

SCL(FOL) Can Simulate Ground Nonredundant Ordered Resolution

Martin Desharnais

March 19, 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

1	Ground Resolution Calculus	4
1.1	Ground Rules	7
1.2	Ground Layer	8
1.3	Correctness	8
1.4	Redundancy Criterion	10
1.5	Refutational Completeness	12
1.6	Move to <i>HOL.Transitive-Closure</i>	54
2	Move to <i>HOL-Library.Multiset</i>	58
3	Move to <i>HOL-Library.FSet</i>	59
4	Move to <i>VeriComp.Simulation</i>	60
5	Move to <i>Simple-Clause-Learning.SCL-FOL</i>	62
6	Move to ground ordered resolution	65
7	Move somewhere?	68

8	Ground ordered resolution for ground terms	75
9	Common definitions and lemmas	76
10	Lemmas about going between ground and first-order terms	92
11	SCL(FOL) for first-order terms	93
12	ORD-RES	95
13	ORD-RES-1 (deterministic)	96
14	Function for full factorization	100
15	ORD-RES-2 (full factorization)	110
16	Function for full resolution	122
17	ORD-RES-3 (full resolve)	146
18	Function for implicit full factorization	151
19	ORD-RES-4 (implicit factorization)	160
20	ORD-RES-5 (explicit model construction)	163
21	ORD-RES-6 (model backjump)	197
22	ORD-RES-7 (clause-guided literal trail construction)	234
23	ORD-RES-8 (atom-guided literal trail construction)	344
24	ORD-RES-9 (factorize when propagating)	375
25	ORD-RES-10 (propagate iff a conflict is produced)	386
26	ORD-RES-11 (SCL strategy)	413
27	ORD-RES-1 (deterministic)	444
28	ORD-RES-2 (full factorization)	446
29	ORD-RES-3 (full resolve)	462
30	ORD-RES-4 (implicit factorization)	479
31	ORD-RES-5 (explicit model construction)	485

32 ORD-RES-6 (model backjump)	491
33 ORD-RES-7 (clause-guided literal trail construction)	502
34 ORD-RES-8 (atom-guided literal trail construction)	526
35 ORD-RES-9 (factorize when propagating)	563
36 ORD-RES-10 (propagate iff a conflict is produced)	565
37 ORD-RES-11 (SCL strategy)	573
38 ORD-RES-11 is a regular SCL strategy	590
theory Isabelle-2024-Compatibility	
imports	
<i>Main</i>	
<i>HOL-Library.Multiset</i>	
begin	
lemmas <i>wfP-def = wfp-def</i>	
lemmas <i>wfP-if-convertible-to-wfP = wfp-if-convertible-to-wfp</i>	
lemmas <i>wfP-imp-asymp = wfp-imp-asymp</i>	
lemmas <i>wfP-induct-rule = wfp-induct-rule</i>	
lemmas <i>wfP-multp = wfp-multp</i>	
end	
theory Ground-Ordered-Resolution	
imports	
<i>Saturation-Framework.Calculus</i>	
<i>Saturation-Framework-Extensions.Clausal-Calculus</i>	
<i>Isabelle-2024-Compatibility</i>	
<i>First-Order-Clause.Ground-Context</i>	
<i>First-Order-Clause.HOL-Extra</i>	
<i>First-Order-Clause.Transitive-Closure-Extra</i>	
<i>Min-Max-Least-Greatest.Min-Max-Least-Greatest-FSet</i>	
<i>Min-Max-Least-Greatest.Min-Max-Least-Greatest-Multiset</i>	
<i>First-Order-Clause.Multiset-Extra</i>	
<i>Superposition-Calculus.Relation-Extra</i>	
begin	
hide-type <i>Inference-System.inference</i>	
hide-const	
<i>Inference-System.Infer</i>	
<i>Inference-System.prem-s-of</i>	
<i>Inference-System.concl-of</i>	
<i>Inference-System.main-prem-of</i>	
primrec <i>mset-lit :: 'a literal \Rightarrow 'a multiset where</i>	
<i>mset-lit (Pos A) = {#A#} </i>	

$mset\text{-}lit (Neg A) = \{\#A, A\#\}$

type-synonym $'t\ atom = 't$

1 Ground Resolution Calculus

locale $ground\text{-}ordered\text{-}resolution\text{-}calculus =$

fixes

$less\text{-}trm :: 'f\ gterm \Rightarrow 'f\ gterm \Rightarrow bool$ (**infix** \prec_t 50) **and**

$select :: 'f\ gterm\ atom\ clause \Rightarrow 'f\ gterm\ atom\ clause$

assumes

$transp\text{-}less\text{-}trm[simp]: transp (\prec_t)$ **and**

$asympt\text{-}less\text{-}trm[intro]: asympt (\prec_t)$ **and**

$wfP\text{-}less\text{-}trm[intro]: wfP (\prec_t)$ **and**

$totalp\text{-}less\text{-}trm[intro]: totalp (\prec_t)$ **and**

$less\text{-}trm\text{-}compatible\text{-}with\text{-}gctxt[simp]: \bigwedge\ ctxt\ t\ t'. t \prec_t t' \Longrightarrow ctxt\langle t \rangle_G \prec_t ctxt\langle t' \rangle_G$

and

$less\text{-}trm\text{-}if\text{-}subterm[simp]: \bigwedge\ ctxt. ctxt \neq \square_G \Longrightarrow t \prec_t ctxt\langle t \rangle_G$ **and**

$select\text{-}subset: \bigwedge\ C. select\ C \subseteq\# C$ **and**

$select\text{-}negative\text{-}lits: \bigwedge\ C\ L. L \in\# select\ C \Longrightarrow is\text{-}neg\ L$

begin

lemma $irreflp\text{-}on\text{-}less\text{-}trm[simp]: irreflp\text{-}on\ A (\prec_t)$

by ($metis\ asymptD\ asympt\text{-}less\text{-}trm\ irreflp\text{-}onI$)

abbreviation $lesseq\text{-}trm$ (**infix** \preceq_t 50) **where**

$lesseq\text{-}trm \equiv (\prec_t)^{==}$

lemma $lesseq\text{-}trm\text{-}if\text{-}subterm\ eq: t \preceq_t ctxt\langle t \rangle_G$

using $less\text{-}trm\text{-}if\text{-}subterm$

by ($metis\ gctxt\text{-}ident\text{-}iff\text{-}eq\text{-}GHole\ reflclp\text{-}iff$)

definition $less\text{-}lit ::$

$'f\ gterm\ atom\ literal \Rightarrow 'f\ gterm\ atom\ literal \Rightarrow bool$ (**infix** \prec_l 50) **where**

$less\text{-}lit\ L1\ L2 \equiv multp (\prec_t) (mset\text{-}lit\ L1) (mset\text{-}lit\ L2)$

abbreviation $lesseq\text{-}lit$ (**infix** \preceq_l 50) **where**

$lesseq\text{-}lit \equiv (\prec_l)^{==}$

abbreviation $less\text{-}cls ::$

$'f\ gterm\ atom\ clause \Rightarrow 'f\ gterm\ atom\ clause \Rightarrow bool$ (**infix** \prec_c 50) **where**

$less\text{-}cls \equiv multp (\prec_l)$

abbreviation $lesseq\text{-}cls$ (**infix** \preceq_c 50) **where**

$lesseq\text{-}cls \equiv (\prec_c)^{==}$

lemma $transp\text{-}on\text{-}less\text{-}lit[simp]: transp\text{-}on\ A (\prec_l)$

by ($smt (verit, best) less\text{-}lit\text{-}def\ transpE\ transp\text{-}less\text{-}trm\ transp\text{-}multp\ transp\text{-}onI$)

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

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

lemma *asympt-on-less-lit*[*simp*]: *asympt-on* *A* (\prec_i)
by (*metis asymptD asympt-less-trm asympt-multp_{HO} asympt-onI less-lit-def multp-eq-multp_{HO} transp-less-trm*)

corollary *asympt-less-lit*[*simp*]: *asympt* (\prec_i)
by *simp*

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

lemma *irreflp-on-less-lit*[*simp*]: *irreflp-on* *A* (\prec_i)
by (*simp only: asympt-on-less-lit irreflp-on-if-asympt-on*)

lemma *wfP-less-lit*[*simp*]: *wfP* (\prec_i)
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_i)
proof (*rule totalp-onI*)
fix *L1 L2* :: 'f *gterm atom literal*
assume *L1* \neq *L2*

show *L1* \prec_i *L2* \vee *L2* \prec_i *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 $\langle L1 \neq L2 \rangle$
by (*cases L1; cases L2*) (*auto simp add: add-eq-conv-ex*)

qed
qed

corollary *totalp-less-lit*: *totalp* (\prec_i)
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 $\bigwedge x y z. x \preceq_t y \implies y \preceq_t z \implies x \preceq_t z$
 by (*meson transpE transp-less-trm transp-on-reflclp*)
next
 show $\bigwedge x y. x \preceq_t y \implies y \preceq_t x \implies x = y$
 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 $\bigwedge x y z. x \preceq_l y \implies y \preceq_l z \implies x \preceq_l z$
 by (*meson transpE transp-less-lit transp-on-reflclp*)
next
 show $\bigwedge x y. x \preceq_l y \implies y \preceq_l x \implies x = y$
 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  $\bigwedge x y z. x \preceq_c y \implies y \preceq_c z \implies x \preceq_c z$ 
    by (meson transpE transp-less-cls transp-on-reflclp)
next
  show  $\bigwedge x y. x \preceq_c y \implies y \preceq_c x \implies x = y$ 
    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 A1 <l Pos A2  $\longleftrightarrow$  A1 <t A2
  Pos A1 <l Neg A2  $\longleftrightarrow$  A1  $\preceq_t$  A2
  Neg A1 <l Neg A2  $\longleftrightarrow$  A1 <t A2
  Neg A1 <l Pos A2  $\longleftrightarrow$  A1 <t A2
  by (auto simp add: less-lit-def)

```

1.1 Ground Rules

abbreviation *is-maximal-lit* :: 'f gterm literal \Rightarrow 'f gterm clause \Rightarrow bool **where**
is-maximal-lit L M \equiv *is-maximal-in-mset-wrt* (<_l) M L

abbreviation *is-strictly-maximal-lit* :: 'f gterm literal \Rightarrow 'f gterm clause \Rightarrow bool
where
is-strictly-maximal-lit L M \equiv *is-greatest-in-mset-wrt* (<_l) M L

inductive *ground-resolution* ::
'f gterm atom clause \Rightarrow 'f gterm atom clause \Rightarrow 'f gterm atom clause \Rightarrow bool
where

```

ground-resolutionI:
  P1 = add-mset (Neg t) P1'  $\implies$ 
  P2 = add-mset (Pos t) P2'  $\implies$ 
  P2 <c P1  $\implies$ 
  select P1 = {#}  $\wedge$  is-maximal-lit (Neg t) P1  $\vee$  Neg t  $\in$  # select P1  $\implies$ 
  select P2 = {#}  $\implies$ 
  is-strictly-maximal-lit (Pos t) P2  $\implies$ 
  C = P1' + P2'  $\implies$ 
  ground-resolution P1 P2 C

```

inductive *ground-factoring* :: 'f gterm atom clause \Rightarrow 'f gterm atom clause \Rightarrow bool
where

```

ground-factoringI:
  P = add-mset (Pos t) (add-mset (Pos t) P')  $\implies$ 
  select P = {#}  $\implies$ 
  is-maximal-lit (Pos t) P  $\implies$ 

```

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

1.2 Ground Layer

definition $G\text{-Inf} :: 'f\ gterm\ atom\ clause\ inference\ set\ \mathbf{where}$

$G\text{-Inf} =$
 $\{\text{Infer } [P_2, P_1]\ C \mid P_2\ P_1\ C.\ \text{ground-resolution } P_1\ P_2\ C\} \cup$
 $\{\text{Infer } [P]\ C \mid P\ C.\ \text{ground-factoring } P\ C\}$

abbreviation $G\text{-Bot} :: 'f\ gterm\ atom\ clause\ set\ \mathbf{where}$

$G\text{-Bot} \equiv \{\{\#\}\}$

definition $G\text{-entails} :: 'f\ gterm\ atom\ clause\ set \Rightarrow 'f\ gterm\ atom\ clause\ set \Rightarrow bool$
where

$G\text{-entails } N_1\ N_2 \iff (\forall (I :: 'f\ gterm\ set). I \models_s N_1 \longrightarrow I \models_s N_2)$

1.3 Correctness

lemma *soundness-ground-resolution:*

assumes

step: ground-resolution P1 P2 C

shows $G\text{-entails } \{P_1, P_2\} \{C\}$

using *step*

proof (*cases P1 P2 C rule: ground-resolution.cases*)

case (*ground-resolutionI t P1' P2'*)

show *?thesis*

unfolding *G-entails-def true-cls-singleton*

unfolding *true-cls-insert*

proof (*intro allI impI, elim conjE*)

fix $I :: 'f\ gterm\ set$

assume $I \models P_1$ **and** $I \models P_2$

then obtain $K_1\ K_2 :: 'f\ gterm\ atom\ literal\ \mathbf{where}$

$K_1 \in\# P_1$ **and** $I \models_l K_1$ **and** $K_2 \in\# P_2$ **and** $I \models_l K_2$

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

show $I \models C$

proof (*cases K1 = Neg t*)

case *K1-def: True*

hence $I \models_l Neg\ t$

using $\langle I \models_l K_1 \rangle$ **by** *simp*

show *?thesis*

proof (*cases K2 = Pos t*)

case *K2-def: True*

hence $I \models_l Pos\ t$

using $\langle I \models_l K_2 \rangle$ **by** *simp*

hence *False*

using $\langle I \models_l Neg\ t \rangle$ **by** *simp*


```

    thus ?thesis ..
  next
    case False
    hence  $K2 \in\# P_2'$ 
      using  $\langle K2 \in\# P2 \rangle$ 
      unfolding ground-resolutionI by simp
    hence  $I \Vdash P_2'$ 
      using  $\langle I \Vdash K2 \rangle$  by blast
    thus ?thesis
      unfolding ground-resolutionI by simp
  qed
next
  case False
  hence  $K1 \in\# P_1'$ 
    using  $\langle K1 \in\# P1 \rangle$ 
    unfolding ground-resolutionI by simp
  hence  $I \Vdash P_1'$ 
    using  $\langle I \Vdash K1 \rangle$  by blast
  thus ?thesis
    unfolding ground-resolutionI by simp
  qed
qed
qed

```

lemma *soundness-ground-factoring*:

```

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

    show  $I \Vdash C$ 
  proof (cases  $K = Pos\ t$ )
    case True
    hence  $I \Vdash Pos\ t$ 
      using  $\langle I \Vdash K \rangle$  by metis
    thus ?thesis
      unfolding ground-factoringI
      by (metis true-cls-add-mset)
  next
    case False

```

```

    hence  $K \in\# P'$ 
      using  $\langle K \in\# P \rangle$ 
      unfolding ground-factoringI
      by auto
    hence  $K \in\# C$ 
      unfolding ground-factoringI
      by simp
    thus ?thesis
      using  $\langle I \models K \rangle$  by blast
  qed
qed
qed

```

interpretation *G*: *sound-inference-system G-Inf G-Bot G-entails*

proof *unfold-locales*

```

  show  $G\text{-Bot} \neq \{\}$ 

```

```

    by simp

```

next

```

  show  $\bigwedge B N. B \in G\text{-Bot} \implies G\text{-entails } \{B\} N$ 

```

```

    by (simp add: G-entails-def)

```

next

```

  show  $\bigwedge N2 N1. N2 \subseteq N1 \implies G\text{-entails } N1 N2$ 

```

```

    by (auto simp: G-entails-def elim!: true-cls-mono[rotated])

```

next

```

  fix  $N1 N2$  assume ball-G-entails:  $\forall C \in N2. G\text{-entails } N1 \{C\}$ 

```

```

  show  $G\text{-entails } N1 N2$ 

```

```

    unfolding G-entails-def

```

```

  proof (intro allI impI)

```

```

    fix  $I :: 'f \text{ gterm set}$ 

```

```

    assume  $I \models N1$ 

```

```

    hence  $\forall C \in N2. I \models \{C\}$ 

```

```

      using ball-G-entails by (simp add: G-entails-def)

```

```

    then show  $I \models N2$ 

```

```

      by (simp add: true-cls-def)

```

```

  qed

```

next

```

  show  $\bigwedge N1 N2 N3. G\text{-entails } N1 N2 \implies G\text{-entails } N2 N3 \implies G\text{-entails } N1 N3$ 

```

```

    using G-entails-def by simp

```

next

```

  show  $\bigwedge \iota. \iota \in G\text{-Inf} \implies G\text{-entails } (\text{set } (\text{prems-of } \iota)) \{\text{concl-of } \iota\}$ 

```

```

    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 P1' P2')
have  $\forall k \in \#P_2'. k \prec_l \text{Pos } t$ 
  using  $\langle \text{is-strictly-maximal-lit } (\text{Pos } t) P_2 \rangle \langle P_2 = \text{add-mset } (\text{Pos } t) P_2' \rangle$ 
  by (simp add: literal-order.is-greatest-in-mset-iff)
moreover have  $\bigwedge A. \text{Pos } A \prec_l \text{Neg } A$ 
  by (simp add: less-lit-def)
ultimately have  $\forall k \in \#P_2'. k \prec_l \text{Neg } t$ 
  by (metis transp-def transp-less-lit)
hence  $P_2' \prec_c \{\# \text{Neg } t\}$ 
  using one-step-implies-multp[of  $\{\# \text{Neg } t\}$   $P_2' (\prec_l) \{\#\}$ ] by simp
hence  $P_2' + P_1' \prec_c \text{add-mset } (\text{Neg } t) P_1'$ 
  using multp-cancel[of  $(\prec_l) P_1' P_2' \{\# \text{Neg } t\}$ ] 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  $\bigwedge \iota. \iota \in G\text{-Inf} \implies \text{prems-of } \iota \neq []$ 
    by (auto simp: G-Inf-def)
next
fix  $\iota$ 
have concl-of  $\iota \prec_c \text{main-prem-of } \iota$ 
  if  $\iota\text{-def: } \iota = \text{Infer } [P_2, P_1] C$  and
  infer: ground-resolution P1 P2 C

```

```

for  $P_2 P_1 C$ 
unfolding  $\iota$ -def
using infer
using ground-resolution-smaller-conclusion
by simp

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

ultimately show  $\iota \in G\text{-Inf} \implies \text{concl-of } \iota \prec_c \text{ main-prem-of } \iota$ 
unfolding  $G\text{-Inf-def}$ 
by fast
qed

```

1.5 Refutational Completeness

context

fixes $N :: 'f \text{ gterm atom clause set}$

begin

function $\text{production} :: 'f \text{ gterm atom clause} \implies 'f \text{ gterm set}$ **where**

$\text{production } C = \{A \mid A C'\}$.

$C \in N \wedge$

$C = \text{add-mset } (\text{Pos } A) C' \wedge$

$\text{select } C = \{\#\} \wedge$

$\text{is-strictly-maximal-lit } (\text{Pos } A) C \wedge$

$\neg (\bigcup D \in \{D \in N. D \prec_c C\}. \text{production } D) \models C\}$

by simp-all

termination production

proof ($\text{relation } \{(x, y). x \prec_c y\}$)

show $\text{wf } \{(x, y). x \prec_c y\}$

using wfp-less-cls

by (simp add: wfp-def)

next

show $\bigwedge C D. D \in \{D \in N. D \prec_c C\} \implies (D, C) \in \{(x, y). x \prec_c y\}$

by simp

qed

declare $\text{production.simps}[\text{simp del}]$

end

lemma *Uniq-strictly-maximal-lit-in-ground-clc:*

$\exists_{\leq 1} L. \text{is-strictly-maximal-lit } L \ C$

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 *asypm-on (set-mset C) (\prec_l)*

by (auto intro: *asypm-on-subset asypm-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 ($\exists A. \text{production N C} = \{A\}$)

proof –

have $\exists_{\leq 1} A. \text{is-strictly-maximal-lit } (Pos \ A) \ C$

using *Uniq-strictly-maximal-lit-in-ground-clc*

by (metis (*mono-tags, lifting*) *Uniq-def literal.inject(1)*)

hence $\exists_{\leq 1} A. \exists C'. C = \text{add-mset } (Pos \ A) \ C' \wedge \text{is-strictly-maximal-lit } (Pos \ A)$

C

by (*simp add: Uniq-def*)

hence *Uniq-production: $\exists_{\leq 1} A. \exists C'.$*

$C \in N \wedge$

$C = \text{add-mset } (Pos \ A) \ C' \wedge$

select C = {#} \wedge

is-strictly-maximal-lit (Pos A) C \wedge

$\neg (\bigcup D \in \{D \in N. D \prec_c C\}. \text{production N D}) \models C$

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 $A \in \text{production N C}$

shows $\text{production N C} = \{A\}$

using *assms production-eq-empty-or-singleton*

by *force*

definition *interp where*

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

lemma *interp-mempty[simp]: interp N {#} = {}*

proof –

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

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

lemma *mem-productionE*:
assumes *C-prod*: $A \in \text{production } N C$
obtains C' **where**
 $C \in N$ **and**
 $C = \text{add-mset } (\text{Pos } A) C'$ **and**
 $\text{select } C = \{\#\}$ **and**
 $\text{is-strictly-maximal-lit } (\text{Pos } A) C$ **and**
 $\neg \text{interp } N C \models C$
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*: $C \prec_c D \implies \text{production } N C \subseteq \text{interp } N D$
unfolding *interp-def*
using *production-unfold* **by** *blast*

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

is-strictly-maximal-lit (Pos A) C
 $\neg \text{interp } N C \models C$
by (*auto elim!*: *mem-productionE*)

assume *production* N D = {A}
hence A \in *production* N D
by *simp*
then obtain D' **where**
 D \in N
 D = *add-mset* (Pos A) D'
is-strictly-maximal-lit (Pos A) D
 $\neg \text{interp } N D \models D$
by (*auto elim!*: *mem-productionE*)

have $\neg (C \prec_c D)$
proof (*rule notI*)
assume $C \prec_c D$
hence *production* N C \subseteq *interp* N D
using *production-subset-if-less-cls* **by** *metis*
hence A \in *interp* N D
unfolding $\langle \text{production } N C = \{A\} \rangle$ **by** *simp*
hence *interp* N D $\models D$
unfolding $\langle D = \text{add-mset } (Pos A) D' \rangle$ **by** *simp*
with $\langle \neg \text{interp } N D \models D \rangle$ **show** *False*
by *metis*
qed

moreover have $\neg (D \prec_c C)$
proof (*rule notI*)
assume $D \prec_c C$
hence *production* N D \subseteq *interp* N C
using *production-subset-if-less-cls* **by** *metis*
hence A \in *interp* N C
unfolding $\langle \text{production } N D = \{A\} \rangle$ **by** *simp*
hence *interp* N C $\models C$
unfolding $\langle C = \text{add-mset } (Pos A) C' \rangle$ **by** *simp*
with $\langle \neg \text{interp } N C \models C \rangle$ **show** *False*
by *metis*
qed

ultimately show C = D
by *order*
qed

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

lemma *subset-Union-mem-CollectI*: $P x \implies f x \subseteq (\bigcup y \in \{z. P z\}. f y)$
by *blast*

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

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

lemma *split-Union-production*:

assumes *D-in*: $D \in N$

shows $(\bigcup C \in N. \text{production } N C) =$

$\text{interp } N D \cup \text{production } N D \cup (\bigcup C \in \{C \in N. D \prec_c C\}. \text{production } N C)$

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 $(\bigcup C \in N. \text{production } N C) =$

$(\bigcup C \in \{C \in N. C \prec_c D\}. \text{production } N C) \cup \text{production } N D \cup (\bigcup C \in \{C \in N. D \prec_c C\}. \text{production } N C)$

by *auto*

thus $(\bigcup C \in N. \text{production } N C) =$

$\text{interp } N D \cup \text{production } N D \cup (\bigcup C \in \{C \in N. D \prec_c C\}. \text{production } N C)$

by (*simp add: interp-def*)

qed

lemma *split-Union-production'*:

assumes *D-in*: $D \in N$

shows $(\bigcup C \in N. \text{production } N C) = \text{interp } N D \cup (\bigcup C \in \{C \in N. D \preceq_c C\}. \text{production } N C)$

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 $\text{interp } N C = \text{interp } N D \cup (\bigcup C' \in \{C' \in N. D \preceq_c C' \wedge C' \prec_c C\}. \text{production } N C')$

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* $\langle D \prec_c C \rangle$ **show** $D \in \{D \in N. D \prec_c C\}$

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

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

lemma *not-interp-to-Interp-imp-le*: $A \notin \text{interp } N C \implies A \in \text{interp } N D \cup \text{production } N D \implies C \preceq_c D$
 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*: $A \in \text{production } N D$
shows $A \in \text{interp } N C$
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 $\langle \text{Pos } A \in \# D \rangle$.
next
show *Neg A* $\in \# C$
 using $\langle \text{Neg } A \in \# C \rangle$.
next
show *Pos A* $\prec_l \text{Neg } A$
 by (*simp add: less-lit-def subset-implies-multp*)

next
show *is-maximal-lit* (Pos A) D
using $\langle \text{is-strictly-maximal-lit (Pos A) D} \rangle$ **by** *auto*
qed *simp-all*
hence $\neg (\prec_c)^{==} C D$
by (*metis asympD asymp-less-cls reflclp-iff*)
thus *?thesis*
proof (*rule contrapos-np*)
from *D-prod* **show** $A \notin \text{interp } N C \implies (\prec_c)^{==} C D$
using *not-interp-to-Interp-imp-le* **by** *simp*
qed
qed

lemma *neg-notin-Interp-not-produce*:

Neg A $\in \# C \implies A \notin \text{interp } N D \cup \text{production } N D \implies C \preceq_c D \implies A \notin$
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: $D \in N$ **and**
D-entailed: $\text{interp } N D \Vdash D$ **and**
C-in: $C \in N$ **and**
D-lt-C: $D \prec_c C$
shows $\text{interp } N C \Vdash D$
proof –
from *D-entailed* **obtain** *L A* **where**
L-in: $L \in \# D$ **and**
L-eq-disj-L-eq: $L = \text{Pos } A \wedge A \in \text{interp } N D \vee L = \text{Neg } A \wedge A \notin \text{interp } N D$
unfolding *true-cls-def true-lit-iff* **by** *metis*

have $\text{interp } N D \subseteq \text{interp } N C$
using *interp-subset-if-less-cls[OF D-lt-C]* .

from *L-eq-disj-L-eq* **show** $\text{interp } N C \Vdash D$

proof (*elim disjE conjE*)

assume $L = \text{Pos } A$ **and** $A \in \text{interp } N D$

thus $\text{interp } N C \Vdash D$

using *L-in* $\langle \text{interp } N D \subseteq \text{interp } N C \rangle$ **by** *auto*

next

assume $L = \text{Neg } A$ **and** $A \notin \text{interp } N D$

hence $A \notin \text{interp } N C$

using *neg-notin-Interp-not-produce*

by (*smt (verit, ccfv-threshold) L-in UN-E interp-def produces-imp-in-interp*)

thus $\text{interp } N C \Vdash D$

using *L-in* $\langle L = \text{Neg } A \rangle$ **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: $\text{interp } N C \models D$

shows $\text{interp } N C \cup \text{production } N C \models D$

proof –

from *D-entailed* **obtain** $L A$ **where**

L-in: $L \in \# D$ **and**

L-eq-disj-L-eq: $L = \text{Pos } A \wedge A \in \text{interp } N C \vee L = \text{Neg } A \wedge A \notin \text{interp } N C$

unfolding *true-cls-def true-lit-iff* **by** *metis*

from *L-eq-disj-L-eq* **show** $\text{interp } N C \cup \text{production } N C \models D$

proof (*elim disjE conjE*)

assume $L = \text{Pos } A$ **and** $A \in \text{interp } N C$

thus $\text{interp } N C \cup \text{production } N C \models D$

using *L-in* **by** *blast*

next

assume $L = \text{Neg } A$ **and** $A \notin \text{interp } N C$

hence $A \notin \text{interp } N C \cup \text{production } N C$

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 $\text{interp } N C \cup \text{production } N C \models D$

using *L-in* $\langle L = \text{Neg } A \rangle$ **by** *blast*

qed

qed

lemma *lift-entailment-to-Union:*

fixes $N D$

assumes

D-in: $D \in N$ **and**

R_D-entails-D: $\text{interp } N D \models D$

shows

$(\bigcup C \in N. \text{production } N C) \models D$

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: $A \in \text{production } N C$ **and**

L-in: $L \in \# D$

shows

lesseq-trm-if-pos: $\text{is-pos } L \implies \text{atm-of } L \preceq_t A$ **and**

$less\text{-}trm\text{-}if\text{-}neg: is\text{-}neg\ L \implies atm\text{-}of\ L \prec_t A$
proof –
from $C\text{-}prod$ **obtain** C' **where**
 $C\text{-}def: C = add\text{-}mset\ (Pos\ A)\ C'$ **and**
 $C\text{-}max\text{-}lit: is\text{-}strictly\text{-}maximal\text{-}lit\ (Pos\ A)\ C$
by $(auto\ elim: mem\text{-}productionE)$

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

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

moreover have $False$ **if** $Pos\ A \prec_l L$
proof –
have $C \prec_c D$
proof $(rule\ multp\text{-}if\text{-}maximal\text{-}of\text{-}lhs\text{-}is\text{-}less)$
show $Pos\ A \in\#\ C$
by $(simp\ add: C\text{-}def)$
next
show $L \in\#\ D$
using $L\text{-}in$ **by** $simp$
next
show $is\text{-}maximal\text{-}lit\ (Pos\ A)\ C$
using $C\text{-}max\text{-}lit$ **by** $auto$
next
show $Pos\ A \prec_l L$
using $that$ **by** $simp$
qed $simp\text{-}all$
with $\langle D \preceq_c C \rangle$ **show** $False$
by $(metis\ asympD\ reflclp\text{-}iff\ wfp\text{-}imp\text{-}asymp\ wfp\text{-}less\text{-}cls)$

qed

ultimately show $is\text{-}pos\ L \implies atm\text{-}of\ L \preceq_t A$ and $is\text{-}neg\ L \implies atm\text{-}of\ L \prec_t A$
by *metis+*

qed

lemma *less-trm-iff-less-cls-if-mem-production*:

assumes $C\text{-}prod: A_C \in production\ N\ C$ and $D\text{-}prod: A_D \in production\ N\ D$
shows $A_C \prec_t A_D \iff C \prec_c D$

proof –

from $C\text{-}prod$ obtain C' where

$C \in N$ and

$C\text{-}def: C = add\text{-}mset\ (Pos\ A_C)\ C'$ and

$is\text{-}strictly\text{-}maximal\text{-}lit\ (Pos\ A_C)\ C$

by (*elim mem-productionE*) *simp*

hence $\forall L \in \# C'. L \prec_l Pos\ A_C$

by (*simp add: literal-order.is-greatest-in-mset-iff*)

from $D\text{-}prod$ obtain D' where

$D \in N$ and

$D\text{-}def: D = add\text{-}mset\ (Pos\ A_D)\ D'$ and

$is\text{-}strictly\text{-}maximal\text{-}lit\ (Pos\ A_D)\ D$

by (*elim mem-productionE*) *simp*

hence $\forall L \in \# D'. L \prec_l Pos\ A_D$

by (*simp add: literal-order.is-greatest-in-mset-iff*)

show *?thesis*

proof (*rule iffI*)

assume $A_C \prec_t A_D$

hence $Pos\ A_C \prec_l Pos\ A_D$

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

moreover hence $\forall L \in \# C'. L \prec_l Pos\ A_D$

using $\langle \forall L \in \# C'. L \prec_l Pos\ A_C \rangle$

by (*meson transp-less-lit transpD*)

ultimately show $C \prec_c D$

using *one-step-implies-multp*[of $D\ C - \{\#\}$]

by (*simp add: D-def C-def*)

next

assume $C \prec_c D$

hence $production\ N\ C \subseteq (\bigcup (production\ N\ \{x \in N. x \prec_c D\}))$

using $\langle C \in N \rangle$ by *auto*

hence $A_C \in (\bigcup (production\ N\ \{x \in N. x \prec_c D\}))$

using $C\text{-}prod$ by *auto*

hence $A_C \neq A_D$

by (*metis D-prod interp-def mem-productionE true-cls-add-mset true-lit-iff*)

moreover have $\neg (A_D \prec_t A_C)$

by (*metis C-def D-prod* $\langle C \prec_c D \rangle$ *asymptD asympt-less-trm lesseq-trm-if-pos*
literal.disc(1))

literal.sel(1) reflclp-iff union-single-eq-member)

ultimately show $A_C \prec_t A_D$
using *totalp-less-trm*[*THEN totalpD*]
using *C-def D-def* **by** *auto*
qed
qed

lemma *false-cls-if-productive-production*:
assumes *C-prod*: $A \in \text{production } N \ C$ **and** $D \in N$ **and** $C \prec_c D$
shows $\neg \text{interp } N \ D \models C - \{\# \text{Pos } A\# \}$
proof –
from *C-prod* **obtain** C' **where**
C-in: $C \in N$ **and**
C-def: $C = \text{add-mset } (\text{Pos } A) \ C'$ **and**
select $C = \{\#\}$ **and**
Pox-A-max: *is-strictly-maximal-lit* $(\text{Pos } A) \ C$ **and**
 $\neg \text{interp } N \ C \models C$
by (*rule mem-productionE*) *blast*

from $\langle D \in N \rangle \langle C \prec_c D \rangle$ **have** $A \in \text{interp } N \ D$
using *C-prod production-subset-if-less-cls* **by** *auto*

from $\langle D \in N \rangle$ **have** $\text{interp } N \ D \subseteq (\bigcup D \in N. \text{production } N \ D)$
by (*auto simp: interp-def*)

have $\neg \text{interp } N \ D \models C'$
unfolding *true-cls-def Set.bex-simps*
proof (*intro ballI*)
fix L **assume** *L-in*: $L \in \# \ C'$
hence $L \in \# \ C$
by (*simp add: C-def*)

have $C' \prec_c C$
by (*simp add: C-def subset-implies-multp*)
hence $C' \preceq_c C$
by *order*

show $\neg \text{interp } N \ D \models_l L$
proof (*cases L*)
case $(\text{Pos } A_L)$
moreover have $A_L \notin \text{interp } N \ D$
proof –
have $\forall y \in \# \ C'. y \prec_l \text{Pos } A$
using *Pox-A-max*
by (*simp add: C-def literal-order.is-greatest-in-mset-iff*)
with *Pos* **have** $A_L \notin \text{insert } A \ (\text{interp } N \ C)$
using *L-in* $\langle \neg \text{interp } N \ C \models C \rangle$ *C-def*
by *blast*

moreover have $A_L \notin (\bigcup D' \in \{D' \in N. C \prec_c D' \wedge D' \prec_c D\}. \text{production})$

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

moreover have $interp\ N\ D =$
 $insert\ A\ (interp\ N\ C) \cup (\bigcup D' \in \{D' \in N. C \prec_c D' \wedge D' \prec_c D\}. production$
 $N\ D')$

$N D'$
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 $\dots = (\bigcup D' \in \{D' \in \{y \in N. y \prec_c C\} \cup \{C\} \cup \{y \in N. C \prec_c$
 $y\}. D' \prec_c D\}. production\ N\ D')$
 using *partition-set-around-element*[*of* $N\ (\prec_c)$, *OF* - $\langle C \in N \rangle$]
totalp-on-subset[*OF* *totalp-less-cls, simplified*]
 by *simp*
 also have $\dots = (\bigcup D' \in \{y \in N. y \prec_c C \wedge y \prec_c D\} \cup \{C\} \cup \{y \in N.$
 $C \prec_c y \wedge y \prec_c D\}. production\ N\ D')$
 using $\langle C \prec_c D \rangle$ **by** *auto*
 also have $\dots = (\bigcup D' \in \{y \in N. y \prec_c C\} \cup \{C\} \cup \{y \in N. C \prec_c y \wedge y$
 $\prec_c D\}. production\ N\ D')$
 by (*metis* (*no-types, opaque-lifting*) *assms*(3) *transpD transp-less-cls*)
 also have $\dots = interp\ N\ C \cup production\ N\ C \cup (\bigcup D' \in \{y \in N. C \prec_c$
 $y \wedge y \prec_c D\}. production\ N\ D')$
 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* $\langle L \in \# C \rangle \langle C \prec_c D \rangle \langle \neg interp\ N\ C \models C \rangle$ *interp-subset-if-less-cls*
 by *blast*
ultimately show *?thesis*
 by *simp*
qed

qed
thus $\neg \text{interp } N D \models C - \{\#Pos A\# \}$
by (*simp add: C-def*)
qed

lemma *production-subset-Union-production*:
 $\bigwedge C N. C \in N \implies \text{production } N C \subseteq (\bigcup D \in N. \text{production } N D)$
by *auto*

lemma *interp-subset-Union-production*:
 $\bigwedge C N. C \in N \implies \text{interp } N C \subseteq (\bigcup D \in N. \text{production } N D)$
by (*simp add: split-Union-production'*)

lemma *model-construction*:
fixes
 $N :: 'f \text{ gterm atom clause set and}$
 $C :: 'f \text{ gterm atom clause}$
assumes *G.saturated N and* $\{\#\} \notin N$ **and** *C-in: C ∈ N*
shows
 $\text{production } N C = \{\} \longleftrightarrow \text{interp } N C \models C$
 $(\bigcup D \in N. \text{production } N D) \models C$
 $D \in N \implies C \prec_c D \implies \text{interp } N D \models C$
unfolding *atomize-conj atomize-imp*
using *wfP-less-cls C-in*
proof (*induction C arbitrary: D rule: wfp-induct-rule*)
case (*less C*)
note $IH = \text{less.IH}$

from $\langle \{\#\} \notin N \rangle \langle C \in N \rangle$ **have** $C \neq \{\#\}$
by *metis*

define $I :: 'f \text{ gterm set where}$
 $I = \text{interp } N C$

have $i: \text{interp } N C \models C \longleftrightarrow (\text{production } N C = \{\})$

proof (*rule iffI*)

show $\text{interp } N C \models C \implies \text{production } N C = \{\}$

by (*smt (z3) Collect-empty-eq interp-def production.elims*)

next

assume $\text{production } N C = \{\}$

show $\text{interp } N C \models C$

proof (*cases* $\exists A. \text{Neg } A \in \# C \wedge (\text{Neg } A \in \# \text{select } C \vee \text{select } C = \{\#\} \wedge \text{is-maximal-lit } (\text{Neg } A) C)$)

case *ex-neg-lit-sel-or-max: True*

then obtain A **where**

$\text{Neg } A \in \# C$ **and**

$\text{sel-or-max: } \text{Neg } A \in \# \text{select } C \vee \text{select } C = \{\#\} \wedge \text{is-maximal-lit } (\text{Neg } A)$

C

by *metis*


