ord-res-6-step
    using right-unique-ord-res-6-step .
next
  show right-unique (constant-context ord-res-7)
    using right-unique-constant-context right-unique-ord-res-7 by metis
next
  show ∀ S. ord-res-6-final S ⟶ (∄ S'. ord-res-6-step S S')
    by (metis finished-def ord-res-6-semantics.final-finished)
next
  show ∀ S. ord-res-7-final S ⟶ (∄ S'. constant-context ord-res-7 S S')
    by (metis finished-def ord-res-7-semantics.final-finished)
next
  show ∀ i S6 S7. match i S6 S7 ⟶ ord-res-6-final S6 ⟷ ord-res-7-final S7
    unfolding match-def
    using ord-res-6-final-iff-ord-res-7-final by metis
next
  show ∀ i S6 S7. match i S6 S7 ⟶
       safe-state ord-res-6-step ord-res-6-final S6 ∧
       safe-state (constant-context ord-res-7) ord-res-7-final S7
  proof (intro allI impI conjI)
    fix i S6 S7
    assume match i S6 S7
    show safe-state ord-res-6-step ord-res-6-final S6
      using ‹match i S6 S7›[unfolded match-def]
      using ord-res-6-safe-state-if-invars
      using ord-res-6-matches-ord-res-7.simps by auto

    show safe-state (constant-context ord-res-7) ord-res-7-final S7
      using ‹match i S6 S7›[unfolded match-def]
      using ord-res-7-safe-state-if-invars
      using ord-res-6-matches-ord-res-7.simps by auto
  qed
next
  show wfp (|⊂|)
    using wfP-pfsubset .
next
  show ∀ i S6 S7 S7'. match i S6 S7 ⟶ constant-context ord-res-7 S7 S7' ⟶
    (∃ i' S6'. ord-res-6-step⁺⁺ S6 S6' ∧ match i' S6' S7') ∨
    (∃ i'. match i' S6 S7' ∧ i' |⊂| i)
    unfolding match-def
    using backward-simulation-between-6-and-7 by metis
qed

end
```

34 ORD-RES-8 (atom-guided literal trail construction)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-8-can-decide-neg* **where**
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
linorder-trm.is-least-in-fset {|*A* |∈| *atms-of-clss* (*N* |∪| U_{er}).
A ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ|} *A* ⟹
ord-res-8-can-decide-neg N U_{er} $\mathcal{F}$ Γ *C*

inductive *ord-res-8-can-skip-undefined-neg* **where**
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
¬(∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ) ⟹
¬ *trail-defined-lit* Γ *L* ⟹
is-neg L ⟹
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *C*

inductive *ord-res-8-can-skip-undefined-pos-ultimate* **where**
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
¬(∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ) ⟹
¬ *trail-defined-lit* Γ *L* ⟹
is-pos L ⟹
¬ *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *L*#} ⟹
¬(∃ *D* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}). *C* ≺$_c$ *D*) ⟹
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *C*

inductive *ord-res-8-can-produce* **where**
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ) ⟹
¬ *trail-defined-lit* Γ *L* ⟹
is-pos L ⟹
trail-false-cls Γ {#*K* ∈# *C*. *K* ≠ *L*#} ⟹
linorder-lit.is-greatest-in-mset C L ⟹
ord-res-8-can-produce N U_{er} $\mathcal{F}$ Γ *C*

inductive *ord-res-8-can-factorize* **where**
¬ *trail-false-cls* Γ *C* ⟹
linorder-lit.is-maximal-in-mset C L ⟹
¬ (∃ *A* |∈| *atms-of-clss* (*N* |∪| U_{er}). *A* ≺$_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ) ⟹
¬ *trail-defined-lit* Γ *L* ⟹
is-pos L ⟹
trail-false-cls Γ {#*K* ∈# *C*. *K* ≠ *L*#} ⟹
¬ *linorder-lit.is-greatest-in-mset C L* ⟹
ord-res-8-can-factorize N U_{er} $\mathcal{F}$ Γ *C*

definition *is-least-nonskipped-clause* **where**
is-least-nonskipped-clause $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \longleftrightarrow$
linorder-cls.is-least-in-fset $\{|C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).$
trail-false-cls $\Gamma\ C \vee$
ord-res-8-can-decide-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-pos-ultimate $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-produce $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-factorize $N\ U_{er}\ \mathcal{F}\ \Gamma\ C|\}\ C$

lemma *is-least-nonskipped-clause-mempty*:
assumes *bot-in*: $\{\#\}\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
shows *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\}$
unfolding *is-least-nonskipped-clause-def linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show $\{\#\}\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
using *bot-in* **.**
next
show *trail-false-cls* $\Gamma\ \{\#\} \vee$
ord-res-8-can-decide-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\} \vee$
ord-res-8-can-skip-undefined-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\} \vee$
ord-res-8-can-skip-undefined-pos-ultimate $N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\} \vee$
ord-res-8-can-produce $N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\} \vee$ *ord-res-8-can-factorize* $N\ U_{er}\ \mathcal{F}\ \Gamma$
$\{\#\}$
by *simp*
next
fix C :: *'f gclause*
assume $C \neq \{\#\}$
thus $\{\#\} \prec_c C$
using *mempty-lesseq-cls* **by** *blast*
qed

lemma *nex-is-least-nonskipped-clause-if*:
assumes
no-undef-atom: $\neg\ (\exists\, A|\in|atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A\ |\notin|\ trail\text{-}atms\ \Gamma)$ **and**
no-false-clause: $\neg\ fBex\ (iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}))\ (trail\text{-}false\text{-}cls\ \Gamma)$
shows $\nexists\, C.\ is\text{-}least\text{-}nonskipped\text{-}clause\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
unfolding *not-ex*
proof (*intro allI notI*)
fix C :: *'f gclause*
assume *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

hence *C-in*: $C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and**
ord-res-8-can-decide-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-pos-ultimate $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-produce $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-factorize $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

unfolding *atomize-conj*
unfolding *is-least-nonskipped-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
using *no-false-clause* **by** *metis*

hence $\exists L.$ *linorder-lit.is-maximal-in-mset* $C\ L \wedge \neg$ *trail-defined-lit* $\Gamma\ L$
proof (*elim disjE*)
assume *ord-res-8-can-decide-neg* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *linorder-trm.is-least-in-ffilter-iff*
ord-res-8-can-decide-neg.cases)
qed (*auto elim*:
ord-res-8-can-skip-undefined-neg.cases
ord-res-8-can-skip-undefined-pos-ultimate.cases
ord-res-8-can-produce.cases
ord-res-8-can-factorize.cases)

hence $\exists L.$ *atm-of* $L \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er}) \wedge$ *atm-of* $L \mid\notin\mid$ *trail-atms* Γ
using *C-in*
unfolding *linorder-lit.is-maximal-in-mset-iff trail-defined-lit-iff-trail-defined-atm*
by (*metis atm-of-in-atms-of-clssI*)

thus *False*
using *no-undef-atom* **by** *metis*
qed

lemma *MAGIC5*:
assumes *invars*: *ord-res-7-invars* $N\ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ **and**
no-more-steps: $\nexists \mathcal{C}'.$ *ord-res-7* $N\ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})\ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}')$
shows $(\forall C.\ \mathcal{C} = \mathit{Some}\ C \longleftrightarrow$ *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C)$
proof (*cases* $\mathcal{C}$)
case *None*

have *trail-atms* $\Gamma =$ *atms-of-clss* $(N \mid\cup\mid U_{er})$
using *None invars*[*unfolded ord-res-7-invars-def*] **by** *simp*

have $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).$ *trail-true-cls* $\Gamma\ C$
using *None invars*[*unfolded ord-res-7-invars-def*] **by** *simp*

hence *no-false*: $\forall C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er}).\ \neg$ *trail-false-cls* $\Gamma\ C$
using *invars*[*unfolded ord-res-7-invars-def*]
by (*meson invars not-trail-true-cls-and-trail-false-cls*
ord-res-7-invars-implies-trail-consistent)

have $\nexists C.$ *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
proof (*rule notI*, *elim exE*)
fix C
assume *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

hence
C-in: $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-spec-disj:
ord-res-8-can-decide-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-neg $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-skip-undefined-pos-ultimate $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-produce $N\ U_{er}\ \mathcal{F}\ \Gamma\ C \vee$
ord-res-8-can-factorize $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
unfolding *atomize-conj*
unfolding *is-least-nonskipped-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
using *no-false* **by** *metis*

thus *False*
proof (*elim disjE*)
assume *ord-res-8-can-decide-neg* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
thus *?thesis*
using ‹*trail-atms* Γ = *atms-of-clss* $(N \mid\cup\mid U_{er})$›
using *ord-res-8-can-decide-neg.cases*
by (*metis* (*no-types*, *lifting*) *linorder-trm.is-least-in-ffilter-iff*)
next
assume *ord-res-8-can-skip-undefined-neg* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
thus *?thesis*
using ‹*trail-atms* Γ = *atms-of-clss* $(N \mid\cup\mid U_{er})$›
using *ord-res-8-can-skip-undefined-neg.cases*
by (*metis C-in atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff trail-defined-lit-iff-trail-defined-atm*)
next
assume *ord-res-8-can-skip-undefined-pos-ultimate* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
thus *?thesis*
using ‹*trail-atms* Γ = *atms-of-clss* $(N \mid\cup\mid U_{er})$›
using *ord-res-8-can-skip-undefined-pos-ultimate.cases*
by (*metis C-in atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff trail-defined-lit-iff-trail-defined-atm*)
next
assume *ord-res-8-can-produce* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
thus *False*
using ‹*trail-atms* Γ = *atms-of-clss* $(N \mid\cup\mid U_{er})$›
using *ord-res-8-can-produce.cases*
by (*metis C-in atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff trail-defined-lit-iff-trail-defined-atm*)
next
assume *ord-res-8-can-factorize* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
thus *False*
using ‹*trail-atms* Γ = *atms-of-clss* $(N \mid\cup\mid U_{er})$›
using *ord-res-8-can-factorize.cases*
by (*metis C-in atm-of-in-atms-of-clssI linorder-lit.is-maximal-in-mset-iff trail-defined-lit-iff-trail-defined-atm*)
qed

```
  qed

  thus ?thesis
    using None by simp
next
  case (Some D)

  have D-in: D |∈| iefac \mathcal{F} |`| (N |∪| U_er)
    using Some invars[unfolded ord-res-7-invars-def] by simp

  have is-least-nonskipped-clause N U_er \mathcal{F} Γ D
  proof (cases D = {#})
    case True
    thus ?thesis
      using D-in is-least-nonskipped-clause-mempty by metis
  next
    case False

    then obtain L_D where D-max-lit: linorder-lit.is-maximal-in-mset D L_D
      using linorder-lit.ex-maximal-in-mset by presburger

    show ?thesis
      unfolding is-least-nonskipped-clause-def
      unfolding linorder-cls.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      show D |∈| iefac \mathcal{F} |`| (N |∪| U_er)
        using D-in .
    next
      show trail-false-cls Γ D ∨
      ord-res-8-can-decide-neg N U_er \mathcal{F} Γ D ∨
      ord-res-8-can-skip-undefined-neg N U_er \mathcal{F} Γ D ∨
      ord-res-8-can-skip-undefined-pos-ultimate N U_er \mathcal{F} Γ D ∨
      ord-res-8-can-produce N U_er \mathcal{F} Γ D ∨ ord-res-8-can-factorize N U_er \mathcal{F} Γ D
      proof (cases trail-false-cls Γ D)
        case True
        then show ?thesis
          by metis
      next
        case D-not-false: False

        then obtain L where D-max-lit: linorder-lit.is-maximal-in-mset D L
          by (metis linorder-lit.ex-maximal-in-mset trail-false-cls-mempty)

        show ?thesis
      proof (cases ∃ A|∈|atms-of-clss (N |∪| U_er). A ≺_t atm-of L ∧ A |∉| trail-atms
Γ)
          case True

          hence ord-res-8-can-decide-neg N U_er \mathcal{F} Γ D
```

```
    using ord-res-8-can-decide-neg.intros[OF D-not-false D-max-lit]
  by (metis (no-types, lifting) equalsffemptyD ffmember-filter linorder-trm.ex-least-in-fset)

 thus ?thesis
  by argo
next
 case no-undef-atm: False
 show ?thesis
 proof (cases trail-defined-lit Γ L)
  case L-defined: True

  hence ∃C'. ord-res-7 N (U_er, F, Γ, C) (U_er, F, Γ, C')
   unfolding ‹C = Some D›
   using ord-res-7.skip-defined[OF D-not-false D-max-lit no-undef-atm]
   by metis

  hence False
   using no-more-steps by contradiction

  thus ?thesis ..
 next
  case L-undef: False
  show ?thesis
  proof (cases L)
   case (Pos A)

   hence L-pos: is-pos L
    by simp

   show ?thesis
   proof (cases trail-false-cls Γ {#K ∈# D. K ≠ L#})
    case D-almost-false: True
    thus ?thesis
     using ord-res-8-can-factorize.intros[
       OF D-not-false D-max-lit no-undef-atm L-undef L-pos]
     using ord-res-8-can-produce.intros[
       OF D-not-false D-max-lit no-undef-atm L-undef L-pos]
     by metis
   next
    case D-not-flagrantly-false: False
    show ?thesis
    proof (cases ∃ E |∈| iefac F |`| (N |∪| U_er). D ≺_c E)
     case True

     hence ∃C'. ord-res-7 N (U_er, F, Γ, C) (U_er, F, Γ, C')
      unfolding ‹C = Some D›
      using ord-res-7.skip-undefined-pos[
          OF D-not-false D-max-lit no-undef-atm L-undef L-pos
D-not-flagrantly-false]
```

by (*metis femptyE ffmember-filter linorder-cls.ex-least-in-fset*)

hence *False*
using *no-more-steps* **by** *contradiction*

thus *?thesis* **..**
next
case *False*

hence *ord-res-8-can-skip-undefined-pos-ultimate N* U_{er} $\mathcal{F}$ Γ *D*
using *ord-res-8-can-skip-undefined-pos-ultimate.intros*[
OF D-not-false D-max-lit no-undef-atm L-undef L-pos D-not-flagrantly-false]
by *metis*

thus *?thesis*
by *argo*
qed
qed
next
case (*Neg A*)
hence *L-neg*: *is-neg L*
by *simp*

hence *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *D*
unfolding ‹$\mathcal{C}$ = *Some D*›
using *ord-res-8-can-skip-undefined-neg.intros*[
OF D-not-false D-max-lit no-undef-atm L-undef]
by *metis*

thus *?thesis*
by *argo*
qed
qed
qed
qed

fix *E* :: *'f gterm literal multiset*
assume
E-in: *E* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}) **and**
E-neq: *E* ≠ *D* **and**
E-spec: *trail-false-cls* Γ *E* ∨
ord-res-8-can-decide-neg N U_{er} $\mathcal{F}$ Γ *E* ∨
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *E* ∨
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *E* ∨
ord-res-8-can-produce N U_{er} $\mathcal{F}$ Γ *E* ∨
ord-res-8-can-factorize N U_{er} $\mathcal{F}$ Γ *E*

have *true-cls-if-lt-D*:

$\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow trail\text{-}true\text{-}cls\ \Gamma\ C$
using *invars*[*unfolded ord-res-7-invars-def*] *Some* **by** *simp*

have Γ-*lower-set*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (*N* |∪| U_{er}))
using *invars*[*unfolded ord-res-7-invars-def*] **by** *simp*

have *FOO*:
$\forall\, C|\in|iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow$
$(\forall\, K.\ ord\text{-}res.is\text{-}maximal\text{-}lit\ K\ C \longrightarrow$
$\neg\ (\exists\, A\ |\in|\ atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ K \wedge A\ |\notin|\ trail\text{-}atms\ \Gamma))$
using *invars*[*unfolded ord-res-7-invars-def*] *Some E-in* **by** *simp*

hence *BAR*:
$\forall\, C\ |\in|\ iefac\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c D \longrightarrow$
$(\forall\, K.\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\ C\ K \longrightarrow$
$\neg\ trail\text{-}defined\text{-}lit\ \Gamma\ K \longrightarrow (trail\text{-}true\text{-}cls\ \Gamma\ \{\#x \in\#\ C.\ x \neq K\#\} \wedge is\text{-}pos\ K))$
using *invars*[*unfolded ord-res-7-invars-def*] *Some* **by** *simp*

show $D \prec_c E$
using *E-spec*
proof (*elim disjE*)
assume *trail-false-cls* Γ *E*
hence ¬ *trail-true-cls* Γ *E*
using *invars not-trail-true-cls-and-trail-false-cls*
ord-res-7-invars-implies-trail-consistent **by** *blast*
thus $D \prec_c E$
using *E-in E-neq true-cls-if-lt-D* **by** *force*
next
assume *ord-res-8-can-decide-neg N* U_{er} $\mathcal{F}$ Γ *E*
thus $D \prec_c E$
proof (*cases N* U_{er} $\mathcal{F}$ Γ *E rule*: *ord-res-8-can-decide-neg.cases*)
case (*1* L_E *A*)
hence $\exists\, A|\in|atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ A \prec_t atm\text{-}of\ L_E \wedge A\ |\notin|\ trail\text{-}atms\ \Gamma$
unfolding *linorder-trm.is-least-in-ffilter-iff* **by** *metis*
thus *?thesis*
using *FOO*[*rule-format*, *OF E-in* - ‹*ord-res.is-maximal-lit* L_E *E*›] *E-in E-neq*
by *force*
qed
next
assume *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *E*
thus $D \prec_c E$
proof (*cases N* U_{er} $\mathcal{F}$ Γ *E rule*: *ord-res-8-can-skip-undefined-neg.cases*)
case *hyps*: (*1* L_E)
thus *?thesis*
using *BAR*[*rule-format*, *OF E-in* - ‹*ord-res.is-maximal-lit* L_E *E*›]

```
          using invars[unfolded ord-res-7-invars-def Some, rule-format, OF refl]
Some E-in
          using E-neq by fastforce
        qed
      next
        assume ord-res-8-can-skip-undefined-pos-ultimate N U_er F Γ E
        thus D ≺_c E
      proof (cases N U_er F Γ E rule: ord-res-8-can-skip-undefined-pos-ultimate.cases)
          case (1 L)
          then show ?thesis using E-neq D-in by force
        qed
      next
        assume ord-res-8-can-produce N U_er F Γ E
        hence ¬ trail-true-cls Γ E
        proof (cases N U_er F Γ E rule: ord-res-8-can-produce.cases)
          case (1 L)
          then show ?thesis
            using invars[THEN ord-res-7-invars-implies-trail-consistent]
         by (smt (verit, ccfv-SIG) mem-Collect-eq not-trail-true-cls-and-trail-false-cls
               set-mset-filter trail-defined-lit-iff-true-or-false trail-true-cls-def)
        qed
        thus D ≺_c E
          using E-in E-neq true-cls-if-lt-D by force
      next
        assume ord-res-8-can-factorize N U_er F Γ E
        hence ¬ trail-true-cls Γ E
        proof (cases N U_er F Γ E rule: ord-res-8-can-factorize.cases)
          case (1 L)
          then show ?thesis
            using invars[THEN ord-res-7-invars-implies-trail-consistent]
         by (smt (verit, ccfv-SIG) mem-Collect-eq not-trail-true-cls-and-trail-false-cls
               set-mset-filter trail-defined-lit-iff-true-or-false trail-true-cls-def)
        qed
        thus D ≺_c E
          using E-in E-neq true-cls-if-lt-D by force
      qed
    qed
  qed

  moreover have Uniq (is-least-nonskipped-clause N U_er F Γ)
    unfolding is-least-nonskipped-clause-def
    using linorder-cls.Uniq-is-least-in-fset
    by simp

  ultimately show ?thesis
    using Some
    by (metis (no-types) The-optional-eq-SomeD The-optional-eq-SomeI)
qed
```

lemma *MAGIC6*:
assumes *invars*: *ord-res-7-invars N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$)
shows $\exists \mathcal{C}'$. (*ord-res-7 N*)** (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}'$) $\wedge$
($\nexists \mathcal{C}''$. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}'$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}''$))
proof −
define R **where**
$R = (\lambda \mathcal{C}\ \mathcal{C}'.$ *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}'$))

define f :: *'f gclause option* $\Rightarrow$ *'f gclause fset* **where**
$f = (\lambda \mathcal{C}.$ *case* $\mathcal{C}$ *of None* $\Rightarrow$ {|||} | *Some* $C \Rightarrow$ {|D |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er}). $C \preceq_c D$|})

let *?less-f* = (|⊂|)

have *Wellfounded.wfp-on* {x'. R^{**} $\mathcal{C}$ x'} R^{-1-1}
proof (*rule wfp-on-if-convertible-to-wfp-on*)
have *wfp* (|⊂|)
by *auto*
thus *Wellfounded.wfp-on* (f ‘ {x'. R^{**} $\mathcal{C}$ x'}) *?less-f*
using *Wellfounded.wfp-on-subset subset-UNIV* **by** *metis*
next
fix $\mathcal{C}_x$ $\mathcal{C}_y$ :: *'f gclause option*

have *rtranclp-with-constsD*: $(\lambda y\ y'.\ R\ (x,\ y)\ (x,\ y'))^{**}\ y\ y' \Longrightarrow$
R^{**} (x, y) (x, y') **for** R x y y'
proof (*induction y arbitrary*: *rule*: *converse-rtranclp-induct*)
case *base*
show *?case*
by *simp*
next
case (*step w*)
thus *?case*
by *force*
qed

assume $\mathcal{C}_x \in$ {x'. R^{**} $\mathcal{C}$ x'} **and** $\mathcal{C}_y \in$ {x'. R^{**} $\mathcal{C}$ x'}
hence
(*ord-res-7 N*)** (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_x$) **and**
(*ord-res-7 N*)** (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_y$)
unfolding *atomize-conj mem-Collect-eq R-def*
by (*auto intro*: *rtranclp-with-constsD*)
hence
x-invars: *ord-res-7-invars N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_x$) **and**
y-invars: *ord-res-7-invars N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_y$)
using *ord-res-7-preserves-invars*
using *invars* **by** (*metis rtranclp-ord-res-7-preserves-ord-res-7-invars*)+

have Γ-*consistent*: *trail-consistent* Γ
using *x-invars* **by** (*metis ord-res-7-invars-implies-trail-consistent*)

have *less-f-if*: *?less-f* (*f* $\mathcal{C}_y$) (*f* $\mathcal{C}_x$)
  if $\mathcal{C}_x$ = *Some C* **and**
    $\mathcal{C}_y$*-disj*: $\mathcal{C}_y$ = *None* $\vee$ $\mathcal{C}_y$ = *Some D* $\wedge$ *C* $\prec_c$ *D* **and**
    C-in: *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
  for *C D*
proof −
  have *f-x*: *f* $\mathcal{C}_x$ = {$|$*D* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}). *C* $\preceq_c$ *D*$|$}
    by (*auto simp add*: ‹$\mathcal{C}_x$ = *Some C*› *f-def*)

  moreover have *C* $|\in|$ *f* $\mathcal{C}_x$
    using *C-in f-x* **by** *simp*

  moreover have *C* $|\notin|$ *f* $\mathcal{C}_y$ $\wedge$ *f* $\mathcal{C}_y$ $|\subseteq|$ *f* $\mathcal{C}_x$
    using $\mathcal{C}_y$*-disj*
  proof (*elim disjE conjE*; *intro conjI*)
    assume $\mathcal{C}_y$ = *None*
    thus *C* $|\notin|$ *f* $\mathcal{C}_y$ **and** *f* $\mathcal{C}_y$ $|\subseteq|$ *f* $\mathcal{C}_x$
      unfolding *f-x*
      by (*simp-all add*: *f-def*)
  next
    assume $\mathcal{C}_y$ = *Some D* **and** *C* $\prec_c$ *D*

    have $\bigwedge x$. *D* $\preceq_c$ *x* $\Longrightarrow$ *C* $\preceq_c$ *x*
      using ‹*C* $\prec_c$ *D*› **by** *auto*

    moreover have *fst-f-y*: *f* $\mathcal{C}_y$ = {$|$*E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}). *D* $\preceq_c$ *E*$|$}
      by (*auto simp add*: ‹$\mathcal{C}_y$ = *Some D*› *f-def*)

    ultimately show *f* $\mathcal{C}_y$ $|\subseteq|$ *f* $\mathcal{C}_x$
      using *f-x* **by** *auto*

    show *C* $|\notin|$ *f* $\mathcal{C}_y$
      using ‹*C* $\prec_c$ *D*› *fst-f-y* **by** *auto*
  qed

  ultimately have *f* $\mathcal{C}_y$ $|\subset|$ *f* $\mathcal{C}_x$
    by *blast*

  thus *?thesis*
    by (*simp add*: *lex-prodp-def*)
qed

have *eres-not-in-if*: *eres D E* $|\notin|$ U_{er}
  if $\mathcal{C}_x$ = *Some E* **and** *E-false*: *trail-false-cls* Γ *E* **and**
    E-max-lit: *ord-res.is-maximal-lit L E* **and** *L-neg*: *is-neg L*
    map-of Γ (− *L*) = *Some* (*Some D*)
  for *D E L*
proof −

have
clauses-lt-E-true:
$\forall\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ C \prec_c E \longrightarrow \mathit{trail\text{-}true\text{-}cls}\ \Gamma\ C$ **and**
Γ-*prop-greatest*:
$\forall\, Ln \in \mathit{set}\ \Gamma.\ \forall\, C.\ \mathit{snd}\ Ln = \mathit{Some}\ C \longrightarrow \mathit{linorder\text{-}lit.is\text{-}greatest\text{-}in\text{-}mset}\ C\ (\mathit{fst}\ Ln)$
using *x-invars* **unfolding** $\langle \mathcal{C}_x = \mathit{Some}\ E\rangle$ *ord-res-7-invars-def* **by** *simp-all*

have $(-\ L,\ \mathit{Some}\ D) \in \mathit{set}\ \Gamma$
using $\langle \mathit{map\text{-}of}\ \Gamma\ (-\ L) = \mathit{Some}\ (\mathit{Some}\ D)\rangle$ **by** (*metis map-of-SomeD*)

hence *D-greatest-lit*: *linorder-lit.is-greatest-in-mset D* $(-\ L)$
using Γ-*prop-greatest* **by** *fastforce*

have *eres D E* $\prec_c$ *E*
using *eres-lt-if*
using *E-max-lit L-neg D-greatest-lit*
by *metis*

hence *eres D E* $\neq$ *E*
by *order*

have $L \in\#$ *E*
using *E-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

hence $-\ L \notin\#$ *E*
using *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*
using Γ-*consistent E-false* **by** *metis*

hence $\forall\, K \in\#$ *eres D E. atm-of K* $\prec_t$ *atm-of L*
using *lit-in-eres-lt-greatest-lit-in-grestest-resolvant*[*OF* $\langle$*eres D E* $\neq$ *E*$\rangle$ *E-max-lit*]
by *metis*

hence $\forall\, K \in\#$ *eres D E.* $K \neq L \wedge K \neq -\ L$
by *fastforce*

moreover have $\forall\, L \in\#$ *eres D E.* $L \in\#$ *D* $\vee$ $L \in\#$ *E*
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have *D-almost-false*: *trail-false-cls* Γ $\{\#K \in\#$ *D.* $K \neq -\ L\#\}$
using *ord-res-7-invars-implies-propagated-clause-almost-false*
using $\langle(-\ L,\ \mathit{Some}\ D) \in \mathit{set}\ \Gamma\rangle$ *x-invars*
by *metis*

ultimately have *trail-false-cls* Γ (*eres D E*)
using *E-false* **unfolding** *trail-false-cls-def* **by** *fastforce*

have *eres D E* $|\notin|$ *N* $|\cup|$ U_{er}

using *eres-not-in-known-clauses-if-trail-false-cls*
using Γ*-consistent clauses-lt-E-true* ‹*eres D E* $\prec_c$ *E*› ‹*trail-false-cls* Γ (*eres D E*)›
by *metis*

thus *eres D E* $|\notin|$ U_{er}
by *blast*
qed

assume R^{-1-1} $\mathcal{C}_y$ $\mathcal{C}_x$
hence *ord-res-*$\mathcal{7}$ *N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_x$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_y$)
unfolding *conversep-iff R-def* **.**
thus *?less-f* (*f* $\mathcal{C}_y$) (*f* $\mathcal{C}_x$)
proof (*cases N* (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_x$) (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}_y$) *rule*: *ord-res-*$\mathcal{7}$*.cases*)
case *step-hyps*: (*decide-neg C L A*)
hence *False* **by** *simp*
thus *?thesis* **..**
next
case *step-hyps*: (*skip-defined C L*)

have *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *ord-res-*$\mathcal{7}$*-invars-def step-hyps*(*1*) *x-invars* **by** *presburger*

moreover have ∃ *D*. $\mathcal{C}_y$ = *None* ∨ $\mathcal{C}_y$ = *Some D* ∧ *C* $\prec_c$ *D*
proof (*cases* $\mathcal{C}_y$)
case *None*
thus *?thesis*
by *simp*
next
case (*Some D*)
thus *?thesis*
using *step-hyps*
by (*metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD*)
qed

ultimately show *?thesis*
using *less-f-if step-hyps* **by** *metis*
next
case *step-hyps*: (*skip-undefined-neg C L*)
hence *False* **by** *simp*
thus *?thesis* **..**
next
case *step-hyps*: (*skip-undefined-pos C L D*)

have *C* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *ord-res-*$\mathcal{7}$*-invars-def step-hyps*(*1*) *x-invars* **by** *presburger*

moreover have ∃ *D*. $\mathcal{C}_y$ = *None* ∨ $\mathcal{C}_y$ = *Some D* ∧ *C* $\prec_c$ *D*
using *step-hyps* **by** (*metis linorder-cls.is-least-in-ffilter-iff*)

ultimately show *?thesis*
using *less-f-if step-hyps* **by** *metis*
next
case *step-hyps*: (*skip-undefined-pos-ultimate C L*)
hence *False* **by** *simp*
thus *?thesis* **..**
next
case *step-hyps*: (*production C L*)
hence *False* **by** *simp*
thus *?thesis* **..**
next
case *step-hyps*: (*factoring C L*)

have C |∈| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er})
using *ord-res-7-invars-def step-hyps*(*1*) *x-invars* **by** *presburger*

moreover have *efac* $C \neq C$
using *step-hyps* **by** (*metis greatest-literal-in-efacI*)

ultimately have C |∉| $\mathcal{F}$
by (*smt* (*verit*, *ccfv-threshold*) *fimage-iff iefac-def ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)

hence *False*
using ‹$\mathcal{F}$ = *finsert C* $\mathcal{F}$› **by** *blast*

thus *?thesis* **..**
next
case *step-hyps*: (*resolution-bot E L D*)
hence *eres D E* |∉| U_{er}
using *eres-not-in-if* **by** *metis*
hence *False*
using ‹U_{er} = *finsert* (*eres D E*) U_{er}› **by** *blast*
thus *?thesis* **..**
next
case (*resolution-pos E L D K*)
hence *eres D E* |∉| U_{er}
using *eres-not-in-if* **by** *metis*
hence *False*
using ‹U_{er} = *finsert* (*eres D E*) U_{er}› **by** *blast*
thus *?thesis* **..**
next
case (*resolution-neg E L D K C*)
hence *eres D E* |∉| U_{er}
using *eres-not-in-if* **by** *metis*
hence *False*
using ‹U_{er} = *finsert* (*eres D E*) U_{er}› **by** *blast*
thus *?thesis* **..**