```

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

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

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

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

have  $\iota \in G\text{-Inf}$ 
  using reso
  by (auto simp only: \iota-def G-Inf-def)

moreover have  $\bigwedge t. t \in \text{set } (\text{prems-of } \iota) \implies t \in N$ 
  using  $\langle C \in N \rangle \langle D \in N \rangle$ 
  by (auto simp add: \iota-def)

```

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

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

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

moreover have $\neg \text{interp } N \ D \models D'$
using $\langle \neg \text{interp } N \ D \models D \rangle$
using $D\text{-def}$ **by force**

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

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

ultimately show $\text{interp } N \ C \models C$
unfolding $C\text{-def}$ **by fast**

next
case False
thus *?thesis*
using $\langle \text{Neg } A \in \# \ C \rangle$
by (*auto simp add: true-cls-def*)

qed

next
case False
hence $\text{select } C = \{\#\}$
using $\text{select-subset select-negative-lits}$
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.prem
      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 $\bigwedge t. t \in \text{set}(\text{prems-of } \iota) \implies t \in N$
using $\langle C \in N \rangle$
by (*auto simp add: ι -def*)

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

moreover have $\forall D \in DD. \text{interp } N C \Vdash D$
using $IH[\text{THEN } \text{conjunct2}, \text{THEN } \text{conjunct2}, \text{rule-format}, \text{of } - C]$
using $\langle C \in N \rangle DD\text{-subset ball-}DD\text{-lt-}C$
by *blast*

ultimately have $\text{interp } N C \Vdash \text{add-mset } (\text{Pos } A) C'$
using $DD\text{-entails-concl}$
unfolding $G.\text{entails-def}$
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 $\text{iaa: } (\bigcup (\text{production } N \text{ ' } N)) \Vdash C$
using $\text{production-eq-empty-or-singleton}[of } N C]$
proof (*elim disjE exE*)
assume $\text{production } N C = \{ \}$
hence $\text{interp } N C \Vdash C$
by (*simp only: i*)
thus *?thesis*
using $\text{lift-entailment-to-Union}[OF } \langle C \in N \rangle]$ **by** *argo*

next
fix A
assume $\text{production } N C = \{ A \}$
hence $\text{prod: } A \in \text{production } N C$
by *simp*

from prod **have** $\text{Pos } A \in \# C$

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

moreover from *prod* **have** $A \in \bigcup$ (*production* N ' N)
using $\langle C \in N \rangle$ *production-subset-Union-production*
by *fast*

ultimately show *?thesis*
by *blast*

qed

moreover have *iib: interp* N $D \Vdash C$ **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 $C \Vdash C$

unfolding *i* **by** *simp*

thus *?thesis*

using *lift-interp-entails*[*OF* $\langle C \in N \rangle$ - *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** $A \in$ *production* (*fset* N) C

shows $\bigwedge A'. A' \prec_t A \implies A' \notin (\bigcup D \in$ *fset* $N.$ *production* (*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 $A \in$ *production* (*fset* N) C

shows $\bigwedge A'. A' \prec_t A \implies A' \notin$ *interp* (*fset* N) $C \implies A' \notin (\bigcup D \in$ *fset* $N.$ *production* (*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 $\bigwedge A'. A' \prec_t$ *atm-of* $L \implies A' \notin$ *interp* (*fset* N) $C \implies A' \notin (\bigcup D \in$ *fset* $N.$ *production* (*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* $L \in$ *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 $A \in \text{interp } N C \longleftrightarrow A \in \text{interp } N D$
proof (*rule iffI*)
show $A \in \text{interp } N C \implies A \in \text{interp } N D$
using *assms(3) interp-subset-if-less-cls* **by** *auto*
next
assume $A \in \text{interp } N D$

then obtain E **where** $A \in \text{production } N E$ **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 $\text{Pos } A \prec_l L$
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.multip_{HO}-if-maximal-less-that-maximal multip_{DM}-imp-multip_{HO}-imp-multip_{DM}*
by *blast*

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

thus $A \in \text{interp } N C$
using $\langle A \in \text{production } N E \rangle$ **by** *auto*
qed

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

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

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

shows $f \text{ ' } X = g \text{ ' } X$
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* $N1 \ C = \text{production } N2 \ C$

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* $N1 \ \{D \in N2. D \prec_c C\} = \text{production } N2 \ \{D \in N2. D \prec_c C\}$

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.prem*s $\langle x \prec_c C \rangle$ *clause-order.order.strict-trans1* **by** *blast*

ultimately **show** *production* $N1 \ x = \text{production } N2 \ x$

using *hyps.IH*[*rule-format*, *of* x] **by** *metis*

qed

hence *BBB*: *interp* $N1 \ C = \text{interp } N2 \ C$

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* $N1 \ C = \text{interp } N2 \ C$

proof –

have *BBB*: *production* $N1 \ \{D \in N2. D \prec_c C\} = \text{production } N2 \ \{D \in N2. D \prec_c C\}$

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*
 $interp' N \equiv (\bigcup C \in N. production N C)$

lemma *interp-eq-interp'*: $interp N D = interp' \{C \in N. C \prec_c D\}$
proof –
have $interp N D = interp \{C \in N. C \prec_c D\} D$
proof (*rule interp-swap-clause-set*)
show $\{Da \in N. Da \prec_c D\} = \{Da \in \{C \in N. C \prec_c D\}. Da \prec_c D\}$
by *blast*
qed

also have $\dots = interp' \{C \in N. C \prec_c D\}$
unfolding *interp-def interp'-def* **by** *blast*

finally show *?thesis* .
qed

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

lemma *Interp-swap-clause-set*:
assumes $agree: \{D \in N1. D \preceq_c C\} = \{D \in N2. D \preceq_c C\}$
shows $interp N1 C \cup production N1 C = interp N2 C \cup production N2 C$
using *production-swap-clause-set[OF agree]*
using *interp-swap-clause-set*
using *agree*
by *blast*

lemma *production-insert-greater-clause*:
assumes $C \prec_c D$
shows $production (insert D N) C = production N C$
proof (*rule production-swap-clause-set*)
show $\{Da \in insert D N. (\prec_c)^{==} Da C\} = \{D \in N. (\prec_c)^{==} D C\}$
using $\langle C \prec_c D \rangle$ **by** *auto*
qed

lemma *interp-insert-greater-clause-strong*:
assumes $C \preceq_c D$

shows $\text{interp } (\text{insert } D \ N) \ C = \text{interp } N \ C$
proof (rule *interp-swap-clause-set*)
show $\{x \in \text{insert } D \ N. x \prec_c C\} = \{x \in N. x \prec_c C\}$
using $\langle C \preceq_c D \rangle$ **by** *auto*
qed

lemma *interp-insert-greater-clause*:
assumes $C \prec_c D$
shows $\text{interp } (\text{insert } D \ N) \ C = \text{interp } N \ C$
proof (rule *interp-swap-clause-set*)
show $\{x \in \text{insert } D \ N. x \prec_c C\} = \{x \in N. x \prec_c C\}$
using $\langle C \prec_c D \rangle$ **by** *auto*
qed

lemma *Interp-insert-greater-clause*:
assumes $C \prec_c D$
shows $\text{interp } (\text{insert } D \ N) \ C \cup \text{production } (\text{insert } D \ N) \ C = \text{interp } N \ C \cup$
 $\text{production } N \ C$
proof (rule *Interp-swap-clause-set*)
show $\{Da \in \text{insert } D \ N. (\prec_c)^{==} Da \ C\} = \{D \in N. (\prec_c)^{==} D \ C\}$
using $\langle C \prec_c D \rangle$ **by** *auto*
qed

lemma *production-add-irrelevant-clause-to-set0*:
assumes
fin: *finite* N **and**
D-irrelevant: $E \in N \ E \subset\# \ D \ \text{set-mset } D = \text{set-mset } E$ **and**
no-select: $\text{select } E = \{\#\}$
shows $\text{production } (\text{insert } D \ N) \ D = \{\}$
proof –
from *D-irrelevant* **have** $E \prec_c D$
using *subset-implies-multp* **by** *metis*
hence *prod-E-subset*: $\text{production } (\text{insert } D \ N) \ E \subseteq \text{interp } (\text{insert } D \ N) \ D$
using *production-subset-if-less-cl* **by** *metis*

show *?thesis*
proof (*cases* $\text{production } (\text{insert } D \ N) \ E = \{\}$)
case *True*
hence $(\nexists A. \text{is-strictly-maximal-lit } (\text{Pos } A) \ E) \vee \text{interp } (\text{insert } D \ N) \ E \models E$
unfolding *production-unfold*
using *no-select*
by (*smt* (*verit*, *del-insts*) *Collect-empty-eq* $\langle E \in N \rangle$ *diff-single-eq-union* *insertI2*
literal-order.is-greatest-in-mset-iff)
hence $(\nexists A. \text{is-strictly-maximal-lit } (\text{Pos } A) \ D) \vee \text{interp } (\text{insert } D \ N) \ D \models D$
proof (*elim disjE*)
assume $\nexists A. \text{is-strictly-maximal-lit } (\text{Pos } A) \ E$
hence $\nexists A. \text{is-strictly-maximal-lit } (\text{Pos } A) \ D$
proof (rule *contrapos-nn*)
show $\exists A. \text{is-strictly-maximal-lit } (\text{Pos } A) \ D \implies \exists A. \text{is-strictly-maximal-lit}$

```

(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 ... =  $\bigcup$  (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 ... =  $\bigcup$  (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:  $\forall D \in N'. \exists E \in N. E \subset\# D \wedge \text{set-mset } D = \text{set-mset } E$  and
    no-select:  $\bigwedge C. \text{select } C = \{\#\}$ 
  shows production (N  $\cup$  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:  $\forall D \in N'. \exists E \in N. E \subset\# D \wedge \text{set-mset } D = \text{set-mset } E$  and
    no-select:  $\bigwedge C. \text{select } C = \{\#\}$ 
  shows production (N  $\cup$  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  $D\text{-lt}: D \prec_c E$  and  $C\text{-entailed}: \text{interp } N D \cup \text{production } N D \models C$ 
  shows  $\text{interp } N E \models C$ 
proof -
  from  $C\text{-entailed}$  obtain  $L$  where  $L \in \# C$  and  $\text{interp } N D \cup \text{production } N D \models_l L$ 
  by (auto simp: true-cls-def)

  show ?thesis
  proof (cases L)
    case (Pos A)
    hence  $A \in \text{interp } N D \cup \text{production } N D$ 
    using  $\langle \text{interp } N D \cup \text{production } N D \models_l L \rangle$  by simp
    moreover from  $D\text{-lt}$  have  $\text{interp } N D \cup \text{production } N D \subseteq \text{interp } N E$ 
    using less-imp-Interp-subseteq-interp by blast
    ultimately have  $A \in \text{interp } N E$ 
    by auto
    thus ?thesis
    using Pos  $\langle L \in \# C \rangle$  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  $\langle L \in \# C \rangle \langle \text{interp } N D \cup \text{production } N D \models_l L \rangle$  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  $C\text{-le}: C \preceq_c D$  and  $D\text{-lt}: D \prec_c E$  and  $C\text{-entailed}: \text{interp } N D \cup \text{production } N D \models C$ 
  shows  $\text{interp } N E \cup \text{production } N E \models C$ 
proof -
  from  $C\text{-entailed}$  obtain  $L$  where  $L \in \# C$  and  $\text{interp } N D \cup \text{production } N D \models_l L$ 

```

by (auto simp: true-cls-def)

show ?thesis

proof (cases L)

case (Pos A)

hence $A \in \text{interp } N D \cup \text{production } N D$

using $\langle \text{interp } N D \cup \text{production } N D \models_l L \rangle$ by simp

moreover from D-lt have $\text{interp } N D \cup \text{production } N D \subseteq \text{interp } N E \cup$
 $\text{production } N E$

using less-imp-Interp-subseteq-interp by blast

ultimately have $A \in \text{interp } N E \cup \text{production } N E$

by auto

thus ?thesis

using Pos $\langle L \in \# C \rangle$ 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 $\langle L \in \# C \rangle \langle \text{interp } N D \cup \text{production } N D$
 $\models_l L \rangle$ 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: $C \prec_c D$ and C-entailed: $\text{interp } N C \cup \text{production } N C \models C$

shows $\text{interp } N D \models C$

using lesser-entailed-clause-stays-entailed'[OF clause-order.order-refl assms] .

lemma entailed-clause-stays-entailed:

assumes C-lt: $C \prec_c D$ and C-entailed: $\text{interp } N C \cup \text{production } N C \models C$

shows $\text{interp } N D \cup \text{production } N D \models C$

using lesser-entailed-clause-stays-entailed[OF clause-order.order-refl assms] .

lemma multp-if-all-left-smaller: $M2 \neq \{\#\} \implies \forall k \in \#M1. \exists j \in \#M2. R k j \implies$
 $\text{multp } R M1 M2$

using one-step-implies-multp

by (metis add-0)

lemma

fixes

$P1 :: 'f \text{ gterm}$ and

$C1 :: 'f \text{ gterm clause}$ and

$N :: 'f \text{ gterm clause set}$

defines

$C1 \equiv \{\#Neg P1\#}$ and

$N \equiv \{C1\}$

assumes

```

    no-select:  $\bigwedge C. \text{select } C = \{\#\}$ 
  shows
    False
proof -
  have interp N C1 = {}
    unfolding interp-def N-def by simp
  have production N C1 = {}
    unfolding production-unfold  $\langle \text{interp } N \ C1 = \{\}\rangle$  C1-def by simp
  hence interp N C1  $\cup$  production N C1  $\models$  C1
    unfolding  $\langle \text{interp } N \ C1 = \{\}\rangle$   $\langle \text{production } N \ C1 = \{\}\rangle$ 
    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  $\equiv$  {#Pos P1, Neg P2#} and
    N  $\equiv$  {C1}
  assumes
    term-order: P1  $\prec_t$  P2 and
    no-select:  $\bigwedge C. \text{select } C = \{\#\}$ 
  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  $\langle \text{interp } N \ C1 = \{\}\rangle$  by simp
  hence interp N C1  $\cup$  production N C1  $\models$  C1
    unfolding  $\langle \text{interp } N \ C1 = \{\}\rangle$   $\langle \text{production } N \ C1 = \{\}\rangle$ 
    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  $\equiv$  {#Neg P1, Neg P2#} and
    C2  $\equiv$  {#Pos P2, Neg P3#} and

```


$C3 \equiv \{\#Pos P1, Pos P2, Pos P4\# \}$ **and**
 $C4 \equiv \{\#Pos P2, Pos P3, Pos P4\# \}$ **and**
 $C5 \equiv \{\#Pos P2, Neg P4\# \}$ **and**
 $N \equiv \{C1, C2, C3, C4, C5\}$

assumes

term-order: $P1 \prec_t P2 P2 \prec_t P3 P3 \prec_t P4$ **and**
no-select: $\bigwedge C. select C = \{\#\}$

shows

$C1 \prec_c C2 C2 \prec_c C3 C3 \prec_c C4 C4 \prec_c C5$

proof –

have *lit-order*: $Pos P1 \prec_l Neg P1 Neg P1 \prec_l Pos P2 Pos P2 \prec_l Neg P2 Neg P2$
 $\prec_l Pos P3$
 $Pos P3 \prec_l Neg P3 Neg P3 \prec_l Pos P4 Pos P4 \prec_l Neg P4$
using *term-order* **by** *simp-all*

show $C1 \prec_c C2$

unfolding *C1-def C2-def*

proof (*rule multp-if-all-left-smaller*)

show $\{\#Pos P2, Neg P3\# \} \neq \{\#\}$

by *simp*

next

show $\forall k \in \#\{\#Neg P1, Neg P2\#\}. \exists j \in \#\{\#Pos P2, Neg P3\#\}. k \prec_l j$

using *lit-order* **by** *simp*

qed

moreover show $C2 \prec_c C3$

unfolding *C2-def C3-def*

proof (*rule multp-if-all-left-smaller*)

show $\{\#Pos P1, Pos P2, Pos P4\# \} \neq \{\#\}$

by *simp*

next

show $\forall k \in \#\{\#Pos P2, Neg P3\#\}. \exists j \in \#\{\#Pos P1, Pos P2, Pos P4\#\}. k \prec_l j$

using *lit-order* **by** *auto*

qed

moreover show $C3 \prec_c C4$

proof –

have $\{\#Pos P1, Pos P2\# \} \prec_c \{\#Pos P2, Pos P3\# \}$

proof (*rule multp-if-all-left-smaller*)

show $\{\#Pos P2, Pos P3\# \} \neq \{\#\}$

by *simp*

next

show $\forall k \in \#\{\#Pos P1, Pos P2\#\}. \exists j \in \#\{\#Pos P2, Pos P3\#\}. k \prec_l j$

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  $C_4 \prec_c C_5$ 
  unfolding  $C_4$ -def  $C_5$ -def
proof (rule multp-if-all-left-smaller)
  show  $\{\#Pos\ P_2, Neg\ P_4\} \neq \{\#\}$ 
  by simp
next
  show  $\forall k \in \{\#Pos\ P_2, Pos\ P_3, Pos\ P_4\}. \exists j \in \{\#Pos\ P_2, Neg\ P_4\}. k \prec_l$ 
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  $N\ C1 \cup$  production  $N\ C1 \models C1$ 
  unfolding  $\langle$ interp  $N\ C1 = \{\}\rangle$   $\langle$ production  $N\ C1 = \{\}\rangle$ 
  unfolding  $C1$ -def
  unfolding true-cls-def true-lit-def
  by (simp add:  $C1$ -def)

have  $\{D \in N. D \prec_c C2\} = \{C1\}$ 
  unfolding  $N$ -def
  using cls-order by auto
hence interp  $N\ C2 =$  interp  $N\ C1 \cup$  production  $N\ C1$ 
  unfolding  $\langle$ interp  $N\ C1 = \{\}\rangle$ 
  unfolding interp-def
  by simp
hence interp  $N\ C2 = \{\}$ 
  unfolding  $\langle$ interp  $N\ C1 = \{\}\rangle$   $\langle$ production  $N\ C1 = \{\}\rangle$  by simp
hence production  $N\ C2 = \{\}$ 
  unfolding production-unfold
  by (simp add:  $C2$ -def)
hence interp  $N\ C2 \cup$  production  $N\ C2 \models C2$ 
  using  $\langle$ interp  $N\ C2 = \{\}\rangle$ 
  by (simp add:  $C2$ -def)

have  $\{D \in N. D \prec_c C3\} = \{C1, C2\}$ 
  unfolding  $N$ -def

```

using *cls-order* **by** *auto*
hence $\text{interp } N \ C3 = \text{interp } N \ C2 \cup \text{production } N \ C2$
unfolding $\langle \text{interp } N \ C2 = \{\} \rangle$
unfolding *interp-def*
by (*simp add: production N C1 = {}*)
hence $\text{interp } N \ C3 = \{\}$
unfolding $\langle \text{interp } N \ C2 = \{\} \rangle \langle \text{production } N \ C2 = \{\} \rangle$ **by** *simp*
have $\text{production } N \ C3 = \{P4\}$
proof –
have $C3 \in N$
by (*simp add: N-def*)
moreover **have** $\exists C3'. C3 = \text{add-mset } (Pos \ P4) \ C3'$
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** $\neg \text{interp } N \ C3 \models C3$
unfolding $\langle \text{interp } N \ C3 = \{\} \rangle$
unfolding *C3-def*
by *simp*
ultimately **have** $P4 \in \text{production } N \ C3$
unfolding *production-unfold*
using *no-select*
by *simp*
thus *?thesis*
using *production-eq-empty-or-singleton* **by** *fastforce*
qed
hence $\text{interp } N \ C3 \cup \text{production } N \ C3 \models C3$
using $\langle \text{interp } N \ C3 = \{\} \rangle$
by (*simp add: C3-def*)

have $\{D \in N. D \prec_c C4\} = \{C1, C2, C3\}$
unfolding *N-def*
using *cls-order* **by** *auto*
hence $\text{interp } N \ C4 = \text{interp } N \ C3 \cup \text{production } N \ C3$
unfolding $\langle \text{interp } N \ C3 = \text{interp } N \ C2 \cup \text{production } N \ C2 \rangle$
unfolding $\langle \text{interp } N \ C2 = \text{interp } N \ C1 \cup \text{production } N \ C1 \rangle$
unfolding $\langle \text{interp } N \ C1 = \{\} \rangle$
unfolding *interp-def*
by *auto*
hence $\text{interp } N \ C4 = \{P4\}$
using $\langle \text{interp } N \ C3 = \{\} \rangle \langle \text{production } N \ C3 = \{P4\} \rangle$ **by** *simp*
hence $\text{interp } N \ C4 \models C4$
using *C4-def* **by** *simp*
hence $\text{production } N \ C4 = \{\}$
unfolding *production-unfold* **by** *simp*
hence $\text{interp } N \ C4 \cup \text{production } N \ C4 \models C4$
using $\langle \text{interp } N \ C4 = \{P4\} \rangle$
by (*simp add: C4-def*)

have $\{D \in N. D \prec_c C5\} = \{C1, C2, C3, C4\}$
unfolding *N-def*
using *cls-order by auto*
hence $\text{interp } N \ C5 = \text{interp } N \ C4 \cup \text{production } N \ C4$
unfolding $\langle \text{interp } N \ C4 = \text{interp } N \ C3 \cup \text{production } N \ C3 \rangle$
unfolding $\langle \text{interp } N \ C3 = \text{interp } N \ C2 \cup \text{production } N \ C2 \rangle$
unfolding $\langle \text{interp } N \ C2 = \text{interp } N \ C1 \cup \text{production } N \ C1 \rangle$
unfolding $\langle \text{interp } N \ C1 = \{\} \rangle$
unfolding *interp-def*
by *auto*
hence $\text{interp } N \ C5 = \{P4\}$
using $\langle \text{interp } N \ C4 = \{P4\} \rangle \langle \text{production } N \ C4 = \{\} \rangle$ **by** *simp*
have $\text{production } N \ C5 = \{\}$
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 $\bigwedge A. \neg \text{is-strictly-maximal-lit } (Pos \ A) \ C5$
by *(meson Uniq-D literal-order.Uniq-is-greatest-in-mset literal.distinct(1))*
thus *?thesis*
unfolding *production-unfold by simp*
qed
hence $\neg \text{interp } N \ C5 \cup \text{production } N \ C5 \models C5$
unfolding $\langle \text{interp } N \ C5 = \{P4\} \rangle \langle \text{production } N \ C5 = \{\} \rangle$
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 $B :: 'f \text{ gterm atom clause and } N :: 'f \text{ gterm atom clause set}$

assume $B \in G\text{-Bot and } G.\text{saturated } N$

hence $B = \{\#\}$

by *simp*

assume $G\text{-entails } N \ \{B\}$

hence $\{\#\} \in N$

unfolding $\langle B = \{\#\} \rangle$

proof *(rule contrapos-pp)*

assume $\{\#\} \notin N$

define $I :: 'f \text{ gterm set where}$

$I = (\bigcup D \in N. \text{production } N \ D)$

show $\neg G\text{-entails } N \ G\text{-Bot}$

unfolding *G-entails-def not-all not-imp*

```

proof (intro exI conjI)
  show  $I \models_s N$ 
    unfolding  $I\text{-def}$ 
    using  $\text{model-construction}(2)[OF \langle G.\text{saturated } N \rangle \langle \{\#\} \notin N \rangle]$ 
    by ( $\text{simp add: true-cls-def}$ )
  next
    show  $\neg I \models_s G\text{-Bot}$ 
      by  $\text{simp}$ 
    qed
  qed
thus  $\exists B' \in G\text{-Bot}. B' \in N$ 
  by  $\text{auto}$ 
qed

end

end
theory  $\text{Lower-Set}$ 
  imports  $\text{Main}$ 
begin

definition  $\text{is-lower-set-wrt} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$  where
   $\text{transp-on } X R \Longrightarrow \text{asyp-on } X R \Longrightarrow$ 
   $\text{is-lower-set-wrt } R L X \longleftrightarrow L \subseteq X \wedge (\forall l \in L. \forall x \in X. R x l \longrightarrow x \in L)$ 

definition  $\text{is-strict-lower-set-wrt} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ 
where
   $\text{transp-on } X R \Longrightarrow \text{asyp-on } X R \Longrightarrow$ 
   $\text{is-strict-lower-set-wrt } R L X \longleftrightarrow L \subset X \wedge (\forall l \in L. \forall x \in X. R x l \longrightarrow x \in L)$ 

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

lemma  $\text{is-lower-set-wrt-refl}$ :
  fixes  $X :: 'a \text{ set}$  and  $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
  assumes  $\text{transp-on } X R$  and  $\text{asyp-on } X R$ 
  shows  $\text{is-lower-set-wrt } R X X$ 
  unfolding  $\text{is-lower-set-wrt-def}[OF \text{assms}]$  by  $\text{simp}$ 

lemma  $\text{is-lower-set-wrt-trans}$ :
  fixes  $X Y Z :: 'a \text{ set}$  and  $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
  assumes
     $\text{transp-on } Z R$  and  $\text{asyp-on } Z R$  and
     $\text{is-lower-set-wrt } R X Y$  and  $\text{is-lower-set-wrt } R Y Z$ 
  shows  $\text{is-lower-set-wrt } R X Z$ 
proof –

```

```

have  $Y \subseteq Z$  and  $Y\text{-lower}$ :  $\forall l \in Y. \forall x \in Z. R\ x\ l \longrightarrow x \in Y$ 
  using  $\langle is\text{-lower-set-wrt}\ R\ Y\ Z \rangle$ 
  unfolding  $is\text{-lower-set-wrt-def}[OF\ \langle transp\text{-on}\ Z\ R \rangle\ \langle asymp\text{-on}\ Z\ R \rangle]$ 
  unfolding  $atomize\text{-conj}$  .

have  $transp\text{-on}\ Y\ R$  and  $asymp\text{-on}\ Y\ R$ 
  unfolding  $atomize\text{-conj}$ 
  using  $\langle transp\text{-on}\ Z\ R \rangle\ \langle asymp\text{-on}\ Z\ R \rangle\ \langle Y \subseteq Z \rangle$ 
  by ( $metis\ asymp\text{-on-subset}\ transp\text{-on-subset}$ )

have  $X \subseteq Y$  and  $X\text{-lower}$ :  $\forall l \in X. \forall x \in Y. R\ x\ l \longrightarrow x \in X$ 
  using  $\langle is\text{-lower-set-wrt}\ R\ X\ Y \rangle$ 
  unfolding  $is\text{-lower-set-wrt-def}[OF\ \langle transp\text{-on}\ Y\ R \rangle\ \langle asymp\text{-on}\ Y\ R \rangle]$ 
  unfolding  $atomize\text{-conj}$  .

have  $X \subseteq Z$ 
  using  $\langle X \subseteq Y \rangle\ \langle Y \subseteq Z \rangle$  by  $order$ 

moreover have ( $\forall l \in X. \forall x \in Z. R\ x\ l \longrightarrow x \in X$ )
  using  $\langle X \subseteq Y \rangle\ X\text{-lower}\ Y\text{-lower}$  by  $blast$ 

ultimately show  $?thesis$ 
  unfolding  $is\text{-lower-set-wrt-def}[OF\ \langle transp\text{-on}\ Z\ R \rangle\ \langle asymp\text{-on}\ Z\ R \rangle]$  ..
qed

lemma  $is\text{-lower-set-wrt-antisym}$ :
  fixes  $X\ Y :: 'a\ set$  and  $R :: 'a \Rightarrow 'a \Rightarrow bool$ 
  assumes
     $transp\text{-on}\ Y\ R$  and  $asymp\text{-on}\ Y\ R$  and
     $is\text{-lower-set-wrt}\ R\ X\ Y$  and  $is\text{-lower-set-wrt}\ R\ Y\ X$ 
  shows  $X = Y$ 
proof –
  have  $X \subseteq Y$ 
    using  $\langle is\text{-lower-set-wrt}\ R\ X\ Y \rangle$ 
    unfolding  $is\text{-lower-set-wrt-def}[OF\ \langle transp\text{-on}\ Y\ R \rangle\ \langle asymp\text{-on}\ Y\ R \rangle]$  ..

  have  $transp\text{-on}\ X\ R$  and  $asymp\text{-on}\ X\ R$ 
    unfolding  $atomize\text{-conj}$ 
    using  $\langle transp\text{-on}\ Y\ R \rangle\ \langle asymp\text{-on}\ Y\ R \rangle\ \langle X \subseteq Y \rangle$ 
    by ( $metis\ asymp\text{-on-subset}\ transp\text{-on-subset}$ )

  have  $Y \subseteq X$ 
    using  $\langle is\text{-lower-set-wrt}\ R\ Y\ X \rangle$ 
    unfolding  $is\text{-lower-set-wrt-def}[OF\ \langle transp\text{-on}\ X\ R \rangle\ \langle asymp\text{-on}\ X\ R \rangle]$  ..

  show  $X = Y$ 
    using  $\langle X \subseteq Y \rangle\ \langle Y \subseteq X \rangle$  by ( $metis\ subset\text{-antisym}$ )
qed

```

lemma *order-is-lower-set-wrt*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
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: $\bigwedge A. \text{transp-on } A \ R$*
using $\langle \text{transp } R \rangle$ **by** (*metis subset-UNIV transp-on-subset*)

have *asym: $\bigwedge A. \text{asymp-on } A \ R$*
using $\langle \text{asymp } R \rangle$ **by** (*metis subset-UNIV asymp-on-subset*)

show $\bigwedge x \ y. \text{is-strict-lower-set-wrt } R \ x \ y = (\text{is-lower-set-wrt } R \ x \ y \wedge \neg \text{is-lower-set-wrt } R \ y \ x)$
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 $\bigwedge x. \text{is-lower-set-wrt } R \ x \ x$
using *is-lower-set-wrt-refl[OF trans asym]* .

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

show $\bigwedge x \ y. \text{is-lower-set-wrt } R \ x \ y \Longrightarrow \text{is-lower-set-wrt } R \ y \ x \Longrightarrow x = y$
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 $\langle \text{transp-on } (\text{insert } x \ X) \ R \rangle$ $\langle \text{asymp-on } (\text{insert } x \ X) \ R \rangle$
by (*metis asymp-on-subset subset-insertI transp-on-subset*)

have *is-lower-set-wrt R (insert x L) X \longleftrightarrow*
 $\text{insert } x \ L \subseteq X \wedge (\forall l \in \text{insert } x \ L. \forall xa \in X. R \ xa \ l \longrightarrow xa \in \text{insert } x \ L)$
unfolding *is-lower-set-wrt-def[OF $\langle \text{transp-on } X \ R \rangle$ $\langle \text{asymp-on } X \ R \rangle$..*

also have $\dots \longleftrightarrow x \in X \wedge L \subseteq X \wedge (\forall l \in \text{insert } x \ L. \forall xa \in X. R \ xa \ l \longrightarrow xa \in \text{insert } x \ L)$
by *simp*

also have $\dots \longleftrightarrow x \in X \wedge L \subseteq X \wedge (\forall w \in X. R \ w \ x \longrightarrow w \in \text{insert } x \ L) \wedge$
 $(\forall l \in L. \forall xa \in X. R \ xa \ l \longrightarrow xa \in \text{insert } x \ L)$
by *simp*

also have $\dots \longleftrightarrow x \in X \wedge L \subseteq X \wedge (\forall w \in X. R w x \longrightarrow w \in L) \wedge$
 $(\forall l \in L. \forall y \in X. R y l \longrightarrow y \in \text{insert } x L)$
using $\langle \text{asympt-on } (\text{insert } x X) R \rangle$ **by** $(\text{smt } (\text{verit}) \text{ asympt-onD insertCI insertE})$

also have $\dots \longleftrightarrow x \in X \wedge (\forall w \in X. R w x \longrightarrow w \in L) \wedge \text{is-lower-set-wrt } R L$
 X
using $\langle \text{is-lower-set-wrt } R L X \rangle$
unfolding $\text{is-lower-set-wrt-def}[OF \langle \text{transp-on } X R \rangle \langle \text{asympt-on } X R \rangle]$
by blast

finally show $?thesis$
using $\text{assms}(\mathcal{B}-)$ **by** argo
qed

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

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

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

sorted-wrt-append subsetD)

lemma *not-in-lower-set-wrtI*:

fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

assumes *trans*: *trans-on* $Y R$ **and** *asym*: *asympt-on* $Y R$

shows *is-lower-set-wrt* $R X Y \implies y \notin X \implies y \in Y \implies R y z \implies z \notin X$

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 trans-on-less asympt-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 *trans* ($<$)

using *trans-on-less* .

next

show *asympt* ($<$)

using *asympt-on-less* .

qed

end

lemmas (**in** *preorder*) *is-lower-set-empty*[*simp*] =

is-lower-set-wrt-empty[*OF trans-on-less asympt-on-less*]

lemmas (**in** *preorder*) *is-lower-set-insertI* =

is-lower-set-wrt-insertI[*OF trans-on-less asympt-on-less*]

lemmas (**in** *preorder*) *lower-set-appendI* =

lower-set-wrt-appendI[*OF trans-on-less asympt-on-less*]

lemmas (**in** *preorder*) *sorted-and-lower-set-appendD-left* =

sorted-and-lower-set-wrt-appendD-left[*OF trans-on-less asympt-on-less*]

lemmas (**in** *preorder*) *sorted-and-lower-set-appendD-right* =

sorted-and-lower-set-wrt-appendD-right[*OF trans-on-less asympt-on-less*]

lemmas (**in** *preorder*) *not-in-lower-setI* =

not-in-lower-set-wrtI[*OF trans-on-less asympt-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 { $x. P x$ }
     $\forall x. \textit{finite}$  { $y. Q x y$ }
  shows finite { $y. \exists x. P x \wedge Q x y$ }
  using assms by auto

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

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

lemma sorted-wrt-dropWhile: sorted-wrt  $R$   $xs \implies \textit{sorted-wrt}$   $R$  (dropWhile  $P$   $xs$ )
  by (auto dest: sorted-wrt-drop simp: dropWhile-eq-drop)

lemma sorted-wrt-takeWhile: sorted-wrt  $R$   $xs \implies \textit{sorted-wrt}$   $R$  (takeWhile  $P$   $xs$ )
  by (subst takeWhile-eq-take) (auto dest: sorted-wrt-take)

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-asymp:  $\forall y \in \text{set } xs. R x y \longrightarrow \neg R y x$  and asymp-on (set xs) R
using Cons.prem1
unfolding atomize-conj
by (metis asymp-on-def list.set-intros(1) list.set-intros(2))

have R-x:  $\forall y \in \text{set } xs. R x y$  and sorted-wrt R xs
using Cons.prem2
unfolding atomize-conj sorted-wrt.simps
by argo

have x  $\notin$  set xs
proof (intro notI)
assume x-in:  $x \in \text{set } xs$ 

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

moreover hence  $\neg R x x$ 
using R-x-asymp x-in by metis

ultimately show False
by contradiction
qed

moreover have distinct xs
using Cons.IH  $\langle \text{asymp-on (set xs) R} \rangle \langle \text{sorted-wrt R xs} \rangle$  by argo

ultimately show ?case
unfolding distinct.simps by argo
qed

lemma dropWhile-append-eq-rhs:
fixes xs ys :: 'a list and P :: 'a  $\Rightarrow$  bool
assumes
 $\bigwedge x. x \in \text{set } xs \Longrightarrow P x$  and
 $\bigwedge y. y \in \text{set } ys \Longrightarrow \neg P y$ 
shows dropWhile P (xs @ ys) = ys
using assms
proof (induction xs)
case Nil
then show ?case
by (metis append-Nil dropWhile-eq-self-iff hd-in-set)
next
case (Cons x xs)
then show ?case
by (metis dropWhile-append dropWhile-cong dropWhile-eq-self-iff member-rec(2))

```

qed

lemma *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone:*

fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ **and** $xs :: 'a \text{ list}$ **and** $P :: 'a \Rightarrow \text{bool}$

assumes *sorted-wrt* R xs **and** *monotone-on* $(\text{set } xs)$ R (\geq) P

shows $x \in \text{set } (\text{dropWhile } P \ xs) \iff \neg P \ x \wedge x \in \text{set } xs$

using *assms*

proof (*induction* xs)

case *Nil*

show *?case*

by *simp*

next

case (*Cons* y xs)

have $\forall z \in \text{set } xs. R \ y \ z$ **and** *sorted-wrt* R xs

using $\langle \text{sorted-wrt } R \ (y \ \# \ xs) \rangle$ **by** *simp-all*

moreover **have** *monotone-on* $(\text{set } xs)$ R (\geq) P

using $\langle \text{monotone-on } (\text{set } (y \ \# \ xs)) \ R \ (\geq) \ P \rangle$

by (*metis monotone-on-subset set-subset-Cons*)

ultimately **have** *IH*: $(x \in \text{set } (\text{dropWhile } P \ xs)) = (\neg P \ x \wedge x \in \text{set } xs)$

using *Cons.IH* $\langle \text{sorted-wrt } R \ xs \rangle$ **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.prem1 Cons.prem2 le-boolD list.set-intros1*)

monotone-on-def

set-ConsD sorted-wrt.simps2)

qed

qed

lemma *ball-set-dropWhile-if-sorted-wrt-and-monotone-on:*

fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ **and** $xs :: 'a \text{ list}$ **and** $P :: 'a \Rightarrow \text{bool}$

assumes *sorted-wrt* R xs **and** *monotone-on* $(\text{set } xs)$ R (\geq) P

shows $\forall x \in \text{set } (\text{dropWhile } P \ xs). \neg P \ x$

using *assms*

proof (*induction* xs)

case *Nil*

show *?case*

```

    by simp
next
case (Cons x xs)
have  $\forall y \in \text{set } xs. R \ x \ y$  and sorted-wrt R xs
  using  $\langle \text{sorted-wrt } R \ (x \ \# \ xs) \rangle$  by simp-all

moreover have monotone-on (set xs) R ( $\geq$ ) P
  using  $\langle \text{monotone-on } (\text{set } (x \ \# \ xs)) \ R \ (\geq) \ P \rangle$ 
  by (metis monotone-on-subset set-subset-Cons)

ultimately have  $\forall x \in \text{set } (\text{dropWhile } P \ xs). \neg P \ x$ 
  using Cons.IH  $\langle \text{sorted-wrt } R \ xs \rangle$  by metis

moreover have  $\neg P \ x \implies \neg P \ y$  if  $y \in \text{set } xs$  for  $y$ 
proof -
  have  $x \in \text{set } (x \ \# \ xs)$ 
  by simp
  moreover have  $y \in \text{set } (x \ \# \ xs)$ 
  using  $\langle y \in \text{set } xs \rangle$  by simp
  moreover have  $R \ x \ y$ 
  using  $\langle \forall y \in \text{set } xs. R \ x \ y \rangle \langle y \in \text{set } xs \rangle$  by metis
  ultimately have  $P \ y \leq P \ x$ 
  using  $\langle \text{monotone-on } (\text{set } (x \ \# \ xs)) \ R \ (\geq) \ P \rangle$  [unfolded monotone-on-def] by
metis
  thus  $\neg P \ x \implies \neg P \ y$ 
  by simp
qed

ultimately show ?case
  by simp
qed

lemma filter-set-eq-filter-set-minus-singleton:
  assumes  $\neg P \ y$ 
  shows  $\{x \in X. P \ x\} = \{x \in X - \{y\}. P \ x\}$ 
  using assms by blast

lemma ex1-subset-eq-image-if-bij-betw:
  fixes  $f :: 'a \Rightarrow 'b$  and  $X :: 'a \text{ set}$  and  $Y :: 'b \text{ set}$ 
  assumes bij-betw  $f \ X \ Y$  and  $Y' \subseteq Y$ 
  shows  $\exists! X'. X' \subseteq X \wedge Y' = f \ ' X'$ 
  using assms
  by (metis bij-betw-def inv-into-image-cancel subset-image-iff)

lemma Collect-eq-image-filter-Collect-if-bij-betw:
  fixes  $f :: 'a \Rightarrow 'b$  and  $X :: 'a \text{ set}$  and  $Y :: 'b \text{ set}$ 
  assumes bij: bij-betw  $f \ X \ Y$  and sub:  $\{y. P \ y\} \subseteq Y$ 
  shows  $\{y. P \ y\} = f \ ' \{x. x \in X \wedge P \ (f \ x)\}$ 
  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 (*in linorder*) *ex1-sorted-list-for-set-if-finite*:

finite X $\implies \exists!xs. \text{sorted-wrt } (<) xs \wedge \text{set } xs = X$

by (*metis local.sorted-list-of-set.finite-set-strict-sorted local.strict-sorted-equal*)

lemma *restrict-map-ident-if-dom-subset*: *dom M $\subseteq A \implies \text{restrict-map } M A = M$*

by (*metis domIff ext in-mono restrict-map-def*)

lemma *dropWhile-ident-if-pred-always-false*:

assumes $\bigwedge x. x \in \text{set } xs \implies \neg P x$

shows *dropWhile P xs = xs*

using *assms dropWhile-eq-self-iff hd-in-set* **by** *auto*

1.6 Move to HOL.Transitive-Closure

lemma *relpoup-right-unique*:

fixes *R :: 'a \Rightarrow 'a \Rightarrow bool* **and** *n :: nat* **and** *x y z :: 'a*

assumes *runique*: $\bigwedge x y z. R x y \implies R x z \implies y = z$

shows $(R \rightsquigarrow n) x y \implies (R \rightsquigarrow n) x z \implies y = z$

proof (*induction n arbitrary: x y z*)

case *0*

thus *?case*

by *simp*

next

case (*Suc n'*)

then obtain *x' :: 'a* **where**

$(R \rightsquigarrow n') x x'$ **and** $R x' y$ **and** $R x' z$

by *auto*

thus $y = z$

using *runique* **by** *simp*

qed

lemma *Uniq-relpoup*:

fixes *n :: nat* **and** *R :: 'a \Rightarrow 'a \Rightarrow bool*

assumes *runiq*: $\forall x. \exists_{\leq 1} y. R x y$

shows $\exists_{\leq 1} y. (R \rightsquigarrow n) x y$

proof (*rule Uniq-I*)

fix *y z*

assume $(R \rightsquigarrow n) x y$ **and** $(R \rightsquigarrow n) x z$

show $y = z$

proof (*rule relpoup-right-unique*)

show $\bigwedge x y z. R x y \implies R x z \implies y = z$

using *runiq* **by** (*auto dest: Uniq-D*)

next

show $(R \rightsquigarrow n) x y$

using $\langle (R \rightsquigarrow n) x y \rangle$.

next

```

    show  $(R \rightsquigarrow n) x z$ 
      using  $\langle (R \rightsquigarrow n) x z \rangle$  .
  qed
qed

lemma relpowp-plus-of-right-unique:
  assumes
    right-unique  $R$ 
     $(R \rightsquigarrow m) x y$  and
     $(R \rightsquigarrow (m + n)) x z$ 
  shows  $(R \rightsquigarrow n) y z$ 
  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  $(R \rightsquigarrow (m + n)) x z$ 
  shows  $\exists y. (R \rightsquigarrow m) x y \wedge (R \rightsquigarrow n) y z$ 
  using assms
  proof (induction  $m$  arbitrary:  $x$ )
    case 0
      thus ?case
        by simp
    next
      case (Suc m)

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

      obtain  $y'$  where  $(R \rightsquigarrow m) y y'$  and  $(R \rightsquigarrow n) y' z$ 
        using Suc.IH[OF  $\langle (R \rightsquigarrow (m + n)) y z \rangle$ ] by metis

      show ?case
      proof (intro exI conjI)
        show  $(R \rightsquigarrow \text{Suc } m) x y'$ 
          using  $\langle R x y \rangle \langle (R \rightsquigarrow m) y y' \rangle$  by (metis relpowp-Suc-I2)
        next
          show  $(R \rightsquigarrow n) y' z$ 
            using  $\langle (R \rightsquigarrow n) y' z \rangle$  .
      qed
  qed

```

lemma *relpowp-Suc-of-right-unique*:

assumes

right-unique R

$R\ x\ y$ **and**

$(R\ \overset{\sim}{\sim}\ \text{Suc}\ n)\ x\ z$

shows $(R\ \overset{\sim}{\sim}\ n)\ y\ z$

using *assms*

by (*metis relpowp-Suc-D2 right-uniqueD*)

lemma *tranclp-if-relpowp*: $n \neq 0 \implies (R\ \overset{\sim}{\sim}\ n)\ x\ y \implies R^{++}\ x\ y$

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* :: $('a \Rightarrow \text{bool}) \Rightarrow 'a\ \text{option}$ **where**

The-optional $P = (\text{if } \exists!x.\ P\ x\ \text{then } \text{Some}\ (\text{THE}\ x.\ P\ x)\ \text{else } \text{None})$

lemma *The-optional-eq-SomeD*: *The-optional* $P = \text{Some}\ x \implies P\ x$

unfolding *The-optional-def*

by (*metis option.discI option.inject theI-unique*)

lemma *Some-eq-The-optionalD*: *Some* $x = \text{The-optional}\ P \implies P\ x$


```

using The-optional-eq-SomeD by metis

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

lemma None-eq-The-optionalD: None = The-optional P  $\implies$   $\nexists!$ 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  $\exists_{\leq 1} y. \text{full-run } R x y$ 
  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  $\exists! y. \text{full-run } R x y$ 
proof -
  have  $\exists_{\leq 1} y. \text{full-run } R x y$ 
    using Uniq-full-run[of R x] Uniq-R by argo

  moreover have  $\exists y. \text{full-run } R x y$ 
    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:  $\bigwedge x y. R x y \implies P x \implies P y$ 
  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  $\longleftrightarrow$  (is-pos L  $\longleftrightarrow$  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: $A \subseteq\# B \implies (multp R)^{==} A B$*

by (*metis reflclp-iff strict-subset-implies-multp subset-mset.le-imp-less-or-eq*)

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

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

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

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

3 Move to *HOL-Library.FSet*

declare *wfP-pfsubset[intro]*

syntax

-FFilter :: *p*trn \Rightarrow 'a *fset* \Rightarrow bool \Rightarrow 'a *fset* ((1{|-| ∈| -/ -|})

translations

{|x| ∈| X. P|} == *CONST ffilter* ($\lambda x. P$) X

lemma *fimage-ffUnion*: $f \ |^{\dagger} \ \text{ffUnion } SS = \text{ffUnion } ((|^{\dagger}) f \ |^{\dagger} \ SS)$

proof (*intro fsubset-antisym fsubsetI*)

fix *x* **assume** $x \in\# \ f \ |^{\dagger} \ \text{ffUnion } SS$

then obtain *y* **where** $y \in\# \ \text{ffUnion } SS$ **and** $x = f \ y$

by *auto*

thus $x \in\# \ \text{ffUnion } ((|^{\dagger}) f \ |^{\dagger} \ SS)$

unfolding *fmember-ffUnion-iff*

by (*metis UN-E ffUnion.rep-eq fimage-eqI*)

next

fix *x* **assume** $x \in\# \ \text{ffUnion } ((|^{\dagger}) f \ |^{\dagger} \ SS)$

then obtain *S* **where** $S \in\# \ SS$ **and** $x \in\# \ f \ |^{\dagger} \ S$

unfolding *fmember-ffUnion-iff* **by** *auto*

then show $x \in\# \ f \ |^{\dagger} \ \text{ffUnion } SS$

by (*metis ffUnion-fsubset-iff fimage-mono fin-mono fsubsetI*)

qed

lemma *ffilter-eq-ffilter-minus-singleton*:

assumes $\neg P \ y$

shows $\{|x| \in\# \ X. P \ x|\} = \{|x| \in\# \ X - \{|y|\}. P \ x|\}$

using *assms* **by** (*induction X*) *auto*

lemma *fun-upd-fimage*: $f(x := y) \ |^{\dagger} \ A = (\text{if } x \in\# \ A \ \text{then } \text{finsert } y \ (f \ |^{\dagger} \ (A - \{|x|\})) \ \text{else } f \ |^{\dagger} \ A)$

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]*: $\text{ffilter } P \{\{\}\} = \{\{\}\}$
by (*metis ex-fin-conv ffilter-member-filter*)

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

lemma (**in** *linorder*) *ex1-sorted-list-for-fset*:
 $\exists!xs. \text{sorted-wrt } (<) \ xs \wedge \text{fset-of-list } xs = X$
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 $x \in\mid X \ P \ x$
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 :: $'state1 \Rightarrow 'state1 \Rightarrow \text{bool}$ **and**
step2 :: $'state2 \Rightarrow 'state2 \Rightarrow \text{bool}$ **and**
final1 :: $'state1 \Rightarrow \text{bool}$ **and**
final2 :: $'state2 \Rightarrow \text{bool}$ +
fixes
match :: $'state1 \Rightarrow 'state2 \Rightarrow \text{bool}$ **and**
measure :: $'state1 \Rightarrow 'index$ **and**
order :: $'index \Rightarrow 'index \Rightarrow \text{bool}$ (**infix** \sqsubset 70)
assumes
wfp-order:
wfp (\sqsubset) **and**
match-final:
 $\text{match } s1 \ s2 \implies \text{final1 } s1 \implies \text{final2 } s2$ **and**
simulation:
 $\text{match } s1 \ s2 \implies \text{step1 } s1 \ s1' \implies$
 $(\exists s2'. \text{step2}^{++} \ s2 \ s2' \wedge \text{match } s1' \ s2') \vee (\text{match } s1' \ s2 \wedge \text{measure } s1' \sqsubset$
 $\text{measure } s1)$
begin
sublocale *forward-simulation* **where**
step1 = *step1* **and** *step2* = *step2* **and** *final1* = *final1* **and** *final2* = *final2* **and**
order = *order* **and**
match = $\lambda i \ x \ y. i = \text{measure } x \wedge \text{match } x \ y$
proof *unfold-locales*