```
    qed
  qed

  then obtain 𝒞' where R** 𝒞 𝒞' and ∄z. R 𝒞' z
    using ex-terminating-rtranclp-strong by metis

  show ?thesis
  proof (intro exI conjI)
    show (ord-res-7 N)** (U_er, ℱ, Γ, 𝒞) (U_er, ℱ, Γ, 𝒞')
      using ‹R** 𝒞 𝒞'›
      by (induction 𝒞 rule: converse-rtranclp-induct) (auto simp: R-def)
  next
    show ∄𝒞''. ord-res-7 N (U_er, ℱ, Γ, 𝒞') (U_er, ℱ, Γ, 𝒞'')
      using ‹∄z. R 𝒞' z›
      by (simp add: R-def)
  qed
qed

inductive ord-res-7-matches-ord-res-8 :: 'f ord-res-7-state ⇒ 'f ord-res-8-state ⇒
bool where
  ord-res-7-invars N (U_er, ℱ, Γ, 𝒞) ⟹
    ord-res-8-invars N (U_er, ℱ, Γ) ⟹
    (∀C. 𝒞 = Some C ⟷ is-least-nonskipped-clause N U_er ℱ Γ C) ⟹
    ord-res-7-matches-ord-res-8 (N, U_er, ℱ, Γ, 𝒞) (N, U_er, ℱ, Γ)

lemma ord-res-7-final-iff-ord-res-8-final:
  fixes S7 S8
  assumes match: ord-res-7-matches-ord-res-8 S7 S8
  shows ord-res-7-final S7 ⟷ ord-res-8-final S8
  using match
proof (cases S7 S8 rule: ord-res-7-matches-ord-res-8.cases)
  case match-hyps: (1 N U_er ℱ Γ 𝒞)

  note invars7 = ‹ord-res-7-invars N (U_er, ℱ, Γ, 𝒞)›[unfolded ord-res-7-invars-def,
      rule-format, OF refl]

  have Γ-consistent: trail-consistent Γ
    using invars7 by (metis trail-consistent-if-sorted-wrt-atoms)

  show ord-res-7-final S7 ⟷ ord-res-8-final S8
  proof (rule iffI)
    assume ord-res-7-final S7
    thus ord-res-8-final S8
      unfolding ‹S7 = (N, U_er, ℱ, Γ, 𝒞)›
    proof (cases (N, U_er, ℱ, Γ, 𝒞) rule: ord-res-7-final.cases)
      case model-found
      show ord-res-8-final S8
        unfolding ‹S8 = (N, U_er, ℱ, Γ)›
      proof (rule ord-res-8-final.model-found)
```

have $\mathcal{C} = \mathit{None} \longrightarrow \mathit{trail\text{-}atms}\ \Gamma = \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er})$
using *invars7* **by** *argo*

hence $\mathit{trail\text{-}atms}\ \Gamma = \mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er})$
using *model-found* **by** *argo*

thus $\neg\ (\exists\, A|\in|\mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).\ A\ |\notin|\ \mathit{trail\text{-}atms}\ \Gamma)$
by *metis*
next
have $\forall\, C\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}true\text{-}cls}\ \Gamma\ C$
using *invars7 model-found* **by** *simp*

moreover have $\neg\ (\mathit{trail\text{-}true\text{-}cls}\ \Gamma\ C \wedge \mathit{trail\text{-}false\text{-}cls}\ \Gamma\ C)$ **for** C
using *not-trail-true-cls-and-trail-false-cls*[*OF* Γ-*consistent*] **.**

ultimately show $\neg\ (\exists\, C|\in|\mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er}).\ \mathit{trail\text{-}false\text{-}cls}\ \Gamma\ C)$
by *metis*
qed
next
case *contradiction-found*
show *ord-res-8-final S8*
unfolding ‹$S8 = (N,\ U_{er},\ \mathcal{F},\ \Gamma)$›
proof (*rule ord-res-8-final.contradiction-found*)
show $\{\#\}\ |\in|\ \mathit{iefac}\ \mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
using *invars7* ‹$\mathcal{C} = \mathit{Some}\ \{\#\}$› **by** *metis*
qed
qed
next
assume *ord-res-8-final S8*
thus *ord-res-7-final S7*
unfolding ‹$S8 = (N,\ U_{er},\ \mathcal{F},\ \Gamma)$›
proof (*cases* $(N,\ U_{er},\ \mathcal{F},\ \Gamma)$ *rule*: *ord-res-8-final.cases*)
case *model-found*

hence $\nexists\, C.\ \mathit{is\text{-}least\text{-}nonskipped\text{-}clause}\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
using *nex-is-least-nonskipped-clause-if* **by** *metis*

hence $\mathcal{C} = \mathit{None}$
using *match-hyps* **by** *simp*

thus *ord-res-7-final S7*
unfolding ‹$S7 = (N,\ U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›
using *ord-res-7-final.model-found* **by** *metis*
next
case *contradiction-found*

hence $\mathit{is\text{-}least\text{-}nonskipped\text{-}clause}\ N\ U_{er}\ \mathcal{F}\ \Gamma\ \{\#\}$
using *is-least-nonskipped-clause-mempty* **by** *metis*

hence $\mathcal{C} = \mathit{Some}\ \{\#\}$
using *match-hyps* **by** *presburger*

thus *ord-res-7-final S7*
unfolding ‹$S7 = (N,\ U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›
using *ord-res-7-final.contradiction-found* **by** *metis*
qed
qed
qed

lemma *backward-simulation-between-7-and-8*:
fixes *i S7 S8 S8'*
assumes *match*: *ord-res-7-matches-ord-res-8 S7 S8* **and** *step*: *constant-context ord-res-8 S8 S8'*
shows $\exists\, S7'.$ (*constant-context ord-res-7*)$^{++}$ *S7 S7'* $\wedge$ *ord-res-7-matches-ord-res-8 S7' S8'*
using *match*
proof (*cases S7 S8 rule*: *ord-res-7-matches-ord-res-8.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ Γ $\mathcal{C}$)

note *S7-def* = ‹$S7 = (N,\ U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})$›

note *invars7* = ‹*ord-res-7-invars N* $(U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$›[*unfolded ord-res-7-invars-def*, *rule-format*, *OF refl*]

have Γ-*sorted*: *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using *invars7* **by** *argo*

have Γ-*consistent*: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms*[*OF* Γ-*sorted*] **.**

have Γ-*lower-set*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N $|\cup|$ U_{er}))
using *invars7* **by** *argo*

have $\mathcal{C}$-*eq-None-implies-all-atoms-defined*: $\mathcal{C} = \mathit{None} \longrightarrow$ *trail-atms* Γ = *atms-of-clss* (N $|\cup|$ U_{er})
using *invars7* **by** *argo*

obtain *s8'* **where**
S8'-def: *S8'* = (N, *s8'*) **and**
step': *ord-res-8 N* (U_{er}, $\mathcal{F}$, Γ) *s8'*
using *step* **unfolding** ‹*S8* = (N, U_{er}, $\mathcal{F}$, Γ)›
by (*auto elim*: *constant-context.cases*)

have *invars-s8'*: *ord-res-8-invars N s8'*
using *ord-res-8-preserves-invars*[*OF step'* ‹*ord-res-8-invars N* (U_{er}, $\mathcal{F}$, Γ)›] **.**

show *?thesis*

```
    using step'
  proof (cases N (U_er, F, Γ) s8' rule: ord-res-8.cases)
    case step-hyps: (decide-neg A Γ')

    have
      A-in: A |∈| atms-of-clss (N |∪| U_er) and
      A-undef: A |∉| trail-atms Γ and
      A-least: ∀ y|∈|atms-of-clss (N |∪| U_er). y ≠ A ⟶ (∀ A₁|∈|trail-atms Γ. A₁
≺_t y) ⟶ A ≺_t y
      using step-hyps(3) unfolding linorder-trm.is-least-in-fset-iff by auto

    have C ≠ None
      using C-eq-None-implies-all-atoms-defined A-in A-undef by metis

    then obtain D :: 'f gclause where C = Some D
      by blast

    hence D-in: D |∈| iefac F |`| (N |∪| U_er)
      by (metis ‹C = Some D› invars7)

    have is-least-nonskipped-clause N U_er F Γ D
      using match-hyps ‹C = Some D› by metis

    moreover have D-not-false: ¬ trail-false-cls Γ D
      using D-in step-hyps by metis

    moreover have ¬ ord-res-8-can-produce N U_er F Γ D
    proof (rule notI)
      assume ord-res-8-can-produce N U_er F Γ D
      thus False
      proof (cases N U_er F Γ D rule: ord-res-8-can-produce.cases)
        case (1 L')
        thus ?thesis
          by (metis A-in A-least A-undef D-in atm-of-in-atms-of-clssI
              atoms-of-trail-lt-atom-of-propagatable-literal clause-could-propagate-def
invars7
              linorder-lit.is-maximal-in-mset-iff literal.collapse(1) step-hyps(4))
      qed
    qed

    moreover have ¬ ord-res-8-can-factorize N U_er F Γ D
    proof (rule notI)
      assume ord-res-8-can-factorize N U_er F Γ D
      thus False
      proof (cases N U_er F Γ D rule: ord-res-8-can-factorize.cases)
        case (1 L')
        thus False
          by (metis A-in A-least A-undef D-in atm-of-in-atms-of-clssI
              atoms-of-trail-lt-atom-of-propagatable-literal clause-could-propagate-def
```

invars7
linorder-lit.is-maximal-in-mset-iff literal.collapse(1) step-hyps(4))
qed
qed

ultimately have *ord-res-8-can-decide-neg N* U_{er} $\mathcal{F}$ Γ *D* $\vee$
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *D* $\vee$
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *D*
unfolding *is-least-nonskipped-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *argo*

then obtain $\mathcal{C}'$ **where** *first-step7*: *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, *Some D*) (U_{er}, $\mathcal{F}$, Γ', $\mathcal{C}'$)
proof (*elim disjE*; *atomize-elim*)
assume *ord-res-8-can-decide-neg N* U_{er} $\mathcal{F}$ Γ *D*
thus $\exists\mathcal{C}'$. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, *Some D*) (U_{er}, $\mathcal{F}$, Γ', $\mathcal{C}'$)
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-decide-neg.cases*)
case *hyps*: (*1 L A'*)
hence $A' = A$
by (*smt* (*verit*, *del-insts*) Γ*-lower-set linorder-trm.is-least-in-ffilter-iff*
linorder-trm.neq-iff linorder-trm.not-in-lower-setI linorder-trm.order.strict-trans
step-hyps(3))
thus *?thesis*
using *hyps* ‹$\Gamma' = (Neg\ A,\ None) \# \Gamma$›
using *ord-res-7.decide-neg*[*of* Γ *D - N* U_{er} *A* Γ' $\mathcal{F}$] **by** *blast*
qed
next
assume *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *D*
thus $\exists\mathcal{C}'$. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, *Some D*) (U_{er}, $\mathcal{F}$, Γ', $\mathcal{C}'$)
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-skip-undefined-neg.cases*)
case *hyps*: (*1 L*)
hence $L = Neg\ A$
by (*smt* (*verit*) *A-in A-least A-undef D-in* Γ*-lower-set atm-of-in-atms-of-clssI*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
linorder-trm.not-in-lower-setI literal.disc(2) literal.expand literal.sel(2)
trail-defined-lit-iff-trail-defined-atm)
thus *?thesis*
using *hyps* ‹$\Gamma' = (Neg\ A,\ None) \# \Gamma$›
using *ord-res-7.skip-undefined-neg* **by** *blast*
qed
next
assume *ord-res-8-can-skip-undefined-pos-ultimate N* U_{er} $\mathcal{F}$ Γ *D*
thus $\exists\mathcal{C}'$. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ, *Some D*) (U_{er}, $\mathcal{F}$, Γ', $\mathcal{C}'$)
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-skip-undefined-pos-ultimate.cases*)
case *hyps*: (*1 L*)
hence $L = Pos\ A$
by (*smt* (*verit*, *best*) *A-in A-least A-undef D-in atm-of-in-atms-of-clssI*
invars7

```
        linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
        linorder-trm.not-in-lower-setI literal.disc(1) literal.expand literal.sel(1)
        trail-defined-lit-iff-trail-defined-atm)
    thus ?thesis
     using hyps ‹Γ' = (Neg A, None) # Γ›
     using ord-res-7.skip-undefined-pos-ultimate by fastforce
   qed
  qed

  moreover have ord-res-7-invars N (U_er, F, Γ', C')
   using ‹C = Some D› first-step7 match-hyps(3) ord-res-7-preserves-invars by
blast

  ultimately obtain C'' where
   following-steps7: (ord-res-7 N)** (U_er, F, Γ', C') (U_er, F, Γ', C'') and
   no-more-step7: (∄C'''. ord-res-7 N (U_er, F, Γ', C'') (U_er, F, Γ', C'''))
   using MAGIC6 by metis

  show ?thesis
  proof (intro exI conjI)
   have (ord-res-7 N)++ (U_er, F, Γ, C) (U_er, F, Γ', C'')
    unfolding ‹C = Some D›
    using first-step7 following-steps7 by simp

   thus (constant-context ord-res-7)++ S7 (N, U_er, F, Γ', C'')
    unfolding S7-def by (simp add: tranclp-constant-context)

   show ord-res-7-matches-ord-res-8 (N, U_er, F, Γ', C'') S8'
    unfolding S8'-def ‹s8' = (U_er, F, Γ')›
   proof (intro ord-res-7-matches-ord-res-8.intros allI)
    show ord-res-7-invars N (U_er, F, Γ', C'')
     using ‹ord-res-7-invars N (U_er, F, Γ', C')› following-steps7
      rtranclp-ord-res-7-preserves-ord-res-7-invars by blast

    show ord-res-8-invars N (U_er, F, Γ')
     using invars-s8' step-hyps(1) by blast

    fix C :: 'f gclause
    show C'' = Some C ⟷ is-least-nonskipped-clause N U_er F Γ' C
     using MAGIC5 ‹ord-res-7-invars N (U_er, F, Γ', C'')› no-more-step7 by
metis
   qed
  qed
 next
  case step-hyps: (propagate A C Γ')

  have
   A-in: A |∈| atms-of-clss (N |∪| U_er) and
   A-undef: A |∉| trail-atms Γ and
```

A-least: $\forall\, y|\in|$*atms-of-clss* $(N\ |\cup|\ U_{er})$. $y \neq A \longrightarrow (\forall\, A_1|\in|$*trail-atms* Γ. $A_1 \prec_t y) \longrightarrow A \prec_t y$
using *step-hyps*(*3*) **unfolding** *linorder-trm.is-least-in-fset-iff* **by** *auto*

have
C-in: $C\ |\in|$ *iefac* $\mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$ **and**
C-can-prop:*clause-could-propagate* $\Gamma\ C$ (*Pos* A) **and**
C-least: $\forall\, D\ |\in|$ *iefac* $\mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$.
$D \neq C \longrightarrow$ *clause-could-propagate* $\Gamma\ D$ (*Pos* A) $\longrightarrow C \prec_c D$
using *step-hyps* **unfolding** *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

hence
Pos-A-undef: $\neg$ *trail-defined-lit* Γ (*Pos* A) **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* C (*Pos* A) **and**
C-almost-false: *trail-false-cls* Γ $\{\#K \in\# C.\ K \neq Pos\ A\#\}$
unfolding *atomize-conj clause-could-propagate-def* **by** *argo*

have *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
unfolding *is-least-nonskipped-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show $C\ |\in|$ *iefac* $\mathcal{F}\ |`|\ (N\ |\cup|\ U_{er})$
using *C-in* **.**
next
have *ord-res-8-can-produce* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$
proof (*rule ord-res-8-can-produce.intros*)
show $\neg$ *trail-false-cls* $\Gamma\ C$
using *step-hyps C-in* **by** *metis*
next
show *ord-res.is-maximal-lit* (*Pos* A) C
using *step-hyps* **by** *blast*
next
show $\neg\ (\exists\, Aa|\in|$*atms-of-clss* $(N\ |\cup|\ U_{er})$. $Aa \prec_t$ *atm-of* (*Pos* A) $\wedge\ Aa\ |\notin|$ *trail-atms* Γ)
unfolding *literal.sel*
using *step-hyps*
by (*smt* (*verit*, *ccfv-threshold*) Γ-*lower-set linorder-trm.dual-order.asym linorder-trm.is-least-in-ffilter-iff linorder-trm.is-lower-set-iff linorder-trm.neq-iff*)
next
show $\neg$ *trail-defined-lit* Γ (*Pos* A)
using *A-undef* **unfolding** *trail-defined-lit-iff-trail-defined-atm literal.sel* **.**
next
show *is-pos* (*Pos* A)
by *simp*
next
show *trail-false-cls* Γ $\{\#K \in\# C.\ K \neq Pos\ A\#\}$
using ‹*clause-could-propagate* $\Gamma\ C$ (*Pos* A)›

```
    unfolding clause-could-propagate-def by argo
  next
    show ord-res.is-strictly-maximal-lit (Pos A) C
      using step-hyps by argo
  qed

  thus trail-false-cls Γ C ∨
    ord-res-8-can-decide-neg N U_er ℱ Γ C ∨
    ord-res-8-can-skip-undefined-neg N U_er ℱ Γ C ∨
    ord-res-8-can-skip-undefined-pos-ultimate N U_er ℱ Γ C ∨
    ord-res-8-can-produce N U_er ℱ Γ C ∨ ord-res-8-can-factorize N U_er ℱ Γ C
    by argo
next
  fix D :: 'f gclause
  assume
    D-in: D |∈| iefac ℱ |‘| (N |∪| U_er) and
    D-neq: D ≠ C and
    D-spec-disj: trail-false-cls Γ D ∨
      ord-res-8-can-decide-neg N U_er ℱ Γ D ∨
      ord-res-8-can-skip-undefined-neg N U_er ℱ Γ D ∨
      ord-res-8-can-skip-undefined-pos-ultimate N U_er ℱ Γ D ∨
      ord-res-8-can-produce N U_er ℱ Γ D ∨
      ord-res-8-can-factorize N U_er ℱ Γ D

  hence
    ord-res-8-can-decide-neg N U_er ℱ Γ D ∨
    ord-res-8-can-skip-undefined-neg N U_er ℱ Γ D ∨
    ord-res-8-can-skip-undefined-pos-ultimate N U_er ℱ Γ D ∨
    ord-res-8-can-produce N U_er ℱ Γ D ∨
    ord-res-8-can-factorize N U_er ℱ Γ D
    using step-hyps D-in by metis

  thus C ≺_c D
  proof (elim disjE)
    assume ord-res-8-can-decide-neg N U_er ℱ Γ D
    thus C ≺_c D
    proof (cases N U_er ℱ Γ D rule: ord-res-8-can-decide-neg.cases)
      case hyps: (1 L A')
      hence A = A'
        using step-hyps
        by (smt (verit, del-insts) Γ-lower-set linorder-trm.antisym-conv3
        linorder-trm.dual-order.strict-implies-not-eq linorder-trm.dual-order.strict-trans
            linorder-trm.is-least-in-ffilter-iff linorder-trm.not-in-lower-setI)
      hence A ≺_t atm-of L
        using hyps
        unfolding linorder-trm.is-least-in-ffilter-iff
        by argo
      hence Pos A ≺_l L
        by (cases L) simp-all
```

thus *?thesis*
using *C-max-lit* ‹*linorder-lit.is-maximal-in-mset D L*›
by (*metis linorder-lit.multp-if-maximal-less-that-maximal*)
qed
next
assume *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *D*
thus *C* $\prec_c$ *D*
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-skip-undefined-neg.cases*)
case *hyps*: (*1 L*)
hence *atm-of L* = *A*
using *step-hyps*
by (*smt* (*verit*, *best*)
A-in A-least A-undef D-in Γ*-lower-set atm-of-in-atms-of-clssI*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
linorder-trm.not-in-lower-setI trail-defined-lit-iff-trail-defined-atm)
hence *Pos A* $\prec_l$ *L*
using ‹*is-neg L*› **by** (*cases L*) *simp-all*
thus *?thesis*
using *C-max-lit* ‹*linorder-lit.is-maximal-in-mset D L*›
by (*metis linorder-lit.multp-if-maximal-less-that-maximal*)
qed
next
assume *ord-res-8-can-skip-undefined-pos-ultimate N* U_{er} $\mathcal{F}$ Γ *D*
thus *C* $\prec_c$ *D*
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-skip-undefined-pos-ultimate.cases*)
case *hyps*: (*1 L*)
thus *?thesis*
by (*meson C-in D-neq linorder-cls.linorder-cases*)
qed
next
assume *ord-res-8-can-produce N* U_{er} $\mathcal{F}$ Γ *D*

then obtain *L* **where**
D-max-lit: *ord-res.is-maximal-lit L D* **and**
¬ (∃ *A*|∈|*atms-of-clss* (*N* |∪| U_{er}). *A* $\prec_t$ *atm-of L* ∧ *A* |∉| *trail-atms* Γ)
and
L-undef: ¬ *trail-defined-lit* Γ *L* **and**
D-almost-false: *trail-false-cls* Γ {#*K* ∈# *D*. *K* ≠ *L*#} **and**
is-pos L
by (*auto elim*: *ord-res-8-can-factorize.cases ord-res-8-can-produce.cases*)

hence *atm-of L* = *A*
using *step-hyps*
by (*smt* (*verit*, *ccfv-SIG*) *A-in A-least A-undef D-in* Γ*-lower-set*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
linorder-trm.not-in-lower-setI atm-of-in-atms-of-clssI
trail-defined-lit-iff-trail-defined-atm)

hence *L* = *Pos A*

```
      using ‹is-pos L› by (cases L) simp-all

    hence clause-could-propagate Γ D (Pos A)
      unfolding clause-could-propagate-def
      using D-almost-false D-max-lit L-undef by metis

    thus C ≺c D
      using D-in D-neq C-least by metis
  next
    assume ord-res-8-can-factorize N Uer F Γ D

    then obtain L where
      D-max-lit: ord-res.is-maximal-lit L D and
      ¬ (∃ A|∈|atms-of-clss (N |∪| Uer). A ≺t atm-of L ∧ A |∉| trail-atms Γ)
and
      L-undef: ¬ trail-defined-lit Γ L and
      D-almost-false: trail-false-cls Γ {#K ∈# D. K ≠ L#} and
      is-pos L
      by (auto elim: ord-res-8-can-factorize.cases ord-res-8-can-produce.cases)

    hence atm-of L = A
      using step-hyps
      by (smt (verit, ccfv-SIG) A-in A-least A-undef D-in Γ-lower-set
          linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
          linorder-trm.not-in-lower-setI atm-of-in-atms-of-clssI
          trail-defined-lit-iff-trail-defined-atm)

    hence L = Pos A
      using ‹is-pos L› by (cases L) simp-all

    hence clause-could-propagate Γ D (Pos A)
      unfolding clause-could-propagate-def
      using D-almost-false D-max-lit L-undef by metis

    thus C ≺c D
      using D-in D-neq C-least by metis
  qed
qed

hence C = Some C
  using match-hyps by metis

define C' where
  C' = The-optional (linorder-cls.is-least-in-fset
    (ffilter ((≺c) C) (iefac F |`| (N |∪| Uer))))

have first-step7: ord-res-7 N (Uer, F, Γ, Some C) (Uer, F, Γ', C')
proof (rule ord-res-7.production)
  show ¬ trail-false-cls Γ C
```

using *C-in step-hyps*(*2*) **by** *blast*
next
show *ord-res.is-maximal-lit* (*Pos A*) *C*
using *C-max-lit* **by** *force*
next
show ¬ (∃ *Aa*|∈|*atms-of-clss* (*N* |∪| U_{er}). *Aa* ≺$_t$ *atm-of* (*Pos A*) ∧ *Aa* |∉| *trail-atms* Γ)
by (*metis A-least* Γ*-lower-set linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI literal.sel*(*1*))
next
show ¬ *trail-defined-lit* Γ (*Pos A*)
using *Pos-A-undef* **.**
next
show *is-pos* (*Pos A*)
by *simp*
next
show *trail-false-cls* Γ {#*K* ∈# *C*. *K* ≠ *Pos A*#}
using *C-almost-false* **.**
next
show *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
using *step-hyps* **by** *argo*
next
show Γ′ = (*Pos A*, *Some C*) # Γ
using *step-hyps* **by** *argo*
next
show $\mathcal{C}'$ = *The-optional* (*linorder-cls.is-least-in-fset* (*ffilter* ((≺$_c$) *C*) (*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er}))))
using $\mathcal{C}'$*-def* **.**
qed

moreover have *ord-res-7-invars N* (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}'$)
using ‹$\mathcal{C}$ = *Some C*› *first-step7 match-hyps*(*3*) *ord-res-7-preserves-invars* **by** *blast*

ultimately obtain $\mathcal{C}''$ **where**
following-steps7: (*ord-res-7 N*)** (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}'$) (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}''$) **and**
no-more-step7: (∄$\mathcal{C}'''$. *ord-res-7 N* (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}''$) (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}'''$))
using *MAGIC6* **by** *metis*

show *?thesis*
proof (*intro exI conjI*)
have (*ord-res-7 N*)$^{++}$ (U_{er}, $\mathcal{F}$, Γ, $\mathcal{C}$) (U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}''$)
unfolding ‹$\mathcal{C}$ = *Some C*›
using *first-step7 following-steps7* **by** *simp*

thus (*constant-context ord-res-7*)$^{++}$ *S7* (*N*, U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}''$)
unfolding *S7-def* **by** (*simp add*: *tranclp-constant-context*)

show *ord-res-7-matches-ord-res-8* (*N*, U_{er}, $\mathcal{F}$, Γ′, $\mathcal{C}''$) *S8′*

unfolding *S8′-def* ‹$s8' = (U_{er}, \mathcal{F}, \Gamma')$›
proof (*intro ord-res-7-matches-ord-res-8.intros allI*)
show *ord-res-7-invars N* $(U_{er}, \mathcal{F}, \Gamma', \mathcal{C}'')$
using ‹*ord-res-7-invars N* $(U_{er}, \mathcal{F}, \Gamma', \mathcal{C}')$› *following-steps7*
rtranclp-ord-res-7-preserves-ord-res-7-invars **by** *blast*

show *ord-res-8-invars N* $(U_{er}, \mathcal{F}, \Gamma')$
using *invars-s8′ step-hyps*(*1*) **by** *blast*

fix $C :: 'f$ *gclause*
show $\mathcal{C}'' = Some\ C \longleftrightarrow$ *is-least-nonskipped-clause N* $U_{er}\ \mathcal{F}\ \Gamma'\ C$
using *MAGIC5* ‹*ord-res-7-invars N* $(U_{er}, \mathcal{F}, \Gamma', \mathcal{C}'')$› *no-more-step7* **by** *metis*
qed
qed
next
case *step-hyps*: (*factorize A C* $\mathcal{F}'$)

have
A-in: $A \mid\in\mid$ *atms-of-clss* $(N \mid\cup\mid U_{er})$ **and**
A-undef: $A \mid\notin\mid$ *trail-atms* Γ **and**
A-least: $\forall\, y \mid\in\mid$*atms-of-clss* $(N \mid\cup\mid U_{er}).\ y \neq A \longrightarrow (\forall\, A_1 \mid\in\mid$*trail-atms* $\Gamma.\ A_1 \prec_t y) \longrightarrow A \prec_t y$
using *step-hyps*(*3*) **unfolding** *linorder-trm.is-least-in-fset-iff* **by** *auto*

have
C-in: $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$ **and**
C-can-prop:*clause-could-propagate* $\Gamma\ C$ (*Pos A*) **and**
C-least: $\forall\, D \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er})$.
$D \neq C \longrightarrow$ *clause-could-propagate* $\Gamma\ D$ (*Pos A*) $\longrightarrow C \prec_c D$
using *step-hyps* **unfolding** *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

hence
Pos-A-undef: ¬ *trail-defined-lit* Γ (*Pos A*) **and**
C-max-lit: *linorder-lit.is-maximal-in-mset C* (*Pos A*) **and**
C-almost-false: *trail-false-cls* Γ $\{\#K \in\# C.\ K \neq Pos\ A\#\}$
unfolding *atomize-conj clause-could-propagate-def* **by** *argo*

have *C-not-false*: ¬ *trail-false-cls* Γ *C*
using *C-in step-hyps* **by** *metis*

have *no-undef-atm-lt-A*:
¬ $(\exists\, Aa \mid\in\mid$*atms-of-clss* $(N \mid\cup\mid U_{er}).\ Aa \prec_t A \wedge Aa \mid\notin\mid$ *trail-atms* $\Gamma)$
by (*metis A-least* Γ*-lower-set linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI*)

have *is-least-nonskipped-clause N* $U_{er}\ \mathcal{F}\ \Gamma\ C$
unfolding *is-least-nonskipped-clause-def*