```

show  $\bigwedge i s1 s2. i = \text{measure } s1 \wedge \text{match } s1 s2 \implies \text{final1 } s1 \implies \text{final2 } s2$ 
using match-final by metis
next
show  $\bigwedge i s1 s2 s1'. i = \text{measure } s1 \wedge \text{match } s1 s2 \implies \text{step1 } s1 s1' \implies$ 
 $(\exists i' s2'. \text{step2}^{++} s2 s2' \wedge i' = \text{measure } s1' \wedge \text{match } s1' s2') \vee$ 
 $(\exists i'. (i' = \text{measure } s1' \wedge \text{match } s1' s2) \wedge i' \sqsubset i)$ 
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  $\Rightarrow$  '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 :: 'state2  $\Rightarrow$  'index and
  order :: 'index  $\Rightarrow$  'index  $\Rightarrow$  bool (infix  $\sqsubset$  70)
assumes
  wfp-order:
    wfp ( $\sqsubset$ ) and
  match-final:
    match s1 s2  $\implies$  final2 s2  $\implies$  final1 s1 and
  simulation:
    match s1 s2  $\implies$  step2 s2 s2'  $\implies$ 
       $(\exists s1'. \text{step1}^{++} s1 s1' \wedge \text{match } s1' s2') \vee (\text{match } s1 s2' \wedge \text{measure } s2' \sqsubset$ 
measure s2)
begin

sublocale backward-simulation where
  step1 = step1 and step2 = step2 and final1 = final1 and final2 = final2 and
  order = order and
  match =  $\lambda i x y. i = \text{measure } y \wedge \text{match } x y$ 
proof unfold-locales
show  $\bigwedge i s1 s2. i = \text{measure } s2 \wedge \text{match } s1 s2 \implies \text{final2 } s2 \implies \text{final1 } s1$ 
using match-final by metis
next
show  $\bigwedge i1 s1 s2 s2'. i1 = \text{measure } s2 \wedge \text{match } s1 s2 \implies \text{step2 } s2 s2' \implies$ 
 $(\exists i2 s1'. \text{step1}^{++} s1 s1' \wedge i2 = \text{measure } s2' \wedge \text{match } s1' s2') \vee$ 
 $(\exists i2. (i2 = \text{measure } s2' \wedge \text{match } s1 s2') \wedge i2 \sqsubset i1)$ 
using simulation by metis

```

next
 show *wfp* (\square)
 using *wfp-order* .
qed

end

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

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

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

definition *trail-true-cls* :: (- literal \times - option) list \Rightarrow - clause \Rightarrow bool **where**
trail-true-cls Γ $C \longleftrightarrow (\exists L \in \# C. \text{trail-true-lit } \Gamma L)$

definition *trail-false-cls* :: (- literal \times - option) list \Rightarrow - clause \Rightarrow bool **where**
trail-false-cls Γ $C \longleftrightarrow (\forall L \in \# C. \text{trail-false-lit } \Gamma L)$

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

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

lemma *trail-defined-lit-iff*: *trail-defined-lit* Γ $L \longleftrightarrow \text{atm-of } L \in \text{atm-of } \text{' fst } \text{' set } \Gamma$
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 \times - option) list \Rightarrow - clause \Rightarrow bool **where**
trail-defined-cls Γ $C \longleftrightarrow (\forall L \in \# C. \text{trail-defined-lit } \Gamma L)$

lemma *trail-defined-lit-iff-true-or-false*:
trail-defined-lit Γ $L \longleftrightarrow \text{trail-true-lit } \Gamma L \vee \text{trail-false-lit } \Gamma L$
unfolding *trail-defined-lit-def trail-false-lit-def trail-true-lit-def* **by** (*rule refl*)

lemma *trail-true-or-false-cls-if-defined*:
trail-defined-cls Γ $C \Longrightarrow \text{trail-true-cls } \Gamma C \vee \text{trail-false-cls } \Gamma C$
unfolding *trail-defined-cls-def trail-false-cls-def trail-true-cls-def*
unfolding *trail-defined-lit-iff-true-or-false*
by *blast*

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

```

fix M
assume M-in:  $M \in \# C$ 

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

from M-in tr-false have tr-false-lit-M: trail-false-lit ((L, Cl) #  $\Gamma$ ) M
  unfolding trail-false-cls-def by simp
thus trail-false-lit  $\Gamma$  M
  unfolding trail-false-lit-def
  using M-neq-L
  by (cases L; cases M) (simp-all add: trail-interp-def trail-false-lit-def)
qed

inductive trail-consistent :: ('a literal  $\times$  'b option) list  $\Rightarrow$  bool where
  Nil[simp]: trail-consistent [] |
  Cons:  $\neg$  trail-defined-lit  $\Gamma$  L  $\Longrightarrow$  trail-consistent  $\Gamma$   $\Longrightarrow$  trail-consistent ((L, u) #  $\Gamma$ )

lemma distinct-lits-if-trail-consistent:
  trail-consistent  $\Gamma$   $\Longrightarrow$  distinct (map fst  $\Gamma$ )
by (induction  $\Gamma$  rule: trail-consistent.induct)
  (simp-all add: image-comp trail-defined-lit-def)

lemma trail-true-lit-if-trail-true-suffix:
  suffix  $\Gamma' \Gamma$   $\Longrightarrow$  trail-true-lit  $\Gamma' K$   $\Longrightarrow$  trail-true-lit  $\Gamma K$ 
by (meson image-mono set-mono-suffix subsetD trail-true-lit-def)

lemma trail-true-cls-if-trail-true-suffix:
  suffix  $\Gamma' \Gamma$   $\Longrightarrow$  trail-true-cls  $\Gamma' C$   $\Longrightarrow$  trail-true-cls  $\Gamma C$ 
using trail-true-cls-def trail-true-lit-if-trail-true-suffix by metis

lemma trail-false-lit-if-trail-false-suffix:
  suffix  $\Gamma' \Gamma$   $\Longrightarrow$  trail-false-lit  $\Gamma' K$   $\Longrightarrow$  trail-false-lit  $\Gamma K$ 
by (meson image-mono set-mono-suffix subsetD trail-false-lit-def)

lemma trail-false-cls-if-trail-false-suffix:
  suffix  $\Gamma' \Gamma$   $\Longrightarrow$  trail-false-cls  $\Gamma' C$   $\Longrightarrow$  trail-false-cls  $\Gamma C$ 
using trail-false-cls-def trail-false-lit-if-trail-false-suffix by metis

lemma trail-defined-lit-if-trail-defined-suffix:
  suffix  $\Gamma' \Gamma$   $\Longrightarrow$  trail-defined-lit  $\Gamma' K$   $\Longrightarrow$  trail-defined-lit  $\Gamma K$ 
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  $\Gamma' \Gamma$   $\Longrightarrow$  trail-defined-cls  $\Gamma' C$   $\Longrightarrow$  trail-defined-cls  $\Gamma C$ 
unfolding trail-defined-cls-def by (metis trail-defined-lit-if-trail-defined-suffix)

lemma not-trail-true-lit-and-trail-false-lit:

```

```

fixes  $\Gamma :: ('a \text{ literal} \times 'b \text{ option}) \text{ list}$  and  $L :: 'a \text{ literal}$ 
shows  $\text{trail-consistent } \Gamma \implies \neg (\text{trail-true-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma L)$ 
proof (induction  $\Gamma$  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  $\Gamma K$  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  $\Gamma :: ('a \text{ literal} \times 'b \text{ option}) \text{ list}$  and  $C :: 'a \text{ clause}$ 
  shows  $\text{trail-consistent } \Gamma \implies \neg (\text{trail-true-cls } \Gamma C \wedge \text{trail-false-cls } \Gamma C)$ 
proof (induction  $\Gamma$  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  $\Gamma L u$ )
  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  $\Gamma$ 
  shows  $\neg (\text{trail-false-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma (\neg L))$ 
  using assms
proof (induction  $\Gamma$ )
  case Nil
  show ?case
  by (simp add: trail-false-lit-def)
next
  case (Cons  $\Gamma K u$ )
  show ?case
  proof (cases  $K = L \vee K = \neg L$ )
    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 $\neg (L \in\# C \wedge - L \in\# C)$
proof (*rule notI*)
assume $L \in\# C \wedge - L \in\# C$

hence *trail-false-lit* Γ $L \wedge$ *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 $\exists_{\leq 1} C.$ *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 $\exists_{\leq 1} C.$ *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 $\bigwedge x y. \text{ground-factoring}^{-1-1} x y \implies x \prec_c y$
 using *ground-factoring-smaller-conclusion* by *simp*
next
 show *wfP* (\prec_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 $\text{atms-of } C \subseteq \text{atms-of } P_1 \cup \text{atms-of } P_2$
 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 $\text{set-mset } P = \text{set-mset } C$
 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 $\text{atms-of } C = \text{atms-of } P$
 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 $\text{is-maximal-lit } L P = \text{is-maximal-lit } L C$
 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 $\text{is-maximal-lit } L P = \text{is-maximal-lit } L C$
 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
 $\text{is-maximal-lit } L P$ and $\text{count } P L = \text{Suc } (\text{Suc } n)$
 shows $\text{is-maximal-lit } L C$ and $\text{count } C L = \text{Suc } n$
 unfolding *atomize-conj*
 using $\langle \text{ground-factoring } P C \rangle$
proof (cases $P C$ rule: *ground-factoring.cases*)
 case (*ground-factoringI* $t P'$)

then show *is-maximal-lit* $L C \wedge \text{count } C L = \text{Suc } n$
by (*metis* *assms*(1) *assms*(3) *assms*(4) *asympt-on-less-lit* *count-add-mset*
ground-factorizing-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-factorizing* $\sim m$) $P C$ **and** $m \leq \text{Suc } n$ **and** *is-pos* L **and**
is-maximal-lit $L P$ **and** $\text{count } P L = \text{Suc } (\text{Suc } n)$
shows *is-maximal-lit* $L C$ **and** $\text{count } C L = \text{Suc } (\text{Suc } n - m)$
unfolding *atomize-conj*
using $\langle (\text{ground-factorizing } \sim m) P C \rangle \langle m \leq \text{Suc } n \rangle$
proof (*induction* m *arbitrary*: C)
case 0
hence $P = C$
by *simp*
then show ?*case*
using *assms*(4,5) **by** *simp*
next
case ($\text{Suc } m$)
then show ?*case*
by (*metis* *Suc-diff-le* *Suc-leD* *assms*(3) *diff-Suc-Suc* *ground-factorizing-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-factorizing* $\sim \text{Suc } n$) $P C$ **and** *L-pos*: *is-pos* L **and**
L-max: *is-maximal-lit* $L P$ **and** *L-count*: $\text{count } P L = \text{Suc } (\text{Suc } n)$
shows *is-maximal-lit* $L C$ **and** $\text{count } C L = \text{Suc } 0$
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* $\text{Suc } 0 P C 0$] **by** *simp*
next
case ($\text{Suc } n$)
from *Suc.prem*s **obtain** P' **where**
ground-factorizing $P P'$ **and** (*ground-factorizing* $\sim \text{Suc } n$) $P' C$
by (*metis* *relpowp-Suc-D2*)

from *Suc.prem*s **have** *is-maximal-lit* $L P'$ **and** $\text{count } P' L = \text{Suc } (\text{Suc } n)$
using *ground-factorizing-reduces-maximal-pos-lit*[*OF* $\langle \text{ground-factorizing } P P' \rangle$
L-pos] **by** *simp-all*

thus ?*case*
using *Suc.IH*[*OF* $\langle (\text{ground-factorizing } \sim \text{Suc } n) P' C \rangle$] **by** *metis*
qed

7 Move somewhere?

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

lemma *lift-tranclp-to-pairs-with-constant-fst*:
 $(R\ x)^{++}\ y\ z \implies (\lambda(x, y)\ (x', z).\ x = x' \wedge R\ x\ y\ z)^{++}\ (x, y)\ (x, z)$
by (*induction* *z arbitrary*: *rule: tranclp-induct*) (*auto simp: tranclp.trancl-into-trancl*)

abbreviation (*in preorder*) *is-lower-fset* **where**
is-lower-fset $X\ Y \equiv$ *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 $\langle zs = xs\ @\ ys \rangle$
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 $\langle zs = xs\ @\ ys \rangle$
by (*smt* (*verit, del-insts*) *Un-iff* $\langle zs = xs\ @\ ys \rangle$ *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*]: $I \Vdash \text{repeat-mset } (Suc\ n)\ C \longleftrightarrow I \Vdash C$
by (*induction n*) *simp-all*

lemma (in *backward-simulation*)
assumes *match i S1 S2* and $\neg L1.\text{inf-step } S1$
shows $\neg L2.\text{inf-step } S2$
using *assms match-inf* by *metis*

lemma (in *scl-fol-calculus*) *grounding-of-cls-ground*:
assumes *is-ground-cls N*
shows *grounding-of-cls N = N*

proof –

have *grounding-of-cls N* = $(\bigcup C \in N. \text{grounding-of-cls } C)$
unfolding *grounding-of-cls-def* by *simp*
also have $\dots = (\bigcup C \in N. \{C\})$
using $\langle \text{is-ground-cls } N \rangle$
by (*simp add: is-ground-cls-def grounding-of-cls-ground*)
also have $\dots = N$
by *simp*
finally show *?thesis* .

qed

lemma (in *scl-fol-calculus*) *propagateI'*:
 $C \Vdash N \mid U \implies C = \text{add-mset } L\ C' \implies \text{is-ground-cls } (C \cdot \gamma) \implies$
 $\forall K \in \# C \cdot \gamma. \text{atm-of } K \preceq_B \beta \implies$
 $C_0 = \{\#K \in \# C'. K \cdot l\ \gamma \neq L \cdot l\ \gamma\} \implies C_1 = \{\#K \in \# C'. K \cdot l\ \gamma = L \cdot l\ \gamma\}$
 $\gamma\} \implies$
 $SCL-FOL.\text{trail-false-cls } \Gamma (C_0 \cdot \gamma) \implies \neg SCL-FOL.\text{trail-defined-lit } \Gamma (L \cdot l\ \gamma)$
 \implies
 $\text{is-ingu } \mu \{\text{atm-of ' set-mset } (\text{add-mset } L\ C_1)\} \implies$
 $\Gamma' = \text{trail-propagate } \Gamma (L \cdot l\ \mu) (C_0 \cdot \mu)\ \gamma \implies$
 $\text{propagate } N\ \beta (\Gamma, U, \text{None}) (\Gamma', U, \text{None})$
using *propagateI* by *metis*

lemma (in *scl-fol-calculus*) *decideI'*:
 $\text{is-ground-lit } (L \cdot l\ \gamma) \implies \neg SCL-FOL.\text{trail-defined-lit } \Gamma (L \cdot l\ \gamma) \implies \text{atm-of } L \cdot a$
 $\gamma \preceq_B \beta \implies$
 $\Gamma' = \text{trail-decide } \Gamma (L \cdot l\ \gamma) \implies$
 $\text{decide } N\ \beta (\Gamma, U, \text{None}) (\Gamma', U, \text{None})$
by (*auto intro!: decideI*)

lemma *ground-iff-vars-term-empty*: $\text{ground } t \longleftrightarrow \text{vars-term } t = \{\}$
proof (*rule iffI*)

show $ground\ t \implies vars-term\ t = \{\}$
by (*rule ground-vars-term-empty*)
next
show $vars-term\ t = \{\} \implies ground\ t$
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* :: $'f\ gterm\ literal \Rightarrow ('f, 'v)\ term\ literal$ **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 $lit-of-glit\ L = lit-of-glit\ K \implies L = K$
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]*: $is_ground_lit\ L \implies lit_of_glit\ (glit_of_lit\ L) = L$

by (*simp add: is-ground-lit-def lit-of-glit-def glit-of-lit-def literal.map-comp literal.expand literal.map-sel*)

lemma *cls-of-gcls-gcls-of-cls-ident[simp]*: $is_ground_cls\ D \implies cls_of_gcls\ (gcls_of_cls\ D) = D$

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 \Rightarrow 'a fset **where**

atms-of-cls C = atm-of | \uparrow fset-mset C

definition *atms-of-clss* :: 'a clause fset \Rightarrow 'a fset **where**

atms-of-clss N = ffUnion (atms-of-cls | \uparrow N)

lemma *atms-of-clss-empty[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 | \cup | atms-of-clss N

unfolding *atms-of-clss-def* **by** *simp*

definition *lits-of-clss* :: 'a clause fset \Rightarrow 'a literal fset **where**

lits-of-clss N = ffUnion (fset-mset | \uparrow N)

definition *lit-occures-in-clss* **where**

lit-occures-in-clss L N \longleftrightarrow fBex N ($\lambda C. L \in \# C$)

inductive *constant-context* **for** *R* **where**

R C D D' \implies constant-context R (C, D) (C, D')

lemma *rtranclp-constant-context*: $(R\ C)^{**}\ D\ D' \implies (constant_context\ R)^{**}\ (C, D)\ (C, D')$

by (*induction D' rule: rtranclp-induct*) (*auto intro: constant-context.intros rtranclp.intros*)

lemma *tranclp-constant-context*: $(R\ C)^{++}\ D\ D' \implies (constant_context\ R)^{++}\ (C, D)\ (C, D')$

by (*induction D' rule: tranclp-induct*) (*auto intro: constant-context.intros tran-*

clp.intros)

lemma *right-unique-constant-context*:

assumes *R-ru*: $\bigwedge C. \text{right-unique } (R\ C)$

shows *right-unique* (*constant-context R*)

proof (*rule right-uniqueI*)

fix *x y z*

show *constant-context R x y* \implies *constant-context R x z* \implies *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:

$\bigwedge N\ s\ s'. \mathcal{R}\ N\ s\ s' \implies \mathcal{I}\ N\ s \implies \mathcal{I}\ N\ s'$ **and**

ex-R-if-not-final:

$\bigwedge N\ s. \neg \mathcal{F}\ (N, s) \implies \mathcal{I}\ N\ s \implies \exists s'. \mathcal{R}\ N\ s\ s'$

assumes *invars*: $\mathcal{I}\ N\ s$

shows *safe-state* (*constant-context R*) $\mathcal{F}\ (N, s)$

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** $(\mathcal{R}\ N)^{**}\ s\ s'$ **and** $\mathcal{I}\ N\ s'$

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 $\neg \mathcal{F}\ (N, s') \implies \exists s''. \mathcal{R}\ N\ s'\ s''$

using *ex-R-if-not-final*[*of N s'*] **by** *argo*

hence $\neg \mathcal{F}\ S' \implies \exists S''. \text{constant-context } \mathcal{R}\ S'\ S''$

unfolding $\langle S' = (N, s') \rangle$ **using** *constant-context.intros* **by** *metis*

hence $\mathcal{F}\ S'$

by (*metis* $\langle \text{stuck-state } (\text{constant-context } \mathcal{R})\ S' \rangle$ *stuck-state-def*)

}

thus ?*thesis*

by (*metis* (*no-types*) *safe-state-def stuck-state-def*)

qed

primrec *trail-atms* :: $(- \text{ literal } \times -) \text{ list } \Rightarrow - \text{ fset}$ **where**

$trail-atms [] = \{\}\mid$
 $trail-atms (Ln \# \Gamma) = finsert (atm-of (fst Ln)) (trail-atms \Gamma)$

lemma *fset-trail-atms*: $fset (trail-atms \Gamma) = atm-of \text{‘}fst \text{‘} set \Gamma$
by (*induction* Γ) *simp-all*

lemma *trail-defined-lit-iff-trail-defined-atm*:
 $trail-defined-lit \Gamma L \longleftrightarrow atm-of L \mid\in\mid trail-atms \Gamma$
proof (*induction* Γ)

case *Nil*
show *?case*
by (*simp add: trail-defined-lit-def*)

next
case (*Cons* $Ln \Gamma$)

have $trail-defined-lit (Ln \# \Gamma) L \longleftrightarrow L = fst Ln \vee \neg L = fst Ln \vee trail-defined-lit \Gamma L$
unfolding *trail-defined-lit-def* **by** *auto*

also have $\dots \longleftrightarrow atm-of L = atm-of (fst Ln) \vee trail-defined-lit \Gamma L$
by (*cases* L ; *cases* $fst Ln$) *simp-all*

also have $\dots \longleftrightarrow atm-of L = atm-of (fst Ln) \vee atm-of L \mid\in\mid trail-atms \Gamma$
unfolding *Cons.IH ..*

also have $\dots \longleftrightarrow atm-of L \mid\in\mid trail-atms (Ln \# \Gamma)$
by *simp*

finally show *?case .*

qed

lemma *trail-atms-subset-if-suffix*:
assumes *suffix* $\Gamma' \Gamma$
shows $trail-atms \Gamma' \mid\subseteq\mid trail-atms \Gamma$

proof –
obtain Γ_0 **where** $\Gamma = \Gamma_0 @ \Gamma'$
using *assms unfolding suffix-def* **by** *metis*

show *?thesis*
unfolding $\langle \Gamma = \Gamma_0 @ \Gamma' \rangle$
by (*induction* Γ_0) *auto*

qed

lemma *dom-model-eq-trail-interp*:
assumes
 $\forall A C. \mathcal{M} A = Some C \longleftrightarrow map-of \Gamma (Pos A) = Some (Some C)$ **and**
 $\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L$
shows $dom \mathcal{M} = trail-interp \Gamma$
proof –

```

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

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

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

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

  also have ... = trail-interp  $(Ln \# \Gamma)$ 
  unfolding trail-interp-Cons[of Ln  $\Gamma$ ] ..

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

```

```

type-synonym 'f gliteral = 'f gterm literal
type-synonym 'f gclause = 'f gterm clause

```

```

locale simulation-SCLFOL-ground-ordered-resolution =
  renaming-apart renaming-vars
  for renaming-vars :: 'v set  $\Rightarrow$  'v  $\Rightarrow$  'v +
  fixes

```

$less-trm :: 'f\ gterm \Rightarrow 'f\ gterm \Rightarrow bool$ (**infix** \prec_t 50)
assumes
 $transp-less-trm[simp]: transp (\prec_t)$ **and**
 $asympt-less-trm[intro]: asympt (\prec_t)$ **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
 $less-trm-if-subterm[simp]: \bigwedge t\ ctxt. ctxt \neq \square_G \Longrightarrow t \prec_t ctxt\langle t \rangle_G$

8 Ground ordered resolution for ground terms

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

sublocale *ord-res: ground-ordered-resolution-calculus* (\prec_t) $\lambda\cdot$ $\{\#\}$
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 asymptD asympt-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 asymptD asympt-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 asymptD ord-res.asympt-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 asymptD ord-res.asympt-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-clcs: 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-clcs reflclp-iff)
next
  show  $\bigwedge x. x \preceq_c x$ 
    by simp
next
  show  $\bigwedge x y z. x \preceq_c y \implies y \preceq_c z \implies x \preceq_c z$ 
    by (meson transpE ord-res.transp-less-clcs transp-on-reflclp)
next
  show  $\bigwedge x y. x \preceq_c y \implies y \preceq_c x \implies x = y$ 
    by (metis asympD ord-res.asymp-less-clcs 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-clcs)
qed

declare linorder-trm.is-least-in-fset-ffilterD[no-atp]
declare linorder-lit.is-least-in-fset-ffilterD[no-atp]
declare linorder-clcs.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 $N C \equiv \text{ord-res.interp } N C \cup \text{ord-res.production } N C$

definition *is-least-false-clause* **where**

is-least-false-clause $N C \longleftrightarrow$

linorder-clcs.is-least-in-fset $\{ | C | \in | N. \neg \text{ord-res-Interp } (fset N) C \models C \} C$

lemma *is-least-false-clause-finsert-smaller-false-clause*:

assumes

D-least: *is-least-false-clause* $N D$ **and**

$C \prec_c D$ **and**

C-false: $\neg \text{ord-res-Interp } (fset (finsert C N)) C \models C$

shows *is-least-false-clause* $(finsert C N) C$

unfolding *is-least-false-clause-def linorder-clcs.is-least-in-ffilter-iff*

proof (*intro conjI ballI impI*)

show $C | \in | finsert C N$

```

    by simp
next
  show  $\neg \text{ord-res-Interp (fset (finsert C N)) C} \models C$ 
    using assms by metis
next
  fix y
  assume  $y \in \text{finsert C N}$  and  $y \neq C$  and y-false:  $\neg \text{ord-res-Interp (fset (finsert C N)) } y \models y$ 
  hence  $y \in N$ 
    by simp

  have  $\neg (y \prec_c C)$ 
  proof (rule notI)
    assume  $y \prec_c C$ 
    hence  $\text{ord-res-Interp (fset (finsert C N)) } y = \text{ord-res-Interp (fset N) } y$ 
      using ord-res.Interp-insert-greater-clause by simp

  hence  $\neg \text{ord-res-Interp (fset N) } y \models y$ 
    using y-false by argo

  moreover have  $y \prec_c D$ 
    using  $\langle y \prec_c C \rangle \langle C \prec_c D \rangle$  by order

  ultimately show False
    using D-least
    by (metis (mono-tags, lifting) \langle y \in N \rangle linorder-cls.is-least-in-filter-iff linorder-cls.less-asym' is-least-false-clause-def)
qed
thus  $C \prec_c y$ 
  using  $\langle y \neq C \rangle$  by order
qed

lemma is-least-false-clause-swap-swap-compatible-fsets:
  assumes  $\{x \in N1. x \preceq_c C\} = \{x \in N2. x \preceq_c C\}$ 
  shows is-least-false-clause N1 C  $\longleftrightarrow$  is-least-false-clause N2 C
proof -
  have is-least-false-clause N2 C if
    subsets-agree:  $\{x \in N1. x \preceq_c C\} = \{x \in N2. x \preceq_c C\}$  and
    C-least: is-least-false-clause N1 C for N1 N2 C
  unfolding is-least-false-clause-def linorder-cls.is-least-in-filter-iff
  proof (intro conjI ballI impI)
    have  $C \in N1$ 
      using C-least
    unfolding is-least-false-clause-def linorder-cls.is-least-in-filter-iff
    by argo
    thus  $C \in N2$ 
      using subsets-agree by auto
  next
    have  $\neg \text{ord-res-Interp (fset N1) C} \models C$ 