```
    unfolding linorder-cls.is-least-in-ffilter-iff
  proof (intro conjI ballI impI)
    show C |∈| iefac F |‘| (N |∪| U_er)
      using C-in .
  next
    have ord-res-8-can-factorize N U_er F Γ C
    proof (intro ord-res-8-can-factorize.intros)
      show ¬ trail-false-cls Γ C
        using C-not-false .
    next
      show ord-res.is-maximal-lit (Pos A) C
        using C-max-lit .
    next
      show ¬ (∃ Aa|∈|atms-of-clss (N |∪| U_er). Aa ≺_t atm-of (Pos A) ∧ Aa |∉|
trail-atms Γ)
        using no-undef-atm-lt-A by simp
    next
      show ¬ trail-defined-lit Γ (Pos A)
        using Pos-A-undef .
    next
      show is-pos (Pos A)
        by simp
    next
      show trail-false-cls Γ {#K ∈# C. K ≠ Pos A#}
        using C-almost-false .
    next
      show ¬ ord-res.is-strictly-maximal-lit (Pos A) C
        using step-hyps by argo
    qed

    thus trail-false-cls Γ C ∨
      ord-res-8-can-decide-neg N U_er F Γ C ∨
      ord-res-8-can-skip-undefined-neg N U_er F Γ C ∨
      ord-res-8-can-skip-undefined-pos-ultimate N U_er F Γ C ∨
      ord-res-8-can-produce N U_er F Γ C ∨
      ord-res-8-can-factorize N U_er F Γ C
      using ‹ord-res-8-can-factorize N U_er F Γ C› by argo
  next
    fix D :: 'f gclause
    assume
      D-in: D |∈| iefac F |‘| (N |∪| U_er) and
      D-neq: D ≠ C and
      D-spec-disj: trail-false-cls Γ D ∨
        ord-res-8-can-decide-neg N U_er F Γ D ∨
        ord-res-8-can-skip-undefined-neg N U_er F Γ D ∨
        ord-res-8-can-skip-undefined-pos-ultimate N U_er F Γ D ∨
        ord-res-8-can-produce N U_er F Γ D ∨
        ord-res-8-can-factorize N U_er F Γ D
```

hence
ord-res-8-can-decide-neg N U_{er} $\mathcal{F}$ Γ *D* ∨
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *D* ∨
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *D* ∨
ord-res-8-can-produce N U_{er} $\mathcal{F}$ Γ *D* ∨
ord-res-8-can-factorize N U_{er} $\mathcal{F}$ Γ *D*
using *step-hyps D-in* **by** *metis*

thus $C \prec_c D$
proof (*elim disjE*)
assume *ord-res-8-can-decide-neg N* U_{er} $\mathcal{F}$ Γ *D*
thus $C \prec_c D$
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-decide-neg.cases*)
case *hyps*: (*1 L A'*)
hence $A = A'$
using *step-hyps*
by (*smt* (*verit*, *del-insts*) Γ-*lower-set linorder-trm.antisym-conv3*
linorder-trm.dual-order.strict-implies-not-eq linorder-trm.dual-order.strict-trans
linorder-trm.is-least-in-ffilter-iff linorder-trm.not-in-lower-setI)
hence $A \prec_t$ *atm-of L*
using *hyps*
unfolding *linorder-trm.is-least-in-ffilter-iff*
by *argo*
hence *Pos A* $\prec_l$ *L*
by (*cases L*) *simp-all*
thus *?thesis*
using *C-max-lit* ‹*linorder-lit.is-maximal-in-mset D L*›
by (*metis linorder-lit.multp-if-maximal-less-that-maximal*)
qed
next
assume *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *D*
thus $C \prec_c D$
proof (*cases N* U_{er} $\mathcal{F}$ Γ *D rule*: *ord-res-8-can-skip-undefined-neg.cases*)
case *hyps*: (*1 L*)
hence *atm-of L* = *A*
using *step-hyps*
by (*smt* (*verit*, *best*)
A-in A-least A-undef D-in Γ-*lower-set atm-of-in-atms-of-clssI*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
linorder-trm.not-in-lower-setI trail-defined-lit-iff-trail-defined-atm)
hence *Pos A* $\prec_l$ *L*
using ‹*is-neg L*› **by** (*cases L*) *simp-all*
thus *?thesis*
using *C-max-lit* ‹*linorder-lit.is-maximal-in-mset D L*›
by (*metis linorder-lit.multp-if-maximal-less-that-maximal*)
qed
next
assume *ord-res-8-can-skip-undefined-pos-ultimate N* U_{er} $\mathcal{F}$ Γ *D*
thus $C \prec_c D$

proof (*cases* $N\ U_{er}\ \mathcal{F}\ \Gamma\ D$ *rule*: *ord-res-8-can-skip-undefined-pos-ultimate.cases*)
case *hyps*: (*1* L)
thus *?thesis*
by (*meson C-in D-neq linorder-cls.linorder-cases*)
qed
next
assume *ord-res-8-can-produce* $N\ U_{er}\ \mathcal{F}\ \Gamma\ D$

then obtain L **where**
D-max-lit: *ord-res.is-maximal-lit* $L\ D$ **and**
$\neg\ (\exists\, A|\in|\mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).\ A \prec_t \mathit{atm\text{-}of}\ L \wedge A\ |\notin|\ \mathit{trail\text{-}atms}\ \Gamma)$
and
L-undef: $\neg$ *trail-defined-lit* $\Gamma\ L$ **and**
D-almost-false: *trail-false-cls* $\Gamma\ \{\#K \in\#\ D.\ K \neq L\#\}$ **and**
is-pos L
by (*auto elim*: *ord-res-8-can-factorize.cases ord-res-8-can-produce.cases*)

hence *atm-of* $L = A$
using *step-hyps*
by (*smt* (*verit*, *ccfv-SIG*) *A-in A-least A-undef D-in* Γ*-lower-set*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3
linorder-trm.not-in-lower-setI atm-of-in-atms-of-clssI
trail-defined-lit-iff-trail-defined-atm)

hence $L = \mathit{Pos}\ A$
using ‹*is-pos* L› **by** (*cases* L) *simp-all*

hence *clause-could-propagate* $\Gamma\ D\ (\mathit{Pos}\ A)$
unfolding *clause-could-propagate-def*
using *D-almost-false D-max-lit L-undef* **by** *metis*

thus $C \prec_c D$
using *D-in D-neq C-least* **by** *metis*
next
assume *ord-res-8-can-factorize* $N\ U_{er}\ \mathcal{F}\ \Gamma\ D$

then obtain L **where**
D-max-lit: *ord-res.is-maximal-lit* $L\ D$ **and**
$\neg\ (\exists\, A|\in|\mathit{atms\text{-}of\text{-}clss}\ (N\ |\cup|\ U_{er}).\ A \prec_t \mathit{atm\text{-}of}\ L \wedge A\ |\notin|\ \mathit{trail\text{-}atms}\ \Gamma)$
and
L-undef: $\neg$ *trail-defined-lit* $\Gamma\ L$ **and**
D-almost-false: *trail-false-cls* $\Gamma\ \{\#K \in\#\ D.\ K \neq L\#\}$ **and**
is-pos L
by (*auto elim*: *ord-res-8-can-factorize.cases ord-res-8-can-produce.cases*)

hence *atm-of* $L = A$
using *step-hyps*
by (*smt* (*verit*, *ccfv-SIG*) *A-in A-least A-undef D-in* Γ*-lower-set*
linorder-lit.is-maximal-in-mset-iff linorder-trm.antisym-conv3

linorder-trm.not-in-lower-setI atm-of-in-atms-of-clssI
trail-defined-lit-iff-trail-defined-atm)

hence $L = Pos\ A$
using ‹*is-pos L*› **by** (*cases L*) *simp-all*

hence *clause-could-propagate* $\Gamma\ D\ (Pos\ A)$
unfolding *clause-could-propagate-def*
using *D-almost-false D-max-lit L-undef* **by** *metis*

thus $C \prec_c D$
using *D-in D-neq C-least* **by** *metis*
qed
qed

hence $\mathcal{C} = Some\ C$
using *match-hyps* **by** *metis*

define $\mathcal{C}'$ **where**
$\mathcal{C}' = Some\ (efac\ C)$

have *first-step*$\mathcal{Y}$: *ord-res-*$\mathcal{Y}$ $N\ (U_{er}, \mathcal{F}, \Gamma, Some\ C)\ (U_{er}, \mathcal{F}', \Gamma, \mathcal{C}')$
unfolding $\mathcal{C}'$*-def*
proof (*rule ord-res-*$\mathcal{Y}$*.factoring*)
show ¬ *trail-false-cls* $\Gamma\ C$
using *C-not-false* **.**
next
show *ord-res.is-maximal-lit* $(Pos\ A)\ C$
using *C-max-lit* **.**
next
show ¬ $(\exists Aa|\in|atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}).\ Aa \prec_t atm\text{-}of\ (Pos\ A) \wedge Aa\ |\notin|\ trail\text{-}atms\ \Gamma)$
using *no-undef-atm-lt-A* **by** *simp*
next
show ¬ *trail-defined-lit* $\Gamma\ (Pos\ A)$
using *Pos-A-undef* **.**
next
show *is-pos* $(Pos\ A)$
by *simp*
next
show *trail-false-cls* $\Gamma\ \{\#K \in\#\ C.\ K \neq Pos\ A\#\}$
using *C-almost-false* **.**
next
show ¬ *ord-res.is-strictly-maximal-lit* $(Pos\ A)\ C$
using *step-hyps* **by** *argo*
next
show $\mathcal{F}' = finsert\ C\ \mathcal{F}$
using *step-hyps* **by** *argo*
qed

moreover have *ord-res-7-invars N* $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}')$
using ‹$\mathcal{C}$ = *Some C*› *first-step7 match-hyps*(*3*) *ord-res-7-preserves-invars* **by** *blast*

ultimately obtain $\mathcal{C}''$ **where**
following-steps7: (*ord-res-7 N*)** $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}')$ $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$ **and**
no-more-step7: ($\nexists\mathcal{C}'''$. *ord-res-7 N* $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$ $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}''')$)
using *MAGIC6* **by** *metis*

show *?thesis*
proof (*intro exI conjI*)
have (*ord-res-7 N*)$^{++}$ $(U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$
unfolding ‹$\mathcal{C}$ = *Some C*›
using *first-step7 following-steps7* **by** *simp*

thus (*constant-context ord-res-7*)$^{++}$ *S7* $(N, U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$
unfolding *S7-def* **by** (*simp add*: *tranclp-constant-context*)

show *ord-res-7-matches-ord-res-8* $(N, U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$ *S8'*
unfolding *S8'-def* ‹*s8'* = $(U_{er}, \mathcal{F}', \Gamma)$›
proof (*intro ord-res-7-matches-ord-res-8.intros allI*)
show *ord-res-7-invars N* $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$
using ‹*ord-res-7-invars N* $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}')$› *following-steps7*
rtranclp-ord-res-7-preserves-ord-res-7-invars **by** *blast*

show *ord-res-8-invars N* $(U_{er}, \mathcal{F}', \Gamma)$
using *invars-s8'* *step-hyps*(*1*) **by** *blast*

fix *C* :: *'f gclause*
show $\mathcal{C}''$ = *Some C* $\longleftrightarrow$ *is-least-nonskipped-clause N* U_{er} $\mathcal{F}'$ Γ *C*
using *MAGIC5* ‹*ord-res-7-invars N* $(U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'')$› *no-more-step7* **by** *metis*
qed
qed
next
case *step-hyps*: (*resolution E A D* U_{er}' Γ')

note *E-max-lit* = ‹*ord-res.is-maximal-lit* (*Neg A*) *E*›

have
E-in: *E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) **and**
E-false: *trail-false-cls* Γ *E* **and**
E-least: $\forall$ *F* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}). *F* $\neq$ *E* $\longrightarrow$ *trail-false-cls* Γ *F* $\longrightarrow$ *E* $\prec_c$ *F*
using *step-hyps*
unfolding *atomize-conj*
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *argo*

have *is-least-nonskipped-clause N* U_{er} $\mathcal{F}$ Γ *E*
unfolding *is-least-nonskipped-clause-def*
unfolding *linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show *E* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er})
using *E-in* **.**
next
show *trail-false-cls* Γ *E* $\vee$
ord-res-8-can-decide-neg N U_{er} $\mathcal{F}$ Γ *E* $\vee$
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *E* $\vee$
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *E* $\vee$
ord-res-8-can-produce N U_{er} $\mathcal{F}$ Γ *E* $\vee$
ord-res-8-can-factorize N U_{er} $\mathcal{F}$ Γ *E*
using *E-false* **by** *argo*
next
fix *F* :: *'f gclause*
assume
F-in: *F* $|\in|$ *iefac* $\mathcal{F}$ $|`|$ (*N* $|\cup|$ U_{er}) **and**
F-neq: *F* $\neq$ *E* **and**
D-spec-disj: *trail-false-cls* Γ *F* $\vee$
ord-res-8-can-decide-neg N U_{er} $\mathcal{F}$ Γ *F* $\vee$
ord-res-8-can-skip-undefined-neg N U_{er} $\mathcal{F}$ Γ *F* $\vee$
ord-res-8-can-skip-undefined-pos-ultimate N U_{er} $\mathcal{F}$ Γ *F* $\vee$
ord-res-8-can-produce N U_{er} $\mathcal{F}$ Γ *F* $\vee$
ord-res-8-can-factorize N U_{er} $\mathcal{F}$ Γ *F*

show *E* $\prec_c$ *F*
using *D-spec-disj*
proof (*elim disjE*)
assume *trail-false-cls* Γ *F*
thus *E* $\prec_c$ *F*
using *E-least F-in F-neq* **by** *metis*
next
assume *ord-res-8-can-decide-neg N* U_{er} $\mathcal{F}$ Γ *F*
thus *E* $\prec_c$ *F*
proof (*cases N* U_{er} $\mathcal{F}$ Γ *F rule*: *ord-res-8-can-decide-neg.cases*)
case *hyps*: (*1 L' A'*)
thus *?thesis*
using *no-undef-atom-le-max-lit-if-lt-false-clause*[
OF Γ*-lower-set E-in E-false E-max-lit F-in* ‹*ord-res.is-maximal-lit L' F*›]
by (*metis* (*no-types*, *lifting*) *F-neq linorder-cls.neq-iff*
linorder-trm.is-least-in-ffilter-iff reflclp-iff)
qed
next
assume *ord-res-8-can-skip-undefined-neg N* U_{er} $\mathcal{F}$ Γ *F*
thus *E* $\prec_c$ *F*
proof (*cases N* U_{er} $\mathcal{F}$ Γ *F rule*: *ord-res-8-can-skip-undefined-neg.cases*)

```
      case (1 L')
      thus ?thesis
        using no-undef-atom-le-max-lit-if-lt-false-clause[
          OF Γ-lower-set E-in E-false E-max-lit F-in ‹ord-res.is-maximal-lit L'
F›]
      by (metis F-in F-neq atm-of-in-atms-of-clssI linorder-cls.not-less-iff-gr-or-eq
          linorder-lit.is-maximal-in-mset-iff reflclp-iff
          trail-defined-lit-iff-trail-defined-atm)
    qed
  next
    assume ord-res-8-can-skip-undefined-pos-ultimate N U_er F Γ F
    thus E ≺_c F
  proof (cases N U_er F Γ F rule: ord-res-8-can-skip-undefined-pos-ultimate.cases)
      case (1 L')
      thus ?thesis
        using no-undef-atom-le-max-lit-if-lt-false-clause[
          OF Γ-lower-set E-in E-false E-max-lit F-in ‹ord-res.is-maximal-lit L'
F›]
      by (metis F-in F-neq atm-of-in-atms-of-clssI linorder-cls.not-less-iff-gr-or-eq
          linorder-lit.is-maximal-in-mset-iff reflclp-iff
          trail-defined-lit-iff-trail-defined-atm)
    qed
  next
    assume ord-res-8-can-produce N U_er F Γ F
    thus E ≺_c F
    proof (cases N U_er F Γ F rule: ord-res-8-can-produce.cases)
      case (1 L')
      thus ?thesis
        using no-undef-atom-le-max-lit-if-lt-false-clause[
          OF Γ-lower-set E-in E-false E-max-lit F-in ‹ord-res.is-maximal-lit L'
F›]
      by (metis F-in F-neq atm-of-in-atms-of-clssI linorder-cls.not-less-iff-gr-or-eq
          linorder-lit.is-maximal-in-mset-iff reflclp-iff
          trail-defined-lit-iff-trail-defined-atm)
    qed
  next
    assume ord-res-8-can-factorize N U_er F Γ F
    thus E ≺_c F
    proof (cases N U_er F Γ F rule: ord-res-8-can-factorize.cases)
      case (1 L')
      thus ?thesis
        using no-undef-atom-le-max-lit-if-lt-false-clause[
          OF Γ-lower-set E-in E-false E-max-lit F-in ‹ord-res.is-maximal-lit L'
F›]
      by (metis F-in F-neq atm-of-in-atms-of-clssI linorder-cls.not-less-iff-gr-or-eq
          linorder-lit.is-maximal-in-mset-iff reflclp-iff
          trail-defined-lit-iff-trail-defined-atm)
    qed
  qed
```

qed

hence $\mathcal{C} = \mathit{Some}\ E$
using *match-hyps* **by** *metis*

obtain $\mathcal{C}'$ **where** *first-step7*: *ord-res-7* N $(U_{er}, \mathcal{F}, \Gamma, \mathit{Some}\ E)$ $(U_{er}', \mathcal{F}, \Gamma', \mathcal{C}')$
proof *atomize-elim*
have $(\mathit{Pos}\ A, \mathit{Some}\ D) \in \mathit{set}\ \Gamma$
using ‹*map-of* Γ $(\mathit{Pos}\ A) = \mathit{Some}\ (\mathit{Some}\ D)$› **by** (*metis map-of-SomeD*)

hence *D-almost-false*: *trail-false-cls* Γ $\{\#K \in\# D.\ K \neq \mathit{Pos}\ A\#\}$
using *ord-res-7-invars-implies-propagated-clause-almost-false*
‹*ord-res-7-invars* N $(U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$› **by** *metis*

have *eres-false*: *trail-false-cls* Γ $(\mathit{eres}\ D\ E)$
unfolding *trail-false-cls-def*
proof (*intro ballI*)
fix $K :: 'f\ \mathit{gliteral}$
assume $K \in\# \mathit{eres}\ D\ E$
hence $K \in\# D \wedge K \neq \mathit{Pos}\ A \vee K \in\# E$
using *strong-lit-in-one-of-resolvents-if-in-eres*[*OF E-max-lit*] **by** *simp*
thus *trail-false-lit* Γ K
proof (*elim disjE conjE*)
assume $K \in\# D$ **and** $K \neq \mathit{Pos}\ A$
thus *trail-false-lit* Γ K
using *D-almost-false* **unfolding** *trail-false-cls-def* **by** *simp*
next
assume $K \in\# E$
thus *trail-false-lit* Γ K
using *E-false* **unfolding** *trail-false-cls-def* **by** *simp*
qed
qed

show $\exists \mathcal{C}'.$ *ord-res-7* N $(U_{er}, \mathcal{F}, \Gamma, \mathit{Some}\ E)$ $(U_{er}', \mathcal{F}, \Gamma', \mathcal{C}')$
proof (*cases eres D E* $= \{\#\}$)
case *True*
hence $\bigwedge Ln.\ \forall K.\ \mathit{ord\text{-}res.is\text{-}maximal\text{-}lit}\ K\ (\mathit{eres}\ D\ E) \longrightarrow \mathit{atm\text{-}of}\ K \preceq_t \mathit{atm\text{-}of}\ (\mathit{fst}\ Ln)$
unfolding *linorder-lit.is-maximal-in-mset-iff*
by *simp*
hence $\Gamma' = \mathit{dropWhile}\ (\lambda Ln.\ \mathit{True})\ \Gamma$
using *step-hyps* **by** *meson*
hence $\Gamma' = []$
by *simp*
show *?thesis*
proof (*intro exI*)
show *ord-res-7* N $(U_{er}, \mathcal{F}, \Gamma, \mathit{Some}\ E)$ $(U_{er}', \mathcal{F}, \Gamma', \mathit{Some}\ \{\#\})$
using *ord-res-7.resolution-bot*[*OF E-false E-max-lit*]

```
        ‹map-of Γ (Pos A) = Some (Some D)› ‹U_er' = finsert (eres D E) U_er›
True ‹Γ' = []›
        by simp
      qed
    next
      case False
      then obtain K where eres-max-lit: ord-res.is-maximal-lit K (eres D E)
        using linorder-lit.ex-maximal-in-mset by presburger
       hence ⋀Ln. (∀ K. ord-res.is-maximal-lit K (eres D E) ⟶ atm-of K ≼_t
atm-of (fst Ln)) ⟷
        atm-of K ≼_t atm-of (fst Ln)
        by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')
      hence Γ'-eq: Γ' = dropWhile (λLn. atm-of K ≼_t atm-of (fst Ln)) Γ
        using step-hyps by meson
      show ?thesis
      proof (cases K)
        case (Pos A_K)
        hence K-pos: is-pos K
          by simp

        then show ?thesis
          using ord-res-7.resolution-pos[OF E-false E-max-lit - - - False Γ'-eq
              eres-max-lit K-pos]
          using step-hyps by fastforce
      next
        case (Neg A_K)

        hence K-neg: is-neg K
          by simp

        have trail-false-lit Γ K
          using eres-false eres-max-lit
          unfolding linorder-lit.is-maximal-in-mset-iff trail-false-cls-def by metis

        hence ∃ opt. (− K, opt) ∈ set Γ
          unfolding trail-false-lit-def by auto

        moreover have ∀ Ln∈set Γ. is-neg (fst Ln) = (snd Ln = None)
          using invars7 by argo

        ultimately obtain C where (− K, Some C) ∈ set Γ
          unfolding Neg uminus-Neg by fastforce

        hence map-of Γ (− K) = Some (Some C)
        proof (rule map-of-is-SomeI[rotated])
          show distinct (map fst Γ)
            using Γ-consistent
            by (metis distinct-lits-if-trail-consistent)
        qed
```

```
        then show ?thesis
          using ord-res-7.resolution-neg[OF E-false E-max-lit - - - False Γ'-eq
              eres-max-lit K-neg]
          using step-hyps by fastforce
      qed
    qed
  qed

  moreover have ord-res-7-invars N (U_er', F, Γ', C')
    using ‹C = Some E› first-step7 match-hyps(3) ord-res-7-preserves-invars by
blast

  ultimately obtain C'' where
    following-steps7: (ord-res-7 N)** (U_er', F, Γ', C') (U_er', F, Γ', C'') and
    no-more-step7: (∄C'''. ord-res-7 N (U_er', F, Γ', C'') (U_er', F, Γ', C'''))
    using MAGIC6 by metis

  show ?thesis
  proof (intro exI conjI)
    have (ord-res-7 N)++ (U_er, F, Γ, C) (U_er', F, Γ', C'')
      unfolding ‹C = Some E›
      using first-step7 following-steps7 by simp

    thus (constant-context ord-res-7)++ S7 (N, U_er', F, Γ', C'')
      unfolding S7-def by (simp add: tranclp-constant-context)

    show ord-res-7-matches-ord-res-8 (N, U_er', F, Γ', C'') S8'
      unfolding S8'-def ‹s8' = (U_er', F, Γ')›
    proof (intro ord-res-7-matches-ord-res-8.intros allI)
      show ord-res-7-invars N (U_er', F, Γ', C'')
        using ‹ord-res-7-invars N (U_er', F, Γ', C')› following-steps7
          rtranclp-ord-res-7-preserves-ord-res-7-invars by blast

      show ord-res-8-invars N (U_er', F, Γ')
        using invars-s8' step-hyps(1) by blast

      fix C :: 'f gclause
      show C'' = Some C ⟷ is-least-nonskipped-clause N U_er' F Γ' C
        using MAGIC5 ‹ord-res-7-invars N (U_er', F, Γ', C'')› no-more-step7 by
metis
    qed
  qed
 qed
qed

theorem bisimulation-ord-res-7-ord-res-8:
 defines match ≡ λ-. ord-res-7-matches-ord-res-8
 shows ∃(MATCH :: nat × nat ⇒ 'f ord-res-7-state ⇒ 'f ord-res-8-state ⇒ bool)
```

```
\mathcal{R}_f \mathcal{R}_b.
    bisimulation
      (constant-context ord-res-7) ord-res-7-final
      (constant-context ord-res-8) ord-res-8-final
      MATCH \mathcal{R}_f \mathcal{R}_b
proof (rule ex-bisimulation-from-backward-simulation)
  show right-unique (constant-context ord-res-7)
    using right-unique-constant-context right-unique-ord-res-7 by metis
next
  show right-unique (constant-context ord-res-8)
    using right-unique-constant-context right-unique-ord-res-8 by metis
next
  show ∀ S. ord-res-7-final S ⟶ (∄ S'. constant-context ord-res-7 S S')
    by (metis finished-def ord-res-7-semantics.final-finished)
next
  show ∀ S. ord-res-8-final S ⟶ (∄ S'. constant-context ord-res-8 S S')
    by (metis finished-def ord-res-8-semantics.final-finished)
next
  show ∀ i S7 S8. match i S7 S8 ⟶ ord-res-7-final S7 ⟷ ord-res-8-final S8
    unfolding match-def
    using ord-res-7-final-iff-ord-res-8-final by metis
next
  show ∀ i S7 S8. match i S7 S8 ⟶
       safe-state (constant-context ord-res-7) ord-res-7-final S7 ∧
       safe-state (constant-context ord-res-8) ord-res-8-final S8
  proof (intro allI impI conjI)
    fix i S7 S8
    assume match: match i S7 S8
    show safe-state (constant-context ord-res-7) ord-res-7-final S7
      using match[unfolded match-def]
      using ord-res-7-safe-state-if-invars
      using ord-res-7-matches-ord-res-8.simps by auto

    show safe-state (constant-context ord-res-8) ord-res-8-final S8
      using match[unfolded match-def]
      using ord-res-8-safe-state-if-invars
      using ord-res-7-matches-ord-res-8.simps by auto
  qed
next
  show wfp (λ- -. False)
    by simp
next
  show ∀ i S7 S8 S8'. match i S7 S8 ⟶ constant-context ord-res-8 S8 S8' ⟶
    (∃ i' S7'. (constant-context ord-res-7)^++ S7 S7' ∧ match i' S7' S8') ∨
    (∃ i'. match i' S7 S8' ∧ False)
    unfolding match-def
    using backward-simulation-between-7-and-8 by metis
qed
```

end

35 ORD-RES-9 (factorize when propagating)

type-synonym *'f ord-res-9-state* =
'f gclause fset × *'f gclause fset* × *'f gclause fset* × (*'f gliteral* × *'f gclause option*) *list*

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-8-matches-ord-res-9* :: *'f ord-res-8-state* ⇒ *'f ord-res-9-state* ⇒ *bool* **where**
ord-res-8-invars N (U_{er}, $\mathcal{F}$, Γ) ⟹
ord-res-8-matches-ord-res-9 (*N*, U_{er}, $\mathcal{F}$, Γ) (*N*, U_{er}, $\mathcal{F}$, Γ)

lemma *ord-res-8-final-iff-ord-res-9-final*:
fixes *S8 S9*
assumes *match*: *ord-res-8-matches-ord-res-9 S8 S9*
shows *ord-res-8-final S8* ⟷ *ord-res-8-final S9*
using *match*
proof (*cases S8 S9 rule*: *ord-res-8-matches-ord-res-9.cases*)
case (*1 N* U_{er} $\mathcal{F}$ Γ)
then show *?thesis*
by *argo*
qed

lemma *backward-simulation-between-8-and-9*:
fixes *S8 S9 S9'*
assumes *match*: *ord-res-8-matches-ord-res-9 S8 S9* **and** *step*: *constant-context ord-res-9 S9 S9'*
shows ∃ *S8'*. (*constant-context ord-res-8*)$^{++}$ *S8 S8'* ∧ *ord-res-8-matches-ord-res-9 S8' S9'*
using *match*
proof (*cases S8 S9 rule*: *ord-res-8-matches-ord-res-9.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ Γ)

note *S8-def* = ‹*S8* = (*N*, U_{er}, $\mathcal{F}$, Γ)›
note *S9-def* = ‹*S9* = (*N*, U_{er}, $\mathcal{F}$, Γ)›
note *invars* = ‹*ord-res-8-invars N* (U_{er}, $\mathcal{F}$, Γ)›

obtain *s9'* **where** *S9'-def*: *S9'* = (*N*, *s9'*) **and** *step'*: *ord-res-9 N* (U_{er}, $\mathcal{F}$, Γ) *s9'*
using *step* **unfolding** *S9-def*
using *constant-context.cases* **by** *blast*

have *ord-res-8 N* (U_{er}, $\mathcal{F}$, Γ) *s9'* ∨ (*ord-res-8 N OO ord-res-8 N*) (U_{er}, $\mathcal{F}$, Γ) *s9'*
using *step' ord-res-9-is-one-or-two-ord-res-9-steps* **by** *metis*