```

```

    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  $\neg$  ord-res-Interp (fset N2) C  $\models$  C
    by argo
next
fix y assume y  $\in$  N2 and  $y \neq C$ 
show  $\neg$  ord-res-Interp (fset N2) y  $\models$  y  $\implies$  C  $\prec_c$  y
proof (erule contrapos-np)
  assume  $\neg$  C  $\prec_c$  y
  hence  $y \preceq_c$  C
  by order
  hence y  $\in$  N1
  using  $\langle y \in N2 \rangle$  using subsets-agree by auto
  hence  $\neg$  ord-res-Interp (fset N1) y  $\models$  y  $\longrightarrow$  C  $\prec_c$  y
  using  $\langle$ is-least-false-clause N1 C $\rangle$   $\langle$ y  $\neq$  C $\rangle$ 
  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  $\{D. D \in N1 \wedge (\prec_c)^{==} D y\} = \{D. D \in N2 \wedge (\prec_c)^{==} D y\}$ 
    using subsets-agree  $\langle$ y  $\preceq_c$  C $\rangle$  by fastforce
  qed
  ultimately show ord-res-Interp (fset N2) y  $\models$  y
    using  $\langle$ y  $\preceq_c$  C $\rangle$  by auto
  qed
qed
thus ?thesis
  using assms by metis
qed

lemma Uniq-is-least-false-clause:  $\exists_{\leq 1} C. \text{is-least-false-clause } N C$ 
proof (rule Uniq-I)
  show  $\bigwedge x y z. \text{is-least-false-clause } x y \implies \text{is-least-false-clause } x z \implies y = z$ 
  unfolding is-least-false-clause-def
  by (meson Uniq-D linorder-cls.Uniq-is-least-in-fset)
qed

lemma mempty-lesseq-cl[simp]:  $\{\#\} \preceq_c C$  for C
by (cases C) (simp-all add: strict-subset-implies-multp)

lemma is-least-false-clause-mempty:  $\{\#\} \in N \implies \text{is-least-false-clause } N \{\#\}$ 
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: $\forall x \in N2 - N1. \text{ord-res.production } (N1 \cup N2) x = \{\}$ and
C-in: $C \in N1$
shows *ord-res.production* $(N1 \cup N2) C = \text{ord-res.production } N1 C$
using *ord-res.wfP-less-cls C-in*
proof (*induction C rule: wfp-induct-rule*)
case (*less C*)
hence *C-in-iff: $C \in N1 \cup N2 \longleftrightarrow C \in N1$*
by *simp*

have *interp-eq: ord-res.interp* $(N1 \cup N2) C = \text{ord-res.interp } N1 C$
proof –
have *ord-res.interp* $(N1 \cup N2) C = \bigcup (\text{ord-res.production } (N1 \cup N2) \text{ ‘ } \{D \in N1 \cup N2. D \prec_c C\}$)
unfolding *ord-res.interp-def ..*
also have $\dots = \bigcup (\text{ord-res.production } (N1 \cup N2) \text{ ‘ } \{D \in N1. D \prec_c C\} \cup \text{ord-res.production } (N1 \cup N2) \text{ ‘ } \{D \in N2 - N1. D \prec_c C\})$
by *auto*
also have $\dots = \bigcup (\text{ord-res.production } (N1 \cup N2) \text{ ‘ } \{D \in N1. D \prec_c C\})$
using *N2-unproductive by simp*
also have $\dots = \bigcup (\text{ord-res.production } N1 \text{ ‘ } \{D \in N1. D \prec_c C\})$
using *less.IH by simp*
also have $\dots = \text{ord-res.interp } N1 C$
unfolding *ord-res.interp-def ..*
finally show *ord-res.interp* $(N1 \cup N2) C = \text{ord-res.interp } N1 C$.
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: $\forall x \in N2. \text{ord-res.production } (N1 \cup N2) x = \{\}$ and
C-in: $C \in N1$
shows *ord-res.production* $(N1 \cup N2) C = \text{ord-res.production } N1 C$
using *production-union-unproductive-strong assms by simp*

lemma *interp-union-unproductive:*

assumes
fin: finite N1 finite N2 and
N2-unproductive: $\forall x \in N2. \text{ord-res.production } (N1 \cup N2) x = \{\}$
shows *ord-res.interp* $(N1 \cup N2) = \text{ord-res.interp } N1$
proof (*rule ext*)
fix *C*
have *ord-res.interp* $(N1 \cup N2) C = \bigcup (\text{ord-res.production } (N1 \cup N2) \text{ ‘ } \{D \in N1 \cup N2. D \prec_c C\})$
unfolding *ord-res.interp-def ..*

also have $\dots = \bigcup (ord-res.production (N1 \cup N2) \text{ ' } \{D \in N1. D \prec_c C\} \cup$
 $ord-res.production (N1 \cup N2) \text{ ' } \{D \in N2. D \prec_c C\})$
by *auto*
also have $\dots = \bigcup (ord-res.production (N1 \cup N2) \text{ ' } \{D \in N1. D \prec_c C\})$
using *N2-unproductive by simp*
also have $\dots = \bigcup (ord-res.production N1 \text{ ' } \{D \in N1. D \prec_c C\})$
using *production-union-unproductive[OF fin N2-unproductive] by simp*
also have $\dots = ord-res.interp N1 C$
unfolding *ord-res.interp-def ..*
finally show $ord-res.interp (N1 \cup N2) C = ord-res.interp N1 C .$
qed

lemma *Interp-union-unproductive:*

assumes

fin: finite N1 finite N2 and

N2-unproductive: $\forall x \in N2. ord-res.production (N1 \cup N2) x = \{\}$

shows $ord-res-Interp (N1 \cup N2) C = ord-res-Interp N1 C$

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: $\forall D \in N'. \exists E \in N. E \subset\# D \wedge set-mset D = set-mset E$ and

C-in: $C \in N$

shows $ord-res-Interp (N \cup N') C \models C \longleftrightarrow ord-res-Interp N C \models C$

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*

lemma *nex-strictly-maximal-pos-lit-if-factorizable:*

assumes *ord-res.ground-factoring C C'*

shows $\nexists L. is-pos L \wedge ord-res.is-strictly-maximal-lit L C$

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 $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$
shows $\text{ord-res.production } N \ C = \{\}$
using *assms* **by** (*simp add: ord-res.production-unfold*)

lemma *ball-unproductive-if-nex-strictly-maximal-pos-lit*:
assumes $\forall C \in N'. \nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$
shows $\forall C \in N'. \text{ord-res.production } (N \cup N') \ C = \{\}$
using *assms unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*

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

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

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

have $E \prec_c D$
proof (*rule E-least[rule-format]*)
show $D \mid \in \mid \text{finsert } C \ N$
using $\langle D \mid \in \mid N \rangle$ **by** *simp*
next
show $D \neq E$
using $\langle D \prec_c C \rangle \langle C = E \rangle$ **by** *order*
next
show $\neg \text{ord-res-Interp } (\text{fset } (\text{finsert } C \ N)) \ D \models D$
using *E-false*
proof (*rule contrapos-nn*)
assume $\text{ord-res-Interp } (\text{fset } (\text{finsert } C \ N)) \ D \models D$
thus $\text{ord-res-Interp } (\text{fset } (\text{finsert } C \ N)) \ E \models E$
using $\langle D \prec_c C \rangle \langle C = E \rangle \langle \{D\} \models_e \{C\} \rangle$ *ord-res.entailed-clause-stays-entailed*

by *auto*
qed

```

qed
hence False
  using  $\langle D \prec_c C \rangle \langle C = E \rangle$  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  $E \in N$ 
    using E-in  $\langle C \neq E \rangle$  by simp
  next
  have  $\text{ord-res-Interp (fset (finsert C N)) } E = \text{ord-res-Interp (fset N) } E$ 
    using C-unproductive Interp-insert-unproductive by simp
  thus  $\neg \text{ord-res-Interp (fset N) } E \models E$ 
    using E-false by argo
  next
  show  $\bigwedge y. y \in N \implies y \neq E \implies \neg \text{ord-res-Interp (fset N) } y \models y \implies E \prec_c y$ 
    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:  $E \in N$  and
  E-false:  $\neg \text{ord-res-Interp (fset N) } E \models E$  and
  E-least:  $(\forall y \in N. y \neq E \longrightarrow \neg \text{ord-res-Interp (fset N) } y \models y \longrightarrow E \prec_c y)$ 
unfolding atomize-conj is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
by metis

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  $E \in \text{finsert C N}$ 
    using E-in by simp
  next
  show  $\neg \text{ord-res-Interp (fset (finsert C N)) } E \models E$ 
    using E-least E-false C-unproductive Interp-insert-unproductive by simp
  next
  fix y
  assume  $y \in \text{finsert C N}$  and  $y \neq E$  and  $\neg \text{ord-res-Interp (fset (finsert C N)) } y \models y$ 
  show  $E \prec_c y$ 
  proof (cases  $y = C$ )
  case True
  thus ?thesis
  using E-least  $\langle \neg \text{ord-res-Interp (fset (finsert C N)) } y \models y \rangle$ 

```

```

    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.premis(1) by auto
    next
    show ∀ C |∈| N2 |-| N1. ∃ D |∈| N1. D <_c C ∧ {D} ||=_e {C}
      using insert.premis(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.premis(1))

```

insert.premis(2)

qed
qed

lemma *is-least-false-clause-funion-cancel-right*:

assumes

$\forall C \in N2. \forall U. \text{ord-res.production } U \ C = \{\}$ **and**

$\forall C \in N2. \exists D \in N1. D \prec_c C \wedge \{D\} \models_e \{C\}$

shows *is-least-false-clause* ($N1 \cup N2$) = *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 $N = (\exists C \in N. \neg \text{ord-res.interp } N \ C \cup \text{ord-res.production } N \ C \models C)$

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 fmember-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 ($\exists C. \text{is-least-false-clause } N \ C$)

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 $N = (\bigcup D \in N. \text{ord-res.production } N \ D)$

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* $N = \text{ord-res.interp } N \ C \cup \text{ord-res.production } N \ C$

proof –

have *ord-res-model* $N = (\bigcup D \in N. \text{ord-res.production } N \ D)$

unfolding *ord-res-model-def* ..

also have $\dots = (\bigcup D \in \{D \in N. D \preceq_c C\}. \text{ord-res.production } N \ D)$

using *C-greatest linorder-cls.is-greatest-in-set-iff* **by** *auto*

also have $\dots = (\bigcup D \in \{D \in N. D \prec_c C\} \cup \{C\}. \text{ord-res.production } N \ D)$

using *C-greatest linorder-cls.is-greatest-in-set-iff* **by** *auto*

also have $\dots = (\bigcup D \in \{D \in N. D \prec_c C\}. \text{ord-res.production } N \ D) \cup \text{ord-res.production}$

$N C$
 by *auto*
 also have $\dots = \text{ord-res.interp } N C \cup \text{ord-res.production } N C$
 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 $\neg \text{ex-false-clause } N$
 shows $\text{ord-res.model } N \models_s N$
 unfolding *true-clss-def*
proof (*intro ballI*)
 from $\langle \neg \text{ex-false-clause } N \rangle$ have *ball-true*:
 $\forall C \in N. \text{ord-res.interp } N C \cup \text{ord-res.production } N C \models C$
 by (*simp add: ex-false-clause-def*)

from $\langle \text{finite } N \rangle \langle N \neq \{\} \rangle$ 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 $\text{ord-res.interp } N C \cup \text{ord-res.production } N C \models C$
 using *ball-true* by *metis*

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

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

thus $\text{ord-res.model } N \models C$
 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 $\{ | C | \in | N. \neg \text{ord-res.Interp } (fset N) C \models C | \}$ *C* and
C-max-lit: ord-res.is-maximal-lit (Pos A) C
 shows $\neg \text{ord-res.is-strictly-maximal-lit } (Pos A) C$
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 |∈| N and
C-false: ¬ ord-res.Interp (fset N) C $\models C$ and
C-lt: $\forall y | \in | N. y \neq C \longrightarrow \neg \text{ord-res.Interp } (fset N) y \models y \longrightarrow C \prec_c y$

unfolding *linorder-cls.is-least-in-filter-iff* **by** *auto*

have $\nexists A. A \in \text{ord-res.production (fset N) C}$
proof (*rule notI, elim exE*)
fix *A* **assume** *A-in*: $A \in \text{ord-res.production (fset N) C}$
have $\text{Pos } A \in \# C$
using *A-in* **by** (*auto elim: ord-res.mem-productionE*)
moreover **have** $A \in \text{ord-res.Interp (fset N) C}$
using *A-in C-in* **by** *blast*
ultimately show *False*
using *C-false* **by** *auto*

qed
hence *C-unproductive*: $\text{ord-res.production (fset N) C} = \{\}$
using *ord-res.production-eq-empty-or-singleton*[*of fset N C*] **by** *simp*

have $\{D \in \text{fset N}. D \preceq_c C\} = \{D \in \text{fset N}. D \prec_c C\} \cup \{D \in \text{fset N}. D = C\}$
by *fastforce*

with *C-unproductive* **have**
 $\text{ord-res.Interp (fset N) C} = \text{ord-res.interp (fset N) C}$
by *simp*

with *C-false* **have** *C-false'*: $\neg \text{ord-res.interp (fset N) C} \models C$
by *simp*

from *C-unproductive* **have** $A \notin \text{ord-res.production (fset N) C}$
by *simp*

thus *?thesis*

proof (*rule contrapos-nn*)
assume *ord-res.is-strictly-maximal-lit (Pos A) C*
then show $A \in \text{ord-res.production (fset N) C}$
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
 $C\text{-max-lit: } \text{ord-res.is-maximal-lit (Pos A) C}$ **and**
 $C\text{-not-max-lit: } \neg \text{ord-res.is-strictly-maximal-lit (Pos A) C}$
shows $\exists C'. \text{ord-res.ground-factoring } C C'$

proof –
from *C-max-lit C-not-max-lit* **have** $\text{count } C (\text{Pos } A) \geq 2$
using *linorder-lit.count-ge-2-if-maximal-in-mset-and-not-greatest-in-mset* **by**
metis

then obtain *C'* **where** *C-def*: $C = \text{add-mset (Pos A) (add-mset (Pos A) C')}$
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*: $L \in\# C \implies I \models_l L \implies I \models C$
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 ($\lambda D. D \prec_c C \wedge \text{ord-res.is-strictly-maximal-lit } (Pos) A D \wedge$
ord-res.production (*fset* N) $D = \{A\}$)
proof –
from *C-least-false* **have**
C-in: $C \in| N$ **and**
C-false: $\neg \text{ord-res.Interp } (fset\ N)\ C \models C$ **and**
C-min: $\forall y \in| N. y \neq C \longrightarrow \neg \text{ord-res.Interp } (fset\ N)\ y \models y \longrightarrow C \prec_c y$
unfolding *atomize-conj* *is-least-false-clause-def*
unfolding *linorder-cls.is-least-in-filter-iff*
by *argo*

have $\nexists A. A \in \text{ord-res.production } (fset\ N)\ C$
proof (*rule notI, elim exE*)
fix A **assume** *A-in*: $A \in \text{ord-res.production } (fset\ N)\ C$
have $Pos\ A \in\# C$
using *A-in* **by** (*auto elim: ord-res.mem-productionE*)
moreover **have** $A \in \text{ord-res.Interp } (fset\ N)\ C$
using *A-in C-in* **by** *blast*
ultimately **show** *False*
using *C-false* **by** *auto*
qed

hence *C-unproductive*: $\text{ord-res.production } (fset\ N)\ C = \{\}$
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** $\neg \text{ord-res.Interp } (fset\ N)\ C \models_l Neg\ A$
using *true-cls-if-true-lit-in*[*OF* $\langle Neg\ A \in\# C \rangle$]
by *meson*
hence $A \in \text{ord-res.Interp } (fset\ N)\ C$
by *simp*
with *C-unproductive* **have** $A \in \text{ord-res.interp } (fset\ N)\ C$
by *blast*
then **obtain** D **where**
D-in: $D \in| N$ **and** *D-lt-C*: $D \prec_c C$ **and** *D-productive*: $A \in \text{ord-res.production}$

(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* $(\lambda D. D \prec_c C \wedge \text{ord-res.is-strictly-maximal-lit } (- L) D \wedge$
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 \prec_c C **and**
D-max-lit: ord-res.is-strictly-maximal-lit (Pos A) D
shows $\exists CD. \text{ord-res.ground-resolution } C D CD$

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: $\forall D \in N'. \exists E \in N. E \subset\# D \wedge \text{set-mset } D = \text{set-mset } E$ **and**
C-in: $C \in N$ **and**
C-not-entailed: $\neg \text{ord-res.interp } N C \cup \text{ord-res.production } N C \models C$
shows $\neg \text{ord-res.interp } (N \cup N') C \cup \text{ord-res.production } (N \cup N') C \models C$
using *C-not-entailed*
proof (*rule contrapos-nn*)
assume $\text{ord-res.interp } (N \cup N') C \cup \text{ord-res.production } (N \cup N') C \models C$
then show $\text{ord-res.interp } N C \cup \text{ord-res.production } N C \models C$
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* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
shows *trail-consistent* Γ
using *assms*
proof (*induction* Γ)
case *Nil*
show *?case*
by *simp*
next
case (*Cons Ln* Γ)

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

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

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

thus $\neg \text{trail-defined-lit } \Gamma L$
unfolding *trail-defined-lit-def* **by** *fastforce*

next
show *trail-consistent* Γ
using *Cons* **by** *simp*

qed
qed

lemma *mono-atms-lt*: *monotone-on* $(\text{set } \Gamma) (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) (\lambda x y. y \leq x)$
 $(\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (fst x))$ **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  $\Gamma$  and
    monotone-on (set  $\Gamma$ ) R ( $\geq$ ) ( $\lambda x. P$  (atm-of (fst x))) and
     $\neg P A$  and
    A  $|\in|$  trail-atms  $\Gamma$ 
  shows A  $|\in|$  trail-atms (dropWhile ( $\lambda Ln. P$  (atm-of (fst Ln)))  $\Gamma$ )
  using assms(1,2,4)
proof (induction  $\Gamma$ )
  case Nil
  thus ?case
    by simp
next
  case (Cons Ln  $\Gamma$ )
  show ?case
  proof (cases P (atm-of (fst Ln)))
    case True

    have A  $|\in|$  trail-atms (dropWhile ( $\lambda Ln. P$  (atm-of (fst Ln)))  $\Gamma$ )
    proof (rule Cons.IH)
      show sorted-wrt R  $\Gamma$ 
        using Cons.prem(1) by simp
    next
      show monotone-on (set  $\Gamma$ ) R ( $\lambda x y. y \leq x$ ) ( $\lambda x. P$  (atm-of (fst x)))
        using Cons.prem(2) by (meson monotone-on-subset set-subset-Cons)
    next
      have  $\neg P A$ 
        using assms by metis
      hence A  $\neq$  atm-of (fst Ln)
        using True by metis
      moreover have A  $|\in|$  trail-atms (Ln #  $\Gamma$ )
        using Cons.prem(3) by metis
      ultimately show A  $|\in|$  trail-atms  $\Gamma$ 
        by (simp add: trail-defined-lit-def)
    qed

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

```

qed
qed

lemma *trail-defined-lit-dropWhileI*:

assumes

sorted-wrt $R \Gamma$ **and**

monotone-on ($\text{set } \Gamma$) $R (\geq) (\lambda x. P (\text{fst } x))$ **and**

$\neg P L \wedge \neg P (- L)$ **and**

trail-defined-lit ΓL

shows *trail-defined-lit* ($\text{dropWhile } (\lambda Ln. P (\text{fst } Ln)) \Gamma$) 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 $R \Gamma$ **and**

monotone-on ($\text{set } \Gamma$) $R (\geq) (\lambda x. P (\text{fst } x))$ **and**

$\forall L \in \# C. \neg P L \wedge \neg P (- L)$ **and**

trail-defined-cls ΓC

shows *trail-defined-cls* ($\text{dropWhile } (\lambda Ln. P (\text{fst } Ln)) \Gamma$) C

using *assms trail-defined-lit-dropWhileI*

by (*metis trail-defined-cls-def*)

lemma *nbeX-less-than-least-in-fset*: $\neg (\exists w \mid \in \mid X. w \prec_c x)$

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

$\mathcal{C} \equiv \text{The-optional } (\text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) D) N))$

assumes

C-lt: $\bigwedge E. \mathcal{C} = \text{Some } E \implies C \prec_c E$ **and**

C-in: $C \mid \in \mid N$

shows $C \preceq_c D$

proof (*cases* \mathcal{C})

case *None*

hence $\neg (\exists E. \text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) D) N) E)$

using *C-def*

by (*metis None-eq-The-optionalD Uniq-D linorder-cls.Uniq-is-least-in-fset*)

hence $\neg (\exists E \mid \in \mid N. D \prec_c E)$

by (*metis emptyE ffilter-member-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 (( $\prec_c$ ) D) N) E
  using C-def by (metis Some-eq-The-optionalD)

hence C  $\prec_c$  D  $\vee$  C = D
  by (metis C-in C-lt Some fmember-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  $\exists!(t' :: ('f, 'v) \text{ term}). \text{ground } t' \wedge t = \text{gterm-of-term } t'$ 
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'  $\implies$  t' = term-of-gterm t
    by (induction t') (simp-all add: map-idI)
qed

```

```

lemma inj-betw-gterm-of-term: inj-betw gterm-of-term {t. ground t} UNIV
  unfolding inj-betw-def
proof (rule injI)
  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 $x\ y \longleftrightarrow \text{ground } x \wedge \text{ground } y \wedge \text{gterm-of-term } x \prec_t \text{gterm-of-term } y$

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 $\beta :: ('f, 'v)\ \text{term}$

have *Collect-gterms-eq*: $\bigwedge P. \{y. P\ y\} = \text{gterm-of-term } \{t. \text{ground } t \wedge P (\text{gterm-of-term } t)\}$

using *Collect-eq-image-filter-Collect-if-bij-betw[OF binj-betw-gterm-of-term subset-UNIV]*

by *auto*

have $\{t_G. t_G \prec_t \text{gterm-of-term } \beta\} = \text{gterm-of-term } \{x. \text{ground } x \wedge \text{gterm-of-term } x \prec_t \text{gterm-of-term } \beta\}$

using *Collect-gterms-eq[of $\lambda t_G. t_G \prec_t \text{gterm-of-term } \beta$]*.

hence *finite* $(\text{gterm-of-term } \{x. \text{ground } x \wedge \text{gterm-of-term } x \prec_t \text{gterm-of-term } \beta\})$

using *finite-less-trm[of gterm-of-term β]* **by** *metis*

moreover **have** *inj-on* $\text{gterm-of-term } \{x. \text{ground } x \wedge \text{gterm-of-term } x \prec_t \text{gterm-of-term } \beta\}$

by *(rule gterm-of-term-inj) simp*

ultimately **have** *finite* $\{x. \text{ground } x \wedge \text{gterm-of-term } x \prec_t \text{gterm-of-term } \beta\}$

using *finite-imageD* **by** *blast*

thus *finite* $\{x. \text{less-B } x\ \beta\}$

unfolding *less-B-def*

using *not-finite-existsD* **by** *force*

qed

end

lemma *trail-atms-eq-trail-atms-if-same-lits*:

assumes *list-all2* $(\lambda x\ y. \text{fst } x = \text{fst } y)\ \Gamma_9\ \Gamma_{10}$

shows *trail-atms* $\Gamma_9 = \text{trail-atms } \Gamma_{10}$

using *assms*

proof *(induction $\Gamma_9\ \Gamma_{10}$ rule: list.rel-induct)*

case *Nil*

show *?case*

by (simp add: trail-atms-def)
 next
 case (Cons Ln₉ Γ₉' Ln₁₀ Γ₁₀')
 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) Γ₉ Γ₁₀
 shows trail-false-lit Γ₉ = trail-false-lit Γ₁₀
 using assms
proof (induction Γ₉ Γ₁₀ rule: list.rel-induct)
 case Nil
 show ?case
 by (simp add: trail-false-lit-def)
 next
 case (Cons Ln₉ Γ₉' Ln₁₀ Γ₁₀')
 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) Γ₉ Γ₁₀
 shows trail-false-cls Γ₉ = trail-false-cls Γ₁₀
 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) Γ₉ Γ₁₀
 shows trail-defined-lit Γ₉ = trail-defined-lit Γ₁₀
 using assms
proof (induction Γ₉ Γ₁₀ rule: list.rel-induct)
 case Nil
 show ?case
 by (simp add: trail-defined-lit-def)
 next
 case (Cons Ln₉ Γ₉' Ln₁₀ Γ₁₀')
 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) Γ₉ Γ₁₀
 shows trail-defined-cls Γ₉ = trail-defined-cls Γ₁₀
 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  $N \longleftrightarrow \neg (\forall C \in N. \text{ord-res-Interp } N \ C \models C)$ 
  unfolding ex-false-clause-def by metis

```

```

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

```

```

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

```

```

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

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

```

```

inductive ord-res-load where
   $N \neq \{\mid\} \implies \text{ord-res-load } N \ N$ 

```

```

sublocale ord-res-antics: semantics where

```

```

  step = ord-res and
  final = ord-res-final
proof unfold-locales
  fix  $N :: 'f \text{ gterm clause fset}$ 
  assume ord-res-final  $N$ 
  thus finished ord-res  $N$ 
    unfolding ord-res-final-def
  proof (elim disjE)
    show  $\{\#\} \mid \in \mid N \implies \text{finished } \text{ord-res } N$ 
      by (simp add: finished-def ord-res.simps)
  next
    show  $\neg \text{ex-false-clause } (\text{fset } N) \implies \text{finished } \text{ord-res } N$ 
      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 $N C \implies$
 linorder-lit.is-maximal-in-mset $C L \implies$
 is-pos $L \implies$
 ord-res.ground-factoring $C C' \implies$
 $N' = \text{finsert } C' N \implies$
 ord-res-1 $N N' \mid$

resolution:

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


```

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 $N \longleftrightarrow \text{ord-res-final } N$

inductive *ord-res-1-load* **where**
 $N \neq \{\mid\} \implies \text{ord-res-1-load } N \ N$

sublocale *ord-res-1-semantic: semantics* **where**

step = *ord-res-1* **and**
final = *ord-res-1-final*

proof *unfold-locales*

fix $N :: 'f \text{ gterm clause fset}$

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 $\{\#\} \mid \in \mid N$

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* $\langle \{\#\} \mid \in \mid N \rangle$ **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.mulp-if-all-left-smaller *set-mset-empty* *true-cls-empty*)

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* $C \ L \ D \ CD$)

from *hyps* $\langle \{\#\} \mid \in \mid N \rangle$ **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.mulp-if-all-left-smaller *set-mset-empty* *true-cls-empty*)

moreover **from** *hyps* **have** $L \in \# \ C$

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 $\neg \text{ex-false-clause } (\text{fset } N)$

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  $\langle \neg \text{ex-false-clause } (fset N) \rangle$  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  $\langle \neg \text{ex-false-clause } (fset N) \rangle$  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  $\neg \text{ord-res-1-final } N$ 
  shows  $\exists N'. \text{ord-res-1 } N N'$ 
proof –
  from assms have  $\{\#\} \notin N$  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  $\langle \text{ex-false-clause } (fset N) \rangle$  obtains-least-false-clause-if-ex-false-clause by
metis

  hence  $C \neq \{\#\}$ 
    using  $\langle \{\#\} \notin N \rangle$ 
    unfolding is-least-false-clause-def linorder-cls.is-least-in-filter-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  $\neg \text{linorder-lit.is-greatest-in-mset } C L$ 
    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
   $D \mid\in\mid N$  and
   $D \prec_c C$  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  $\vee$  ( $\exists N'$ . 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  $\Rightarrow$  'f gterm clause where
  efac C = (THE C'. ord-res.ground-factoring** C C'  $\wedge$  ( $\nexists C''$ . ord-res.ground-factoring C' C''))

```

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

```

lemma ex1-efac-eq-factoring-chain:

```

```

   $\exists! C'$ . efac C = C'  $\wedge$  ord-res.ground-factoring** C C'  $\wedge$  ( $\nexists C''$ . ord-res.ground-factoring C' C'')

```

```

proof –

```

have *right-unique* ($\lambda x y. \text{ord-res.ground-factorizing}^{**} x y \wedge (\nexists z. \text{ord-res.ground-factorizing } y z)$)
using *ord-res.unique-ground-factorizing right-unique-terminating-rtranclp right-unique-iff*
by *blast*

moreover obtain C' **where** $\text{ord-res.ground-factorizing}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factorizing } C' C'')$
using *ex-terminating-rtranclp[OF ord-res.termination-ground-factorizing]*
by *metis*

ultimately have $\text{efac } C = C'$
by (*simp add: efac-def right-unique-def the-equality*)

then show *?thesis*
using $\langle \text{ord-res.ground-factorizing}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factorizing } C' C'') \rangle$ **by** *blast*
qed

lemma *efac-eq-disj*:
 $\text{efac } C = C \vee (\exists! C'. \text{efac } C = C' \wedge \text{ord-res.ground-factorizing}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factorizing } C' C''))$
using *ex1-efac-eq-factorizing-chain*
by (*metis is-pos-def*)

lemma *member-mset-if-count-eq-Suc*: $\text{count } X x = \text{Suc } n \implies x \in \# X$
by (*simp add: count-inI*)

lemmas *member-fsetE = mset-add*

lemma *ord-res-ground-factorizing-iff*: $\text{ord-res.ground-factorizing } C C' \longleftrightarrow (\exists A. \text{ord-res.is-maximal-lit } (Pos A) C \wedge (\exists n. \text{count } C (Pos A) = \text{Suc } (\text{Suc } n) \wedge C' = C - \{\#Pos A\}))$
proof (*rule iffI*)
assume $\text{ord-res.ground-factorizing } C C'$
thus $\exists A. \text{ord-res.is-maximal-lit } (Pos A) C \wedge (\exists n. \text{count } C (Pos A) = \text{Suc } (\text{Suc } n) \wedge C' = C - \{\#Pos A\})$
proof (*cases C C' rule: ord-res.ground-factorizing.cases*)
case (*ground-factorizingI A P'*)
show *?thesis*
proof (*intro exI conjI*)
show $\text{ord-res.is-maximal-lit } (Pos A) C$
using $\langle \text{ord-res.is-maximal-lit } (Pos A) C \rangle$.
next
show $\text{count } C (Pos A) = \text{Suc } (\text{Suc } (\text{count } P' (Pos A)))$
unfolding $\langle C = \text{add-mset } (Pos A) (\text{add-mset } (Pos A) P') \rangle$ **by** *simp*
next
show $C' = \text{remove1-mset } (Pos A) C$
unfolding $\langle C = \text{add-mset } (Pos A) (\text{add-mset } (Pos A) P') \rangle \langle C' = \text{add-mset } (Pos A) P' \rangle$ **by** *simp*

qed
qed
next
assume $\exists A. \text{ord-res.is-maximal-lit } (Pos\ A)\ C \wedge$
 $(\exists n. \text{count } C\ (Pos\ A) = \text{Suc } (\text{Suc } n) \wedge C' = C - \{\#Pos\ A\# \})$
then obtain $A\ n$ **where**
 $\text{ord-res.is-maximal-lit } (Pos\ A)\ C$ **and**
 $\text{count } C\ (Pos\ A) = \text{Suc } (\text{Suc } n)$ **and**
 $C' = C - \{\#Pos\ A\# \}$
by *metis*

have $Pos\ A \in\# C$
using $\langle \text{count } C\ (Pos\ A) = \text{Suc } (\text{Suc } n) \rangle$ *member-mset-if-count-eq-Suc* **by** *metis*
then obtain C'' **where** $C = \text{add-mset } (Pos\ A)\ C''$
by *(auto elim: member-fsetE)*
with $\langle \text{count } C\ (Pos\ A) = \text{Suc } (\text{Suc } n) \rangle$ **have** $\text{count } C''\ (Pos\ A) = \text{Suc } n$
by *simp*
hence $Pos\ A \in\# C''$
using *member-mset-if-count-eq-Suc* **by** *metis*
then obtain C''' **where** $C'' = \text{add-mset } (Pos\ A)\ C'''$
by *(auto elim: member-fsetE)*

show *ord-res.ground-factorizing* $C\ C'$
proof *(rule ord-res.ground-factorizingI)*
show $C = \text{add-mset } (Pos\ A)\ (\text{add-mset } (Pos\ A)\ C''')$
using $\langle C = \text{add-mset } (Pos\ A)\ C'' \rangle \langle C'' = \text{add-mset } (Pos\ A)\ C''' \rangle$ **by** *metis*
next
show *ord-res.is-maximal-lit* $(Pos\ A)\ C$
using $\langle \text{ord-res.is-maximal-lit } (Pos\ A)\ C \rangle$.
next
show $C' = \text{add-mset } (Pos\ A)\ C'''$
using $\langle C' = C - \{\#Pos\ A\# \} \rangle \langle C = \text{add-mset } (Pos\ A)\ C'' \rangle \langle C'' = \text{add-mset } (Pos\ A)\ C''' \rangle$ **by** *simp*
qed *simp-all*
qed