```
  hence steps8: (ord-res-8 N)^++ (U_er, F, Γ) s9'
    by auto

  show ?thesis
  proof (intro exI conjI)
    show (constant-context ord-res-8)^++ S8 (N, s9')
      unfolding S8-def
      using steps8 by (simp add: tranclp-constant-context)
  next
    have ord-res-8-invars N s9'
      using invars steps8 tranclp-ord-res-8-preserves-invars by metis

    thus ord-res-8-matches-ord-res-9 (N, s9') S9'
      unfolding S9'-def
      by (metis ord-res-8-matches-ord-res-9.intros prod-cases3)
  qed
qed

theorem bisimulation-ord-res-8-ord-res-9:
  defines match ≡ λ-. ord-res-8-matches-ord-res-9
  shows ∃(MATCH :: nat × nat ⇒ 'f ord-res-8-state ⇒ 'f ord-res-9-state ⇒ bool)
R_f R_b.
    bisimulation
      (constant-context ord-res-8) ord-res-8-final
      (constant-context ord-res-9) ord-res-8-final
      MATCH R_f R_b
proof (rule ex-bisimulation-from-backward-simulation)
  show right-unique (constant-context ord-res-8)
    using right-unique-constant-context right-unique-ord-res-8 by metis
next
  show right-unique (constant-context ord-res-9)
    using right-unique-constant-context right-unique-ord-res-9 by metis
next
  show ∀S. ord-res-8-final S ⟶ (∄S'. constant-context ord-res-8 S S')
    by (metis finished-def ord-res-8-semantics.final-finished)
next
  show ∀S. ord-res-8-final S ⟶ (∄S'. constant-context ord-res-9 S S')
    by (metis finished-def ord-res-9-semantics.final-finished)
next
  show ∀i S8 S9. match i S8 S9 ⟶ ord-res-8-final S8 ⟷ ord-res-8-final S9
    unfolding match-def
    using ord-res-8-final-iff-ord-res-9-final by metis
next
  show ∀i S8 S9. match i S8 S9 ⟶
      safe-state (constant-context ord-res-8) ord-res-8-final S8 ∧
      safe-state (constant-context ord-res-9) ord-res-8-final S9
  proof (intro allI impI conjI)
    fix i S8 S9
    assume match: match i S8 S9
```

show *safe-state* (*constant-context ord-res-8*) *ord-res-8-final S8*
using *match*[*unfolded match-def*]
using *ord-res-8-safe-state-if-invars*
using *ord-res-8-matches-ord-res-9.simps* **by** *auto*

show *safe-state* (*constant-context ord-res-9*) *ord-res-8-final S9*
using *match*[*unfolded match-def*]
using *ord-res-9-safe-state-if-invars*
using *ord-res-8-matches-ord-res-9.simps* **by** *auto*
qed
next
show *wfp* (λ- -. *False*)
by *simp*
next
show ∀ *i S8 S9 S9′*. *match i S8 S9* ⟶ *constant-context ord-res-9 S9 S9′* ⟶
(∃ *i′ S8′*. (*constant-context ord-res-8*)$^{++}$ *S8 S8′* ∧ *match i′ S8′ S9′*) ∨
(∃ *i′*. *match i′ S8 S9′* ∧ *False*)
unfolding *match-def*
using *backward-simulation-between-8-and-9* **by** *metis*
qed

end

36 ORD-RES-10 (propagate iff a conflict is produced)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-9-matches-ord-res-10* :: *′f ord-res-9-state* ⇒ *′f ord-res-10-state* ⇒ *bool* **where**
ord-res-8-invars N (U_{er}, $\mathcal{F}$, Γ_9) ⟹
ord-res-10-invars N (U_{er}, $\mathcal{F}$, Γ_{10}) ⟹
list-all2 (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10} ⟹
list-all2 (λ*x y*. *snd y* ≠ *None* ⟶ *x* = *y*) Γ_9 Γ_{10} ⟹
ord-res-9-matches-ord-res-10 (*N*, U_{er}, $\mathcal{F}$, Γ_9) (*N*, U_{er}, $\mathcal{F}$, Γ_{10})

lemma *ord-res-9-final-iff-ord-res-10-final*:
fixes *S9 S10*
assumes *match*: *ord-res-9-matches-ord-res-10 S9 S10*
shows *ord-res-8-final S9* ⟷ *ord-res-8-final S10*
using *match*
proof (*cases S9 S10 rule*: *ord-res-9-matches-ord-res-10.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ Γ_9 Γ_{10})
then show *?thesis*
using *trail-atms-eq-trail-atms-if-same-lits*[*OF* ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}›]
using *trail-false-cls-eq-trail-false-cls-if-same-lits*[*OF* ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}›]

unfolding *ord-res-8-final.simps*
by *simp*
qed

lemma *backward-simulation-between-9-and-10*:
fixes *S9 S10 S10′*
assumes
match: *ord-res-9-matches-ord-res-10 S9 S10* **and**
step: *constant-context ord-res-10 S10 S10′*
shows ∃ *S9′*. (*constant-context ord-res-9*)$^{++}$ *S9 S9′* ∧ *ord-res-9-matches-ord-res-10 S9′ S10′*
using *match*
proof (*cases S9 S10 rule*: *ord-res-9-matches-ord-res-10.cases*)
case *match-hyps*: (*1 N* U_{er} $\mathcal{F}$ Γ_9 Γ_{10})

note *S9-def* = ‹*S9* = (*N*, U_{er}, $\mathcal{F}$, Γ_9)›
note *S10-def* = ‹*S10* = (*N*, U_{er}, $\mathcal{F}$, Γ_{10})›
note *invars9* = ‹*ord-res-8-invars N* (U_{er}, $\mathcal{F}$, Γ_9)›
note *invars10* = ‹*ord-res-10-invars N* (U_{er}, $\mathcal{F}$, Γ_{10})›

have *trail-atms* Γ_9 = *trail-atms* Γ_{10}
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}› *trail-atms-eq-trail-atms-if-same-lits*
by *metis*

have *trail-false-lit* Γ_9 = *trail-false-lit* Γ_{10}
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}› *trail-false-lit-eq-trail-false-lit-if-same-lits*
by *metis*

have *trail-false-cls* Γ_9 = *trail-false-cls* Γ_{10}
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}› *trail-false-cls-eq-trail-false-cls-if-same-lits*
by *metis*

have *trail-defined-lit* Γ_9 = *trail-defined-lit* Γ_{10}
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}›
trail-defined-lit-eq-trail-defined-lit-if-same-lits **by** *metis*

have *trail-defined-cls* Γ_9 = *trail-defined-cls* Γ_{10}
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}›
trail-defined-cls-eq-trail-defined-cls-if-same-lits **by** *metis*

have *clause-could-propagate* Γ_9 = *clause-could-propagate* Γ_{10}
unfolding *clause-could-propagate-def*
unfolding ‹*trail-defined-lit* Γ_9 = *trail-defined-lit* Γ_{10}›
unfolding ‹*trail-false-cls* Γ_9 = *trail-false-cls* Γ_{10}› **..**

have Γ_9-*sorted*: *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) ≺$_t$ *atm-of* (*fst x*)) Γ_9
using *invars9*[*unfolded ord-res-8-invars-def trail-is-sorted-def*, *simplified*] **by** *argo*

```
  obtain s10' where S10' = (N, s10') and step10: ord-res-10 N (U_er, F, Γ_10)
s10'
    using step unfolding S10-def by (auto elim: constant-context.cases)

  show ?thesis
    using step10
  proof (cases N (U_er, F, Γ_10) s10' rule: ord-res-10.cases)
    case step-hyps: (decide-neg A Γ_10')

    define Γ_9' where
      Γ_9' = (Neg A, None) # Γ_9

    show ?thesis
    proof (intro exI conjI)
      have step9: ord-res-9 N (U_er, F, Γ_9) (U_er, F, Γ_9')
      proof (rule ord-res-9.decide-neg)
        show ¬ (∃ C|∈|iefac F |'| (N |∪| U_er). trail-false-cls Γ_9 C)
          using step-hyps ‹trail-false-cls Γ_9 = trail-false-cls Γ_10› by argo
      next
        show linorder-trm.is-least-in-fset
          {|A_2 |∈| atms-of-clss (N |∪| U_er). ∀ A_1 |∈| trail-atms Γ_9. A_1 ≺_t A_2|} A
          using step-hyps ‹trail-atms Γ_9 = trail-atms Γ_10› by metis
      next
        show ¬ (∃ C|∈|iefac F |'| (N |∪| U_er). clause-could-propagate Γ_9 C (Pos
A))
          using step-hyps ‹clause-could-propagate Γ_9 = clause-could-propagate Γ_10›
by metis
      next
        show Γ_9' = (Neg A, None) # Γ_9
          using Γ_9'-def .
      qed

      thus (constant-context ord-res-9)^++ S9 (N, U_er, F, Γ_9') S10'
        unfolding S9-def by (auto intro: constant-context.intros)

      show ord-res-9-matches-ord-res-10 (N, U_er, F, Γ_9') S10'
        unfolding ‹S10' = (N, s10')› ‹s10' = (U_er, F, Γ_10')›
      proof (rule ord-res-9-matches-ord-res-10.intros)
        show ord-res-8-invars N (U_er, F, Γ_9')
          using invars9 step9 ord-res-9-preserves-invars by metis
      next
        show ord-res-10-invars N (U_er, F, Γ_10')
          using invars10 step10 ord-res-10-preserves-invars ‹s10' = (U_er, F, Γ_10')›
by metis
      next
        show list-all2 (λx y. fst x = fst y) Γ_9' Γ_10'
          unfolding ‹Γ_9' = (Neg A, None) # Γ_9› ‹Γ_10' = (Neg A, None) # Γ_10›
          using ‹list-all2 (λx y. fst x = fst y) Γ_9 Γ_10› by simp
      next
```

```
      show list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9' Γ10'
        unfolding ‹Γ9' = (Neg A, None) # Γ9› ‹Γ10' = (Neg A, None) # Γ10›
        using ‹list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9 Γ10›
        by simp
    qed
  qed
next
  case step-hyps: (decide-pos A C Γ10' F')

  define Γ9' where
    Γ9' = (Pos A, Some (efac C)) # Γ9

  show ?thesis
  proof (intro exI conjI)
    have step9: ord-res-9 N (U_er, F, Γ9) (U_er, F', Γ9')
    proof (rule ord-res-9.propagate)
      show ¬ (∃ C|∈|iefac F |`| (N |∪| U_er). trail-false-cls Γ9 C)
        using step-hyps ‹trail-false-cls Γ9 = trail-false-cls Γ10› by argo
    next
      show linorder-trm.is-least-in-fset
        {|A2 |∈| atms-of-clss (N |∪| U_er). ∀ A1|∈|trail-atms Γ9. A1 ≺t A2|} A
        using step-hyps ‹trail-atms Γ9 = trail-atms Γ10› by metis
    next
      show linorder-cls.is-least-in-fset
        {|C |∈| iefac F |`| (N |∪| U_er). clause-could-propagate Γ9 C (Pos A)|} C
        using step-hyps ‹clause-could-propagate Γ9 = clause-could-propagate Γ10›
by metis
    next
      show Γ9' = (Pos A, Some (efac C)) # Γ9
        using Γ9'-def .
    next
      show F' = (if ord-res.is-strictly-maximal-lit (Pos A) C then F else finsert
C F)
        using step-hyps by argo
    qed

    thus (constant-context ord-res-9)^++ S9 (N, U_er, F', Γ9')
      unfolding S9-def by (auto intro: constant-context.intros)

    show ord-res-9-matches-ord-res-10 (N, U_er, F', Γ9') S10'
      unfolding ‹S10' = (N, s10')› ‹s10' = (U_er, F', Γ10')›
    proof (rule ord-res-9-matches-ord-res-10.intros)
      show ord-res-8-invars N (U_er, F', Γ9')
        using invars9 step9 ord-res-9-preserves-invars by metis
    next
      show ord-res-10-invars N (U_er, F', Γ10')
       using invars10 step10 ord-res-10-preserves-invars ‹s10' = (U_er, F', Γ10')›
by metis
    next
```

```
      show list-all2 (λx y. fst x = fst y) Γ9' Γ10'
        unfolding ‹Γ9' = (Pos A, Some (efac C)) # Γ9› ‹Γ10' = (Pos A, None)
# Γ10›
        using ‹list-all2 (λx y. fst x = fst y) Γ9 Γ10› by simp
      next
      show list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9' Γ10'
        unfolding ‹Γ9' = (Pos A, Some (efac C)) # Γ9› ‹Γ10' = (Pos A, None)
# Γ10›
        using ‹list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9 Γ10›
        by simp
      qed
    qed
  next
    case step-hyps: (propagate A C Γ10' F')

    define Γ9' where
      Γ9' = (Pos A, Some (efac C)) # Γ9

    show ?thesis
    proof (intro exI conjI)
      have step9: ord-res-9 N (U_er, F, Γ9) (U_er, F', Γ9')
      proof (rule ord-res-9.propagate)
        show ¬ (∃ C|∈|iefac F |'| (N |∪| U_er). trail-false-cls Γ9 C)
          using step-hyps ‹trail-false-cls Γ9 = trail-false-cls Γ10› by argo
      next
        show linorder-trm.is-least-in-fset
          {|A2 |∈| atms-of-clss (N |∪| U_er). ∀ A1|∈|trail-atms Γ9. A1 ≺t A2|} A
          using step-hyps ‹trail-atms Γ9 = trail-atms Γ10› by metis
      next
        show linorder-cls.is-least-in-fset
          {|C |∈| iefac F |'| (N |∪| U_er). clause-could-propagate Γ9 C (Pos A)|} C
          using step-hyps ‹clause-could-propagate Γ9 = clause-could-propagate Γ10›
by metis
      next
        show Γ9' = (Pos A, Some (efac C)) # Γ9
          using Γ9'-def .
      next
        show F' = (if ord-res.is-strictly-maximal-lit (Pos A) C then F else finsert
C F)
          using step-hyps by argo
      qed

      thus (constant-context ord-res-9)++ S9 (N, U_er, F', Γ9') S10'
        unfolding S9-def by (auto intro: constant-context.intros)

      show ord-res-9-matches-ord-res-10 (N, U_er, F', Γ9') S10'
        unfolding ‹S10' = (N, s10')› ‹s10' = (U_er, F', Γ10')›
      proof (rule ord-res-9-matches-ord-res-10.intros)
        show ord-res-8-invars N (U_er, F', Γ9')
```

```
      using invars9 step9 ord-res-9-preserves-invars by metis
    next
      show ord-res-10-invars N (U_er, F', Γ_10')
       using invars10 step10 ord-res-10-preserves-invars ‹s10' = (U_er, F', Γ_10')›
by metis
    next
      show list-all2 (λx y. fst x = fst y) Γ_9' Γ_10'
       unfolding ‹Γ_9' = (Pos A, Some (efac C)) # Γ_9› ‹Γ_10' = (Pos A, Some
(efac C)) # Γ_10›
        using ‹list-all2 (λx y. fst x = fst y) Γ_9 Γ_10› by simp
    next
      show list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ_9' Γ_10'
       unfolding ‹Γ_9' = (Pos A, Some (efac C)) # Γ_9› ‹Γ_10' = (Pos A, Some
(efac C)) # Γ_10›
        using ‹list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ_9 Γ_10›
        by simp
    qed
  qed
 next
  case step-hyps: (resolution D A C U_er' Γ_10')

  have ∀ Ln Γ'. Γ_10 = Ln # Γ' ⟶
    (snd Ln ≠ None) = fBex (iefac F |`| (N |∪| U_er)) (trail-false-cls Γ_10) ∧
    (∀ x∈set Γ'. snd x = None)
    using invars10 by (simp add: ord-res-10-invars.simps)

  then obtain Γ_10'' where Γ_10 = (Pos A, Some C) # Γ_10''
    using ‹map-of Γ_10 (Pos A) = Some (Some C)›
    by (metis list.set-cases map-of-SomeD not-Some-eq snd-conv)

  then obtain Γ_9'' where Γ_9 = (Pos A, Some C) # Γ_9''
    using ‹list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ_9 Γ_10›
    by (smt (verit, best) list-all2-Cons2 option.discI snd-conv)

  define Γ_9' where
    Γ_9' = dropWhile (λLn. ∀ K. ord-res.is-maximal-lit K (eres C D) ⟶
      atm-of K ≺_t atm-of (fst Ln)) Γ_9

  show ?thesis
  proof (intro exI conjI)
    have step9: ord-res-9 N (U_er, F, Γ_9) (U_er', F, Γ_9')
    proof (rule ord-res-9.resolution)
      show linorder-cls.is-least-in-fset (ffilter (trail-false-cls Γ_9) (iefac F |`| (N
|∪| U_er))) D
       using step-hyps ‹trail-false-cls Γ_9 = trail-false-cls Γ_10› by argo
    next
      show ord-res.is-maximal-lit (Neg A) D
       using step-hyps by argo
    next
```

show *map-of* Γ_9 (*Pos A*) = *Some* (*Some C*)
unfolding ‹Γ_9 = (*Pos A*, *Some C*) # Γ_9''› **by** *simp*
next
show U_{er}' = *finsert* (*eres C D*) U_{er}
using *step-hyps* **by** *argo*
next
show Γ_9' = *dropWhile* (λ*Ln*. ∀ *K*. *ord-res.is-maximal-lit K* (*eres C D*) ⟶ *atm-of K* ≼$_t$ *atm-of* (*fst Ln*)) Γ_9
using Γ_9'-*def* **.**
qed

thus (*constant-context ord-res-9*)$^{++}$ *S9* (*N*, U_{er}', $\mathcal{F}$, Γ_9')
unfolding *S9-def* **by** (*auto intro*: *constant-context.intros*)

show *ord-res-9-matches-ord-res-10* (*N*, U_{er}', $\mathcal{F}$, Γ_9') *S10'*
unfolding ‹*S10'* = (*N*, *s10'*)› ‹*s10'* = (U_{er}', $\mathcal{F}$, Γ_{10}')›
proof (*rule ord-res-9-matches-ord-res-10.intros*)
show *ord-res-8-invars N* (U_{er}', $\mathcal{F}$, Γ_9')
using *invars9 step9 ord-res-9-preserves-invars* **by** *metis*
next
show *ord-res-10-invars N* (U_{er}', $\mathcal{F}$, Γ_{10}')
using *invars10 step10 ord-res-10-preserves-invars* ‹*s10'* = (U_{er}', $\mathcal{F}$, Γ_{10}')›
by *metis*
next
define *P* :: *'f gterm literal* × *'f gterm literal multiset option* ⇒ *bool* **where**
P ≡ λ*Ln*. ∀ *K*. *ord-res.is-maximal-lit K* (*eres C D*) ⟶ *atm-of K* ≼$_t$ *atm-of* (*fst Ln*)

have *length* (*takeWhile P* Γ_9) = *length* (*takeWhile P* Γ_{10})
using ‹*list-all2* (λ*x y*. *fst x* = *fst y*) Γ_9 Γ_{10}›
proof (*induction* Γ_9 Γ_{10} *rule*: *list.rel-induct*)
case *Nil*
show *?case*
by *simp*
next
case (*Cons x xs y ys*)
then show *?case*
by (*simp add*: *P-def*)
qed

moreover have Γ_9 = *takeWhile P* Γ_9 @ Γ_9'
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* ‹Γ_9' = *dropWhile* - Γ_9› **by** *simp*

moreover have Γ_{10} = *takeWhile P* Γ_{10} @ Γ_{10}'
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* ‹Γ_{10}' = *dropWhile* - Γ_{10}› **by** *simp*

ultimately have ⋀*Q*. *list-all2 Q* Γ_9 Γ_{10} ⟷

```
      (list-all2 Q (takeWhile P Γ9) (takeWhile P Γ10) ∧ list-all2 Q Γ9' Γ10')
      using list-all2-append by metis

    thus
      list-all2 (λx y. fst x = fst y) Γ9' Γ10'
      list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9' Γ10'
      unfolding atomize-conj
      using ‹list-all2 (λx y. fst x = fst y) Γ9 Γ10›
      using ‹list-all2 (λx y. snd y ≠ None ⟶ x = y) Γ9 Γ10›
      by (simp only:)
  qed
 qed
qed
qed

theorem bisimulation-ord-res-9-ord-res-10:
 defines match ≡ λ-. ord-res-9-matches-ord-res-10
 shows ∃(MATCH :: nat × nat ⇒ 'f ord-res-8-state ⇒ 'f ord-res-9-state ⇒ bool)
ℛf ℛb.
   bisimulation
     (constant-context ord-res-9) ord-res-8-final
     (constant-context ord-res-10) ord-res-8-final
     MATCH ℛf ℛb
proof (rule ex-bisimulation-from-backward-simulation)
 show right-unique (constant-context ord-res-9)
   using right-unique-constant-context right-unique-ord-res-9 by metis
next
 show right-unique (constant-context ord-res-10)
   using right-unique-constant-context right-unique-ord-res-10 by metis
next
 show ∀S. ord-res-8-final S ⟶ (∄S'. constant-context ord-res-9 S S')
   by (metis finished-def ord-res-9-semantics.final-finished)
next
 show ∀S. ord-res-8-final S ⟶ (∄S'. constant-context ord-res-10 S S')
   by (metis finished-def ord-res-10-semantics.final-finished)
next
 show ∀i S9 S10. match i S9 S10 ⟶ ord-res-8-final S9 ⟷ ord-res-8-final S10
   unfolding match-def
   using ord-res-9-final-iff-ord-res-10-final by metis
next
 show ∀i S9 S10. match i S9 S10 ⟶
      safe-state (constant-context ord-res-9) ord-res-8-final S9 ∧
      safe-state (constant-context ord-res-10) ord-res-8-final S10
 proof (intro allI impI conjI)
   fix i S9 S10
   assume match: match i S9 S10
   show safe-state (constant-context ord-res-9) ord-res-8-final S9
     using match[unfolded match-def]
     using ord-res-9-safe-state-if-invars
```

using *ord-res-9-matches-ord-res-10.simps* **by** *auto*

show *safe-state* (*constant-context ord-res-10*) *ord-res-8-final S10*
using *match*[*unfolded match-def*]
using *ord-res-10-safe-state-if-invars*
using *ord-res-9-matches-ord-res-10.simps* **by** *auto*
qed
next
show *wfp* (λ- -. *False*)
by *simp*
next
show ∀ *i S9 S10 S10'*. *match i S9 S10* ⟶ *constant-context ord-res-10 S10 S10'* ⟶
(∃ *i' S9'*. (*constant-context ord-res-9*)$^{++}$ *S9 S9'* ∧ *match i' S9' S10'*) ∨
(∃ *i'*. *match i' S9 S10'* ∧ *False*)
unfolding *match-def*
using *backward-simulation-between-9-and-10* **by** *metis*
qed

end

37 ORD-RES-11 (SCL strategy)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

inductive *ord-res-10-matches-ord-res-11* :: *'f ord-res-10-state* ⇒ *'f ord-res-11-state* ⇒ *bool* **where**
ord-res-10-invars N (U_{er10}, $\mathcal{F}$, Γ) ⟹
ord-res-11-invars N (U_{er11}, $\mathcal{F}$, Γ, $\mathcal{C}$) ⟹
U_{er11} = U_{er10} − {|{#}|} ⟹
if {#} |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er10}) *then* Γ = [] ∧ $\mathcal{C}$ = *Some* {#} *else* $\mathcal{C}$ = *None* ⟹
ord-res-10-matches-ord-res-11 (*N*, U_{er10}, $\mathcal{F}$, Γ) (*N*, U_{er11}, $\mathcal{F}$, Γ, $\mathcal{C}$)

lemma *ord-res-10-final-iff-ord-res-11-final*:
fixes *S10 S11*
assumes *match*: *ord-res-10-matches-ord-res-11 S10 S11*
shows *ord-res-8-final S10* ⟷ *ord-res-11-final S11*
using *match*
proof (*cases S10 S11 rule*: *ord-res-10-matches-ord-res-11.cases*)
case *match-hyps*: (*1 N* U_{er10} $\mathcal{F}$ Γ U_{er11} $\mathcal{C}$)
show *?thesis*
proof (*rule iffI*)
assume *ord-res-8-final S10*
thus *ord-res-11-final S11*
unfolding ‹*S10* = -›
proof (*cases* (*N*, U_{er10}, $\mathcal{F}$, Γ) *rule*: *ord-res-8-final.cases*)
case *model-found*
hence {#} |∉| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er10})

```
      using trail-false-cls-mempty by blast
    hence C = None
      using match-hyps by argo
    moreover have U_er11 = U_er10
      using match-hyps
       by (metis ‹{#} |∉| iefac F |‘| (N |∪| U_er10)› fimage-eqI finsert-fminus1
finsert-iff
          fminus-finsert-absorb funionI2 iefac-mempty)
    ultimately show ?thesis
      unfolding ‹S11 = -›
      using model-found
      using ord-res-11-final.model-found
      by metis
  next
    case contradiction-found
    hence Γ = [] ∧ C = Some {#}
      using match-hyps by argo
    thus ?thesis
      unfolding ‹S11 = -›
      using ord-res-11-final.contradiction-found by metis
  qed
 next
  assume ord-res-11-final S11
  thus ord-res-8-final S10
    unfolding ‹S11 = -›
  proof (cases (N, U_er11, F, Γ, C) rule: ord-res-11-final.cases)
    case model-found
    show ?thesis
      unfolding ‹S10 = -›
    proof (rule ord-res-8-final.model-found)
      show ¬ (∃ A |∈| atms-of-clss (N |∪| U_er10). A |∉| trail-atms Γ)
       by (metis (no-types, lifting) fimage-iff fminus-finsert-absorb fminus-idemp
funionCI
           iefac-mempty local.model-found(1,2) match-hyps(5,6) option.simps(3))
    next
      show ¬ (∃ C |∈| iefac F |‘| (N |∪| U_er10). trail-false-cls Γ C)
     by (metis finsertCI finsert-fminus1 fminus-finsert-absorb funionI2 iefac-mempty
         local.model-found(1,3) match-hyps(5,6) option.simps(3) rev-fimage-eqI)
    qed
  next
    case contradiction-found
    show ?thesis
      unfolding ‹S10 = -›
    proof (rule ord-res-8-final.contradiction-found)
      show {#} |∈| iefac F |‘| (N |∪| U_er10)
        using match-hyps contradiction-found
        by auto
    qed
  qed
```

```
  qed
qed

lemma forward-simulation-between-10-and-11:
  fixes S10 S11 S10'
  assumes
    match: ord-res-10-matches-ord-res-11 S10 S11 and
    step: constant-context ord-res-10 S10 S10'
  shows ∃ S11'. (constant-context ord-res-11)^++ S11 S11' ∧ ord-res-10-matches-ord-res-11
S10' S11'
  using match
proof (cases S10 S11 rule: ord-res-10-matches-ord-res-11.cases)
  case match-hyps: (1 N U_er10 F Γ U_er11 C)

  note S10-def = ‹S10 = (N, U_er10, F, Γ)›
  note S11-def = ‹S11 = (N, U_er11, F, Γ, C)›
  note invars10 = ‹ord-res-10-invars N (U_er10, F, Γ)›
  note invars11 = ‹ord-res-11-invars N (U_er11, F, Γ, C)›

  have mempty-not-in-if-no-false-cls: {#} |∉| iefac F |`| (N |∪| U_er10)
    if ¬ fBex (iefac F |`| (N |∪| U_er10)) (trail-false-cls Γ)
    using that by force

  have C-eq-None-if-no-false-cls: C = None
    if ¬ fBex (iefac F |`| (N |∪| U_er10)) (trail-false-cls Γ)
    using match-hyps mempty-not-in-if-no-false-cls[OF that] by argo

  have mempty-not-in-if: {#} |∉| N |∪| U_er10
    if ¬ fBex (iefac F |`| (N |∪| U_er10)) (trail-false-cls Γ)
    using that
   by (metis (no-types, opaque-lifting) fimageI iefac-mempty trail-false-cls-mempty)

  have U_er11-eq-U_er10-if: U_er11 = U_er10
    if ¬ fBex (iefac F |`| (N |∪| U_er10)) (trail-false-cls Γ)
    using mempty-not-in-if[OF that] ‹U_er11 = U_er10 |−| {|{#}|}›
    by (metis (no-types, opaque-lifting) finsertI1 finsert-ident fminusD2 funionCI
        funion-fempty-right funion-finsert-right funion-fminus-cancel2)

  obtain s10' where S10' = (N, s10') and step10: ord-res-10 N (U_er10, F, Γ)
s10'
    using step unfolding S10-def by (auto elim: constant-context.cases)

  show ?thesis
    using step10
  proof (cases N (U_er10, F, Γ) s10' rule: ord-res-10.cases)
    case step-hyps: (decide-neg A Γ')

    have C = None
      using step-hyps C-eq-None-if-no-false-cls by argo
```

```
  have {#} |∉| N |∪| U_er10
    using step-hyps mempty-not-in-if by argo

  have U_er11 = U_er10
    using step-hyps U_er11-eq-U_er10-if by argo

  show ?thesis
  proof (intro exI conjI)
    have step11: ord-res-11 N (U_er11, F, Γ, None) (U_er11, F, Γ', None)
    proof (rule ord-res-11.decide-neg)
      show ¬ (∃ C|∈|iefac F |'| (N |∪| U_er11). trail-false-cls Γ C)
        using step-hyps unfolding ‹U_er11 = U_er10› by argo
    next
      show linorder-trm.is-least-in-fset
        {|A_2 |∈| atms-of-clss (N |∪| U_er11). ∀ A_1|∈|trail-atms Γ. A_1 ≺_t A_2|} A
        using step-hyps unfolding ‹U_er11 = U_er10› by argo
    next
      show ¬ (∃ C|∈|iefac F |'| (N |∪| U_er11). clause-could-propagate Γ C (Pos
A))
        using step-hyps unfolding ‹U_er11 = U_er10› by argo
    next
      show Γ' = (Neg A, None) # Γ
        using step-hyps by argo
    qed

    thus (constant-context ord-res-11)^++ S11 (N, U_er11, F, Γ', None)
      unfolding S11-def ‹C = None› by (auto intro: constant-context.intros)

    show ord-res-10-matches-ord-res-11 S10' (N, U_er11, F, Γ', None)
      unfolding ‹S10' = (N, s10')› ‹s10' = -›
    proof (rule ord-res-10-matches-ord-res-11.intros)
      show ord-res-10-invars N (U_er10, F, Γ')
        using step10 ‹s10' = -› invars10 ord-res-10-preserves-invars by metis
    next
      show ord-res-11-invars N (U_er11, F, Γ', None)
        using step11 invars11 ‹C = None› ord-res-11-preserves-invars by metis
    next
      show U_er11 = U_er10 |-| {|{#}|}
        unfolding ‹U_er11 = U_er10›
        using ‹{#} |∉| N |∪| U_er10› by simp
    next
      have {#} |∉| iefac F |'| (N |∪| U_er10)
        using ‹{#} |∉| N |∪| U_er10› by (simp add: iefac-def)
     thus if {#} |∈| iefac F |'| (N |∪| U_er10) then Γ' = [] ∧ None = Some {#}
else None = None
        by argo
    qed
  qed
```

next
case *step-hyps*: (*decide-pos A C* Γ' $\mathcal{F}'$)

have $\mathcal{C}$ = *None*
using *step-hyps* $\mathcal{C}$*-eq-None-if-no-false-cls* **by** *argo*

have {#} |∉| *N* |∪| U_{er10}
using *step-hyps mempty-not-in-if* **by** *argo*

have U_{er11} = U_{er10}
using *step-hyps* U_{er11}*-eq-*U_{er10}*-if* **by** *argo*

show *?thesis*
proof (*intro exI conjI*)
have *step11*: *ord-res-11 N* (U_{er11}, $\mathcal{F}$, Γ, *None*) (U_{er11}, $\mathcal{F}'$, Γ', *None*)
proof (*rule ord-res-11.decide-pos*)
show ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er11}). *trail-false-cls* Γ *C*)
using *step-hyps* **unfolding** ‹U_{er11} = U_{er10}› **by** *argo*
next
show *linorder-trm.is-least-in-fset*
{|*A*$_2$ |∈| *atms-of-clss* (*N* |∪| U_{er11}). ∀ *A*$_1$|∈|*trail-atms* Γ. *A*$_1$ ≺$_t$ *A*$_2$|} *A*
using *step-hyps* **unfolding** ‹U_{er11} = U_{er10}› **by** *argo*
next
show *linorder-cls.is-least-in-fset*
{|*C* |∈| *iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er11}). *clause-could-propagate* Γ *C* (*Pos A*)|} *C*
using *step-hyps* **unfolding** ‹U_{er11} = U_{er10}› **by** *argo*
next
show Γ' = (*Pos A*, *None*) # Γ
using *step-hyps* **by** *argo*
next
show ¬ (∃ *C*|∈|*iefac* $\mathcal{F}$ |‘| (*N* |∪| U_{er11}). *trail-false-cls* Γ' *C*)
using *step-hyps* **unfolding** ‹U_{er11} = U_{er10}› **by** *argo*
next
show $\mathcal{F}'$ = (*if ord-res.is-strictly-maximal-lit* (*Pos A*) *C then* $\mathcal{F}$ *else finsert*
C $\mathcal{F}$)
using *step-hyps* **by** *argo*
qed

thus (*constant-context ord-res-11*)$^{++}$ *S11* (*N*, U_{er11}, $\mathcal{F}'$, Γ', *None*)
unfolding *S11-def* ‹$\mathcal{C}$ = *None*› **by** (*auto intro*: *constant-context.intros*)

show *ord-res-10-matches-ord-res-11 S10'* (*N*, U_{er11}, $\mathcal{F}'$, Γ', *None*)
unfolding ‹*S10'* = (*N*, *s10'*)› ‹*s10'* = -›
proof (*rule ord-res-10-matches-ord-res-11.intros*)
show *ord-res-10-invars N* (U_{er10}, $\mathcal{F}'$, Γ')
using *step10* ‹*s10'* = -› *invars10 ord-res-10-preserves-invars* **by** *metis*
next
show *ord-res-11-invars N* (U_{er11}, $\mathcal{F}'$, Γ', *None*)
using *step11 invars11* ‹$\mathcal{C}$ = *None*› *ord-res-11-preserves-invars* **by** *metis*

next
show $U_{er11} = U_{er10} \mid - \mid \{\mid\{\#\}\mid\}$
unfolding ‹$U_{er11} = U_{er10}$›
using ‹$\{\#\} \mid\notin\mid N \mid\cup\mid U_{er10}$› **by** *simp*
next
have $\{\#\} \mid\notin\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er10})$
using ‹$\{\#\} \mid\notin\mid N \mid\cup\mid U_{er10}$› **by** (*simp add*: *iefac-def*)
thus *if* $\{\#\} \mid\in\mid$ *iefac* $\mathcal{F}' \mid`\mid (N \mid\cup\mid U_{er10})$ *then* $\Gamma' = [] \wedge$ *None* = *Some* $\{\#\}$ *else None* = *None*
by *argo*
qed
qed
next
case *step-hyps*: (*propagate A C* Γ' $\mathcal{F}'$)

have $\mathcal{C}$ = *None*
using *step-hyps* $\mathcal{C}$*-eq-None-if-no-false-cls* **by** *argo*

have $\{\#\} \mid\notin\mid N \mid\cup\mid U_{er10}$
using *step-hyps mempty-not-in-if* **by** *argo*

have $U_{er11} = U_{er10}$
using *step-hyps* U_{er11}*-eq-*U_{er10}*-if* **by** *argo*

show *?thesis*
proof (*intro exI conjI*)
have *step11*: *ord-res-11 N* (U_{er11}, $\mathcal{F}$, Γ, *None*) (U_{er11}, $\mathcal{F}'$, Γ', *None*)
proof (*rule ord-res-11.propagate*)
show $\neg$ ($\exists$ $C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er11})$. *trail-false-cls* Γ C)
using *step-hyps* **unfolding** ‹$U_{er11} = U_{er10}$› **by** *argo*
next
show *linorder-trm.is-least-in-fset*
$\{\mid A_2 \mid\in\mid$ *atms-of-clss* ($N \mid\cup\mid U_{er11}$). $\forall A_1 \mid\in\mid$*trail-atms* Γ. $A_1 \prec_t A_2 \mid\}$ A
using *step-hyps* **unfolding** ‹$U_{er11} = U_{er10}$› **by** *argo*
next
show *linorder-cls.is-least-in-fset*
$\{\mid C \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er11})$. *clause-could-propagate* Γ C (*Pos A*)$\mid\}$ C
using *step-hyps* **unfolding** ‹$U_{er11} = U_{er10}$› **by** *argo*
next
show Γ' = (*Pos A*, *Some* (*efac C*)) # Γ
using *step-hyps* **by** *argo*
next
show $\exists$ $C \mid\in\mid$*iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er11})$. *trail-false-cls* Γ' C
using *step-hyps* **unfolding** ‹$U_{er11} = U_{er10}$› **by** *argo*
next
show $\mathcal{F}'$ = (*if ord-res.is-strictly-maximal-lit* (*Pos A*) *C then* $\mathcal{F}$ *else finsert C* $\mathcal{F}$)
using *step-hyps* **by** *argo*
qed

thus (*constant-context ord-res-11*)$^{++}$ *S11* (N, U_{er11}, $\mathcal{F}'$, Γ', *None*)
unfolding *S11-def* ‹$\mathcal{C}$ = *None*› **by** (*auto intro*: *constant-context.intros*)

show *ord-res-10-matches-ord-res-11 S10'* (N, U_{er11}, $\mathcal{F}'$, Γ', *None*)
unfolding ‹*S10'* = (N, *s10'*)› ‹*s10'* = -›
proof (*rule ord-res-10-matches-ord-res-11.intros*)
show *ord-res-10-invars N* (U_{er10}, $\mathcal{F}'$, Γ')
using *step10* ‹*s10'* = -› *invars10 ord-res-10-preserves-invars* **by** *metis*
next
show *ord-res-11-invars N* (U_{er11}, $\mathcal{F}'$, Γ', *None*)
using *step11 invars11* ‹$\mathcal{C}$ = *None*› *ord-res-11-preserves-invars* **by** *metis*
next
show U_{er11} = U_{er10} |−| {|{#}|}
unfolding ‹U_{er11} = U_{er10}›
using ‹{#} |∉| N |∪| U_{er10}› **by** *simp*
next
have {#} |∉| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er10})
using ‹{#} |∉| N |∪| U_{er10}› **by** (*simp add*: *iefac-def*)
thus *if* {#} |∈| *iefac* $\mathcal{F}'$ |‘| (N |∪| U_{er10}) *then* Γ' = [] ∧ *None* = *Some* {#} *else None* = *None*
by *argo*
qed
qed
next
case *step-hyps*: (*resolution D A C* U_{er10}' Γ')

note *D-max-lit* = ‹*ord-res.is-maximal-lit* (*Neg A*) *D*›

have {#} |∉| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er10})
using ‹*linorder-cls.is-least-in-fset - D*› ‹*linorder-lit.is-maximal-in-mset D* -›
unfolding *linorder-cls.is-least-in-ffilter-iff linorder-lit.is-maximal-in-mset-iff*
by (*metis* (*no-types*, *lifting*) *empty-iff linorder-cls.leD mempty-lesseq-cls set-mset-empty*
trail-false-cls-mempty)

have $\mathcal{C}$ = *None*
using *match-hyps* ‹{#} |∉| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er10})› **by** *argo*

have U_{er11} = U_{er10}
using *match-hyps* ‹{#} |∉| *iefac* $\mathcal{F}$ |‘| (N |∪| U_{er10})› **by** *force*

have *step11-conf*: *ord-res-11 N* (U_{er11}, $\mathcal{F}$, Γ, *None*) (U_{er11}, $\mathcal{F}$, Γ, *Some D*)
proof (*rule ord-res-11.conflict*)
show *linorder-cls.is-least-in-fset*
(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F}$ |‘| (N |∪| U_{er11}))) *D*
using *step-hyps* **unfolding** ‹U_{er11} = U_{er10}› **by** *argo*
qed

have Γ-*spec*: $\forall\, Ln\ \Gamma'.\ \Gamma = Ln\ \#\ \Gamma' \longrightarrow$
(*snd Ln* $\neq$ *None*) = *fBex* (*iefac* $\mathcal{F}$ |‘| (*N* |$\cup$| U_{er10})) (*trail-false-cls* Γ) $\wedge$
($\forall\, x{\in}$*set* Γ'. *snd x* = *None*)
using *invars10* **by** (*simp add*: *ord-res-10-invars.simps*)

then obtain Γ''' **where** Γ = (*Pos A*, *Some C*) # Γ'''
using ‹*map-of* Γ (*Pos A*) = *Some* (*Some C*)›
by (*metis list.set-cases map-of-SomeD not-Some-eq snd-conv*)

have *C-max-lit*: *linorder-lit.is-greatest-in-mset C* (*Pos A*)
using *invars10* **by** (*simp add*: *ord-res-10-invars.simps* ‹Γ = (*Pos A*, *Some C*) # Γ'''›)

have *ord-res.ground-resolution D C* ((*D* − {#*Neg A*#}) + (*C* − {#*Pos A*#}))
proof (*rule ord-res.ground-resolutionI*)
show *D* = *add-mset* (*Neg A*) (*D* − {#*Neg A*#})
using *D-max-lit* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *simp*
next
show *C* = *add-mset* (*Pos A*) (*C* − {#*Pos A*#})
using *C-max-lit* **unfolding** *linorder-lit.is-greatest-in-mset-iff* **by** *simp*
next
show *C* $\prec_c$ *D*
using *C-max-lit D-max-lit*
by (*simp add*: *clause-lt-clause-if-max-lit-comp*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one)
next
show {#} = {#} $\wedge$ *ord-res.is-maximal-lit* (*Neg A*) *D* $\vee$ *Neg A* $\in\#$ {#}
using *D-max-lit* **by** *argo*
next
show {#} = {#}
by *argo*
next
show *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
using *C-max-lit* **.**
next
show *remove1-mset* (*Neg A*) *D* + *remove1-mset* (*Pos A*) *C* = (*D* − {#*Neg A*#}) + (*C* − {#*Pos A*#})
..
qed

hence *eres C D* ≠ *D*
unfolding *eres-ident-iff not-not ground-resolution-def* **by** *metis*

have *D-false*: *trail-false-cls* Γ *D*
using *step-hyps* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *argo*

have *clause-could-propagate* Γ''' *C* (*Pos A*)
using *invars10* ‹Γ = (*Pos A*, *Some C*) # Γ'''› **by** (*simp add*: *ord-res-10-invars.simps*)

hence *trail-false-cls* Γ''' $\{\#L \in\# C.\ L \neq Pos\ A\#\}$
unfolding *clause-could-propagate-def* **by** *argo*

hence *C-almost-false*: *trail-false-cls* Γ $\{\#L \in\# C.\ L \neq Pos\ A\#\}$
unfolding $\langle\Gamma = (Pos\ A, Some\ C) \#\ \Gamma'''\rangle$
by (*meson suffix-ConsD suffix-order.dual-order.refl trail-false-cls-if-trail-false-suffix*)

have *eres-false*: *trail-false-cls* Γ (*eres C D*)
unfolding *trail-false-cls-def*
proof (*intro ballI*)
fix K :: *'f gliteral*
assume $K \in\#$ *eres C D*
hence $K \in\# C \wedge K \neq Pos\ A \vee K \in\# D$
using *strong-lit-in-one-of-resolvents-if-in-eres*[*OF D-max-lit*] **by** *simp*
thus *trail-false-lit* Γ K
proof (*elim disjE conjE*)
assume $K \in\# C$ **and** $K \neq Pos\ A$
thus *trail-false-lit* Γ K
using *C-almost-false* **unfolding** *trail-false-cls-def* **by** *simp*
next
assume $K \in\# D$
thus *trail-false-lit* Γ K
using *D-false* **unfolding** *trail-false-cls-def* **by** *simp*
qed
qed

have $\forall Ln \in set\ \Gamma.\ \forall C.\ snd\ Ln = Some\ C \longrightarrow$ *ord-res.is-strictly-maximal-lit* (*fst Ln*) *C*
using *invars10* **by** (*simp add*: *ord-res-10-invars.simps*)

hence *C-max-lit*: *ord-res.is-strictly-maximal-lit* (*Pos A*) *C*
unfolding $\langle\Gamma = (Pos\ A, Some\ C) \#\ \Gamma'''\rangle$ **by** *simp*

have *steps11-reso*: (*ord-res-11 N*)** (U_{er11}, $\mathcal{F}$, Γ, *Some D*) (U_{er11}, $\mathcal{F}$, Γ, *Some* (*eres C D*))
unfolding $\langle\Gamma = (Pos\ A, Some\ C) \#\ \Gamma'''\rangle$
using *C-max-lit rtrancl-ord-res-11-all-resolution-steps* **by** *metis*

define *strict-P* :: *'f gterm literal* $\times$ *'f gterm literal multiset option* $\Rightarrow$ *bool* **where**
strict-P $\equiv \lambda Ln.\ \forall K.$ *ord-res.is-maximal-lit K* (*eres C D*) $\longrightarrow$ *atm-of K* $\prec_t$ *atm-of* (*fst Ln*)

have $\forall K \in\# eres\ C\ D.\ -K \in fst\ `\ set\ \Gamma$
using *eres-false* **unfolding** *trail-false-cls-def trail-false-lit-def* **.**

moreover have Γ-*sorted*: *sorted-wrt* ($\lambda x\ y.$ *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ
using *invars10* **by** (*simp add*: *ord-res-10-invars.simps*)

ultimately have *dropWhile strict-P* Γ = *dropWhile* ($\lambda Ln.\ -$ *fst Ln* $\notin\#$ *eres*

```
C D) Γ
  proof (induction Γ)
    case Nil
    show ?case by simp
  next
    case (Cons Ln Γ)

    show ?case
    proof (cases eres C D = {#})
      case True
      thus ?thesis
        unfolding strict-P-def linorder-lit.is-maximal-in-mset-iff by simp
    next
      case False

      then obtain L where eres-max-lit: ord-res.is-maximal-lit L (eres C D)
        using linorder-lit.ex-maximal-in-mset by metis

      hence strict-P-Ln-iff: strict-P Ln ⟷ atm-of L ≺t atm-of (fst Ln)
        unfolding strict-P-def
        by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')

      show ?thesis
      proof (cases atm-of L ≺t atm-of (fst Ln))
        case True

        moreover have − fst Ln ∉# eres C D
          using True
       by (smt (verit, best) atm-of-uminus eres-max-lit linorder-lit.dual-order.strict-trans
          linorder-lit.is-maximal-in-mset-iff linorder-lit.neq-iff linorder-trm.order.irrefl
              literal.exhaust-sel ord-res.less-lit-simps(4))

         moreover have dropWhile strict-P Γ = dropWhile (λLn. − fst Ln ∉#
eres C D) Γ
        proof (rule Cons.IH)
          show ∀K∈#eres C D. − K ∈ fst ‘ set Γ
            using Cons.prems True ‹− fst Ln ∉# eres C D›
            using image-iff by fastforce
        next
          show sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) Γ
            using Cons.prems by simp
        qed

        ultimately show ?thesis
          unfolding dropWhile.simps
          unfolding strict-P-Ln-iff
          by simp
      next
        case False
```

```
        hence atm-of (fst Ln) ⪯t atm-of L
          by order

        hence atm-of (fst Ln) = atm-of L
          using Cons.prems
            using atm-of-eq-atm-of eres-max-lit linorder-lit.is-maximal-in-mset-iff
by fastforce

        hence L = fst Ln ∨ L = − fst Ln
          by (metis atm-of-eq-atm-of)

        moreover have − fst Ln ∉ fst ‘ set Γ
          using ‹sorted-wrt (λx y. atm-of (fst y) ≺t atm-of (fst x)) (Ln # Γ)›
          by fastforce

        moreover have − L ∈ fst ‘ (set (Ln # Γ))
          using Cons.prems(1) eres-max-lit linorder-lit.is-maximal-in-mset-iff by
blast

        ultimately have − fst Ln ∈# eres C D
          using eres-max-lit linorder-lit.is-maximal-in-mset-iff by auto

        then show ?thesis
          unfolding dropWhile.simps
          unfolding strict-P-Ln-iff
          using False
          by argo
      qed
    qed
  qed

  hence steps11-skip: (ord-res-11 N)**
    (U_er11, F, Γ, Some (eres C D))
    (U_er11, F, dropWhile strict-P Γ, Some (eres C D))
    using rtrancl-ord-res-11-all-skip-steps by metis

  have most-steps11: (ord-res-11 N)++ (U_er11, F, Γ, None)
   (U_er11, F, dropWhile strict-P Γ, Some (eres C D))
    using step11-conf steps11-reso steps11-skip by simp

  show ?thesis
  proof (cases eres C D = {#})
    case True

    hence dropWhile strict-P Γ = []
      unfolding strict-P-def ‹eres C D = {#}›
      unfolding linorder-lit.is-maximal-in-mset-iff
      by simp
```

have $\Gamma' = []$
unfolding ‹$\Gamma' =$ *dropWhile* - Γ› ‹*eres C D* $= \{\#\}$›
unfolding *linorder-lit.is-maximal-in-mset-iff*
by *simp*

show *?thesis*
proof (*intro exI conjI*)
show (*constant-context ord-res-11*)$^{++}$ *S11* (N, U_{er11}, $\mathcal{F}$, [], *Some* $\{\#\}$)
unfolding *S11-def* ‹$\mathcal{C} =$ *None*›
using *most-steps11*[*unfolded* ‹*dropWhile strict-P* $\Gamma = []$› ‹*eres C D* $= \{\#\}$›]
using *tranclp-constant-context* **by** *metis*

show *ord-res-10-matches-ord-res-11 S10'* (N, U_{er11}, $\mathcal{F}$, [], *Some* $\{\#\}$)
unfolding ‹*S10'* $= (N,$ *s10'*)› ‹*s10'* $=$ -› ‹$\Gamma' = []$›
proof (*rule ord-res-10-matches-ord-res-11.intros*)
show *ord-res-10-invars N* (U_{er10}', $\mathcal{F}$, [])
using *step10* ‹*s10'* $=$ -› ‹$\Gamma' = []$› *invars10 ord-res-10-preserves-invars*
by *metis*
next
show *ord-res-11-invars N* (U_{er11}, $\mathcal{F}$, [], *Some* $\{\#\}$)
using *most-steps11 invars11*
unfolding ‹$\mathcal{C} =$ *None*› ‹*dropWhile strict-P* $\Gamma = []$› ‹*eres C D* $= \{\#\}$›
by (*metis tranclp-ord-res-11-preserves-invars*)
next
show $U_{er11} = U_{er10}' \mid\!-\!\mid \{\mid\{\#\}\mid\}$
unfolding ‹$U_{er11} = U_{er10}$› ‹$U_{er10}' =$ *finsert* (*eres C D*) U_{er10}› ‹*eres C D* $= \{\#\}$›
using ‹$\{\#\} \mid\notin\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er10})$›
by *force*
next
show *if* $\{\#\} \mid\in\mid$ *iefac* $\mathcal{F} \mid`\mid (N \mid\cup\mid U_{er10}')$ *then*
$[] = [] \wedge$ *Some* $\{\#\} =$ *Some* $\{\#\}$ *else Some* $\{\#\} =$ *None*
unfolding ‹$U_{er10}' =$ *finsert* (*eres C D*) U_{er10}› ‹*eres C D* $= \{\#\}$›
by *simp*
qed
qed
next
case *False*

then obtain L **where** *eres-max-lit*: *ord-res.is-maximal-lit L* (*eres C D*)
using *linorder-lit.ex-maximal-in-mset* **by** *metis*

hence $L \in\#$ *eres C D*
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence $L \in\#$ $C \wedge L \neq$ *Pos A* $\vee$ $L \in\#$ $D \wedge L \neq$ *Neg A*
using *stronger-lit-in-one-of-resolvents-if-in-eres* ‹*eres C D* $\neq D$› *D-max-lit*

by (*metis uminus-Neg*)

hence $L \prec_l Neg\ A$
using *C-max-lit D-max-lit*
unfolding *linorder-lit.is-greatest-in-mset-iff linorder-lit.is-maximal-in-mset-iff*
by (*metis C-max-lit D-max-lit* ‹$L \in\#$ *eres C D*› *eres-lt-if ord-res.asymp-less-lit ord-res.less-lit-simps*(3,4) *ord-res.transp-less-lit in-remove1-mset-neq linorder-lit.less-than-maximal-if-multp*$_{HO}$ *linorder-lit.order.not-eq-order-implies-strict literal.disc*(2) *multp-eq-multp*$_{HO}$ *uminus-Neg*)

have *dropWhile strict-P* Γ = *dropWhile* (λ*Ln. atm-of L* $\prec_t$ *atm-of* (*fst Ln*)) Γ
unfolding *strict-P-def*
using *eres-max-lit*
by (*metis* (*no-types*) *Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

also have ... = (−, *L*, *None*) # *dropWhile* (λ*Ln. atm-of L* $\preceq_t$ *atm-of* (*fst Ln*)) Γ
proof −
have − $L \neq Pos\ A$
using ‹$L \prec_l Neg\ A$› **by** (*cases L*) *simp-all*

hence − $L \in$ *fst* ‘ *set* Γ'''
using *eres-false* ‹$L \in\#$ *eres C D*›
unfolding *trail-false-cls-def trail-false-lit-def*
unfolding ‹Γ = (*Pos A*, *Some C*) # Γ'''›
by *auto*

hence (−, *L*, *None*) ∈ *set* Γ'''
using Γ-*spec* **unfolding** ‹Γ = (*Pos A*, *Some C*) # Γ'''› **by** *auto*

then obtain Γ_0 Γ_1 **where** $\Gamma''' = \Gamma_1$ @ (−, *L*, *None*) # Γ_0
by (*meson split-list*)

hence Γ = (*Pos A*, *Some C*) # Γ_1 @ (−, *L*, *None*) # Γ_0
unfolding ‹Γ = (*Pos A*, *Some C*) # Γ'''› **by** (*simp only*:)

have *AAA*: ∀ *Ln* ∈ *set* ((*Pos A*, *Some C*) # Γ_1). *atm-of L* $\prec_t$ *atm-of* (*fst Ln*)
using Γ-*sorted* **unfolding** ‹Γ = (*Pos A*, *Some C*) # Γ_1 @ (−, *L*, *None*) # Γ_0›
by (*simp add*: *sorted-wrt-append*)

hence *BBB*: ∀ *Ln* ∈ *set* ((*Pos A*, *Some C*) # Γ_1 @ [(−, *L*, *None*)]). *atm-of L* $\preceq_t$ *atm-of* (*fst Ln*)
by *simp*

have ∀ *Ln* ∈ *set* Γ_0. *atm-of* (*fst Ln*) $\prec_t$ *atm-of L*
using Γ-*sorted* **unfolding** ‹Γ = (*Pos A*, *Some C*) # Γ_1 @ (−, *L*, *None*)

```
# Γ₀›
        by (simp add: sorted-wrt-append)

      hence CCC: ∀ Ln ∈ set Γ₀. ¬ atm-of L ≼t atm-of (fst Ln)
        using linorder-trm.leD by blast

      have dropWhile (λLn. atm-of L ≺t atm-of (fst Ln)) Γ = (− L, None) # Γ₀
        unfolding ‹Γ = (Pos A, Some C) # Γ₁ @ (− L, None) # Γ₀›
        using dropWhile-append2 AAA by simp

       also have ... = (− L, None) # dropWhile (λLn. atm-of L ≼t atm-of (fst
Ln)) Γ₀
        using CCC by simp

       also have ... = (− L, None) # dropWhile (λLn. atm-of L ≼t atm-of (fst
Ln)) Γ
        unfolding ‹Γ = (Pos A, Some C) # Γ₁ @ (− L, None) # Γ₀›
        using dropWhile-append2 BBB by simp

      finally show ?thesis .
    qed

    also have ... = (− L, None) # Γ'
      unfolding ‹Γ' = dropWhile - Γ›
      using eres-max-lit
      by (metis (no-types) Uniq-D linorder-lit.Uniq-is-maximal-in-mset)

    finally have dropWhile strict-P Γ = (− L, None) # Γ' .

    have step10-back: ord-res-11 N
      (U_er11, F, dropWhile strict-P Γ, Some (eres C D))
      (finsert (eres C D) U_er11, F, Γ', None)
      unfolding ‹dropWhile strict-P Γ = (− L, None) # Γ'›
    proof (rule ord-res-11.backtrack)
      show (− L, None) # Γ' = (− L, None) # Γ' ..
    next
      show − (− L) ∈# eres C D
        unfolding uminus-of-uminus-id
        using eres-max-lit
        unfolding linorder-lit.is-maximal-in-mset-iff by argo
    qed

    hence all-steps11: (ord-res-11 N)⁺⁺ (U_er11, F, Γ, None)
      (finsert (eres C D) U_er11, F, Γ', None)
      using most-steps11 by simp

    show ?thesis
    proof (intro exI conjI)
      show (constant-context ord-res-11)⁺⁺ S11 (N, finsert (eres C D) U_er11,
```