lemma *tranclp-ord-res-ground-factorizing-iff*:
 $\text{ord-res.ground-factorizing}^{++}\ C\ C' \wedge (\exists C''. \text{ord-res.ground-factorizing } C'\ C'') \longleftrightarrow$
 $(\exists A. \text{ord-res.is-maximal-lit } (Pos\ A)\ C \wedge (\exists n. \text{count } C\ (Pos\ A) = \text{Suc } (\text{Suc } n) \wedge$
 $C' = C - \text{replicate-mset } (\text{Suc } n)\ (Pos\ A)))$
proof *(intro iffI; elim exE conjE)*
assume $\text{ord-res.ground-factorizing}^{++}\ C\ C'$ **and** $(\exists C''. \text{ord-res.ground-factorizing } C'\ C'')$
then show $\exists A. \text{ord-res.is-maximal-lit } (Pos\ A)\ C \wedge (\exists n. \text{count } C\ (Pos\ A) =$
 $\text{Suc } (\text{Suc } n) \wedge$
 $C' = C - \text{replicate-mset } (\text{Suc } n)\ (Pos\ A))$
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 $n \neq 0$
then obtain C'' **where** $C' = \text{add-mset } (Pos\ A) (\text{add-mset } (Pos\ A)\ C'')$
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* $C' (\text{add-mset } (Pos\ A)\ C'')$
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.prem*s **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.prem*s **obtain** A n **where**
ord-res.is-maximal-lit (Pos A) C'' **and**
count C'' (Pos A) = *Suc* (*Suc* n) **and**
 $C' = C'' - \text{replicate-mset } (Suc\ n) (Pos\ A)$
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**
 $C'' = \text{remove1-mset } (Pos\ A')\ C$
unfolding *ord-res-ground-factoring-iff* **by** *metis*

have $A' = A$
using $\langle \text{ord-res.is-maximal-lit } (Pos\ A)\ C'' \rangle \langle \text{ord-res.is-maximal-lit } (Pos\ A')\ C \rangle$
by (*metis* $\langle C'' = \text{remove1-mset } (Pos\ A')\ C \rangle \langle \text{count } C (Pos\ A') = \text{Suc } (Suc\ m) \rangle$
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 $\langle \text{count } C'' (Pos\ A) = \text{Suc } (Suc\ n) \rangle \langle \text{count } C (Pos\ A') = \text{Suc } (Suc\ m) \rangle$
unfolding $\langle C'' = \text{remove1-mset } (Pos\ A')\ C \rangle \langle A' = A \rangle$
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)  $\implies$  C' = C - replicate-mset (Suc n) (Pos
A)  $\implies$ 
  ord-res.ground-factorizing++ C C'  $\wedge$  ( $\nexists$  C''. ord-res.ground-factorizing C' C'')
proof (induction n arbitrary: C)
  case 0
  hence (ord-res.is-maximal-lit (Pos A) C  $\wedge$ 
    (count C (Pos A) = Suc (Suc 0)  $\wedge$ 
      C' = remove1-mset (Pos A) C))
    by (metis replicate-mset-0 replicate-mset-Suc)
  hence ord-res.ground-factorizing C C'  $\wedge$  ( $\nexists$  a. ord-res.ground-factorizing C' a)
    unfolding ord-res.ground-factorizing-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-factorizing++ (C - {#Pos A#}) C'  $\wedge$  ( $\nexists$  a. ord-res.ground-factorizing
C' a)
    proof (rule Suc.IH)
      show count (remove1-mset (Pos A) C) (Pos A) = Suc (Suc n)
        using Suc.prem by simp
    next
      show C' = remove1-mset (Pos A) C - replicate-mset (Suc n) (Pos A)
        using Suc.prem by simp
    next
      show ord-res.is-maximal-lit (Pos A) (remove1-mset (Pos A) C)
        using Suc.prem
      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
  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.premis by metis
next
  show count  $C$  ( $Pos A$ ) = Suc (Suc (Suc  $n$ ))
    using Suc.premis 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  $x$   $\{\#y \in\# X. y \neq x\# \}$ 
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  $x$   $\{\#y \in\# X. y \neq x\# \}$ )
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**  $C$   $C' \wedge (\exists C''. ord-res.ground-factoring C' C'')$ 
 $\longleftrightarrow$ 
  (( $\exists A. ord-res.is-maximal-lit (Pos A) C \wedge count C (Pos A) \geq 2$ )  $\wedge C = C' \vee$ 
  ( $\exists A. ord-res.is-maximal-lit (Pos A) C \wedge C' = add-mset (Pos A) \{\#L \in\# C. L \neq Pos A\# \}$ ))
proof (intro iffI; elim exE conjE disjE)
  show ord-res.ground-factoring**  $C$   $C' \implies \exists C''. ord-res.ground-factoring C' C''$ 
 $\implies$ 
  ( $\exists A. ord-res.is-maximal-lit (Pos A) C \wedge 2 \leq count C (Pos A)$ )  $\wedge C = C' \vee$ 
  ( $\exists A. ord-res.is-maximal-lit (Pos A) C \wedge C' = add-mset (Pos A) \{\#L \in\# C. L \neq Pos A\# \}$ )
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-factorizing++ y C'  $\wedge$  ( $\nexists x$ . ord-res.ground-factorizing C' x)
    by simp
  thus ?case
    unfolding tranclp-ord-res-ground-factorizing-iff
    by (metis minus-mset-replicate-mset-eq-add-mset-filter-mset)
qed
next
  assume  $\nexists A$ . ord-res.is-maximal-lit (Pos A) C  $\wedge$   $2 \leq$  count C (Pos A) and C
  = C'
  thus ord-res.ground-factorizing** C C'  $\wedge$  ( $\nexists C''$ . ord-res.ground-factorizing C' C'')
    by (metis One-nat-def Suc-1 Suc-le-eq Suc-le-mono ord-res.ground-factorizing-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  $\langle$ ord-res.is-maximal-lit (Pos A) C $\rangle$  show C' = add-mset (Pos A) {#L  $\in$  #
  C. L  $\neq$  Pos A#}  $\implies$ 
    ord-res.ground-factorizing** C C'  $\wedge$  ( $\nexists C''$ . ord-res.ground-factorizing C' C'')
  proof (induction n arbitrary: C)
    case 0
      have ( $\nexists a$ . ord-res.ground-factorizing C a)
      proof (intro notI, elim exE)
        fix D assume ord-res.ground-factorizing C D
        thus False
      proof (cases rule: ord-res.ground-factorizing.cases)
        case (ground-factorizingI A' P')
          hence A' = A
            using  $\langle$ ord-res.is-maximal-lit (Pos A) C $\rangle$ 
            using linorder-lit.Uniq-is-maximal-in-mset
            by (metis Uniq-D literal.inject(1))
          thus False
            using  $\langle$ count C (Pos A) = Suc 0 $\rangle$   $\langle$ C = add-mset (Pos A') (add-mset
            (Pos A') P' $\rangle$  by simp
      qed
    qed
    thus ?case
      by (metis 0.premis(1) 0.premis(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-factorizing-iff)
  qed
qed

```

lemma *efac-spec*: $efac\ C = C \vee$
 $(\exists A. ord-res.is-maximal-lit\ (Pos\ A)\ C \wedge efac\ C = add-mset\ (Pos\ A)\ \{\#L \in\#$
 $C. L \neq Pos\ A\#\})$
using *efac-eq-disj*[*of C*]
proof (*elim disjE*)
assume $efac\ C = C$
thus $efac\ C = C \vee$
 $(\exists A. ord-res.is-maximal-lit\ (Pos\ A)\ C \wedge efac\ C = add-mset\ (Pos\ A)\ \{\#L \in\#$
 $C. L \neq Pos\ A\#\})$
by *metis*
next
assume $\exists!C'. efac\ C = C' \wedge ord-res.ground-factoring^{**}\ C\ C' \wedge$
 $(\nexists C''. ord-res.ground-factoring\ C'\ C'')$
then obtain C' **where**
 $efac\ C = C'$ **and**
 $ord-res.ground-factoring^{**}\ C\ C' \wedge (\nexists C''. ord-res.ground-factoring\ C'\ C'')$
by *metis*
thus $efac\ C = C \vee$
 $(\exists A. ord-res.is-maximal-lit\ (Pos\ A)\ C \wedge efac\ C = add-mset\ (Pos\ A)\ \{\#L \in\#$
 $C. L \neq Pos\ A\#\})$
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 $efac\ C = add-mset\ L\ \{\#K \in\# C. K \neq L\#\}$
proof –
from *assms* **obtain** C' **where**
 $efac\ C = C'$ **and**
 $ord-res.ground-factoring^{**}\ C\ C' \wedge (\nexists C''. ord-res.ground-factoring\ C'\ C'')$
using *ex1-efac-eq-factoring-chain* **by** *metis*
thus *?thesis*
unfolding *rtrancl-ground-factoring-iff*
proof (*elim disjE conjE*)
assume *hyps*: $\nexists A. ord-res.is-maximal-lit\ (Pos\ A)\ C \wedge 2 \leq count\ C\ (Pos\ A)$
 $C = C'$
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 $C = add-mset\ L\ \{\#K \in\# C. K \neq L\#\}$
by (*metis One-nat-def diff-zero minus-mset-replicate-mset-eq-add-mset-filter-mset*
replicate-mset-0)
thus $efac\ C = add-mset\ L\ \{\#K \in\# C. K \neq L\#\}$
using $\langle efac\ C = C' \rangle \langle C = C' \rangle$ **by** *argo*
next
assume $\exists A. ord-res.is-maximal-lit\ (Pos\ A)\ C \wedge C' = add-mset\ (Pos\ A)\ \{\#L$
 $\in\# C. L \neq Pos\ A\#\}$

thus $\text{efac } C = \text{add-mset } L \{ \#K \in \# C. K \neq L \# \}$
by (*metis L-max Uniq-D <efac C = C'> linorder-lit.Uniq-is-maximal-in-mset*)
qed
qed

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

lemma *set-mset-efac[simp]*: $\text{set-mset } (\text{efac } C) = \text{set-mset } C$
using *efac-spec[of C]*
proof (*elim disjE exE conjE*)
show $\text{efac } C = C \implies \text{set-mset } (\text{efac } C) = \text{set-mset } C$
by *simp*

next
fix A
assume *ord-res.is-maximal-lit (Pos A) C*
hence $\text{Pos } A \in \# C$
by (*simp add: linorder-lit.is-maximal-in-mset-iff*)

assume *efac-C-eq*: $\text{efac } C = \text{add-mset } (\text{Pos } A) \{ \#L \in \# C. L \neq \text{Pos } A \# \}$
show $\text{set-mset } (\text{efac } C) = \text{set-mset } C$
proof (*intro Set.subset-antisym Set.subsetI*)
fix L **assume** $L \in \# \text{efac } C$
then show $L \in \# C$
unfolding *efac-C-eq*
using $\langle \text{Pos } A \in \# C \rangle$ **by** *auto*

next
fix L **assume** $L \in \# C$
then show $L \in \# \text{efac } C$
unfolding *efac-C-eq*
by *simp*

qed
qed

lemma *efac-subset*: $\text{efac } C \subseteq \# C$
using *efac-spec[of C]*
proof (*elim disjE exE conjE*)
show $\text{efac } C = C \implies \text{efac } C \subseteq \# C$
by *simp*

next
fix A
assume *ord-res.is-maximal-lit (Pos A) C and*
efac-C-eq: $\text{efac } C = \text{add-mset } (\text{Pos } A) \{ \#L \in \# C. L \neq \text{Pos } A \# \}$
then show $\text{efac } C \subseteq \# C$
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 $\mathcal{I} :: 'f\ gterm\ set$ and $C :: 'f\ gclause$
shows $\mathcal{I} \models_{efac} C \longleftrightarrow \mathcal{I} \models C$
by (metis set-mset-efac true-cls-iff-set-mset-eq)

lemma obtains-positive-greatest-lit-if-efac-not-ident:

assumes $efac\ C \neq C$

obtains L where $is_pos\ L$ and $linorder_lit.is_greatest_in_mset\ (efac\ C)\ L$

proof –

from $\langle efac\ C \neq C \rangle$ obtain A where

$Pos\text{-}A\text{-maximal}$: $linorder_lit.is_maximal_in_mset\ C\ (Pos\ A)$ and

$efac\text{-}C\text{-eq}$: $efac\ C = add_mset\ (Pos\ A)\ \{\#L \in\# C.\ L \neq Pos\ A\#\}$

using $efac_spec$ by metis

assume hyp: $\bigwedge L. is_pos\ L \implies linorder_lit.is_greatest_in_mset\ (efac\ C)\ L \implies$

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 $efac\text{-}C\text{-eq}$ $linorder_lit.is_greatest_in_mset_iff$

using $Pos\text{-}A\text{-maximal}$ [unfolded $linorder_lit.is_maximal_in_mset_iff$]

by auto

qed

qed

lemma mempty-in-image-efac-iff[simp]: $\{\#\} \in efac\ 'N \longleftrightarrow \{\#\} \in N$

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 $C\text{-max-lit}$: $linorder_lit.is_maximal_in_mset\ C\ L$

shows $linorder_lit.is_greatest_in_mset\ (efac\ C)\ L$

unfolding $efac_spec\text{-if-}pos\text{-lit-is-maximal}$ [OF assms] $linorder_lit.is_greatest_in_mset_iff$

proof (intro conjI ballI)

show $L \in\# add_mset\ L\ \{\#K \in\# C.\ K \neq L\#\}$

by simp

next

fix $y :: 'f\ gterm\ literal$

assume $y \in\# remove1_mset\ L\ (add_mset\ L\ \{\#K \in\# C.\ K \neq L\#\})$

then show $y \prec_l L$

using $C\text{-max-lit}$ [unfolded $linorder_lit.is_maximal_in_mset_iff$]

by auto

qed

lemma $linorder_lit.is_maximal_in_mset\ (efac\ C)\ L \longleftrightarrow linorder_lit.is_maximal_in_mset\ C\ L$

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 \longleftrightarrow 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 $C \neq \text{efac } C$
shows $\exists C'. \text{ord-res.ground-factoring } C C'$
using *assms*
by (*metis converse-rtranclpE ex1-efac-eq-factoring-chain*)

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

lemma *efac-properties-if-not-ident:*
assumes $\text{efac } C \neq C$
shows $\text{efac } C \prec_c C$ **and** $\{\text{efac } C\} \models_e \{C\}$
proof –
have $\text{efac } C \subseteq\# C$
using *efac-subset .*
hence $\text{efac } C \preceq_c C$
using *subset-implies-reflclp-multp by blast*
thus $\text{efac } C \prec_c C$
using $\langle \text{efac } C \neq C \rangle$ **by** *order*

show $\{\text{efac } C\} \models_e \{C\}$
using $\langle \text{efac } C \subseteq\# C \rangle$ *true-cls-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 $(N \mid \cup \mid U_r \mid \cup \mid U_{ef}) C \implies$
linorder-lit.is-maximal-in-mset $C L \implies$
is-pos $L \implies$
 $U_{ef}' = \text{finsert } (efac C) U_{ef} \implies$
ord-res-2 $N (U_r, U_{ef}) (U_r, U_{ef}') \mid$

resolution:

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

inductive ord-res-2-step where

ord-res-2 $N S S' \implies \text{ord-res-2-step } (N, S) (N, S')$

inductive ord-res-2-final where

ord-res-final $(N \mid \cup \mid U_r \mid \cup \mid U_{ef}) \implies \text{ord-res-2-final } (N, (U_r, U_{ef}))$

inductive ord-res-2-load where

$N \neq \{\mid\} \implies \text{ord-res-2-load } N (N, (\{\mid\}, \{\mid\}))$

sublocale ord-res-2-semantic: semantics where

step = *ord-res-2-step* **and**

final = *ord-res-2-final*

proof *unfold-locales*

fix $S :: 'f \text{ gterm clause } fset \times 'f \text{ gterm clause } fset \times 'f \text{ gterm clause } fset$

obtain $N U_r U_{ef} :: 'f \text{ gterm clause } fset$ **where**

S-def: $S = (N, (U_r, U_{ef}))$

by (*metis prod.exhaust*)

assume *ord-res-2-final* S

hence $\{\#\} \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef} \vee \neg \text{ex-false-clause } (fset (N \mid \cup \mid U_r \mid \cup \mid U_{ef}))$

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 ‹{#} |∈| N |∪| Ur |∪| Uef›]
    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
next
case hyps: (resolution C L D CD Uef')
from hyps ‹{#} |∈| N |∪| Ur |∪| Uef› have C = {#}
  using is-least-false-clause-mempty[OF ‹{#} |∈| N |∪| Ur |∪| Uef›]
  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 |∪| Ur |∪| Uef))
have False if ord-res-2 N (Ur, Uef) x for x
  using that[unfolded S-def]
proof (cases N (Ur, Uef) x rule: ord-res-2.cases)
case hyps: (factoring C L Uef')
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 Uef')
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: $\text{linorder-cls.is-least-in-fset } \{ | C | \in | N. \neg \text{ord-res-Interp } (fset N) C \models C | \} C$
shows $\text{linorder-cls.is-least-in-fset } \{ | C | \in | N \cup | N'. \neg \text{ord-res-Interp } (fset (N \cup | N')) C \models C | \} C$
proof –
have
C-in: $C | \in | N$ **and**
C-not-entailed: $\neg \text{ord-res-Interp } (fset N) C \models C$ **and**
C-lt: $\forall x | \in | N. x \neq C \longrightarrow \neg \text{ord-res-Interp } (fset N) x \models x \longrightarrow C \prec_c x$
using *C-least linorder-cls.is-least-in-filter-iff* **by** *simp-all*

have $C | \in | N \cup | N'$
using *C-in* **by** *simp*

moreover have $\neg \text{ord-res-Interp } (fset (N \cup | N')) C \models C$
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 $C \prec_c x$
if
x-in: $x | \in | N \cup | N'$ **and**
x-neg: $x \neq C$ **and**
x-not-entailed: $\neg \text{ord-res-Interp } (fset (N \cup | N')) x \models x$
for x
proof –

from *x-in* **have** $x | \in | N \vee x | \in | N'$
by *simp*
thus $C \prec_c x$
proof (*elim disjE*)
assume *x-in*: $x | \in | N$

moreover have $\neg \text{ord-res-Interp } (fset N) x \models x$
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 $C \prec_c x$
using *C-lt*[*rule-format, of x*] *x-neg* **by** *argo*

next
assume $x | \in | N'$
then obtain x' **where** $x' | \in | N$ **and** $x' \subset \# x$ *set-mset* $x' = \text{set-mset } x$
using *irrelevant* **by** *metis*

have $x' \prec_c x$
using $\langle x' \subset \# x \rangle$ **by** (*metis strict-subset-implies-multp*)

```

moreover have  $C \preceq_c x'$ 
proof (cases  $x' = C$ )
  case True
  thus ?thesis
    by order
next
  case False

  have  $C \prec_c x'$ 
  proof (rule C-lt[rule-format])
    show  $x' \in N$ 
      using  $\langle x' \in N \rangle$  .
  next
  show  $x' \neq C$ 
    using False .
  next
  have  $\neg \text{ord-res-Interp } (fset (N \cup N')) x \models x'$ 
    using x-not-entailed  $\langle \text{set-mset } x' = \text{set-mset } x \rangle$ 
    by (metis true-cls-def)
  hence  $\neg \text{ord-res-Interp } (fset (N \cup N')) x' \models x'$ 
    by (metis  $\langle x' \prec_c x \rangle$  ord-res.entailed-clause-stays-entailed)
  thus  $\neg \text{ord-res-Interp } (fset N) x' \models x'$ 
    using extended-partial-model-entails-iff-partial-model-entails[
      of fset N fset N' x', OF finite-fset finite-fset irrelevant]
    using  $\langle x' \in N \rangle$  by simp
  qed
thus ?thesis
  by order
qed

```

```

ultimately show  $C \prec_c x$ 
  by order
qed

```

```

qed

```

```

ultimately show linorder-cls.is-least-in-fset  $\{|C \in N \cup N'\}$ 
   $\neg \text{ord-res-Interp } (fset (N \cup N')) C \models C\}$   $C$ 
  using C-in C-not-entailed
  unfolding linorder-cls.is-least-in-filter-iff by metis
qed

```

```

primrec fset-upto ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat fset}$  where
  fset-upto  $i\ 0 = (\text{if } i = 0 \text{ then } \{|0|\} \text{ else } \{|\})$  |
  fset-upto  $i\ (\text{Suc } n) = (\text{if } i \leq \text{Suc } n \text{ then } \text{finsert } (\text{Suc } n) (\text{fset-upto } i\ n) \text{ else } \{|\})$ 

```

```

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  $i \leq 1 + j \implies \text{List.upto } i (1 + j) = \text{List.upto } i j @ [1 + j]$ 
using upto-rec2 by simp

lemma fset-of-append-singleton:  $\text{fset-of-list } (xs @ [x]) = \text{finsert } x (\text{fset-of-list } xs)$ 
by simp

lemma fset-of-list ( $\text{List.upto } (int\ i) (int\ j)$ ) =  $int \mid^{\cdot}$  fset-upto  $i\ j$ 
proof (induction  $j$ )
  case 0
  show ?case
  by simp
next
  case (Suc  $j$ )
  show ?case
  proof (cases  $i \leq \text{Suc } j$ )
    case True
    hence AAA:  $int\ i \leq 1 + int\ j$ 
    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]:  $\text{fset } (\text{fset-upto } m\ n) = \{m..n\}$ 
apply (induction  $n$ )
apply simp
apply simp
using atLeastAtMostSuc-conv by presburger

lemma minus-mset-replicate-strict-subset-minus-msetI:
  assumes  $m < n$  and  $n < \text{count } C\ L$ 
  shows  $C - \text{replicate-mset } n\ L \subset\# C - \text{replicate-mset } m\ L$ 
proof -

```

from $\langle m < n \rangle$ **obtain** $k1$ **where** $n\text{-def}: n = m + \text{Suc } k1$
using less-natE **by** auto

with $\langle n < \text{count } C L \rangle$ **obtain** $k2$ **where**
 $\text{count-eq}: \text{count } C L = m + \text{Suc } k1 + \text{Suc } k2$
by $(\text{metis add.commute add-Suc group-cancel.add1 less-natE})$

define C_0 **where**
 $C_0 = \{\#K \in\# C. K \neq L\# \}$

have $C\text{-eq}: C = C_0 + \text{replicate-mset } m L + \text{replicate-mset } (\text{Suc } k1) L + \text{replicate-mset } (\text{Suc } k2) L$
using $C_0\text{-def count-eq}$
by $(\text{metis (mono-tags, lifting) filter-mset-eq group-cancel.add1 replicate-mset-plus union-filter-mset-complement})$

have $C - \text{replicate-mset } n L = C_0 + \text{replicate-mset } (\text{Suc } k2) L$
unfolding $C\text{-eq } n\text{-def}$
by $(\text{simp add: replicate-mset-plus})$

also have $\dots \subset\# C_0 + \text{replicate-mset } (\text{Suc } k1) L + \text{replicate-mset } (\text{Suc } k2) L$
by simp

also have $\dots = C - \text{replicate-mset } m L$
unfolding $C\text{-eq}$
by $(\text{simp add: replicate-mset-plus})$

finally show $?thesis$.

qed

lemma $\text{factoring-all-is-between-efac-and-original-clause}$:
fixes z
assumes
 $\text{is-pos } L$ **and** $\text{ord-res.is-maximal-lit } L C$ **and** $\text{count } C L = \text{Suc } (\text{Suc } n)$
 $m' \leq n$ **and**
 $z\text{-in}: z \in | \lambda i. C - \text{replicate-mset } i L | \uparrow \text{fset-upto } 0 m'$
shows $\text{efac } C \subset\# z$ **and** $z \subseteq\# C$

proof –

from $z\text{-in}$ **obtain** i **where**
 $i \leq m'$ **and**
 $z\text{-def}: z = C - \text{replicate-mset } i L$
by auto

have $i \leq n$
using $\langle i \leq m' \rangle \langle m' \leq n \rangle$ **by** presburger

hence $i < \text{count } C L$
using $\langle \text{count } C L = \text{Suc } (\text{Suc } n) \rangle$ **by** presburger

thus $z \subseteq\# C$
unfolding $z\text{-def}$ **by** simp

show $\text{efac } C \subset\# z$

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-fset-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
qed

```

lemma *ground-factoring-preserves-efac*:
assumes *ord-res.ground-factoring* P C
shows $\text{efac } P = \text{efac } C$
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 $\text{efac } P = \text{efac } C$
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 $\neg \text{ord-res-2-final } S$
shows $\exists S'. \text{ord-res-2-step } S S'$

proof –

from *assms* **obtain** N U_r U_{ef} **where**
 $S\text{-def: } S = (N, (U_r, U_{ef}))$ **and**
 $\{\#\} \notin N \cup U_r \cup U_{ef}$ **and**
 $\text{ex-false-clause } (fset (N \cup U_r \cup U_{ef}))$
by (*metis ord-res-2-final.intros ord-res-final-def surj-pair*)

obtain C **where** $C\text{-least-false: is-least-false-clause } (N \cup U_r \cup U_{ef})$ C
using $\langle \text{ex-false-clause } (fset (N \cup U_r \cup U_{ef})) \rangle$ *obtains-least-false-clause-if-ex-false-clause*
by *metis*

hence $C \neq \{\#\}$
using $\langle \{\#\} \notin N \cup U_r \cup U_{ef} \rangle$
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *metis*

then obtain L **where** $C\text{-max: linorder-lit.is-maximal-in-mset } C$ 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]* $S\text{-def}$ *is-pos-def*

by (*metis ord-res-2-step.intros*)

next

case (*Neg* A)

then obtain D **where**

$D \in N \cup U_r \cup U_{ef}$ **and**

$D \prec_c C$ **and**

ord-res.is-strictly-maximal-lit (*Pos A D*) **and**
ord-res.production (*fset (N | \cup | U_r | \cup | U_{ef}) D = {A}*)
using *bx-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 \vee ($\exists S'$. 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: $\forall C | \in | N2. ord-res.production (fset (N1 | \cup | N2)) C = \{\}$

and

C-in: C | \in | N1 **and**

C-least-false: is-least-false-clause (N1 | \cup | 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 | \in | N1*

using *C-in* .

next

have $\neg ord-res.Interp (fset (N1 | \cup | N2)) C \models C$

using *C-least-false[unfolded is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff]*
by *argo*

thus $\neg ord-res.interp (fset N1) C \cup ord-res.production (fset N1) C \models C$

unfolding *Interp-union-unproductive[of fset N1 fset N2, folded union-fset,*
OF finite-fset finite-fset N2-unproductive] .

next

fix *D*

have $\forall D | \in | N1 | \cup | N2. D \neq C \longrightarrow \neg ord-res.Interp (fset (N1 | \cup | N2)) D \models D \longrightarrow C \prec_c D$

using *C-least-false[unfolded is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff]*
by *argo*

moreover assume *D | \in | N1* **and** *D \neq C* **and** $\neg ord-res.Interp (fset N1) D \models D$

ultimately show *C \prec_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 $m \leq Suc n \implies (ord-res.ground-factoring \widehat{\sim} m)$
 $(C_0 + replicate_mset (Suc (Suc n)) (Pos A))$
 $(C_0 + replicate_mset (Suc (Suc n - m)) (Pos A))$
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*: $C \in N$ **and** *L-greatest*: *linorder-lit.is-greatest-in-mset* $C L$ **and**
L-pos: *is-pos* L
shows *ord-res-Interp* $N C \Vdash C$
proof (*cases ord-res.interp N C* $\Vdash C$)
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  $A \in \text{ord-res.production } N C$ 
  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  $s s' s'' :: 'f \text{ gterm clause fset} \times 'f \text{ gterm clause fset}$ 
  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 Ur1 Uef1 C1 L1 Uef'1)
      show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-2.cases)
      case (factoring Ur2 Uef2 C2 L2 Uef'2)
        with hyps1 show ?thesis
        by (metis Uniq-D Uniq-is-least-false-clause prod.inject)
      next
        case (resolution Ur Uef C L D CD Ur')
          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 Ur1 Uef1 C1 L1 D1 CD1 Ur'1)
      show ?thesis
      using step2
    proof (cases N s s'' rule: ord-res-2.cases)
      case (factoring Ur2 Uef2 C2 L2 Uef'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 Ur Uef C L D CD Ur')
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  $\implies$  ord-res-2-step x z  $\implies$  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 $\bigwedge x y. (ground-resolution D)^{-1-1} x y \implies x \prec_c y$

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 $DC \implies D \prec_c C$
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 $DC \implies D \prec_c C$
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 $DC \implies DC \prec_c C$
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 $DC \implies DC \prec_c C$

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 $C = (\text{THE } DC. \text{full-run } (\text{ground-resolution } D) C DC)$

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*: $\nexists DC. ground-resolution\ D\ C\ DC$

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** $Uniq: \exists_{\leq 1}\ y. full-run\ (ground-resolution\ D)\ C\ y$

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: $\nexists DDC. ground-resolution\ D\ DC\ DDC$

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** $Uniq: \exists_{\leq 1}\ y. full-run\ (ground-resolution\ D)\ C\ y$