```
$\mathcal{F}$, $\Gamma'$, None)
        unfolding S11-def ‹$\mathcal{C}$ = None›
        using all-steps11 tranclp-constant-context by metis

      have {#} |∉| iefac $\mathcal{F}$ |‘| (N |∪| $U_{er10}'$)
          by (smt (verit, del-insts) False ‹{#} |∉| iefac $\mathcal{F}$ |‘| (N |∪| $U_{er10}$)› fimage.rep-eq
            fimageE fimageI finsertE funion-iff iefac-def mempty-in-image-efac-iff step-hyps(5))

      show ord-res-10-matches-ord-res-11 S10' (N, finsert (eres C D) $U_{er11}$, $\mathcal{F}$, $\Gamma'$, None)
        unfolding ‹S10' = (N, s10')› ‹s10' = -›
      proof (rule ord-res-10-matches-ord-res-11.intros)
        show ord-res-10-invars N ($U_{er10}'$, $\mathcal{F}$, $\Gamma'$)
          using step10 ‹s10' = -› invars10 ord-res-10-preserves-invars by metis
      next
        show ord-res-11-invars N (finsert (eres C D) $U_{er11}$, $\mathcal{F}$, $\Gamma'$, None)
          using all-steps11 invars11
          unfolding ‹$\mathcal{C}$ = None›
          by (metis tranclp-ord-res-11-preserves-invars)
      next
        show finsert (eres C D) $U_{er11}$ = $U_{er10}'$ |−| {|{#}|}
          unfolding ‹$U_{er11}$ = $U_{er10}$› ‹$U_{er10}'$ = finsert (eres C D) $U_{er10}$›
          using ‹{#} |∉| iefac $\mathcal{F}$ |‘| (N |∪| $U_{er10}'$)›
          using False ‹$U_{er11}$ = $U_{er10}$› ‹$U_{er11}$ = $U_{er10}$ |−| {|{#}|}› by auto
      next
        show if {#} |∈| iefac $\mathcal{F}$ |‘| (N |∪| $U_{er10}'$) then
          $\Gamma'$ = [] ∧ None = Some {#} else None = None
          using ‹{#} |∉| iefac $\mathcal{F}$ |‘| (N |∪| $U_{er10}'$)› by simp
      qed
    qed
  qed
qed

theorem bisimulation-ord-res-10-ord-res-11:
  defines match ≡ λ-. ord-res-10-matches-ord-res-11
  shows ∃(MATCH :: nat × nat ⇒ 'f ord-res-10-state ⇒ 'f ord-res-11-state ⇒ bool) $\mathcal{R}_f$ $\mathcal{R}_b$.
    bisimulation
      (constant-context ord-res-10) ord-res-8-final
      (constant-context ord-res-11) ord-res-11-final
      MATCH $\mathcal{R}_f$ $\mathcal{R}_b$
proof (rule ex-bisimulation-from-forward-simulation)
  show right-unique (constant-context ord-res-10)
    using right-unique-constant-context right-unique-ord-res-10 by metis
next
  show right-unique (constant-context ord-res-11)
```

using *right-unique-constant-context right-unique-ord-res-11* **by** *metis*
next
show $\forall S.$ *ord-res-8-final S* $\longrightarrow$ ($\nexists S'.$ *constant-context ord-res-10 S S'*)
by (*metis finished-def ord-res-10-semantics.final-finished*)
next
show $\forall S.$ *ord-res-11-final S* $\longrightarrow$ ($\nexists S'.$ *constant-context ord-res-11 S S'*)
by (*metis finished-def ord-res-11-semantics.final-finished*)
next
show $\forall$ *i S10 S11*. *match i S10 S11* $\longrightarrow$ *ord-res-8-final S10* $\longleftrightarrow$ *ord-res-11-final S11*
unfolding *match-def*
using *ord-res-10-final-iff-ord-res-11-final* **by** *metis*
next
show $\forall$ *i S10 S11*. *match i S10 S11* $\longrightarrow$
safe-state (*constant-context ord-res-10*) *ord-res-8-final S10* $\wedge$
safe-state (*constant-context ord-res-11*) *ord-res-11-final S11*
proof (*intro allI impI conjI*)
fix *i S10 S11*
assume *match*: *match i S10 S11*
show *safe-state* (*constant-context ord-res-10*) *ord-res-8-final S10*
using *match*[*unfolded match-def*]
using *ord-res-10-safe-state-if-invars*
unfolding *ord-res-10-matches-ord-res-11.simps* **by** *auto*

show *safe-state* (*constant-context ord-res-11*) *ord-res-11-final S11*
using *match*[*unfolded match-def*]
using *ord-res-11-safe-state-if-invars*
using *ord-res-10-matches-ord-res-11.simps* **by** *auto*
qed
next
show *wfp* (λ- -. *False*)
by *simp*
next
show $\forall$ *i S10 S11 S10'*. *match i S10 S11* $\longrightarrow$ *constant-context ord-res-10 S10 S10'* $\longrightarrow$
($\exists$ *i' S11'*. (*constant-context ord-res-11*)$^{++}$ *S11 S11'* $\wedge$ *match i' S10' S11'*) $\vee$
($\exists$ *i'*. *match i' S10' S11* $\wedge$ *False*)
unfolding *match-def*
using *forward-simulation-between-10-and-11* **by** *metis*
qed

end

type-synonym *bisim-index-1-2* = *nat* $\times$ *nat*
type-synonym *bisim-index-1-3* = *bisim-index-1-2* $\times$ (*nat* $\times$ *nat*)
type-synonym *bisim-index-1-4* = *bisim-index-1-3* $\times$ (*nat* $\times$ *nat*)
type-synonym *bisim-index-1-5* = *bisim-index-1-4* $\times$ (*nat* $\times$ *nat*)
type-synonym *bisim-index-1-6* = *bisim-index-1-5* $\times$ (*nat* $\times$ *nat*)
type-synonym *bisim-index-1-7* = *bisim-index-1-6* $\times$ (*nat* $\times$ *nat*)

type-synonym *bisim-index-1-8* = *bisim-index-1-7* × (*nat* × *nat*)
type-synonym *bisim-index-1-9* = *bisim-index-1-8* × (*nat* × *nat*)
type-synonym *bisim-index-1-10* = *bisim-index-1-9* × (*nat* × *nat*)
type-synonym *bisim-index-1-11* = *bisim-index-1-10* × (*nat* × *nat*)

context *simulation-SCLFOL-ground-ordered-resolution* **begin**

theorem *bisimulation-ord-res-1-ord-res-11*:
obtains
MATCH :: *bisim-index-1-11* ⇒ *'f ord-res-1-state* ⇒ *'f ord-res-11-state* ⇒ *bool*
and
$\mathcal{R}_f$ $\mathcal{R}_b$ :: *bisim-index-1-11* ⇒ *bisim-index-1-11* ⇒ *bool*
where
bisimulation
ord-res-1 ord-res-1-final
(*constant-context ord-res-11*) *ord-res-11-final*
MATCH $\mathcal{R}_f$ $\mathcal{R}_b$
apply *atomize-elim*
using *bisimulation-ord-res-1-ord-res-2*
bisimulation-ord-res-2-ord-res-3
bisimulation-ord-res-3-ord-res-4
bisimulation-ord-res-4-ord-res-5
bisimulation-ord-res-5-ord-res-6
bisimulation-ord-res-6-ord-res-7
bisimulation-ord-res-7-ord-res-8
bisimulation-ord-res-8-ord-res-9
bisimulation-ord-res-9-ord-res-10
bisimulation-ord-res-10-ord-res-11
using *bisimulation-composition* **by** *meson*

theorem
obtains
MATCH :: *bisim-index-1-11* ⇒ *'f ord-res-1-state* ⇒ *'f ord-res-11-state* ⇒ *bool*
and
$\mathcal{R}_f$ $\mathcal{R}_b$ :: *bisim-index-1-11* ⇒ *bisim-index-1-11* ⇒ *bool*
where
bisimulation
ord-res-1 ord-res-1-final
(*constant-context ord-res-11*) *ord-res-11-final*
MATCH $\mathcal{R}_f$ $\mathcal{R}_b$ **and**
$\bigwedge j$ *S1 S11*. *MATCH j S1 S11* ⟹ *ord-res-1-final S1* ⟷ *ord-res-11-final S11*
using *bisimulation-ord-res-1-ord-res-11 bisimulation.agree-on-final*
by (*metis* (*no-types*, *opaque-lifting*))

38 ORD-RES-11 is a regular SCL strategy

definition *gtrailelem-of-trailelem* **where**
gtrailelem-of-trailelem ≡ λ(*L*, *opt*).
(*lit-of-glit L*, *map-option* (λ*C*. (*cls-of-gcls* {#*K* ∈# *C*. *K* ≠ *L*#}, *lit-of-glit L*,

Var)) *opt*)

fun *state-of-gstate* :: - ⇒ ('*f*, '*v*) *SCL-FOL.state* **where**
state-of-gstate (U_G, -, Γ_G, $\mathcal{C}_G$) =
(*let*
Γ = *map gtrailelem-of-trailelem* Γ_G;
U = *cls-of-gcls* |`| U_G;
$\mathcal{C}$ = *map-option* (λC_G. (*cls-of-gcls* C_G, *Var*)) $\mathcal{C}_G$
in (Γ, U, $\mathcal{C}$))

lemma *fst-case-prod-simp*: *fst* (*case p of* (*x*, *y*) ⇒ (*f x*, *g x y*)) = *f* (*fst p*)
by (*cases p*) *simp*

lemma *trail-false-cls-nonground-iff-trail-false-cls-ground*:
fixes Γ_G **and** D_G :: '*f gclause*
fixes Γ :: ('*f*, '*v*) *SCL-FOL.trail* **and** D :: ('*f*, '*v*) *term clause*
defines Γ ≡ *map gtrailelem-of-trailelem* Γ_G **and** D ≡ *cls-of-gcls* D_G
shows *trail-false-cls* Γ D ⟷ *trail-false-cls* Γ_G D_G
proof −
have *trail-false-cls* Γ D ⟷ (∀ L ∈# D. *trail-false-lit* Γ L)
unfolding *trail-false-cls-def* **..**
also have ... ⟷ (∀ L_G ∈# D_G. *trail-false-lit* Γ (*lit-of-glit* L_G))
unfolding *D-def cls-of-gcls-def* **by** *simp*
also have ... ⟷ (∀ L_G ∈# D_G. *trail-false-lit* Γ_G L_G)
proof −
have *trail-false-lit* Γ (*lit-of-glit* L_G) ⟷ *trail-false-lit* Γ_G L_G
for L_G :: '*f gterm literal*
proof −
have *trail-false-lit* Γ (*lit-of-glit* L_G) ⟷ − *lit-of-glit* L_G ∈ *fst* ` *set* Γ
unfolding *trail-false-lit-def* **..**
also have ... ⟷
− (*lit-of-glit* L_G :: ('*f*, '*v*) *term literal*) ∈ *set* (*map* (λ*x*. *lit-of-glit* (*fst x*)) Γ_G)
unfolding Γ*-def image-set list.map-comp*
unfolding *gtrailelem-of-trailelem-def*
unfolding *list.map-comp*
unfolding *comp-def fst-case-prod-simp* **..**
also have ... ⟷ (*lit-of-glit* (− L_G) :: ('*f*, '*v*) *term literal*) ∈ *lit-of-glit* ` *fst* ` *set* Γ_G
by (*cases* L_G) (*auto simp*: *lit-of-glit-def*)
also have ... ⟷ − L_G ∈ *fst* ` *set* Γ_G
using *inj-image-mem-iff inj-lit-of-glit* **by** *metis*
also have ... ⟷ *trail-false-lit* Γ_G L_G
unfolding *trail-false-lit-def* **..**
finally show *trail-false-lit* Γ (*lit-of-glit* L_G) = *trail-false-lit* Γ_G L_G **.**
qed
thus *?thesis* **by** *metis*
qed

also have ... $\longleftrightarrow$ *trail-false-cls* Γ_G D_G
unfolding *trail-false-cls-def* **..**
finally show *?thesis* **.**
qed

theorem *ord-res-11-is-strategy-for-regular-scl*:
fixes
N_G :: *'f gclause fset* **and**
N :: *('f, 'v) term clause fset* **and**
β_G :: *'f gterm* **and**
β :: *('f, 'v) term* **and**
S_G S_G' :: *'f gclause fset* × *'f gclause fset* × *('f gliteral* × *'f gclause option) list* × *'f gclause option* **and**
S S' :: *('f, 'v) SCL-FOL.state*
defines
$N \equiv$ *cls-of-gcls* |‘| N_G **and**
$\beta \equiv$ *term-of-gterm* β_G **and**
$S \equiv$ *state-of-gstate* S_G **and**
$S' \equiv$ *state-of-gstate* S_G'
assumes
ball-le-β_G: $\forall A_G$ |∈| *atms-of-clss* N_G. $A_G \preceq_t \beta_G$ **and**
run: *(ord-res-11* N_G*)*** $(\{|||\}, \{|||\}, [], None)$ S_G **and**
step: *ord-res-11* N_G S_G S_G'
shows
scl-fol.regular-scl N β S S'
proof −
have *ord-res-11-invars* N_G $(\{|||\}, \{|||\}, [], None)$
using *ord-res-11-invars-initial-state* **.**

hence *ord-res-11-invars* N_G S_G
using *run rtranclp-ord-res-11-preserves-invars* **by** *metis*

obtain U_G $\mathcal{F}$ Γ_G $\mathcal{C}_G$ **where** S_G*-def*: $S_G = (U_G, \mathcal{F}, \Gamma_G, \mathcal{C}_G)$
by (*metis surj-pair*)

obtain Γ U $\mathcal{C}$ **where** *S-def*: $S = (\Gamma, U, \mathcal{C})$
by (*metis surj-pair*)

have Γ*-def*: $\Gamma =$ *map gtrailelem-of-trailelem* Γ_G
using *S-def* S_G*-def* ‹$S \equiv$ *state-of-gstate* S_G› **by** *simp*

have *U-def*: $U =$ *cls-of-gcls* |‘| U_G
using *S-def* S_G*-def* ‹$S \equiv$ *state-of-gstate* S_G› **by** *simp*

have $\mathcal{C}$*-def*: $\mathcal{C} =$ *map-option* $(\lambda C_G.$ *(cls-of-gcls* C_G, *Var))* $\mathcal{C}_G$
using *S-def* S_G*-def* ‹$S \equiv$ *state-of-gstate* S_G› **by** *simp*

obtain $\mathcal{F}'$ U_G' :: *'f gclause fset* **and** Γ_G' :: *- list* **and** $\mathcal{C}_G'$ :: *- option* **where**
S_G'*-def*: $S_G' = (U_G', \mathcal{F}', \Gamma_G', \mathcal{C}_G')$

by (*metis surj-pair*)

obtain Γ' :: - *list* **and** U' :: - *fset* **and** $\mathcal{C}'$ :: - *option* **where**
S'*-def*: $S' = (\Gamma',\ U',\ \mathcal{C}')$
by (*metis surj-pair*)

have Γ'*-def*: $\Gamma' =$ *map gtrailelem-of-trailelem* ${\Gamma_G}'$
using S'*-def* ${S_G}'$*-def* ‹$S' \equiv$ *state-of-gstate* ${S_G}'$› **by** *simp*

have U'*-def*: $U' =$ *cls-of-gcls* $|`|$ ${U_G}'$
using S'*-def* ${S_G}'$*-def* ‹$S' \equiv$ *state-of-gstate* ${S_G}'$› **by** *simp*

have $\mathcal{C}'$*-def*: $\mathcal{C}' =$ *map-option* $(\lambda C_G.\ (\textit{cls-of-gcls}\ C_G,\ \textit{Var}))$ ${\mathcal{C}_G}'$
using S'*-def* ${S_G}'$*-def* ‹$S' \equiv$ *state-of-gstate* ${S_G}'$› **by** *simp*

have *atms-of-clss* U_G $|\subseteq|$ *atms-of-clss* N_G
using ‹*ord-res-11-invars* N_G S_G›[*unfolded* S_G*-def*]
unfolding *ord-res-11-invars.simps* **by** *simp*

have *atms-of-clss* $(N_G$ $|\cup|$ $U_G)$ = *atms-of-clss* N_G $|\cup|$ *atms-of-clss* U_G
by (*simp add*: *atms-of-clss-def fimage-funion*)

also have $\ldots$ = *atms-of-clss* N_G
using ‹*atms-of-clss* U_G $|\subseteq|$ *atms-of-clss* N_G› **by** *auto*

finally have *atms-of-clss* $(N_G$ $|\cup|$ $U_G)$ = *atms-of-clss* N_G **.**

have *clauses-in-*$\mathcal{F}$*-have-pos-max-lit*: $\forall\, C |\in| \mathcal{F}.\ \exists\, L.$ *is-pos* $L \wedge$ *ord-res.is-maximal-lit* L C
using ‹*ord-res-11-invars* N_G S_G›[*unfolded* S_G*-def ord-res-11-invars.simps*]
by *simp*

have *nex-conflict-if-nbex-trail-false*:
$\neg$ *fBex* (*iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$) (*trail-false-cls* Γ_G) $\Longrightarrow \neg$ *Ex* (*scl-fol.conflict* N β S)
proof (*elim contrapos-nn exE*)
fix x :: $('f, 'v)$ *state*
assume *scl-fol.conflict* N β S x
hence *fBex* $(N_G$ $|\cup|$ $U_G)$ (*trail-false-cls* Γ_G)
unfolding S*-def*
proof (*cases* N β $(\Gamma,\ U,\ \mathcal{C})$ x *rule*: *scl-fol.conflict.cases*)
case (*conflictI* D γ)

obtain D_G **where** D_G $|\in|$ N_G $|\cup|$ U_G **and** D*-def*: $D =$ *cls-of-gcls* D_G
using ‹D $|\in|$ N $|\cup|$ U›
unfolding N*-def* U*-def* **by** *blast*

moreover have *trail-false-cls* Γ_G D_G
proof $-$

```
      have is-ground-cls D
        using ‹D = cls-of-gcls D_G› by simp
      hence D · γ = D
        by simp
      hence trail-false-cls Γ D
        using conflictI
        unfolding SCL-FOL.trail-false-cls-def trail-false-cls-def
        unfolding SCL-FOL.trail-false-lit-def trail-false-lit-def
        by argo

      thus ?thesis
        unfolding Γ-def D-def
        unfolding trail-false-cls-nonground-iff-trail-false-cls-ground .
    qed
    ultimately show ?thesis by metis
  qed

  thus fBex (iefac F |`| (N_G |∪| U_G)) (trail-false-cls Γ_G)
    unfolding bex-trail-false-cls-simp .
qed

have nex-conflict-if-alread-in-conflict: C_G = Some C_G ⟹ ¬ Ex (scl-fol.conflict
N β S) for C_G
  unfolding S-def C-def by (simp add: scl-fol.conflict.simps)

have nex-conflict-if-no-clause-could-propagate-comp:
  ¬ Ex (scl-fol.conflict N β ((lit-of-glit L_G, None) # Γ, U, C))
  if
    nex-false-clause-wrt-Γ_G: ¬ fBex (iefac F |`| (N_G |∪| U_G)) (trail-false-cls Γ_G)
and
    ball-lt-atm-L_G: ∀ x |∈| trail-atms Γ_G. x ≺_t atm-of L_G and
    nex-clause-that-propagate: ¬ (∃ C |∈| iefac F |`| (N_G |∪| U_G).
      clause-could-propagate Γ_G C (− L_G))
  for L_G
proof (intro notI, elim exE)
  fix S'' :: ('f, 'v) SCL-FOL.state
  assume scl-fol.conflict N β ((lit-of-glit L_G, None) # Γ, U, C) S''
  thus False
proof (cases N β ((lit-of-glit L_G, None) # Γ, U, C) S'' rule: scl-fol.conflict.cases)
    case (conflictI D γ)

    obtain D_G where D_G |∈| N_G |∪| U_G and D-def: D = cls-of-gcls D_G
      using ‹D |∈| N |∪| U› N-def U-def by blast

    have (lit-of-glit L_G :: ('f, 'v) term literal, None) # Γ =
      (map gtrailelem-of-trailelem ((L_G, None) # Γ_G) :: ('f, 'v) SCL-FOL.trail)
      by (simp add: Γ-def gtrailelem-of-trailelem-def)

    moreover have D · γ = cls-of-gcls D_G
```

unfolding *D-def* **by** *simp*

ultimately have *trail-false-cls*
(*map gtrailelem-of-trailelem* ((*L*$_G$, *None*) # Γ$_G$) :: (*'f*, *'v*) *SCL-FOL.trail*)
(*cls-of-gcls D*$_G$)
using ‹*SCL-FOL.trail-false-cls* ((*lit-of-glit L*$_G$, *None*) # Γ) (*D* · γ)›
unfolding *SCL-FOL.trail-false-cls-def trail-false-cls-def*
unfolding *SCL-FOL.trail-false-lit-def trail-false-lit-def*
by *metis*

hence *trail-false-cls* ((*L*$_G$, *None*) # Γ$_G$) *D*$_G$
using *trail-false-cls-nonground-iff-trail-false-cls-ground* **by** *blast*

hence *trail-false-cls* Γ$_G$ {#*K*$_G$ ∈# *D*$_G$. *K*$_G$ ≠ − *L*$_G$#}
unfolding *trail-false-cls-def trail-false-lit-def*
by *auto*

moreover have *ord-res.is-maximal-lit* (− *L*$_G$) *D*$_G$
unfolding *linorder-lit.is-maximal-in-mset-iff*
proof (*intro conjI ballI impI*)
show − *L*$_G$ ∈# *D*$_G$
using ‹*D*$_G$ |∈| *N*$_G$ |∪| *U*$_G$› ‹*trail-false-cls* ((*L*$_G$, *None*) # Γ$_G$) *D*$_G$›
subtrail-falseI
nex-false-clause-wrt-Γ$_G$
unfolding *bex-trail-false-cls-simp*
by *blast*
next
fix *K*$_G$ **assume** *K*$_G$ ∈# *D*$_G$ **and** *K*$_G$ ≠ − *L*$_G$
hence *trail-false-lit* Γ$_G$ *K*$_G$
using ‹*trail-false-cls* Γ$_G$ {#*K*$_G$ ∈# *D*$_G$. *K*$_G$ ≠ − *L*$_G$#}›
unfolding *trail-false-cls-def* **by** *simp*
hence *trail-defined-lit* Γ$_G$ *K*$_G$
by (*simp add*: *trail-defined-lit-iff-true-or-false*)
hence *atm-of K*$_G$ |∈| *trail-atms* Γ$_G$
unfolding *trail-defined-lit-iff-trail-defined-atm* **.**
hence *atm-of K*$_G$ ≺$_t$ *atm-of L*$_G$
using *ball-lt-atm-L*$_G$ **by** *metis*
hence *K*$_G$ ≺$_l$ − *L*$_G$
by (*cases L*$_G$; *cases K*$_G$) *simp-all*
thus ¬ − *L*$_G$ ≺$_l$ *K*$_G$
by *order*
qed

moreover have ¬ *trail-defined-lit* Γ$_G$ (− *L*$_G$)
by (*metis atm-of-uminus linorder-trm.less-irrefl that*(*2*)
trail-defined-lit-iff-trail-defined-atm)

ultimately have *clause-could-propagate* Γ$_G$ *D*$_G$ (− *L*$_G$)
unfolding *clause-could-propagate-def* **by** *argo*

```
    hence ∃ C|∈| N_G |∪| U_G. clause-could-propagate Γ_G C (− L_G)
      using ‹D_G |∈| N_G |∪| U_G› by metis

    hence False
      using nex-clause-that-propagate
      unfolding bex-clause-could-propagate-simp
      by contradiction

    thus ?thesis .
  qed
qed

show ?thesis
  using step unfolding S_G-def S_G'-def
proof (cases N_G (U_G, F, Γ_G, C_G) (U_G', F', Γ_G', C_G') rule: ord-res-11.cases)
  case step-hyps: (decide-neg A_G)

  define A :: ('f, 'v) term where
    A = term-of-gterm A_G

  let ?f = gtrailelem-of-trailelem
  have Γ' = map ?f Γ_G'
    unfolding Γ'-def ..
  also have ... = map ?f ((Neg A_G, None) # Γ_G)
    unfolding ‹Γ_G' = (Neg A_G, None) # Γ_G› ..
  also have ... = ?f (Neg A_G, None) # map ?f Γ_G
    unfolding list.map ..
  also have ... = ?f (Neg A_G, None) # Γ
    unfolding Γ-def ..
  also have ... = (lit-of-glit (Neg A_G), None) # Γ
    unfolding gtrailelem-of-trailelem-def prod.case option.map ..
  also have ... = (Neg (term-of-gterm A_G), None) # Γ
    unfolding lit-of-glit-def literal.map ..
  also have ... = (Neg A, None) # Γ
    unfolding A-def ..
  finally have Γ' = decide-lit (Neg A) # Γ
    unfolding decide-lit-def .

  have U' = U
    unfolding U'-def ‹U_G' = U_G› U-def ..

  have ¬ Ex (scl-fol.conflict N β S)
  using ‹¬ fBex (iefac F |`| (N_G |∪| U_G)) (trail-false-cls Γ_G)› nex-conflict-if-nbex-trail-false
    by metis

  moreover have scl-fol.reasonable-scl N β S S'
    unfolding scl-fol.reasonable-scl-def
  proof (intro conjI impI notI ; (elim exE) ?)
```

have *scl-fol.decide N β S S′*
unfolding *S-def S′-def* ‹*U′ = U*› *C-def C′-def* ‹*C_G = None*› ‹*C_G′ = None*› *option.map*
proof (*rule scl-fol.decideI′*)
show *is-ground-lit* (*Neg A ·l Var*)
by (*simp add*: *A-def*)
next
have ∀ *x* |∈| *trail-atms Γ_G*. *x* ≺_t *A_G*
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence *A_G* |∉| *trail-atms Γ_G*
by *blast*
hence *A_G* ∉ *atm-of ‘ fst ‘ set Γ_G*
unfolding *fset-trail-atms* **.**
hence *term-of-gterm A_G* ∉ *term-of-gterm ‘ atm-of ‘ fst ‘ set Γ_G*
using *inj-image-mem-iff inj-term-of-gterm* **by** *metis*
hence *term-of-gterm A_G* ∉ *set* (*map* (λ*x*. *term-of-gterm* (*atm-of* (*fst x*))) *Γ_G*)
unfolding *image-set list.map-comp comp-def* **.**
hence *A* ∉ *set* (*map* (λ*x*. *atm-of* (*lit-of-glit* (*fst x*))) *Γ_G*)
unfolding *A-def atm-of-lit-of-glit-conv* **.**
hence *A* ∉ *atm-of ‘ fst ‘ set Γ*
unfolding *image-set list.map-comp comp-def Γ-def gtrailelem-of-trailelem-def fst-case-prod-simp* **.**
hence *A* |∉| *trail-atms Γ*
unfolding *fset-trail-atms* **.**
hence ¬ *trail-defined-lit Γ* (*Neg A ·l Var*)
by (*simp add*: *trail-defined-lit-iff-trail-defined-atm*)
thus ¬ *SCL-FOL.trail-defined-lit Γ* (*Neg A ·l Var*)
by (*simp add*: *SCL-FOL.trail-defined-lit-def trail-defined-lit-def*)
next
have *A_G* |∈| *atms-of-clss* (*N_G* |∪| *U_G*)
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *blast*
hence *A_G* |∈| *atms-of-clss N_G*
unfolding ‹*atms-of-clss* (*N_G* |∪| *U_G*) = *atms-of-clss N_G*› **.**
hence *A_G* ≼_t *β_G*
using *ball-le-β_G* **by** *metis*
moreover have *gterm-of-term A* = *A_G*
by (*simp add*: *A-def*)
moreover have *gterm-of-term β* = *β_G*
by (*simp add*: *β-def*)
ultimately have *gterm-of-term A* ≼_t *gterm-of-term β*
by *argo*
thus *less-B*^== (*atm-of* (*Neg A*) ·*a Var*) *β*
using *inj-term-of-gterm*[*THEN injD*]
by (*auto simp*: *less-B-def A-def β-def*)
next
show *Γ′* = *trail-decide Γ* (*Neg A ·l Var*)
using ‹*Γ′ = decide-lit* (*Neg A*) # *Γ*›
unfolding *subst-lit-id-subst* **.**

```
    qed

    thus scl-fol.scl N β S S'
      unfolding scl-fol.scl-def by argo
  next
    fix S'' :: ('f, 'v) SCL-FOL.state
    assume scl-fol.conflict N β S' S''

    moreover have ∄S''. scl-fol.conflict N β S' S''
    proof −
      have ¬ Ex (scl-fol.conflict N β ((lit-of-glit (Neg A_G), None) # Γ, U, C))
      proof (rule nex-conflict-if-no-clause-could-propagate-comp)
        show ¬ fBex (iefac F |¶| (N_G |∪| U_G)) (trail-false-cls Γ_G)
          using step-hyps by argo
      next
        show ∀x |∈| trail-atms Γ_G. x ≺_t atm-of (Neg A_G)
          unfolding literal.sel
          using step-hyps linorder-trm.is-least-in-fset-iff by simp
      next
        show ¬ (∃C |∈| iefac F |¶| (N_G |∪| U_G). clause-could-propagate Γ_G C
(− Neg A_G))
          using step-hyps by simp
      qed
      moreover have lit-of-glit (Neg A_G) = Neg A
        unfolding A-def lit-of-glit-def literal.map ..
      ultimately show ?thesis
        unfolding S'-def ‹Γ' = decide-lit (Neg A) # Γ› decide-lit-def
        using C'-def C-def ‹U' = U› step-hyps(1,4) by argo
    qed

    ultimately show False
      by metis
  qed

  ultimately show ?thesis
    unfolding scl-fol.regular-scl-def by argo
next
  case step-hyps: (decide-pos A_G)

  define A :: ('f, 'v) term where
    A = term-of-gterm A_G

  let ?f = gtrailelem-of-trailelem
  have Γ' = map ?f Γ_G'
    unfolding Γ'-def ..
  also have ... = map ?f ((Pos A_G, None) # Γ_G)
    unfolding ‹Γ_G' = (Pos A_G, None) # Γ_G› ..
  also have ... = ?f (Pos A_G, None) # map ?f Γ_G
    unfolding list.map ..
```

also have $\ldots = ?f\ (Pos\ A_G,\ None)\ \#\ \Gamma$
unfolding Γ*-def* **..**
also have $\ldots = (lit\text{-}of\text{-}glit\ (Pos\ A_G),\ None)\ \#\ \Gamma$
unfolding *prod.case option.map gtrailelem-of-trailelem-def* **..**
also have $\ldots = (Pos\ (term\text{-}of\text{-}gterm\ A_G),\ None)\ \#\ \Gamma$
unfolding *lit-of-glit-def literal.map* **..**
also have $\ldots = (Pos\ A,\ None)\ \#\ \Gamma$
unfolding *A-def* **..**
finally have $\Gamma' = decide\text{-}lit\ (Pos\ A)\ \#\ \Gamma$
unfolding *decide-lit-def* **.**

have $U' = U$
unfolding *U'-def* $\langle U_G' = U_G \rangle$ *U-def* **..**

have $\neg\ Ex\ (scl\text{-}fol.conflict\ N\ \beta\ S)$
using *step-hyps nex-conflict-if-nbex-trail-false* **by** *metis*

moreover have *scl-fol.reasonable-scl N* β *S S'*
unfolding *scl-fol.reasonable-scl-def*
proof (*intro conjI impI notI* ; (*elim exE*) *?*)
have *scl-fol.decide N* β *S S'*
unfolding *S-def S'-def* $\langle U' = U \rangle$ $\mathcal{C}$*-def* $\mathcal{C}'$*-def* $\langle \mathcal{C}_G = None \rangle$ $\langle \mathcal{C}_G' = None \rangle$ *option.map*
proof (*rule scl-fol.decideI'*)
show *is-ground-lit* (*Pos A* $\cdot l$ *Var*)
by (*simp add*: *A-def*)
next
have $\forall\, x\ |\in|\ trail\text{-}atms\ \Gamma_G.\ x \prec_t A_G$
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence $A_G\ |\notin|\ trail\text{-}atms\ \Gamma_G$
by *blast*
hence $A_G \notin atm\text{-}of\ `\ fst\ `\ set\ \Gamma_G$
unfolding *fset-trail-atms* **.**
hence *term-of-gterm* $A_G \notin$ *term-of-gterm* $`$ *atm-of* $`$ *fst* $`$ *set* Γ_G
using *inj-image-mem-iff inj-term-of-gterm* **by** *metis*
hence *term-of-gterm* $A_G \notin set\ (map\ (\lambda x.\ term\text{-}of\text{-}gterm\ (atm\text{-}of\ (fst\ x)))\ \Gamma_G)$
unfolding *image-set list.map-comp comp-def* **.**
hence $A \notin set\ (map\ (\lambda x.\ atm\text{-}of\ (lit\text{-}of\text{-}glit\ (fst\ x)))\ \Gamma_G)$
unfolding *A-def atm-of-lit-of-glit-conv* **.**
hence $A \notin atm\text{-}of\ `\ fst\ `\ set\ \Gamma$
unfolding *image-set list.map-comp comp-def* Γ*-def gtrailelem-of-trailelem-def fst-case-prod-simp* **.**
hence $A\ |\notin|\ trail\text{-}atms\ \Gamma$
unfolding *fset-trail-atms* **.**
hence $\neg\ trail\text{-}defined\text{-}lit\ \Gamma\ (Pos\ A\ \cdot l\ Var)$
by (*simp add*: *trail-defined-lit-iff-trail-defined-atm*)
thus $\neg\ SCL\text{-}FOL.trail\text{-}defined\text{-}lit\ \Gamma\ (Pos\ A\ \cdot l\ Var)$
by (*simp add*: *SCL-FOL.trail-defined-lit-def trail-defined-lit-def*)

next
have A_G $|\in|$ *atms-of-clss* (N_G $|\cup|$ U_G)
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence A_G $|\in|$ *atms-of-clss* N_G
unfolding ‹*atms-of-clss* (N_G $|\cup|$ U_G) = *atms-of-clss* N_G› **.**
hence $A_G \preceq_t \beta_G$
using *ball-le-*β_G **by** *metis*
moreover have *gterm-of-term* A = A_G
by (*simp add*: *A-def*)
moreover have *gterm-of-term* β = β_G
by (*simp add*: β*-def*)
ultimately have *gterm-of-term* $A \preceq_t$ *gterm-of-term* β
by *argo*
thus *less-B*$^{==}$ (*atm-of* (*Pos A*) ·*a Var*) β
using *inj-term-of-gterm*[*THEN injD*]
by (*auto simp*: *less-B-def A-def* β*-def*)
next
show Γ' = *trail-decide* Γ (*Pos A* ·*l Var*)
using ‹Γ' = *decide-lit* (*Pos A*) # Γ›
unfolding *subst-lit-id-subst* **.**
qed

thus *scl-fol.scl N* β *S S'*
unfolding *scl-fol.scl-def* **by** *argo*
next
fix *S''* :: ('*f*, '*v*) *SCL-FOL.state*
assume *scl-fol.conflict N* β *S' S''*

moreover have ∄*S''*. *scl-fol.conflict N* β *S' S''*
proof −
have ¬ *Ex* (*scl-fol.conflict N* β ((*lit-of-glit* (*Pos* A_G), *None*) # Γ, U, $\mathcal{C}$))
proof (*rule nex-conflict-if-no-clause-could-propagate-comp*)
show ¬ *fBex* (*iefac* $\mathcal{F}$ |‘| (N_G $|\cup|$ U_G)) (*trail-false-cls* Γ_G)
using *step-hyps* **by** *argo*
next
show ∀ *x*$|\in|$ *trail-atms* Γ_G. *x* ≺$_t$ *atm-of* (*Pos* A_G)
unfolding *literal.sel*
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
next
have *clause-could-propagate* Γ_G *C* (*Neg* A_G) ⟹ *trail-false-cls* Γ_G' *C* **for** *C*
unfolding ‹Γ_G' = (*Pos* A_G, *None*) # Γ_G›
using *trail-false-if-could-have-propagatated* **by** *fastforce*

thus ¬ (∃ *C* $|\in|$ *iefac* $\mathcal{F}$ |‘| (N_G $|\cup|$ U_G). *clause-could-propagate* Γ_G *C* (− *Pos* A_G))
unfolding *uminus-Pos*
using *step-hyps* **by** *metis*
qed

```
      moreover have lit-of-glit (Pos A_G) = Pos A
        unfolding A-def lit-of-glit-def literal.map ..
      ultimately show ?thesis
        unfolding S'-def ‹Γ' = decide-lit (Pos A) # Γ› decide-lit-def
        using C'-def C-def ‹U' = U› step-hyps(1) step-hyps(3) by argo
    qed

    ultimately show False by metis
  qed

  ultimately show ?thesis
    unfolding scl-fol.regular-scl-def by argo
next
  case step-hyps: (propagate A_G C_G)

  have C_G |∈| iefac F |‘| (N_G |∪| U_G) and C_G-prop: clause-could-propagate Γ_G
C_G (Pos A_G)
    using step-hyps linorder-cls.is-least-in-fset-iff by simp-all

  define A :: ('f, 'v) term where
    A = term-of-gterm A_G

  define C :: ('f, 'v) term clause where
    C = cls-of-gcls C_G

  have ord-res.is-maximal-lit (Pos A_G) C_G and trail-false-cls Γ_G {#K ∈# C_G.
K ≠ Pos A_G#}
    using ‹clause-could-propagate Γ_G C_G (Pos A_G)›
    unfolding clause-could-propagate-def by metis+

  then obtain C_G' where C_G = add-mset (Pos A_G) C_G'
    by (metis linorder-lit.is-maximal-in-mset-iff mset-add)

  define C' :: ('f, 'v) term clause where
    C' = cls-of-gcls C_G'

  let ?f = gtrailelem-of-trailelem
  have Γ' = map ?f Γ_G'
    unfolding Γ'-def ..
  also have ... = map ?f ((Pos A_G, Some (efac C_G)) # Γ_G)
    unfolding ‹Γ_G' = (Pos A_G, Some -) # Γ_G› ..
  also have ... = ?f (Pos A_G, Some (efac C_G)) # map ?f Γ_G
    unfolding list.map ..
  also have ... = ?f (Pos A_G, Some (efac C_G)) # Γ
    unfolding Γ-def ..
  also have ... = (lit-of-glit (Pos A_G),
    Some (cls-of-gcls {#K ∈# efac C_G. K ≠ Pos A_G#}, lit-of-glit (Pos A_G),
Var)) # Γ
    unfolding gtrailelem-of-trailelem-def prod.case option.map ..
```

also have ... = (*lit-of-glit* (*Pos* A_G),
Some (*cls-of-gcls* {#K ∈# *add-mset* (*Pos* A_G) {#K ∈# C_G. $K \neq$ *Pos* A_G#}. $K \neq$ *Pos* A_G#},
lit-of-glit (*Pos* A_G), *Var*)) # Γ
proof −
have *is-pos* (*Pos* A_G)
by *simp*

moreover have *linorder-lit.is-maximal-in-mset* C_G (*Pos* A_G)
using C_G*-prop* **unfolding** *clause-could-propagate-def* **by** *argo*

ultimately show *?thesis*
using *efac-spec-if-pos-lit-is-maximal* **by** *metis*
qed
also have ... = (*lit-of-glit* (*Pos* A_G),
Some (*cls-of-gcls* {#K ∈# C_G. $K \neq$ *Pos* A_G#}, *lit-of-glit* (*Pos* A_G), *Var*)) # Γ
by (*simp add*: *filter-filter-mset*)
also have ... = (*Pos* A, *Some* (*cls-of-gcls* {#K ∈# C_G. $K \neq$ *Pos* A_G#}, *Pos* A, *Var*)) # Γ
by (*simp add*: A*-def lit-of-glit-def*)
also have ... = (*Pos* A, *Some* (*cls-of-gcls* {#L ∈# C_G. *lit-of-glit* $L \neq$ *Pos* A#}, *Pos* A, *Var*)) # Γ
by (*metis* A*-def glit-of-lit-lit-of-glit lit-of-glit-def literal.simps*(9))
also have ... = (*Pos* A, *Some* ({#L ∈# *cls-of-gcls* C_G. $L \neq$ *Pos* A#}, *Pos* A, *Var*)) # Γ
unfolding *cls-of-gcls-def*
unfolding *image-mset-filter-mset-swap*[*of lit-of-glit* λL. $L \neq$ *Pos* A C_G]
unfolding *cls-of-gcls-def*[*symmetric*] **..**
also have ... = (*Pos* A ·*l* *Var*, *Some* ({#L ∈# *cls-of-gcls* C_G. $L \neq$ *Pos* A#}, *Pos* A, *Var*)) # Γ
by *simp*
also have ... = (*Pos* A ·*l* *Var*, *Some* ({#L ∈# C. $L \neq$ *Pos* A#}, *Pos* A, *Var*)) # Γ
unfolding C*-def* **..**
finally have Γ′ = *propagate-lit* (*Pos* A) {#L ∈# C. $L \neq$ *Pos* A#} *Var* # Γ
unfolding *propagate-lit-def* **.**

have $U' = U$
unfolding U'*-def* ‹$U_G' = U_G$› U*-def* **..**

obtain C_{G0} **where** C_{G0} |∈| N_G |∪| U_G **and** C_G = *iefac* $\mathcal{F}$ C_{G0}
using ‹C_G |∈| *iefac* $\mathcal{F}$ |‘| (N_G |∪| U_G)› **by** *blast*

define C_0 :: ('f, 'v) *term clause* **where**
C_0 = *cls-of-gcls* C_{G0}

have *ord-res.is-maximal-lit* (*Pos* A_G) C_{G0}
using ‹*ord-res.is-maximal-lit* (*Pos* A_G) C_G› ‹C_G = *iefac* $\mathcal{F}$ C_{G0}›

by (*metis iefac-def linorder-lit.is-maximal-in-mset-iff set-mset-efac*)

have ¬ *Ex* (*scl-fol.conflict N β S*)
using *step-hyps nex-conflict-if-nbex-trail-false* **by** *metis*

moreover have *scl-fol.reasonable-scl N β S S′*
unfolding *scl-fol.reasonable-scl-def*
proof (*intro conjI impI notI* ; (*elim exE*) *?*)
have *scl-fol.propagate N β S S′*
unfolding *S-def S′-def* ‹*U′ = U*› *C-def C′-def* ‹$\mathcal{C}_G$ = *None*› ‹$\mathcal{C}_G$′ = *None*› *option.map*
proof (*rule scl-fol.propagateI′*)
show C_0 |∈| *N* |∪| *U*
unfolding C_0*-def N-def U-def*
using ‹C_{G0} |∈| N_G |∪| U_G›
by *blast*
next
show *is-ground-cls* (C_0 · *Var*)
by (*simp add*: C_0*-def*)

have *Pos A* ∈# C_0
unfolding *A-def* C_0*-def*
by (*metis* (*no-types*, *lifting*) ‹C_G = *iefac* $\mathcal{F}$ C_{G0}› ‹*ord-res.is-maximal-lit* (*Pos* A_G) C_G›
cls-of-gcls-def iefac-def image-eqI linorder-lit.is-maximal-in-mset-iff lit-of-glit-def literal.map(*1*) *multiset.set-map set-mset-efac*)

then show C_0 = *add-mset* (*Pos A*) (C_0 − {#*Pos A*#})
by *simp*

have A_G |∈| *atms-of-clss* (N_G |∪| U_G)
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence A_G |∈| *atms-of-clss* N_G
unfolding ‹*atms-of-clss* (N_G |∪| U_G) = *atms-of-clss* N_G› **.**
hence A_G ≼$_t$ β_G
using *ball-le-*β_G **by** *metis*
moreover have *gterm-of-term A* = A_G
by (*simp add*: *A-def*)
moreover have *gterm-of-term* β = β_G
by (*simp add*: β*-def*)
ultimately have *gterm-of-term A* ≼$_t$ *gterm-of-term* β
by *argo*
hence *less-B*$^{==}$ *A* β
by (*auto simp*: *less-B-def A-def* β*-def*)

show ∀ *K* ∈# C_0 · *Var*. *less-B*$^{==}$ (*atm-of K*) β
unfolding *subst-cls-id-subst*
proof (*intro ballI*)
fix *K* :: (′*f*, ′*v*) *Term.term literal*

assume $K \in\# C_0$
then obtain K_G **where** $K_G \in\# C_{G0}$ **and** *K-def*: $K = \textit{lit-of-glit}\ K_G$
unfolding C_0*-def cls-of-gcls-def* **by** *blast*

have $K_G \preceq_l \textit{Pos}\ A_G$
using ‹*ord-res.is-maximal-lit* (*Pos* A_G) C_{G0}› ‹$K_G \in\# C_{G0}$›
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *fastforce*

hence *atm-of* $K_G \preceq_t A_G$
by (*metis literal.collapse*(*1*) *literal.collapse*(*2*) *literal.sel*(*1*)
ord-res.less-lit-simps(*1*) *ord-res.less-lit-simps*(*4*) *reflclp-iff*)

hence *less-B*$^{==}$ (*atm-of K*) *A*
by (*auto simp*: *less-B-def K-def A-def atm-of-lit-of-glit-conv*)

then show *less-B*$^{==}$ (*atm-of K*) β
using ‹*less-B*$^{==}$ *A* β› **by** *order*
qed

show $\{\#K \in\# C_0.\ K \neq \textit{Pos}\ A\#\} = \{\#K \in\# \textit{remove1-mset}\ (\textit{Pos}\ A)\ C_0.\ K \cdot l\ \textit{Var} \neq \textit{Pos}\ A \cdot l\ \textit{Var}\#\}$
by *simp*

show $\{\#K \in\# \textit{remove1-mset}\ (\textit{Pos}\ A)\ C_0.\ K = \textit{Pos}\ A\#\} =$
$\{\#K \in\# \textit{remove1-mset}\ (\textit{Pos}\ A)\ C_0.\ K \cdot l\ \textit{Var} = \textit{Pos}\ A \cdot l\ \textit{Var}\#\}$
by *simp*

have *trail-false-cls* Γ_G $(\{\#K_G \in\# C_{G0}.\ K_G \neq \textit{Pos}\ A_G\#\})$
by (*smt* (*verit*, *ccfv-threshold*) ‹$C_G = \textit{iefac}\ \mathcal{F}\ C_{G0}$›
‹*trail-false-cls* Γ_G $\{\#K \in\# C_G.\ K \neq \textit{Pos}\ A_G\#\}$› *iefac-def mem-Collect-eq set-mset-efac*
set-mset-filter trail-false-cls-def)

moreover have (*cls-of-gcls* $\{\#K_G \in\# C_{G0}.\ K_G \neq \textit{Pos}\ A_G\#\}$ :: $('f, 'v)$ *term clause*) =
$\{\#K \in\# C_0.\ K \neq \textit{Pos}\ A\#\}$
by (*smt* (*verit*) *A-def* C_0*-def cls-of-gcls-def filter-mset-cong0 glit-of-lit-lit-of-glit*
image-mset-filter-mset-swap lit-of-glit-def literal.map(*1*))

ultimately have *trail-false-cls* Γ $(\{\#K \in\# C_0.\ K \neq \textit{Pos}\ A\#\} \cdot \textit{Var})$
unfolding *subst-cls-id-subst*
using *trail-false-cls-nonground-iff-trail-false-cls-ground*[*THEN iffD2*]
by (*metis* Γ*-def*)

thus *SCL-FOL.trail-false-cls* Γ $(\{\#K \in\# C_0.\ K \neq \textit{Pos}\ A\#\} \cdot \textit{Var})$
unfolding *SCL-FOL.trail-false-cls-def trail-false-cls-def*
unfolding *SCL-FOL.trail-false-lit-def trail-false-lit-def*
by *argo*

have $\forall x \mid\in\mid \mathit{trail\text{-}atms}\ \Gamma_G.\ x \prec_t A_G$
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence $A_G \mid\notin\mid \mathit{trail\text{-}atms}\ \Gamma_G$
by *blast*
hence $A_G \notin \mathit{atm\text{-}of}\ `\ \mathit{fst}\ `\ \mathit{set}\ \Gamma_G$
unfolding *fset-trail-atms* **.**
hence $\mathit{term\text{-}of\text{-}gterm}\ A_G \notin \mathit{term\text{-}of\text{-}gterm}\ `\ \mathit{atm\text{-}of}\ `\ \mathit{fst}\ `\ \mathit{set}\ \Gamma_G$
using *inj-image-mem-iff inj-term-of-gterm* **by** *metis*
hence $\mathit{term\text{-}of\text{-}gterm}\ A_G \notin \mathit{set}\ (\mathit{map}\ (\lambda x.\ \mathit{term\text{-}of\text{-}gterm}\ (\mathit{atm\text{-}of}\ (\mathit{fst}\ x)))\ \Gamma_G)$
unfolding *image-set list.map-comp comp-def* **.**
hence $A \notin \mathit{set}\ (\mathit{map}\ (\lambda x.\ \mathit{atm\text{-}of}\ (\mathit{lit\text{-}of\text{-}glit}\ (\mathit{fst}\ x)))\ \Gamma_G)$
unfolding *A-def atm-of-lit-of-glit-conv* **.**
hence $A \notin \mathit{atm\text{-}of}\ `\ \mathit{fst}\ `\ \mathit{set}\ \Gamma$
unfolding *image-set list.map-comp comp-def* Γ*-def gtrailelem-of-trailelem-def fst-case-prod-simp* **.**
hence $A \mid\notin\mid \mathit{trail\text{-}atms}\ \Gamma$
unfolding *fset-trail-atms* **.**
hence $\neg\ \mathit{trail\text{-}defined\text{-}lit}\ \Gamma\ (\mathit{Pos}\ A \cdot l\ \mathit{Var})$
by (*simp add*: *trail-defined-lit-iff-trail-defined-atm*)
thus $\neg\ \mathit{SCL\text{-}FOL.trail\text{-}defined\text{-}lit}\ \Gamma\ (\mathit{Pos}\ A \cdot l\ \mathit{Var})$
by (*simp add*: *SCL-FOL.trail-defined-lit-def trail-defined-lit-def*)

have $\mathit{set\text{-}mset}\ (\mathit{add\text{-}mset}\ (\mathit{Pos}\ A)\ \{\#K \in\# \mathit{remove1\text{-}mset}\ (\mathit{Pos}\ A)\ C_0.\ K = \mathit{Pos}\ A\#\}) =$
$\{\mathit{Pos}\ A\}$
by (*smt* (*verit*) *Collect-cong insert-compr mem-Collect-eq set-mset-add-mset-insert set-mset-filter singletonD*)
hence *is-unifier Var* (*atm-of* ‘ *set-mset*
(*add-mset* (*Pos A*) {#*K* ∈# *remove1-mset* (*Pos A*) C_0. *K* = *Pos A*#}))
by (*metis* (*no-types*, *lifting*) *finite-imageI finite-set-mset image-empty image-insert*
is-unifier-alt singletonD)
hence *is-unifiers Var* {*atm-of* ‘ *set-mset*
(*add-mset* (*Pos A*) {#*K* ∈# *remove1-mset* (*Pos A*) C_0. *K* = *Pos A*#})}
unfolding *SCL-FOL.is-unifiers-def* **by** *simp*
thus *SCL-FOL.is-imgu Var* {*atm-of* ‘ *set-mset*
(*add-mset* (*Pos A*) {#*K* ∈# *remove1-mset* (*Pos A*) C_0. *K* = *Pos A*#})}
unfolding *SCL-FOL.is-imgu-def* **by** *simp*

have $\{\#K \in\# C_{G0}.\ K \neq \mathit{Pos}\ A_G\#\} = \{\#K \in\# C_G.\ K \neq \mathit{Pos}\ A_G\#\}$
using ‹*ord-res.is-maximal-lit* (*Pos* A_G) C_G›
using ‹*ord-res.is-maximal-lit* (*Pos* A_G) C_{G0}›
unfolding ‹C_G = *iefac* $\mathcal{F}$ C_{G0}›
by (*metis add-mset-remove-trivial efac-spec-if-pos-lit-is-maximal ex1-efac-eq-factoring-chain factorizable-if-neq-efac iefac-def literal.disc*(*1*))

hence $\{\#K \in\# C_0.\ K \neq \mathit{Pos}\ A\#\} = \{\#K \in\# C.\ K \neq \mathit{Pos}\ A\#\}$
unfolding C_0*-def C-def*

by (*smt* (*verit*, *ccfv-SIG*) *A-def cls-of-gcls-def filter-mset-cong0 glit-of-lit-lit-of-glit image-mset-filter-mset-swap lit-of-glit-def literal.map*(*1*))

thus $\Gamma' = \mathit{trail\text{-}propagate}\ \Gamma\ (\mathit{Pos}\ A \cdot l\ \mathit{Var})\ (\{\#K \in\#\ C_0.\ K \neq \mathit{Pos}\ A\#\} \cdot \mathit{Var})\ \mathit{Var}$
unfolding *subst-cls-id-subst subst-lit-id-subst*
using ‹$\Gamma' = \mathit{propagate\text{-}lit}\ (\mathit{Pos}\ A)\ \{\#L \in\#\ C.\ L \neq \mathit{Pos}\ A\#\}\ \mathit{Var}\ \#\ \Gamma$›
by *argo*
qed

thus *scl-fol.scl N β S S′*
unfolding *scl-fol.scl-def* **by** *argo*
next
fix S'' :: (′*f*, ′*v*) *SCL-FOL.state*
assume *scl-fol.decide N β S S′*
thus *False*
unfolding *S-def S′-def*
proof (*cases N β* (Γ, *U*, $\mathcal{C}$) (Γ′, *U′*, $\mathcal{C}'$) *rule*: *scl-fol.decide.cases*)
case (*decideI L γ*)
show *False*
using ‹$\Gamma' = \mathit{decide\text{-}lit}\ (L \cdot l\ \gamma)\ \#\ \Gamma$›
using ‹$\Gamma' = \mathit{propagate\text{-}lit}\ (\mathit{Pos}\ A)\ \{\#L \in\#\ C.\ L \neq \mathit{Pos}\ A\#\}\ \mathit{Var}\ \#\ \Gamma$›
unfolding *decide-lit-def propagate-lit-def*
by *blast*
qed
qed

ultimately show *?thesis*
unfolding *scl-fol.regular-scl-def* **by** *argo*
next
case *step-hyps*: (*conflict* C_G)

have $\Gamma' = \Gamma$
unfolding Γ′*-def* ‹${\Gamma_G}' = \Gamma_G$› Γ*-def* **..**

have $U' = U$
unfolding *U′-def* ‹${U_G}' = U_G$› *U-def* **..**

have
C_G*-in*: $C_G \mid\in\mid \mathit{iefac}\ \mathcal{F}\ \mid`\mid\ (N_G \mid\cup\mid\ U_G)$ **and**
C_G*-false*: *trail-false-cls* Γ_G C_G **and**
C_G*-lt*: $\forall\, E_G \mid\in\mid \mathit{iefac}\ \mathcal{F}\ \mid`\mid\ (N_G \mid\cup\mid\ U_G).\ E_G \neq C_G \longrightarrow \mathit{trail\text{-}false\text{-}cls}\ \Gamma_G\ E_G \longrightarrow C_G \prec_c E_G$
using ‹*linorder-cls.is-least-in-fset* - C_G›
unfolding *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

have $\nexists L.\ \mathit{is\text{-}pos}\ L \wedge \mathit{ord\text{-}res.is\text{-}maximal\text{-}lit}\ L\ C_G$
proof (*rule notI* , *elim exE conjE*)
fix *L* :: ′*f gliteral*

assume *is-pos L* **and** C_G*-max-lit*: *ord-res.is-maximal-lit L* C_G

have $\{\#\}$ $|\notin|$ *iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$
using C_G*-lt*
by (*metis* (*full-types*) C_G*-max-lit bot-fset.rep-eq fBex-fempty linorder-cls.leD linorder-lit.is-maximal-in-mset-iff mempty-lesseq-cls set-mset-empty trail-false-cls-mempty*)

have *trail-false-lit* Γ_G *L*
using C_G*-max-lit* C_G*-false*
unfolding *linorder-lit.is-maximal-in-mset-iff trail-false-cls-def*
by *metis*

then obtain *Ln* Γ_{G0} **where** Γ_G = *Ln* # Γ_{G0}
unfolding *trail-false-lit-def*
by (*metis* (*no-types*) *List.insert-def image-iff insert-Nil neq-Nil-conv*)

moreover have
AAA: $\forall$ *x xs*. Γ_G = *x* # *xs* $\longrightarrow$
((*snd x* $\neq$ *None*) $\longleftrightarrow$ *fBex* (*iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$) (*trail-false-cls* (*x* # *xs*))) $\wedge$
(*snd x* $\neq$ *None* $\longrightarrow$ *is-pos* (*fst x*)) $\wedge$
($\forall$ *x* $\in$ *set xs*. *snd x* = *None*) **and**
BBB: ($\forall$ Γ_1 *Ln* Γ_0. Γ_G = Γ_1 @ *Ln* # Γ_0 $\longrightarrow$ *snd Ln* = *None* $\longrightarrow$
$\neg$ *fBex* (*iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$) (*trail-false-cls* (*Ln* # Γ_0))) **and**
Γ_G*-sorted*: *sorted-wrt* (λ*x y*. *atm-of* (*fst y*) $\prec_t$ *atm-of* (*fst x*)) Γ_G
using ‹*ord-res-11-invars* N_G S_G›[*unfolded* S_G*-def ord-res-11-invars.simps ord-res-10-invars.simps*]
by *simp-all*

moreover have *fBex* (*iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$) (*trail-false-cls* Γ_G)
using C_G*-in* C_G*-false* **by** *metis*

ultimately have *snd Ln* $\neq$ *None* **and** *is-pos* (*fst Ln*) **and** $\forall$ *x* $\in$ *set* Γ_{G0}. *snd x* = *None*
unfolding *atomize-conj* **by** *metis*

have $\neg$ *fBex* (*iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$) (*trail-false-cls* Γ_{G0})
proof (*cases* Γ_{G0})
case *Nil*
then show *?thesis*
using ‹$\{\#\}$ $|\notin|$ *iefac* $\mathcal{F}$ $|`|$ $(N_G$ $|\cup|$ $U_G)$›
unfolding *trail-false-cls-def trail-false-lit-def*
by *simp*
next
case (*Cons x xs*)
then show *?thesis*
using ‹Γ_G = *Ln* # Γ_{G0}›
using ‹$\forall$ *x* $\in$ *set* Γ_{G0}. *snd x* = *None*›