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: $\nexists DDDC. ground-resolution\ D\ DDC\ DDDC$

shows $eres\ D\ C = DDC$

proof –

from *step1* **have** $(\text{ground-resolution } D)^{**} C DC$
by *auto*

with *step2* **have** $(\text{ground-resolution } D)^{**} C DDC$
by $(\text{metis rtranclp.simps})$

with *stuck* **have** $\text{full-run } (\text{ground-resolution } D) C DDC$
unfolding *full-run-def* **by** *argo*

moreover **have** $\text{Uniq: } \exists_{\leq 1} y. \text{full-run } (\text{ground-resolution } D) C y$
by $(\text{metis Uniq-ground-resolution Uniq-full-run})$

ultimately show *?thesis*
unfolding *eres-def* **by** $(\text{metis the1-equality}')$
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: \nexists DDDDC. ground-resolution D DDDC DDDDC
shows $\text{eres } D C = DDDC$

proof –
from *step1* **have** $(\text{ground-resolution } D)^{**} C DC$
by *auto*

with *step2* **have** $(\text{ground-resolution } D)^{**} C DDC$
by $(\text{metis rtranclp.simps})$

with *step3* **have** $(\text{ground-resolution } D)^{**} C DDDC$
by $(\text{metis rtranclp.simps})$

with *stuck* **have** $\text{full-run } (\text{ground-resolution } D) C DDDC$
unfolding *full-run-def* **by** *argo*

moreover **have** $\text{Uniq: } \exists_{\leq 1} y. \text{full-run } (\text{ground-resolution } D) C y$
by $(\text{metis Uniq-ground-resolution Uniq-full-run})$

ultimately show *?thesis*
unfolding *eres-def* **by** $(\text{metis the1-equality}')$
qed

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

lemma *eres-empty-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*: $\exists! DC. \text{eres } D \ C = DC \wedge \text{full-run (ground-resolution } D) \ C \ DC$
using *ex1-full-run[of ground-resolution D C]*
by (*metis Uniq-ground-resolution eres-def ground-resolution-terminates theI'*)

lemma *eres-le*: $\text{eres } D \ C \preceq_c \ C$
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 $\bigwedge x \ y. \text{ground-resolution } D \ x \ y \implies x \preceq_c \ C \implies y \preceq_c \ C$
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.multip-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 $\text{eres } D \ E \prec_c \ E$
proof –
have $\text{eres } D \ E \neq \ E$
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-rtranclE 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  $\exists!DC.$  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  $\nexists L.$  is-pos L  $\wedge$  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 $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$
using *assms nex-maximal-pos-lit-if-resolvable* **by** *blast*

corollary *nex-maximal-pos-lit-if-neq-eres*:

assumes $C \neq \text{eres } D \ C$

shows $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$

using *assms resolvable-if-neq-eres nex-maximal-pos-lit-if-resolvable* **by** *metis*

corollary *nex-strictly-maximal-pos-lit-if-neq-eres*:

assumes $C \neq \text{eres } D \ C$

shows $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$

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-lit.is-greatest-in-mset $D \ (\text{Pos } A) \wedge$

linorder-lit.is-maximal-in-mset $C \ (\text{Neg } A) \wedge$

$D = \text{add-mset } (\text{Pos } A) \ D' \wedge$

$C = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + C' \wedge \text{Neg } A \notin\# \ C' \wedge$

$DC = D' + \text{replicate-mset } m \ (\text{Neg } A) + C'$

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** $\text{count } C \ (\text{Neg } A) = \text{Suc } m$

by *simp*

define C'' **where**

$C'' = \{\#L \in\# \ C. \ L \neq \text{Neg } A\# \}$

have $C = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + C''$

using $\langle \text{count } C \ (\text{Neg } A) = \text{Suc } m \rangle \ C''\text{-def}$

by (*metis filter-eq-replicate-mset union-filter-mset-complement*)

show *?thesis*

proof (*intro exI conjI*)

show *linorder-lit.is-greatest-in-mset* $D \ (\text{Pos } A)$

using $\langle \text{linorder-lit.is-greatest-in-mset } D \ (\text{Pos } A) \rangle$.

next

show *linorder-lit.is-maximal-in-mset* $C \ (\text{Neg } A)$

using *ground-resolutionI* **by** *simp*

next

show $D = \text{add-mset } (\text{Pos } A) \ D'$

using $\langle D = \text{add-mset } (\text{Pos } A) \ D' \rangle$.

next

show $C = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + C''$

using $\langle C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'' \rangle$.
next
show $\text{Neg } A \not\# C''$
by (*simp add: C''-def*)
next
show $DC = D' + \text{replicate-mset } m (\text{Neg } A) + C''$
using $\langle DC = C' + D' \rangle \langle C = \text{add-mset } (\text{Neg } A) C' \rangle \langle C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'' \rangle$
by *simp*
qed
qed

lemma *relpowp-ground-resolutionD*:

assumes $n \neq 0$ **and** (*ground-resolution* $D \rightsquigarrow n$) $C \text{ Dn} C$
shows $\exists m A D' C'. \text{Suc } m \geq n \wedge$
 $\text{linorder-lit.is-greatest-in-mset } D (\text{Pos } A) \wedge$
 $\text{linorder-lit.is-maximal-in-mset } C (\text{Neg } A) \wedge$
 $D = \text{add-mset } (\text{Pos } A) D' \wedge$
 $C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C' \wedge \text{Neg } A \not\# C' \wedge$
 $Dn C = \text{repeat-mset } n D' + \text{replicate-mset } (\text{Suc } m - n) (\text{Neg } A) + C'$
using *assms*
proof (*induction n arbitrary: C*)
case 0
hence *False*
by *simp*
thus *?case ..*
next
case (*Suc n'*)
then obtain DC **where**
 $\text{ground-resolution } D C DC$ **and** (*ground-resolution* $D \rightsquigarrow n'$) $DC \text{ Dn} C$
by (*metis relpowp-Suc-E2*)

then obtain $m A D' C'$ **where**

$\text{linorder-lit.is-greatest-in-mset } D (\text{Pos } A)$ **and**
 $\text{linorder-lit.is-maximal-in-mset } C (\text{Neg } A)$
 $D = \text{add-mset } (\text{Pos } A) D'$ **and**
 $C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'$ **and**
 $\text{Neg } A \not\# C'$ **and**
 $DC = D' + \text{replicate-mset } m (\text{Neg } A) + C'$
using $\langle \text{ground-resolution } D C DC \rangle$ [*THEN ground-resolutionD*] **by** *metis*

have $\text{Neg } A \not\# D'$

using $\langle \text{linorder-lit.is-greatest-in-mset } D (\text{Pos } A) \rangle$
unfolding $\langle D = \text{add-mset } (\text{Pos } A) D' \rangle$
unfolding *linorder-lit.is-greatest-in-mset-iff*
by *auto*

show *?case*

proof (*cases n'*)

case 0
hence $DnC = DC$
using $\langle \text{ground-resolution } D \overset{\sim}{\sim} n' \rangle DC DnC \rangle$ **by** *simp*

show *?thesis*
unfolding 0 $\langle DnC = DC \rangle$
unfolding *repeat-mset-Suc repeat-mset-0 empty-neutral*
unfolding *diff-Suc-Suc minus-nat.diff-0*
using $\langle C = \text{replicate-mset } (Suc\ m) (Neg\ A) + C' \rangle \langle D = \text{add-mset } (Pos\ A)$
 $D' \rangle$
 $\langle DC = D' + \text{replicate-mset } m (Neg\ A) + C' \rangle \langle Neg\ A \notin\# C' \rangle$
 $\langle \text{linorder-lit.is-greatest-in-mset } D (Pos\ A) \rangle \langle \text{linorder-lit.is-maximal-in-mset}$
 $C (Neg\ A) \rangle$
using *linorder-lit.is-greatest-in-mset-iff*
by *blast*

next
case $(Suc\ n')$
hence $n' \neq 0$
by *presburger*
then obtain $m' A' D'' DC'$ **where** $n' \leq Suc\ m'$ **and**
ord-res.is-strictly-maximal-lit (Pos A') D **and**
ord-res.is-maximal-lit (Neg A') DC **and**
 $D = \text{add-mset } (Pos\ A') D''$ **and**
 $DC = \text{replicate-mset } (Suc\ m') (Neg\ A') + DC'$ **and**
 $Neg\ A' \notin\# DC'$ **and**
 $DnC = \text{repeat-mset } n' D'' + \text{replicate-mset } (Suc\ m' - n') (Neg\ A') + DC'$
using *Suc.IH[OF - $\langle \text{ground-resolution } D \overset{\sim}{\sim} n' \rangle DC DnC \rangle]$*
by *metis*

have $A' = A$
using $\langle \text{ord-res.is-strictly-maximal-lit } (Pos\ A') D \rangle \langle \text{ord-res.is-strictly-maximal-lit}$
 $(Pos\ A) D \rangle$
by $(\text{meson } \text{Uniq-D } \text{linorder-lit.Uniq-is-maximal-in-mset}$
 $\text{linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset } \text{literal.inject}(1))$

hence $D'' = D'$
using $\langle D = \text{add-mset } (Pos\ A') D'' \rangle \langle D = \text{add-mset } (Pos\ A) D' \rangle$ **by** *auto*

have $m = Suc\ m'$
using $\langle DC = D' + \text{replicate-mset } m (Neg\ A) + C' \rangle \langle DC = \text{replicate-mset}$
 $(Suc\ m') (Neg\ A') + DC' \rangle$
 $\langle Neg\ A \notin\# D' \rangle \langle Neg\ A \notin\# C' \rangle \langle Neg\ A' \notin\# DC' \rangle$
unfolding $\langle A' = A \rangle$
by $(\text{metis } \text{add-0 } \text{count-eq-zero-iff } \text{count-replicate-mset } \text{count-union } \text{union-commute})$

hence $DC' = D' + C'$
using $\langle DC = D' + \text{replicate-mset } m (Neg\ A) + C' \rangle \langle DC = \text{replicate-mset}$
 $(Suc\ m') (Neg\ A') + DC' \rangle$
by $(\text{simp } \text{add: } \langle A' = A \rangle)$

```

show ?thesis
proof (intro exI conjI)
  show  $Suc\ n' \leq Suc\ (Suc\ m')$ 
    using  $\langle n' \leq Suc\ m' \rangle$  by presburger
next
  show  $ord-res.is-strictly-maximal-lit\ (Pos\ A)\ D$ 
    using  $\langle ord-res.is-strictly-maximal-lit\ (Pos\ A)\ D \rangle$  .
next
  show  $ord-res.is-maximal-lit\ (Neg\ A)\ C$ 
    using  $\langle ord-res.is-maximal-lit\ (Neg\ A)\ C \rangle$  by metis
next
  show  $D = add-mset\ (Pos\ A)\ D'$ 
    using  $\langle D = add-mset\ (Pos\ A)\ D' \rangle$  .
next
  show  $C = replicate-mset\ (Suc\ (Suc\ m'))\ (Neg\ A) + C'$ 
    using  $\langle C = replicate-mset\ (Suc\ m)\ (Neg\ A) + C' \rangle$   $\langle m = Suc\ m' \rangle$  by argo
next
  show  $Neg\ A \notin\# C'$ 
    using  $\langle Neg\ A \notin\# C' \rangle$  .
next
  show  $DnC = repeat-mset\ (Suc\ n')\ D' + replicate-mset\ (Suc\ (Suc\ m') - Suc\ n')\ (Neg\ A) + C'$ 
    using  $\langle DnC = repeat-mset\ n'\ D'' + replicate-mset\ (Suc\ m' - n')\ (Neg\ A) + DC' \rangle$ 
    unfolding  $\langle A' = A \rangle$   $\langle D'' = D' \rangle$   $diff-Suc-Suc$   $\langle DC' = D' + C' \rangle$ 
    by simp
  qed
qed
qed

```

lemma *tranclp-ground-resolutionD*:

```

assumes  $(ground-resolution\ D)^{++}\ C\ DnC$ 
shows  $\exists n\ m\ A\ D'\ C'.\ Suc\ m \geq Suc\ n \wedge$ 
   $linorder-lit.is-greatest-in-mset\ D\ (Pos\ A) \wedge$ 
   $linorder-lit.is-maximal-in-mset\ C\ (Neg\ A) \wedge$ 
   $D = add-mset\ (Pos\ A)\ D' \wedge$ 
   $C = replicate-mset\ (Suc\ m)\ (Neg\ A) + C' \wedge Neg\ A \notin\# C' \wedge$ 
   $DnC = repeat-mset\ (Suc\ n)\ D' + replicate-mset\ (Suc\ m - Suc\ n)\ (Neg\ A) +$ 
   $C'$ 

```

proof –

```

from assms obtain  $n :: nat$  where
   $(ground-resolution\ D \rightsquigarrow Suc\ n)\ C\ DnC$ 
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 $\exists m A D' C'$.

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 $\bigwedge n. \text{Suc } n \neq 0$

by *presburger*

obtain *n* **where**

steps: (ground-resolution D \sim Suc n) C (eres D C) and

stuck: $\nexists x. \text{ground-resolution } D (eres D C) x$

using $\langle \text{eres } D C \neq C \rangle$ *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 $\langle \text{Suc } n \neq 0 \rangle$ 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 $\langle \text{eres } D C = \text{add-mset } (Neg A) \text{ ERES}' \rangle$ **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 $\neg \text{Neg } A \prec_l L$

proof (*elim disjE*)

```

    assume  $L \in \# \text{ repeat-mset } (\text{Suc } n) D'$ 
    hence  $L \in \# D'$ 
      using member-mset-repeat-msetD by metis
    hence  $L \prec_l \text{ Pos } A$ 
      using D-max-lit
    unfolding D-eq linorder-lit.is-greatest-in-mset-iff by simp
    also have  $\text{Pos } A \prec_l \text{ Neg } A$ 
      by simp
    finally show ?thesis
      by order
  next
    assume  $L \in \# C'$ 
    thus ?thesis
      using C-eq  $\langle L \neq \text{Neg } A \rangle$  C-max-lit linorder-lit.is-maximal-in-mset-iff by
auto
  qed
  qed

  moreover have  $D \prec_c \text{ eres } D C$ 
    using D-max-lit
    using  $\langle \text{ord-res.is-maximal-lit } (\text{Neg } A) (\text{eres } D C) \rangle$ 
    using linorder-lit.multipHO-if-maximal-less-than-maximal[of D Pos A eres D
C Neg A, simplified]
    using multipDM-imp-multip multipHO-imp-multipDM by blast

  ultimately show  $\exists x. \text{ ground-resolution } D (\text{eres } D C) x$ 
    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  $\langle \text{eres } D C = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (\text{Suc } m - \text{Suc } n) (\text{Neg } A) + C' \rangle$ 
    using  $\langle \text{Suc } n \leq \text{Suc } m \rangle$  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  $D = \text{add-mset } (\text{Pos } A) D'$ 
      using D-eq .
  next
    show  $C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'$ 

```

```

    using C-eq .
  next
    show Neg A  $\notin$  C'
      using  $\langle$ Neg A  $\notin$  C' $\rangle$  .
  next
    show eres D C = repeat-mset (Suc m) D' + C'
      using eres-eq unfolding  $\langle$ m = n $\rangle$  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  $\in$  C  $\vee$  L  $\in$  D
  shows L  $\in$  C  $\vee$  L  $\in$  D
proof (cases eres C D = D)
  assume eres C D = D
  thus L  $\in$  C  $\vee$  L  $\in$  D
    using  $\langle$ L  $\in$  eres C D $\rangle$  by argo
next
  assume eres C D  $\neq$  D
  thus L  $\in$  C  $\vee$  L  $\in$  D
    using  $\langle$ L  $\in$  eres C D $\rangle$ 
  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  $\in$  C  $\vee$  K  $\in$  D
  shows K  $\in$  C  $\wedge$  K  $\neq$  -L  $\vee$  K  $\in$  D
proof (cases eres C D = D)
  assume eres C D = D
  thus K  $\in$  C  $\wedge$  K  $\neq$  -L  $\vee$  K  $\in$  D
    using K-in by argo
next
  assume eres C D  $\neq$  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  $\notin$  D' and
    eres C D = repeat-mset (Suc m) C' + D'
    using eres-not-identD by metis

  have L-eq: L = Neg A

```

using $D\text{-max-lit } D\text{-max-lit}'$ **by** (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have $K \in\# \text{repeat-mset } (\text{Suc } m) C' + D'$
using $K\text{-in unfolding } \langle \text{eres } C D = \text{repeat-mset } (\text{Suc } m) C' + D' \rangle$.

hence $K \in\# \text{repeat-mset } (\text{Suc } m) C' \vee K \in\# D'$
unfolding *Multiset.union-iff* .

hence $K \in\# C' \vee K \in\# D'$
unfolding *member-mset-repeat-mset-Suc* .

thus $K \in\# C \wedge K \neq -L \vee K \in\# D$
proof (*elim disjE*)
assume $K \in\# C'$

hence $K \in\# C \wedge K \neq -L$
using *C-max-lit*
unfolding *C-eq L-eq linorder-lit.is-greatest-in-mset-iff* **by** *auto*

thus $K \in\# C \wedge K \neq -L \vee K \in\# D$..

next
assume $K \in\# D'$

hence $K \in\# D$
unfolding *D-eq* **by** *simp*

thus $K \in\# C \wedge K \neq -L \vee K \in\# D$..

qed
qed

lemma *stronger-lit-in-one-of-resolvents-if-in-eres*:
fixes $K L :: 'f \text{ gterm literal}$ **and** $C D :: 'f \text{ gclause}$
assumes $\text{eres } C D \neq D$ **and**
D-max-lit: linorder-lit.is-maximal-in-mset D L **and**
K-in-eres: K \in\# eres C D

shows $K \in\# C \wedge K \neq -L \vee K \in\# D \wedge K \neq L$
proof –

obtain $m :: \text{nat}$ **and** $A :: 'f \text{ gterm}$ **and** $C' D' :: 'f \text{ gterm literal multiset}$ **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 \notin\# D' **and**
eres C D = repeat-mset (Suc m) C' + D'
using $\langle \text{eres } C D \neq D \rangle$ [*THEN eres-not-identD*] **by** *metis*

have $L = \text{Neg } A$
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 $K \in\# \text{repeat-mset } (\text{Suc } m) C' + D'$
using $K\text{-in-eres unfolding } \langle \text{eres } C D = \text{repeat-mset } (\text{Suc } m) C' + D' \rangle .$

hence $K \in\# \text{repeat-mset } (\text{Suc } m) C' \vee K \in\# D'$
unfolding $\text{Multiset.union-iff} .$

hence $K \in\# C' \vee K \in\# D'$
unfolding $\text{member-mset-repeat-mset-Suc} .$

thus $K \in\# C \wedge K \neq -L \vee K \in\# D \wedge K \neq L$
proof (elim disjE)
assume $K \in\# C'$

hence $K \in\# C \wedge K \neq -L$
using $C\text{-max-lit}[\text{unfolded linorder-lit.is-greatest-in-mset-iff}]$
unfolding $\langle C = \text{add-mset } (\text{Pos } A) C' \rangle \langle L = \text{Neg } A \rangle$
by auto

thus $?thesis$
by argo

next

assume $K \in\# D'$

hence $K \in\# D \wedge K \neq L$
unfolding $\langle D = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + D' \rangle \langle L = \text{Neg } A \rangle$
using $\langle \text{Neg } A \notin\# D' \rangle$
by auto

thus $?thesis$
by argo

qed

qed

lemma $\text{lit-in-eres-lt-greatest-lit-in-greatest-resolvent}$:

fixes $K L :: 'f \text{ gterm literal}$ **and** $C D :: 'f \text{ gclause}$

assumes $\text{eres } C D \neq D$ **and**

$D\text{-max-lit: linorder-lit.is-maximal-in-mset } D L$ **and**

$-L \notin\# D$ **and**

$K\text{-in-eres: } K \in\# \text{eres } C D$

shows $\text{atm-of } K \prec_t \text{atm-of } L$

proof –

obtain $m :: \text{nat}$ **and** $A :: 'f \text{ gterm}$ **and** $C' D' :: 'f \text{ gterm literal multiset}$ **where**

$C\text{-max-lit: ord-res.is-strictly-maximal-lit } (\text{Pos } A) C$ **and**

$C\text{-def: } C = \text{add-mset } (\text{Pos } A) C'$ **and**

$D = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + D'$ **and**

$\text{Neg } A \notin\# D'$ **and**

$\text{eres } C D = \text{repeat-mset } (\text{Suc } m) C' + D'$


```

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' uni-
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}  $\models_e$  {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  $\notin\#$  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-resolution] by
metis

fix I :: 'f gterm set
assume I  $\models$  eres C D0
show I  $\models$  D
proof (cases eres C D0 = D)
  case True
  thus ?thesis
    using ⟨I  $\models$  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  $\notin\#$  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  $\models$  repeat-mset (Suc k) C' + D'
    using ⟨I  $\models$  eres C D0⟩
    unfolding ⟨eres C D0 = eres C D⟩ ⟨eres C D = repeat-mset (Suc k) C' +
D'⟩ .

  hence I  $\models$  D' ∨ I  $\models$  repeat-mset (Suc k) C'
    by auto

```

```

thus  $I \Vdash D$ 
proof (elim disjE)
  assume  $I \Vdash D'$ 
  thus  $I \Vdash D$ 
    unfolding  $\langle D = \text{replicate-mset } (\text{Suc } k) (\text{Neg } A) + D' \rangle$ 
    by simp
  next
  assume  $I \Vdash \text{repeat-mset } (\text{Suc } k) C'$ 
  thus  $I \Vdash D$ 
    using  $\langle D = \text{replicate-mset } (\text{Suc } k) (\text{Neg } A) + D' \rangle$ 
    using  $\langle D = \text{repeat-mset } (\text{Suc } n) C' + \text{replicate-mset } (\text{Suc } m - \text{Suc } n) (\text{Neg } A) + D_0' \rangle$ 
    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  $N D \neq \{\}$  and
  C-true: ord-res-Interp  $N DC \Vdash DC$ 
shows ord-res-Interp  $N C \Vdash C$ 
proof –
  obtain  $n$  where
    steps: (ground-resolution D)  $\rightsquigarrow$   $\text{Suc } n) C DC$ 
    using  $\langle (\text{ground-resolution } D)^{++} C DC \rangle$ 
    by (metis less-not-refl not0-implies-Suc tranclp-power)

```

obtain $m A D' C'$ **where**

```

 $n \leq m$  and
ord-res.is-strictly-maximal-lit (Pos A)  $D$  and
ord-res.is-maximal-lit (Neg A)  $C$  and
 $D = \text{add-mset } (\text{Pos } A) D'$  and
 $C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'$  and
Neg A  $\notin\# C'$  and
 $DC = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C'$ 
using relpowp-ground-resolutionD[OF Suc-not-Zero steps]
by (metis diff-Suc-Suc Suc-le-mono)

```

have *Neg A* $\notin\# D'$

```

by (metis  $\langle D = \text{add-mset } (\text{Pos } A) D' \rangle \langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D \rangle$ 
  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 $DC \prec_c C$

```

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 ∈# DC
  hence L ∈# D' ∨ L ∈# C'
  unfolding ⟨DC = repeat-mset (Suc n) D' + replicate-mset (m - n) (Neg
A) + C'⟩ ⟨m = n⟩
  using member-mset-repeat-msetD by fastforce
  hence L <i Neg A
  using ⟨ord-res.is-strictly-maximal-lit (Pos A) D'⟩ ⟨ord-res.is-maximal-lit (Neg
A) C'⟩
  unfolding ⟨D = add-mset (Pos A) D'⟩ ⟨C = replicate-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 ∉# 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 ∈# C
  by (simp add: ⟨C = replicate-mset (Suc m) (Neg A) + C'⟩)

  ultimately show ∃ K ∈# C. L <i K
  by metis
qed
next
case False
hence n < m
using ⟨n ≤ m⟩ by presburger

have multpHO (<i) DC C
proof (rule linorder-lit.multpHO-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 ∈# DC
  unfolding ⟨DC = repeat-mset (Suc n) D' + replicate-mset (m - n) (Neg
A) + C'⟩
  using ⟨n < m⟩ by simp
  next
  fix L
  assume L ∈# DC and L ≠ Neg A
  hence L ∈# D' ∨ L ∈# C'
  unfolding ⟨DC = repeat-mset (Suc n) D' + replicate-mset (m - n) (Neg
A) + C'⟩

```

```

    by (metis in-replicate-mset member-mset-repeat-msetD union-iff)
  thus  $\neg \text{Neg } A \prec_l L$ 
    using  $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D \rangle \langle \text{ord-res.is-maximal-lit } (\text{Neg } A) C \rangle$ 
    unfolding  $\langle D = \text{add-mset } (\text{Pos } A) D' \rangle \langle C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C' \rangle$ 
    unfolding linorder-lit.is-maximal-in-mset-iff linorder-lit.is-greatest-in-mset-iff
    by (metis  $\langle L \neq \text{Neg } A \rangle$  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  $\langle \text{ord-res.is-maximal-lit } (\text{Neg } A) C \rangle$  .
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  $\langle DC = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C' \rangle$  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:  $\langle \text{Neg } A \notin\# C' \rangle \langle \text{Neg } A \notin\# D' \rangle$  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:  $\langle \text{Neg } A \notin\# C' \rangle$  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  $\langle C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C' \rangle$  by simp
  finally show count DC (Neg A) < count C (Neg A) .
qed
thus ?thesis
  by (simp add: multpDM-imp-multp multpHO-imp-multpDM)
qed

with C-true have ord-res-Interp N C  $\models DC$ 
  using ord-res.entailed-clause-stays-entailed by metis

thus ord-res-Interp N C  $\models C$ 
  unfolding true-cls-def
  proof (elim bexE)