```
      using BBB[rule-format, of [Ln], unfolded append-Cons append-Nil]
      by simp
  qed

  hence ¬ trail-false-cls Γ_G0 C_G
    using C_G-in by metis

  hence fst Ln = − L
  using C_G-false C_G-max-lit Γ_G-sorted[unfolded ‹Γ_G = Ln # Γ_G0› sorted-wrt.simps]
    by (smt (verit, ccfv-SIG) Neg-atm-of-iff ‹Γ_G = Ln # Γ_G0› atm-of-uminus
        ord-res.less-lit-simps(4) imageE image-insert insertE
        linorder-lit.dual-order.strict-trans linorder-lit.is-maximal-in-mset-iff
      linorder-lit.neq-iff linorder-trm.order.irrefl list.simps(15) literal.collapse(1)
        ord-res.ground-ordered-resolution-calculus-axioms trail-false-cls-def
        trail-false-lit-def)

  hence ¬ is-pos L
    using ‹is-pos (fst Ln)›
    by (simp add: is-pos-neg-not-is-pos)

  thus False
    using ‹is-pos L› by contradiction
qed

hence C_G |∈| N_G |∪| U_G
proof −
  obtain C where C |∈| N_G |∪| U_G and C_G = iefac F C
    using C_G-in by blast

  hence C_G = C
    using ‹∄L. is-pos L ∧ ord-res.is-maximal-lit L C_G›
   by (metis clauses-in-F-have-pos-max-lit ex1-efac-eq-factoring-chain iefac-def
        ord-res.ground-factorings-preserves-maximal-literal)

  thus ?thesis
    using ‹C |∈| N_G |∪| U_G› by simp
qed

have scl-fol.conflict N β S S'
  unfolding S-def S'-def ‹Γ' = Γ› ‹U' = U› C-def C'-def ‹C_G = None› ‹C_G'
= Some C_G› option.map
proof (rule scl-fol.conflictI)
  show cls-of-gcls C_G |∈| N |∪| U
    unfolding N-def U-def
    using ‹C_G |∈| N_G |∪| U_G› by auto
next
  show is-ground-cls (cls-of-gcls C_G · (Var::'v ⇒ ('f, -) Term.term))
    by simp
next
```

```
    have trail-false-cls Γ_G C_G
      using ‹trail-false-cls Γ_G C_G› .

    hence trail-false-cls Γ (cls-of-gcls C_G · Var)
      unfolding Γ-def subst-cls-id-subst
      using trail-false-cls-nonground-iff-trail-false-cls-ground by metis

    thus SCL-FOL.trail-false-cls Γ (cls-of-gcls C_G · Var)
      unfolding SCL-FOL.trail-false-cls-def trail-false-cls-def
      unfolding SCL-FOL.trail-false-lit-def trail-false-lit-def
      by argo
  qed

  thus ?thesis
    unfolding scl-fol.regular-scl-def by argo
next
  case step-hyps: (skip L_G C_G n_G)

  have Γ = gtrailelem-of-trailelem (L_G, n_G) # Γ'
    unfolding Γ-def Γ'-def ‹Γ_G = (L_G, n_G) # Γ_G'› by simp

  have U' = U
    unfolding U'-def ‹U_G' = U_G› U-def ..

  have ¬ Ex (scl-fol.conflict N β S)
    using ‹C_G = Some C_G› nex-conflict-if-alread-in-conflict by metis

  moreover have scl-fol.reasonable-scl N β S S'
    unfolding scl-fol.reasonable-scl-def
  proof (intro conjI impI notI ; (elim exE) ?)
    have scl-fol.skip N β S S'
      unfolding S-def S'-def ‹U' = U› C-def C'-def ‹C_G = Some C_G› ‹C_G' =
Some C_G› option.map
      unfolding ‹Γ = - # Γ'› gtrailelem-of-trailelem-def prod.case
    proof (rule scl-fol.skipI)
      have - lit-of-glit L_G = lit-of-glit (- L_G)
        by (cases L_G) (simp-all add: lit-of-glit-def)
      show - lit-of-glit L_G ∉# cls-of-gcls C_G · Var
        unfolding subst-cls-id-subst
        unfolding ‹- lit-of-glit L_G = lit-of-glit (- L_G)›
        unfolding cls-of-gcls-def
        using ‹- L_G ∉# C_G› inj-image-mset-mem-iff[OF inj-lit-of-glit]
        by metis
    qed

    thus scl-fol.scl N β S S'
      unfolding scl-fol.scl-def by argo
  next
    fix S'' :: ('f, 'v) SCL-FOL.state
```

```
    assume scl-fol.conflict N β S' S''

    moreover have ∄S''. scl-fol.conflict N β S' S''
    unfolding S'-def C'-def ‹C_G' = Some C_G› by (simp add: scl-fol.conflict.simps)

    ultimately show False
      by metis
  qed

  ultimately show ?thesis
    unfolding scl-fol.regular-scl-def by argo
next
  case step-hyps: (resolution L_G C_G Γ_G'' D_G)

  have Γ' = Γ
    unfolding Γ'-def ‹Γ_G' = Γ_G› Γ-def ..

  have U' = U
    unfolding U'-def ‹U_G' = U_G› U-def ..

  have C = Some (cls-of-gcls D_G, Var)
    unfolding C-def ‹C_G = Some D_G› option.map ..

  hence C-eq: C = Some
   (add-mset (− lit-of-glit L_G) (remove1-mset (− lit-of-glit L_G) (cls-of-gcls D_G)),
Var)
    by (smt (verit, best) add-mset-remove-trivial atm-of-eq-atm-of atm-of-lit-of-glit-conv
      cls-of-gcls-def glit-of-lit-lit-of-glit image-mset-add-mset insert-DiffM step-hyps(7)
        uminus-not-id')

  have C' = Some (
    remove1-mset (− (lit-of-glit L_G)) (cls-of-gcls D_G) +
    remove1-mset (lit-of-glit L_G) (cls-of-gcls C_G), Var)
    unfolding C'-def ‹C_G' = Some -› option.map
    apply (simp add: cls-of-gcls-def)
   by (smt (verit, ccfv-threshold) add-diff-cancel-right' add-mset-add-single atm-of-eq-atm-of
       atm-of-lit-of-glit-conv diff-single-trivial glit-of-lit-lit-of-glit
     image-mset-remove1-mset-if insert-DiffM is-pos-neg-not-is-pos msed-map-invR)
  hence C'-eq: C' = Some (
    (remove1-mset (− (lit-of-glit L_G)) (cls-of-gcls D_G) · Var +
    remove1-mset (lit-of-glit L_G) (cls-of-gcls C_G) · Var) · Var, Var)
    by simp

  have linorder-lit.is-greatest-in-mset C_G L_G
    using ‹ord-res-11-invars N_G S_G›[unfolded S_G-def ‹Γ_G = (L_G, Some C_G) #
Γ_G''›]
    unfolding ord-res-11-invars.simps ord-res-10-invars.simps
    by simp
```

hence *add-mset* L_G $\{\#y \in\# C_G.\ y \neq L_G\#\} = C_G$
using *linorder-lit.explode-greatest-in-mset* **by** *metis*

hence $C_G - \{\#L_G\#\} = \{\#K \in\# C_G.\ K \neq L_G\#\}$
by (*metis add-mset-remove-trivial*)

hence *cls-of-gcls* $(C_G - \{\#L_G\#\}) =$ *cls-of-gcls* $\{\#K \in\# C_G.\ K \neq L_G\#\}$
by *argo*

moreover have *cls-of-gcls* $(C_G - \{\#L_G\#\}) =$ *cls-of-gcls* $C_G -$ *cls-of-gcls* $\{\#L_G\#\}$
unfolding *cls-of-gcls-def*
proof (*rule image-mset-Diff*)
show $\{\#L_G\#\} \subseteq\# C_G$
by (*metis* ‹*ord-res.is-strictly-maximal-lit* L_G C_G› *linorder-lit.is-greatest-in-mset-iff single-subset-iff*)
qed

ultimately have *cls-of-gcls* $C_G - \{\#$*lit-of-glit* $L_G\#\} =$ *cls-of-gcls* $\{\#K \in\# C_G.\ K \neq L_G\#\}$
by (*metis* ‹*add-mset* L_G $\{\#y \in\# C_G.\ y \neq L_G\#\} = C_G$› *cls-of-gcls-def image-mset-remove1-mset-if union-single-eq-member*)

have $\neg$ *Ex* (*scl-fol.conflict* N β S)
using ‹$\mathcal{C}_G$ = *Some* -› *nex-conflict-if-alread-in-conflict* **by** *metis*

moreover have *scl-fol.reasonable-scl* N β S S'
unfolding *scl-fol.reasonable-scl-def*
proof (*intro conjI impI notI* ; (*elim exE*) *?*)
have *scl-fol.resolve* N β S S'
unfolding *S-def* S'*-def* ‹$\Gamma' = \Gamma$› ‹$U' = U$›
unfolding $\mathcal{C}$*-eq* $\mathcal{C}'$*-eq*
proof (*rule scl-fol.resolveI*)
show $\Gamma =$ *trail-propagate* (*map gtrailelem-of-trailelem* ${\Gamma_G}''$) (*lit-of-glit* L_G) (*remove1-mset* (*lit-of-glit* L_G) (*cls-of-gcls* C_G)) *Var*
unfolding Γ*-def* ‹$\Gamma_G = (L_G,$ *Some* $C_G) \#$ ${\Gamma_G}''$› *gtrailelem-of-trailelem-def list.map prod.case*
unfolding *propagate-lit-def subst-lit-id-subst option.map*
unfolding ‹*remove1-mset* (*lit-of-glit* L_G) (*cls-of-gcls* C_G) = *cls-of-gcls* $\{\#K \in\# C_G.\ K \neq L_G\#\}$›
by *argo*
next
show *lit-of-glit* L_G ·l *Var* = $-$ ($-$ *lit-of-glit* L_G ·l *Var*)
by *simp*
next
show *SCL-FOL.is-renaming Var*
by *simp*
next

```
      show SCL-FOL.is-renaming Var
        by simp
    next
      show
        vars-cls (add-mset (− lit-of-glit L_G)
          (remove1-mset (− lit-of-glit L_G) (cls-of-gcls D_G)) · Var) ∩
        vars-cls (add-mset (lit-of-glit L_G)
          (remove1-mset (lit-of-glit L_G) (cls-of-gcls C_G)) · Var) =
        {}
        by (metis (no-types, lifting) boolean-algebra.conj-zero-right cls-of-gcls-def
            diff-single-trivial image-mset-add-mset insert-DiffM subst-cls-id-subst
            vars-cls-cls-of-gcls)
    next
      show SCL-FOL.is-imgu Var
        {{atm-of (− lit-of-glit L_G) ·a Var, atm-of (lit-of-glit L_G) ·a Var}}
      by (simp add: SCL-FOL.is-imgu-def SCL-FOL.is-unifiers-def SCL-FOL.is-unifier-def)
    next
      show is-grounding-merge Var
        (vars-cls
          (add-mset (− lit-of-glit L_G) (remove1-mset (− lit-of-glit L_G) (cls-of-gcls
D_G)) · Var))
        (rename-subst-domain Var Var)
        (vars-cls
          (add-mset (lit-of-glit L_G) (remove1-mset (lit-of-glit L_G) (cls-of-gcls C_G))
· Var))
        (rename-subst-domain Var Var)
        by (simp add: is-grounding-merge-def)
    qed

    thus scl-fol.scl N β S S'
      unfolding scl-fol.scl-def by argo
  next
    fix S'' :: ('f, 'v) SCL-FOL.state
    assume scl-fol.conflict N β S' S''

    moreover have ∄S''. scl-fol.conflict N β S' S''
    unfolding S'-def C'-def ‹C_G' = Some -› by (simp add: scl-fol.conflict.simps)

    ultimately show False
      by metis
  qed

  ultimately show ?thesis
    unfolding scl-fol.regular-scl-def by argo
next
  case step-hyps: (backtrack L_G C_G)

  define K :: ('f, 'v) term literal where
    K = − lit-of-glit L_G
```

define $D :: ('f, 'v)$ *term clause* **where**
$D =$ *cls-of-gcls* $C_G - \{\#K\#\}$

have *add-mset* K $D =$ *cls-of-gcls* C_G
by (*smt* (*verit*, *best*) *D-def K-def add-mset-remove-trivial atm-of-eq-atm-of atm-of-lit-of-glit-conv cls-of-gcls-def glit-of-lit-lit-of-glit image-mset-add-mset insert-DiffM step-hyps*(6) *uminus-not-id'*)

have $U' =$ *finsert* (*add-mset* K D) U
unfolding *U-def U'-def* ‹$U_G' =$ *finsert* C_G U_G›
by (*smt* (*verit*, *ccfv-SIG*) *D-def K-def add-mset-remove-trivial atm-of-eq-atm-of atm-of-lit-of-glit-conv cls-of-gcls-def fimage-finsert glit-of-lit-lit-of-glit image-mset-add-mset insert-DiffM step-hyps*(6) *uminus-not-id'*)

have $\mathcal{C} =$ *Some* (*add-mset* K D, *Var*)
by (*smt* (*verit*) *D-def K-def* $\mathcal{C}$*-def add-mset-remove-trivial atm-of-eq-atm-of atm-of-lit-of-glit-conv cls-of-gcls-def glit-of-lit-lit-of-glit image-mset-add-mset insert-DiffM option.map*(2) *step-hyps*(1,6) *uminus-not-id'*)

have $\mathcal{C}' =$ *None*
unfolding $\mathcal{C}'$*-def* ‹$\mathcal{C}_G' =$ *None*› *option.map* **..**

have ¬ *Ex* (*scl-fol.conflict* N β S)
using ‹$\mathcal{C}_G =$ *Some* -› *nex-conflict-if-alread-in-conflict* **by** *metis*

moreover have *scl-fol.reasonable-scl* N β S S'
unfolding *scl-fol.reasonable-scl-def*
proof (*intro conjI impI notI* ; (*elim exE*) *?*)
have *scl-fol.backtrack* N β S S'
unfolding *S-def S'-def*
unfolding ‹$U' =$ *finsert* (*add-mset* K D) U› ‹$\mathcal{C} =$ *Some* (*add-mset* K D, *Var*)› ‹$\mathcal{C}' =$ *None*›
proof (*rule scl-fol.backtrackI*)
show $\Gamma =$ *trail-decide* ([] @ Γ') (*lit-of-glit* L_G)
unfolding *append-Nil*
unfolding *decide-lit-def*
unfolding Γ*-def* ‹$\Gamma_G =$ - # -› *list.map* Γ'*-def*[*symmetric*]
unfolding *gtrailelem-of-trailelem-def prod.case option.map*
..
next
show *lit-of-glit* $L_G = -$ (K ·*l* *Var*)
unfolding *K-def* **by** *simp*
next
have *sorted-wrt* (λx y. *atm-of* (*fst* y) $\prec_t$ *atm-of* (*fst* x)) Γ_G
using ‹*ord-res-11-invars* N_G S_G›[*unfolded* S_G*-def*]
unfolding *ord-res-11-invars.simps ord-res-10-invars.simps*
by *fast*

hence *trail-consistent* Γ_G
using *trail-consistent-if-sorted-wrt-atoms* **by** *metis*

hence ¬ *trail-defined-lit* $\Gamma_G{}'$ (− L_G)
by (*metis trail-consistent.cases atm-of-eq-atm-of list.distinct*(*1*) *list.inject prod.sel*(*1*) *step-hyps*(*5*) *trail-defined-lit-iff-trail-defined-atm*)

hence ¬ *trail-false-lit* $\Gamma_G{}'$ (− L_G)
using *trail-defined-lit-iff-true-or-false* **by** *metis*

hence ¬ *trail-false-cls* $\Gamma_G{}'$ C_G
using ‹− L_G ∈# C_G›
unfolding *trail-false-cls-def* **by** *metis*

hence ¬ *trail-false-cls* Γ' (*add-mset K D*)
unfolding Γ'*-def* ‹*add-mset K D* = *cls-of-gcls* C_G›
unfolding *trail-false-cls-nonground-iff-trail-false-cls-ground* **.**

moreover have *is-ground-cls* (*add-mset K D*)
using $\mathcal{C}$*-def* ‹$\mathcal{C}$ = *Some* (*add-mset K D*, *Var*)› *step-hyps*(*1*) **by** *auto*

ultimately have ∄γ. *is-ground-cls* (*add-mset K D* · γ) ∧ *trail-false-cls* Γ' (*add-mset K D* · γ)
by *simp*

thus ∄γ. *is-ground-cls* (*add-mset K D* · γ) ∧ *SCL-FOL.trail-false-cls* Γ' (*add-mset K D* · γ)
unfolding *SCL-FOL.trail-false-cls-def trail-false-cls-def*
unfolding *SCL-FOL.trail-false-lit-def trail-false-lit-def*
by *argo*
qed

thus *scl-fol.scl N* β *S S*$'$
unfolding *scl-fol.scl-def* **by** *argo*
next
fix *S*$''$ :: (′*f*, ′*v*) *SCL-FOL.state*
assume *scl-fol.decide N* β *S S*$'$
thus *False*
unfolding *S-def* ‹$\mathcal{C}$ = *Some* -›
by (*auto elim*!: *scl-fol.decide.cases*)
qed

ultimately show *?thesis*
unfolding *scl-fol.regular-scl-def* **by** *argo*
qed
qed

end

lemma *wfp-on-antimono-stronger*:
fixes
$A :: 'a\ set$ **and** $B :: 'b\ set$ **and**
$f :: 'a \Rightarrow 'b$ **and**
$R :: 'b \Rightarrow 'b \Rightarrow bool$ **and** $Q :: 'a \Rightarrow 'a \Rightarrow bool$
assumes
wf: *wfp-on B R* **and**
sub: $f\ `\ A \subseteq B$ **and**
mono: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ x\ y \Longrightarrow R\ (f\ x)\ (f\ y)$
shows *wfp-on A Q*
unfolding *wfp-on-iff-ex-minimal*
proof (*intro allI impI*)
fix $A' :: 'a\ set$
assume $A' \subseteq A$ **and** $A' \neq \{\}$
have $f\ `\ A' \subseteq B$
using $\langle A' \subseteq A \rangle$ *sub* **by** *blast*
moreover have $f\ `\ A' \neq \{\}$
using $\langle A' \neq \{\} \rangle$ **by** *blast*
ultimately have $\exists z \in f\ `\ A'.\ \forall y.\ R\ y\ z \longrightarrow y \notin f\ `\ A'$
using *wf wfp-on-iff-ex-minimal* **by** *blast*
hence $\exists z \in A'.\ \forall y.\ R\ (f\ y)\ (f\ z) \longrightarrow y \notin A'$
by *blast*
thus $\exists z \in A'.\ \forall y.\ Q\ y\ z \longrightarrow y \notin A'$
using $\langle A' \subseteq A \rangle$ *mono* **by** *blast*
qed

For AFP-devel, delete $[\![$*wfp-on ?B ?R*; *?f ` ?A* $\subseteq$ *?B*; $\bigwedge x\ y.\ [\![x \in ?A;\ y \in ?A;\ ?Q\ x\ y]\!] \Longrightarrow ?R\ (?f\ x)\ (?f\ y)]\!] \Longrightarrow$ *wfp-on ?A ?Q* as it is available in *HOL.Wellfounded*.

corollary (**in** *scl-fol-calculus*) *termination-projectable-strategy*:
fixes
$N :: ('f, 'v)\ Term.term\ clause\ fset$ **and**
$\beta :: ('f, 'v)\ Term.term$ **and**
strategy **and** *strategy-init* **and** *proj*
assumes *strategy-restricts-regular-scl*:
$\bigwedge S\ S'.\ strategy^{**}\ strategy\text{-}init\ S \Longrightarrow strategy\ S\ S' \Longrightarrow regular\text{-}scl\ N\ \beta\ (proj\ S)\ (proj\ S')$ **and**
initial-state: *proj strategy-init* = *initial-state*
shows *wfp-on* $\{S.\ strategy^{**}\ strategy\text{-}init\ S\}\ strategy^{-1-1}$
proof (*rule wfp-on-antimono-stronger*)
show *wfp-on* $\{proj\ S \mid S.\ strategy^{**}\ strategy\text{-}init\ S\}\ (regular\text{-}scl\ N\ \beta)^{-1-1}$
proof (*rule wfp-on-subset*)
show *wfp-on* $\{S.\ (regular\text{-}scl\ N\ \beta)^{**}\ initial\text{-}state\ S\}\ (regular\text{-}scl\ N\ \beta)^{-1-1}$
using *termination-regular-scl* **by** *metis*
next
show $\{proj\ S \mid S.\ strategy^{**}\ strategy\text{-}init\ S\} \subseteq \{S.\ (regular\text{-}scl\ N\ \beta)^{**}\ initial\text{-}state\ S\}$
proof (*intro Collect-mono impI*, *elim exE conjE*)
fix *s S* **assume** *s* = *proj S* **and** *strategy*** *strategy-init S*

show (*regular-scl N β*)** *initial-state s*
unfolding ‹*s* = *proj S*›
using ‹*strategy*** *strategy-init S*›
proof (*induction S rule*: *rtranclp-induct*)
case *base*
thus *?case*
unfolding *initial-state* **by** *simp*
next
case (*step y z*)
thus *?case*
using *strategy-restricts-regular-scl*
by (*meson rtranclp.simps*)
qed
qed
qed
next
show *proj* ‘ {*S*. *strategy*** *strategy-init S*} ⊆ {*proj S* |*S*. *strategy*** *strategy-init S*}
by *blast*
next
show ⋀*S′ S*. *S* ∈ {*S*. *strategy*** *strategy-init S*} ⟹ *strategy*$^{-1-1}$ *S′ S* ⟹
(*regular-scl N β*)$^{-1-1}$ (*proj S′*) (*proj S*)
using *strategy-restricts-regular-scl* **by** *simp*
qed

For AFP-devel, delete ⟦*scl-fol-calculus ?renaming-vars ?less-B*; ⋀*S S′*. ⟦*?strategy*** *?strategy-init S*; *?strategy S S′*⟧ ⟹ *scl-fol-calculus.regular-scl ?less-B ?N ?β* (*?proj S*) (*?proj S′*); *?proj ?strategy-init* = *initial-state*⟧ ⟹ *wfp-on* {*S*. *?strategy*** *?strategy-init S*} *?strategy*$^{-1-1}$ as it is available in *Simple-Clause-Learning.Termination*.

corollary (**in** *simulation-SCLFOL-ground-ordered-resolution*) *ord-res-11-termination*:
fixes *N* :: *'f gclause fset*
shows *wfp-on* {*S*. (*ord-res-11 N*)** ({||}, {||}, [], *None*) *S*} (*ord-res-11 N*)$^{-1-1}$
proof (*rule scl-fol.termination-projectable-strategy*)
fix *S S′*
assume *run*: (*ord-res-11 N*)** ({||}, {||}, [], *None*) *S* **and** *step*: *ord-res-11 N S S′*

define *β* :: *'f gterm* **where**
β = (*THE A*. *linorder-trm.is-greatest-in-fset* (*atms-of-clss N*) *A*)

show *scl-fol.regular-scl* (*cls-of-gcls* |‘| *N*) (*term-of-gterm β*) (*state-of-gstate S*) (*state-of-gstate S′*)
proof (*rule ord-res-11-is-strategy-for-regular-scl*)
show ∀ *A*$_G$ |∈| *atms-of-clss N*. *A*$_G$ ≼$_t$ *β*
proof (*cases atms-of-clss N* = {||})
case *True*
thus *?thesis*
by *simp*

```
    next
      case False
      then show ?thesis
        unfolding β-def
        by (metis (full-types) linorder-trm.Uniq-is-greatest-in-fset
            linorder-trm.ex-greatest-in-fset linorder-trm.is-greatest-in-fset-iff sup2CI
            the1-equality')
    qed
  next
    show (ord-res-11 N)** ({||}, {||}, [], None) S
      using run .
  next
    show ord-res-11 N S S'
      using step .
  qed
next
  show state-of-gstate ({||}, {||}, [], None) = SCL-FOL.initial-state
    by simp
qed

corollary (in scl-fol-calculus) static-non-subsumption-projectable-strategy:
  fixes strategy and strategy-init and proj
  assumes
    run: strategy** strategy-init S and
    step: backtrack N β (proj S) S' and
    strategy-restricts-regular-scl:
      ⋀S S'. strategy** strategy-init S ⟹ strategy S S' ⟹ regular-scl N β (proj
S) (proj S') and
    initial-state: proj strategy-init = initial-state
  defines
    U ≡ state-learned (proj S)
  shows ∃ C γ. state-conflict (proj S) = Some (C, γ) ∧ ¬ (∃ D |∈| N |∪| U.
subsumes D C)
proof −
  have (regular-scl N β)** initial-state (proj S)
    using run
  proof (induction S rule: rtranclp-induct)
    case base
    thus ?case
      unfolding initial-state by simp
  next
    case (step y z)
    thus ?case
      using strategy-restricts-regular-scl
      by (meson rtranclp.simps)
  qed

  moreover have backtrack N β (proj S) S'
    using step by simp
```

ultimately show *?thesis*
unfolding *U-def*
using *static-non-subsumption-regular-scl*
by *simp*
qed

For AFP-devel, delete ⟦*scl-fol-calculus ?renaming-vars ?less-B*; *?strategy*** *?strategy-init ?S*; *scl-fol-calculus.backtrack ?N ?β* (*?proj ?S*) *?S′*; ⋀*S S′*. ⟦*?strategy*** *?strategy-init S*; *?strategy S S′*⟧ ⟹ *scl-fol-calculus.regular-scl ?less-B ?N ?β* (*?proj S*) (*?proj S′*); *?proj ?strategy-init* = *initial-state*⟧ ⟹ ∃ *C γ*. *state-conflict* (*?proj ?S*) = *Some* (*C*, *γ*) ∧ ¬ (∃ *D*|∈| *?N* |∪| *state-learned* (*?proj ?S*). *subsumes D C*) as it is available in *Simple-Clause-Learning.Non-Redundancy*.

corollary (**in** *simulation-SCLFOL-ground-ordered-resolution*) *ord-res-11-non-subsumption*:
fixes N_G :: *'f gclause fset* **and** *s* :: - × - × - × -
defines
β ≡ (*THE A*. *linorder-trm.is-greatest-in-fset* (*atms-of-clss* N_G) *A*)
assumes
run: (*ord-res-11* N_G)** ({||}, {||}, [], *None*) *s* **and**
step: *scl-fol.backtrack* (*cls-of-gcls* |‘| N_G) (*term-of-gterm β*) (*state-of-gstate s*) *s′*
shows ∃ U_{er} *F Γ D*. *s* = (U_{er}, *F*, Γ, *Some D*) ∧ ¬ (∃ *C* |∈| N_G |∪| U_{er}. *C* ⊆# *D*)
proof −
have ∃ *C γ*. *state-conflict* (*state-of-gstate s*) = *Some* (*C*, *γ*) ∧
¬ (∃ *D*|∈| *cls-of-gcls* |‘| N_G |∪| *state-learned* (*state-of-gstate s*). *subsumes D C*)
proof (*rule scl-fol.static-non-subsumption-projectable-strategy*[
of ord-res-11 N_G - - - - *state-of-gstate* , *OF run step*])
fix *S S′*
assume *run*: (*ord-res-11* N_G)** ({||}, {||}, [], *None*) *S* **and** *step*: *ord-res-11* N_G *S S′*
show *scl-fol.regular-scl* (*cls-of-gcls* |‘| N_G) (*term-of-gterm β*) (*state-of-gstate S*) (*state-of-gstate S′*)
proof (*intro ord-res-11-is-strategy-for-regular-scl ballI*)
fix A_G :: *'f gterm*
assume A_G |∈| *atms-of-clss* N_G
show A_G ≼$_t$ *β*
proof (*cases atms-of-clss* N_G = {||})
case *True*
thus *?thesis*
using ‹A_G |∈| *atms-of-clss* N_G›
by *simp*
next
case *False*
then show *?thesis*
using ‹A_G |∈| *atms-of-clss* N_G›
unfolding *β-def*
by (*metis* (*full-types*) *linorder-trm.Uniq-is-greatest-in-fset*

linorder-trm.ex-greatest-in-fset linorder-trm.is-greatest-in-fset-iff sup2CI the1-equality')
    qed
  next
    show (*ord-res-11* N_G)** ({||}, {||}, [], *None*) *S*
      using *run* **.**
  next
    show *ord-res-11* N_G *S S'*
      using *step* **.**
  qed
next
  show *state-of-gstate* ({||}, {||}, [], *None*) = *initial-state*
    by *simp*
qed

moreover obtain U_G $\mathcal{F}$ Γ *D* **where** *s* = (U_G, $\mathcal{F}$, Γ, *Some D*)
proof *atomize-elim*
  obtain U_G $\mathcal{F}$ Γ $\mathcal{C}$ **where** *s* = (U_G, $\mathcal{F}$, Γ, $\mathcal{C}$)
    by (*metis prod.exhaust*)

  moreover obtain *D* **where** $\mathcal{C}$ = *Some D*
    using *step*
    by (*auto simp*: ‹*s* = (U_G, $\mathcal{F}$, Γ, $\mathcal{C}$)› *elim*: *scl-fol.backtrack.cases*)

  ultimately show ∃ U_{er} $\mathcal{F}$ Γ *D*. *s* = (U_{er}, $\mathcal{F}$, Γ, *Some D*)
    by *metis*
qed

ultimately have ¬ (∃ *C* |∈| N_G |∪| U_G.
  subsumes (*cls-of-gcls C* :: ('*f*, '*v*) *term clause*) (*cls-of-gcls D*))
  by *auto*

hence ¬ (∃ *C* |∈| N_G |∪| U_G. (*cls-of-gcls C* :: ('*f*, '*v*) *term clause*) ⊆# (*cls-of-gcls D*))
  by (*simp add*: *subsumes-def*)

hence ¬ (∃ *C* |∈| N_G |∪| U_G. *C* ⊆# *D*)
  by (*metis cls-of-gcls-def image-mset-subseteq-mono*)

thus *?thesis*
  unfolding ‹*s* = (U_G, $\mathcal{F}$, Γ, *Some D*)› **by** *metis*
qed

end

References

[1] M. Bromberger, C. Jain, and C. Weidenbach. SCL(FOL) can simulate non-redundant superposition clause learning. In B. Pientka and C. Tinelli, editors, *Automated Deduction – CADE 29*, pages 134–152, Cham, 2023. Springer Nature Switzerland.