```

fix L
assume
 $L\text{-in}: L \in\# DC$ **and**
 $L\text{-true}: \text{ord-res-Interp } N C \models_l L$

from $L\text{-in}$ **have** $L \in\# D' \vee L = \text{Neg } A \vee L \in\# C'$
unfolding $\langle DC = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C' \rangle$
by $(\text{metis in-replicate-mset member-mset-repeat-msetD union-iff})$

moreover have $L \notin\# D'$
proof (rule notI)
assume $L \in\# D'$

moreover have $\neg \text{ord-res.interp } N (\text{add-mset } (\text{Pos } A) D') \models \text{add-mset } (\text{Pos } A) D'$
using $D\text{-productive}[\text{unfolded } \langle D = \text{add-mset } (\text{Pos } A) D' \rangle]$
unfolding $\text{ord-res.production-unfold}$
by fast

ultimately have $\neg \text{ord-res.interp } N (\text{add-mset } (\text{Pos } A) D') \models_l L$
by auto

have $L \prec_l \text{Pos } A$
using $\langle D = \text{add-mset } (\text{Pos } A) D' \rangle \langle L \in\# D' \rangle \langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D \rangle$
 $\text{linorder-lit.is-greatest-in-mset-iff}$ **by** fastforce

have $\neg \text{ord-res-Interp } N C \models_l L$
proof $(\text{cases } L)$
case $(\text{Pos } B)$
hence $B \notin \text{ord-res.interp } N (\text{add-mset } (\text{Pos } A) D')$
using $\langle \neg \text{ord-res.interp } N (\text{add-mset } (\text{Pos } A) D') \models_l L \rangle$ **by** simp

moreover have $\text{add-mset } (\text{Pos } A) D' \prec_c C$
by $(\text{metis } \langle D = \text{add-mset } (\text{Pos } A) D' \rangle \langle \bigwedge \text{thesis. } (\bigwedge m A D' C'. \llbracket n \leq m; \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D; \text{ord-res.is-maximal-lit } (\text{Neg } A) C; D = \text{add-mset } (\text{Pos } A) D'; C = \text{replicate-mset } (\text{Suc } m) (\text{Neg } A) + C'; \text{Neg } A \notin\# C'; DC = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C' \rrbracket \implies \text{thesis} \rangle \implies \text{thesis} \rangle \text{linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.multip}_{HO}\text{-if-maximal-less-than-maximal multip}_{DM}\text{-imp-multip multip}_{HO}\text{-imp-multip}_{DM} \text{ord-res.less-lit-simps}(2) \text{ reflclp-iff})$

ultimately have $B \notin \text{ord-res.interp } N C$
using $\langle L \prec_l \text{Pos } A \rangle [\text{unfolded Pos, simplified}]$
using $\text{ord-res.interp-fixed-for-smaller-literals}$
by $(\text{metis } \langle D = \text{add-mset } (\text{Pos } A) D' \rangle \langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D \rangle$
 $\text{linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.sel}(1))$

moreover have $B \notin \text{ord-res.production } N \ C$
by (*metis* $\langle \text{Uniq-D } \langle \text{ord-res.is-maximal-lit } (Neg \ A) \ C \rangle \text{ ground-ordered-resolution-calculus.mem-production.}$
 $\text{linorder-lit.Uniq-is-maximal-in-mset linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset}$
 $\text{literal.simps}(4) \text{ ord-res.ground-ordered-resolution-calculus-axioms} \rangle$)

ultimately show *?thesis*
unfolding Pos **by** *simp*

next

case ($Neg \ B$)

hence $B \in \text{ord-res.interp } N \ (\text{add-mset } (Pos \ A) \ D')$

using $\langle \neg \text{ord-res.interp } N \ (\text{add-mset } (Pos \ A) \ D') \models_l L \rangle$ **by** *simp*

moreover have $\text{add-mset } (Pos \ A) \ D' \prec_c C$

by (*metis* $\langle D = \text{add-mset } (Pos \ A) \ D' \rangle \langle \wedge \text{thesis. } (\wedge m \ A \ D' \ C'. \llbracket n$
 $\leq m; \text{ord-res.is-strictly-maximal-lit } (Pos \ A) \ D; \text{ord-res.is-maximal-lit } (Neg \ A) \ C;$
 $D = \text{add-mset } (Pos \ A) \ D'; C = \text{replicate-mset } (Suc \ m) \ (Neg \ A) + C'; Neg$
 $A \notin\# C'; DC = \text{repeat-mset } (Suc \ n) \ D' + \text{replicate-mset } (m - n) \ (Neg \ A)$
 $+ C \rrbracket \implies \text{thesis} \rangle \implies \text{thesis} \rangle \text{linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset}$
 $\text{linorder-lit.mulp}_{HO}\text{-if-maximal-less-than-maximal mulp}_{DM}\text{-imp-mulp mulp}_{HO}\text{-imp-mulp}_{DM}$
 $\text{ord-res.less-lit-simps}(2) \text{ reflclp-iff} \rangle$)

ultimately have $B \in \text{ord-res.interp } N \ C$

by (*metis* $Un\text{-iff ord-res.not-interp-to-Interp-imp-le linorder-cls.leD}$)

then show *?thesis*

unfolding Neg

by *simp*

qed

with $L\text{-true}$ **show** $False$

by *contradiction*

qed

ultimately have $L \in\# C$

unfolding $\langle C = \text{replicate-mset } (Suc \ m) \ (Neg \ A) + C' \rangle$ **by** *simp*

with $L\text{-true}$ **show** $\exists L \in\# C. \text{ord-res-Interp } N \ C \models_l L$

by *metis*

qed

qed

lemma *clause-true-if-eres-true:*

assumes

$(\text{ground-resolution } D1)^{++} \ D2 \ C$ **and**

$C \neq \text{eres } D1 \ C$ **and**

$\text{eres-C-true: ord-res-Interp } N \ (\text{eres } D1 \ C) \models \text{eres } D1 \ C$

shows $\text{ord-res-Interp } N \ C \models C$

proof –

obtain n **where**

steps: $\langle \text{ground-resolution } D1 \rightsquigarrow \text{Suc } n \rangle D2 \ C$
using $\langle \text{ground-resolution } D1 \rangle^{++} D2 \ C \rangle$
by $(\text{metis less-not-refl not0-implies-Suc tranclp-power})$

obtain $m \ A \ D' \ C'$ **where**

$n \leq m$ **and**
ord-res.is-strictly-maximal-lit $(\text{Pos } A) \ D1$ **and**
ord-res.is-maximal-lit $(\text{Neg } A) \ D2$ **and**
 $D1 = \text{add-mset } (\text{Pos } A) \ D'$ **and**
 $D2 = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + C'$ **and**
 $\text{Neg } A \notin\# C'$ **and**
 $C = \text{repeat-mset } (\text{Suc } n) \ D' + \text{replicate-mset } (m - n) \ (\text{Neg } A) + C'$
using $\text{relpowp-ground-resolutionD}[OF \ \text{Suc-not-Zero steps}]$
by $(\text{metis diff-Suc-Suc Suc-le-mono})$

have $\text{Neg } A \notin\# D'$

by $(\text{metis } \langle D1 = \text{add-mset } (\text{Pos } A) \ D' \rangle \langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ D1 \rangle$
 $\text{ord-res.less-lit-simps}(4) \ \text{linorder-lit.is-greatest-in-mset-iff linorder-trm.eq-refl}$
 $\text{linorder-trm.leD remove1-mset-add-mset-If})$

obtain $m' \ C''$ **where**

$C = \text{replicate-mset } (\text{Suc } m') \ (\text{Neg } A) + C''$ **and**
 $\text{Neg } A \notin\# C''$ **and**
 $\text{eres } D1 \ C = \text{repeat-mset } (\text{Suc } m') \ D' + C''$
using $\langle C \neq \text{eres } D1 \ C \rangle \ \text{eres-not-identD}$
using $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ D1 \rangle \ \text{linorder-lit.Uniq-is-greatest-in-mset}$
using $\langle D1 = \text{add-mset } (\text{Pos } A) \ D' \rangle$
by $(\text{metis Uniq-D add-mset-remove-trivial literal.inject}(1))$

have $m - n = \text{Suc } m'$

proof –

have $\text{count } C \ (\text{Neg } A) = \text{count } (\text{repeat-mset } (\text{Suc } n) \ D') \ (\text{Neg } A) +$
 $\text{count } (\text{replicate-mset } (m - n) \ (\text{Neg } A)) \ (\text{Neg } A) + \text{count } C' \ (\text{Neg } A)$
using $\langle C = \text{repeat-mset } (\text{Suc } n) \ D' + \text{replicate-mset } (m - n) \ (\text{Neg } A) + C' \rangle$
by *simp*
also have $\dots = \text{count } D' \ (\text{Neg } A) * \text{Suc } n + \text{count } \{\#\text{Neg } A\#\} \ (\text{Neg } A) * (m$
 $- n) +$
 $\text{count } C' \ (\text{Neg } A)$
by *simp*
also have $\dots = 0 * \text{Suc } n + 1 * (m - n) + 0$
using $\langle \text{Neg } A \notin\# D' \rangle \langle \text{Neg } A \notin\# C' \rangle$ **by** $(\text{simp add: count-eq-zero-iff})$
also have $\dots = m - n$
by *presburger*
finally have $\text{count } C \ (\text{Neg } A) = m - n .$

have $\text{count } C \ (\text{Neg } A) = \text{count } (\text{replicate-mset } (\text{Suc } m') \ (\text{Neg } A)) \ (\text{Neg } A) +$
 $\text{count } C'' \ (\text{Neg } A)$

using $\langle C = \text{replicate-mset } (\text{Suc } m') (\text{Neg } A) + C'' \rangle$ **by** *simp*
also have $\dots = \text{count } \{\#\text{Neg } A\# \} (\text{Neg } A) * \text{Suc } m' + \text{count } C'' (\text{Neg } A)$
by *simp*
also have $\dots = 1 * \text{Suc } m' + 0$
using $\langle \text{Neg } A \notin \# C'' \rangle$ **by** (*simp add: count-eq-zero-iff*)
also have $\dots = \text{Suc } m'$
by *presburger*
finally have $\text{count } C (\text{Neg } A) = \text{Suc } m'$.

show *?thesis*
using $\langle \text{count } C (\text{Neg } A) = m - n \rangle \langle \text{count } C (\text{Neg } A) = \text{Suc } m' \rangle$ **by** *argo*
qed

hence $C'' = \text{repeat-mset } (\text{Suc } n) D' + C'$
using $\langle C = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C' \rangle$
 $\langle C = \text{replicate-mset } (\text{Suc } m') (\text{Neg } A) + C'' \rangle$
by *simp*

hence *eres-D1-C-eq*: $\text{eres } D1 C = \text{repeat-mset } (\text{Suc } m' + \text{Suc } n) D' + C'$
using $\langle \text{eres } D1 C = \text{repeat-mset } (\text{Suc } m') D' + C'' \rangle$ **by** *simp*

have *ord-res-Interp N (eres D1 C) \models eres D1 C*
using *eres-C-true* .

moreover have *eres D1 C \prec_c C*
using *eres-le[of D1 C] $\langle C \neq \text{eres } D1 C \rangle$ by order*

ultimately have *ord-res-Interp N C \models eres D1 C*
using *ord-res.entailedd-clause-stays-entailedd by metis*

thus *ord-res-Interp N C \models C*
unfolding *true-cls-def*
proof (*elim bexE*)
fix *L*
assume
L-in: L $\in \#$ eres D1 C and
L-true: ord-res-Interp N C \models_l L

from *L-in* **have** $L \in \# D' \vee L \in \# C'$
unfolding *eres-D1-C-eq*
using *member-mset-repeat-msetD* **by** *fastforce*
hence $L \in \# C$
by (*auto simp: $\langle C = \text{repeat-mset } (\text{Suc } n) D' + \text{replicate-mset } (m - n) (\text{Neg } A) + C' \rangle$*)
with *L-true* **show** $\exists L \in \# C. \text{ord-res-Interp } N C \models_l L$
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 $(N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) C \implies$

linorder-lit.is-maximal-in-mset $C L \implies$

is-pos $L \implies$

$U_{ef}' = \text{finsert } (efac C) U_{ef} \implies$

ord-res-3 $N (U_{er}, U_{ef}) (U_{er}, U_{ef}') \mid$

resolution:

is-least-false-clause $(N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) C \implies$

linorder-lit.is-maximal-in-mset $C L \implies$

is-neg $L \implies$

$D \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef} \implies$

$D \prec_c C \implies$

ord-res.production $(\text{fset } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef})) D = \{\text{atm-of } L\} \implies$

$U_{er}' = \text{finsert } (eres D C) U_{er} \implies$

ord-res-3 $N (U_{er}, U_{ef}) (U_{er}', U_{ef})$

inductive *ord-res-3-step* **where**

ord-res-3 $N s s' \implies \text{ord-res-3-step } (N, s) (N, s')$

inductive *ord-res-3-final* **where**

ord-res-final $(N \mid \cup \mid U_{rr} \mid \cup \mid U_{ef}) \implies \text{ord-res-3-final } (N, (U_{rr}, U_{ef}))$

inductive *ord-res-3-load* **where**

$N \neq \{\mid\} \implies \text{ord-res-3-load } N (N, (\{\mid\}, \{\mid\}))$

sublocale *ord-res-3-semantic: semantics* **where**

step = *ord-res-3-step* **and**

final = *ord-res-3-final*

proof *unfold-locales*

fix $S3 :: 'f \text{ gterm clause fset} \times 'f \text{ gterm clause fset} \times 'f \text{ gterm clause fset}$

obtain $N U_{rr} U_{ef} :: 'f \text{ gterm clause fset}$ **where**

S3-def: $S3 = (N, (U_{rr}, U_{ef}))$

by (*metis prod.exhaust*)

assume *ord-res-3-final S3*

hence $\{\#\} \in N \cup U_{rr} \cup U_{ef} \vee \neg \text{ex-false-clause} (fset (N \cup U_{rr} \cup U_{ef}))$
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 $\{\#\} \in N \cup U_{rr} \cup U_{ef}$

hence *is-least-false-clause* $(N \cup U_{rr} \cup U_{ef}) \{\#\}$

using *is-least-false-clause-mempty* **by** *metis*

hence $\nexists C L. \text{is-least-false-clause} (N \cup U_{rr} \cup U_{ef}) C \wedge \text{linorder-lit.is-maximal-in-mset } C L$

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 $\nexists x. \text{ord-res-3 } N (U_{rr}, U_{ef}) x$

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 $\neg \text{ex-false-clause} (fset (N \cup U_{rr} \cup U_{ef}))$

hence $\nexists C. \text{is-least-false-clause} (N \cup U_{rr} \cup U_{ef}) C$

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 $\nexists x. \text{ord-res-3 } N (U_{rr}, U_{ef}) x$

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 $\forall C \in U_{pr}. \exists D1 \in N \cup U_{er} \cup U_{ef}. \exists D2 \in N \cup U_{er} \cup U_{ef}. (\text{ground-resolution } D1)^{++} D2 C \wedge C \neq \text{eres } D1 D2 \wedge \text{eres } D1 D2 \in U_{er}$

shows *is-least-false-clause* $(N \cup U_{pr} \cup U_{er} \cup U_{ef}) = \text{is-least-false-clause} (N \cup U_{er} \cup U_{ef})$

proof –

have *is-least-false-clause* $(N \cup U_{pr} \cup U_{er} \cup U_{ef}) = \text{is-least-false-clause} (N \cup U_{er} \cup U_{ef} \cup U_{pr})$

by (*simp add: sup-commute sup-left-commute*)

also have $\dots = \text{is-least-false-clause} (N \cup U_{er} \cup U_{ef})$

proof (*rule is-least-false-clause-union-cancel-right*)

show $\forall C \in U_{pr}. \forall U. \text{ord-res.production } U C = \{\}$

```

proof (intro ballI)
  fix C
  assume C |∈| Upr
  hence ∃ D |∈| N |∪| Uer |∪| Uef. (∃ C'. ground-resolution D C C')
  using assms by (metis eres-eq-after-tranclp-ground-resolution resolvable-if-neq-eres)
  hence ∄ L. is-pos L ∧ 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 |∈| Upr. ∃ D |∈| N |∪| Uer |∪| Uef. D <c C ∧ {D} ||e {C}
proof (intro ballI)
  fix C
  assume C |∈| Upr
  then obtain D1 D2 where
    D1 |∈| N |∪| Uer |∪| Uef and
    D2 |∈| N |∪| Uer |∪| Uef and
    (ground-resolution D1)++ D2 C and
    C ≠ eres D1 D2 and
    eres D1 D2 |∈| Uer
    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 |∪| Uer |∪| Uef. 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 |∪| Uer |∪| Uef
    using ⟨eres D1 D2 |∈| Uer⟩ 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  $\implies$  ord-res-3-step x z  $\implies$  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  $\neg$  ord-res-3-final S
shows  $\exists S'$ . ord-res-3-step S S'
proof -

```

```

from assms obtain N Ur Uef where
  S-def: S = (N, (Ur, Uef)) and
  {#} | $\notin$ | N | $\cup$ | Ur | $\cup$ | Uef and
  ex-false-clause (fset (N | $\cup$ | Ur | $\cup$ | Uef))
by (metis ord-res-3-final.intros ord-res-final-def surj-pair)

```

```

obtain C where C-least-false: is-least-false-clause (N | $\cup$ | Ur | $\cup$ | Uef) C
using  $\langle$ ex-false-clause (fset (N | $\cup$ | Ur | $\cup$ | Uef)) $\rangle$  obtains-least-false-clause-if-ex-false-clause
by metis

```

```

hence C  $\neq$  {#}
using  $\langle$ {#} | $\notin$ | N | $\cup$ | Ur | $\cup$ | Uef $\rangle$ 
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.factorizing[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**

```

     $D \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef}$  and
     $D \prec_c C$  and
    ord-res.is-strictly-maximal-lit (Pos A)  $D$  and
    ord-res.production (fset ( $N \mid \cup \mid U_r \mid \cup \mid U_{ef}$ ))  $D = \{A\}$ 
using bx-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  $\vee$  ( $\exists S'$ . 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  $\mathcal{F} C = (\text{if } C \mid \in \mid \mathcal{F} \text{ then } \text{efac } C \text{ else } C)$ 

```

```

lemma iefac-mempty[simp]:
    fixes  $\mathcal{F} :: 'f \text{ gclause fset}$ 
    shows iefac  $\mathcal{F} \{\#\} = \{\#\}$ 
    by (metis efac-mempty iefac-def)

```

```

lemma fset-mset-iefac[simp]:
    fixes  $\mathcal{F} :: 'f \text{ gclause fset}$  and  $C :: 'f \text{ gclause}$ 
    shows fset-mset (iefac  $\mathcal{F} C$ ) = fset-mset  $C$ 
proof (cases  $C \mid \in \mid \mathcal{F}$ )
    case True
    hence iefac  $\mathcal{F} C = \text{efac } C$ 
    unfolding iefac-def by simp

```

```

thus ?thesis
  by auto
next
  case False
  hence  $\text{iefac } \mathcal{F} \ C = C$ 
    unfolding  $\text{iefac-def}$  by  $\text{simp}$ 
  thus ?thesis by  $\text{simp}$ 
qed

lemma  $\text{atms-of-cls-iefac[simp]}$ :
  fixes  $\mathcal{F} :: 'f \text{ gclause fset}$  and  $C :: 'f \text{ gclause}$ 
  shows  $\text{atms-of-cls } (\text{iefac } \mathcal{F} \ C) = \text{atms-of-cls } C$ 
  by ( $\text{simp add: atms-of-cls-def}$ )

lemma  $\text{iefac-le}$ :
  fixes  $\mathcal{F} :: 'f \text{ gclause fset}$  and  $C :: 'f \text{ gclause}$ 
  shows  $\text{iefac } \mathcal{F} \ C \preceq_c C$ 
  by ( $\text{metis efac-subset iefac-def reflclp-iff subset-implies-reflclp-multp}$ )

lemma  $\text{true-cls-iefac-iff[simp]}$ :
  fixes  $\mathcal{I} :: 'f \text{ gterm set}$  and  $\mathcal{F} :: 'f \text{ gclause fset}$  and  $C :: 'f \text{ gclause}$ 
  shows  $\mathcal{I} \models \text{iefac } \mathcal{F} \ C \longleftrightarrow \mathcal{I} \models C$ 
  by ( $\text{metis iefac-def true-cls-efac-iff}$ )

lemma  $\text{funion-funion-eq-funion-funion-fimage-iefac-if}$ :
  assumes  $U_{ef}\text{-eq: } U_{ef} = \text{iefac } \mathcal{F} \ |\cdot| \ \{C \mid C \in N \ \cup \ U_{er}. \text{iefac } \mathcal{F} \ C \neq C\}$ 
  shows  $N \ \cup \ U_{er} \ \cup \ U_{ef} = N \ \cup \ U_{er} \ \cup \ (\text{iefac } \mathcal{F} \ |\cdot| \ (N \ \cup \ U_{er}))$ 
proof ( $\text{intro fsubset-antisym fsubsetI}$ )
  fix  $C :: 'f \text{ gterm clause}$ 
  assume  $C \in N \ \cup \ U_{er} \ \cup \ U_{ef}$ 
  hence  $C \in N \ \cup \ U_{er} \vee C \in U_{ef}$ 
    by  $\text{simp}$ 
  thus  $C \in N \ \cup \ U_{er} \ \cup \ (\text{iefac } \mathcal{F} \ |\cdot| \ (N \ \cup \ U_{er}))$ 
proof ( $\text{elim disjE}$ )
  assume  $C \in N \ \cup \ U_{er}$ 
  thus ?thesis
    by  $\text{simp}$ 
next
  assume  $C \in U_{ef}$ 
  hence  $C \in \text{iefac } \mathcal{F} \ |\cdot| \ \{C \mid C \in N \ \cup \ U_{er}. \text{iefac } \mathcal{F} \ C \neq C\}$ 
    using  $U_{ef}\text{-eq}$  by  $\text{argo}$ 
  then obtain  $C_0 :: 'f \text{ gterm clause}$  where
     $C_0 \in N \ \cup \ U_{er}$  and  $\text{iefac } \mathcal{F} \ C_0 \neq C_0$  and  $C = \text{iefac } \mathcal{F} \ C_0$ 
    by  $\text{auto}$ 
  thus ?thesis
    by  $\text{simp}$ 
qed
next

```



```

fix C :: 'f gterm clause
assume C |∈| N |∪| Uer |∪| (iefac F |' (N |∪| Uer))
hence C |∈| N |∪| Uer ∨ C |∈| iefac F |' (N |∪| Uer)
  by simp
thus C |∈| N |∪| Uer |∪| Uef
proof (elim disjE)
  assume C |∈| N |∪| Uer
  thus ?thesis
    by simp
next
assume C |∈| iefac F |' (N |∪| Uer)
then obtain C0 :: 'f gterm clause where
  C0 |∈| N |∪| Uer and C = iefac F C0
  by auto

show ?thesis
proof (cases iefac F C0 = C0)
  case True
  hence C = C0
    using ⟨C = iefac F C0⟩ by argo
  thus ?thesis
    using ⟨C0 |∈| N |∪| Uer⟩ by simp
next
  case False
  hence C |∈| Uef
    unfolding Uef-eq
    using ⟨C0 |∈| N |∪| Uer⟩ ⟨C = iefac F C0⟩ by simp
  thus ?thesis
    by simp
qed
qed
qed

```

lemma *clauses-for-iefac-are-unproductive:*
 $\forall C |∈| N \dashv| iefac \mathcal{F} |' N. \forall U. ord-res.production \ U \ C = \{\}$

```

proof (intro ballI allI)
fix C U
assume C |∈| N \dashv| 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:*

$\forall C \mid \in \mid N \mid - \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N. \exists D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N. D \prec_c C \wedge \{D\} \models_e \{C\}$

proof (*intro ballI allI*)

fix C

assume $C \mid \in \mid N \mid - \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N$

hence $C \mid \in \mid N$ **and** $C \not\mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N$

by *simp-all*

hence $\text{iefac } \mathcal{F} C \neq C$

by (*metis fimage-iff*)

hence $\text{efac } C \neq C$

by (*metis iefac-def*)

show $\exists D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N. D \prec_c C \wedge \{D\} \models_e \{C\}$

proof (*intro ballI conjI*)

show $\text{efac } C \prec_c C$ **and** $\{\text{efac } C\} \models_e \{C\}$

using *efac-properties-if-not-ident*[*OF* $\langle \text{efac } C \neq C \rangle$] **by** *simp-all*

next

show $\text{efac } C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N$

using $\langle C \mid \in \mid N \rangle \langle \text{iefac } \mathcal{F} C \neq C \rangle$ **by** (*metis fimageI iefac-def*)

qed

qed

lemma *is-least-false-clause-with-iefac-conv:*

is-least-false-clause $(N \mid \cup \mid U_{er} \mid \cup \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})) =$

is-least-false-clause $(\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}))$

using *is-least-false-clause-union-cancel-right-strong*[*OF* *clauses-for-iefac-are-unproductive*
clauses-for-iefac-have-smaller-entailing-clause]

by (*simp add: sup-commute*)

lemma *MAGIC4:*

fixes $N \mathcal{F} \mathcal{F}' U_{er} U_{er}'$

defines

$N1 \equiv \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**

$N2 \equiv \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er}')$

assumes

subsets-agree: $\{x \mid \in \mid N1. x \prec_c C\} = \{x \mid \in \mid N2. x \prec_c C\}$ **and**

is-least-false-clause $N1 D$ **and**

is-least-false-clause $N2 E$ **and**

$C \prec_c D$

shows $C \preceq_c E$

proof –

have $\neg E \prec_c C$

proof (*rule notI*)

assume $E \prec_c C$

have *is-least-false-clause* $N2 E \longleftrightarrow \text{is-least-false-clause } \{x \mid \in \mid N2. x \preceq_c E\}$

E

proof (*rule is-least-false-clause-swap-swap-compatible-fsets*)
show $\{|x \in N2. (\prec_c)^{==} x E|\} = \{|x \in \{|x \in N2. (\prec_c)^{==} x E|\}. (\prec_c)^{==} x E|\}$
using $\langle E \prec_c C \rangle$ **by force**
qed

also have $\dots \longleftrightarrow$ *is-least-false-clause* $\{|x \in N1. x \preceq_c E|\}$ E
proof (*rule is-least-false-clause-swap-swap-compatible-fsets*)
show $\{|x \in \{|x \in N2. (\prec_c)^{==} x E|\}. (\prec_c)^{==} x E|\} = \{|x \in \{|x \in N1. (\prec_c)^{==} x E|\}. (\prec_c)^{==} x E|\}$
using *subsets-agree* $\langle E \prec_c C \rangle$ **by fastforce**
qed

also have $\dots \longleftrightarrow$ *is-least-false-clause* $N1 E$
proof (*rule is-least-false-clause-swap-swap-compatible-fsets*)
show $\{|x \in \{|x \in N1. (\prec_c)^{==} x E|\}. (\prec_c)^{==} x E|\} = \{|x \in N1. (\prec_c)^{==} x E|\}$
using $\langle E \prec_c C \rangle$ **by fastforce**
qed

finally have *is-least-false-clause* $N1 E$
using \langle *is-least-false-clause* $N2 E \rangle$ **by argo**

hence $D = E$
using \langle *is-least-false-clause* $N1 D \rangle$
by (*metis Uniq-is-least-false-clause the1-equality'*)

thus *False*
using $\langle E \prec_c C \rangle \langle C \prec_c D \rangle$ **by order**
qed
thus $C \preceq_c E$
by order
qed

lemma *atms-of-clss-fimage-iefac[simp]*:
atms-of-clss (*iefac* $\mathcal{F} \mid\! \uparrow N$) = *atms-of-clss* N

proof –
have *atms-of-clss* (*iefac* $\mathcal{F} \mid\! \uparrow N$) = *ffUnion* (*atms-of-clss* $\mid\! \uparrow$ *iefac* $\mathcal{F} \mid\! \uparrow N$)
unfolding *atms-of-clss-def* ..

also have $\dots =$ *ffUnion* ((*atms-of-clss* \circ *iefac* \mathcal{F}) $\mid\! \uparrow N$)
by *simp*

also have $\dots =$ *ffUnion* (*atms-of-clss* $\mid\! \uparrow N$)
unfolding *comp-def atms-of-clss-iefac* ..

also have $\dots =$ *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*: $C \in\text{iefac } \mathcal{F} \mid\uparrow N$
 shows *atm-of* $L \in\text{atms-of-clss } N$

proof –
 have *atm-of* $L \in\text{atms-of-clss } C$
 unfolding *atms-of-clss-def*
 using *L-in* by *simp*

hence *atm-of* $L \in\text{atms-of-clss } (\text{iefac } \mathcal{F} \mid\uparrow N)$
 unfolding *atms-of-clss-def*
 using *C-in* by (*metis fmember-ffUnion-iff*)

thus *atm-of* $L \in\text{atms-of-clss } N$
 by *simp*
 qed

lemma *clause-almost-almost-definedI*:

fixes $\Gamma D K$

assumes

D-in: $D \in\text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er})$ and

D-max-lit: *ord-res.is-maximal-lit* $K D$ and

no-undef-atm: $\neg (\exists A \in\text{atms-of-clss } (N \mid\cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin$

trail-atms $\Gamma)$

shows *trail-defined-clss* $\Gamma \{\#L \in\# D. L \neq K \wedge L \neq -K\}$

unfolding *trail-defined-clss-def*

proof (*intro ballI*)

have $K \in\# D$ and *lit-in-D-le-K*: $\bigwedge L. L \in\# D \implies L \preceq_l K$

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 $L :: \text{'f gterm literal}$

assume $L \in\# \{\#L \in\# D. L \neq K \wedge L \neq -K\}$

hence $L \in\# D$ and $L \neq K$ and $L \neq -K$

by *simp-all*

have *atm-of* $L \in\text{atms-of-clss } (N \mid\cup U_{er})$

using $\langle L \in\# D \rangle$ *D-in* by (*metis atm-of-in-atms-of-clssI*)

hence *atm-of* $L \prec_t \text{atm-of } K \implies \text{atm-of } L \in\text{trail-atms } \Gamma$

using *no-undef-atm* by *metis*

moreover have *atm-of* $L \prec_t \text{atm-of } K$

using *lit-in-D-le-K* [*OF* $\langle L \in\# D \rangle$] $\langle L \neq K \rangle$ $\langle L \neq -K \rangle$

by (*cases L*; *cases K*) *simp-all*

ultimately show *trail-defined-lit* ΓL
using *trail-defined-lit-iff-trail-defined-atm* **by** *auto*
qed

lemma *clause-almost-definedI*:

fixes $\Gamma D K$

assumes

D-in: $D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**

D-max-lit: *ord-res.is-maximal-lit* $K D$ **and**

no-undef-atm: $\neg (\exists A \in \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \text{trail-atms } \Gamma)$ **and**

K-defined: *trail-defined-lit* ΓK

shows *trail-defined-cls* $\Gamma \{\#Ka \in \# D. Ka \neq K\#\}$

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

$\mathcal{F} :: \text{'f gclause fset}$ **and**

$\Gamma :: \text{'f gliteral} \times \text{'f gclause option}$) *list*

assumes

Γ -*consistent*: *trail-consistent* Γ **and**

clauses-lt-E-true: $\forall C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow \text{trail-true-cls } \Gamma C$ **and**

eres $D E \prec_c E$ **and**

trail-false-cls Γ (*eres* $D E$)

shows *eres* $D E \notin N \mid \cup \mid U_{er}$

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 $\langle \text{eres } D E \prec_c E \rangle$ **by** *order*

moreover assume *eres* $D E \in N \mid \cup \mid U_{er}$

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* $\langle \text{trail-false-cls } \Gamma$ (*eres* $D E$) \rangle

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: $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**

D-false: *trail-false-cls* Γ *D* **and**

D-max-lit: *linorder-lit.is-maximal-in-mset* *D* *L*

shows $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \preceq_t \text{atm-of } L \wedge A \mid \notin \mid \text{trail-atms } \Gamma)$

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* $\mid \in \mid \text{trail-atms } \Gamma$

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: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**

C-max-lit: *linorder-lit.is-maximal-in-mset* *C* *K* **and**

no-undef-atom-le-K:

$\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \preceq_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma)$

shows *trail-defined-cls* Γ *C*

proof –

have $\bigwedge x. x \in \# C \implies \text{atm-of } x \preceq_t \text{atm-of } K$

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 $\bigwedge x. x \in \# C \implies \text{trail-defined-lit } \Gamma$ *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* ($N \mid \cup \mid U_{er}$))

and

D-in: $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**

D-false: *trail-false-cls* Γ *D* **and**

D-max-lit: *linorder-lit.is-maximal-in-mset* *D L* **and**

C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**

C-max-lit: *linorder-lit.is-maximal-in-mset* *C K* **and**

C-lt: $C \prec_c D$

shows $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \preceq_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma)$

proof –

have $K \preceq_t L$

using *C-lt C-max-lit D-max-lit*

using *linorder-cls.less-imp-not-less linorder-lit.multip-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 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \preceq_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma)$

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 $\mathcal{F} N \Gamma$

shows $fBex (\text{iefac } \mathcal{F} \mid \uparrow \mid N) (\text{trail-false-cls } \Gamma) \longleftrightarrow fBex N (\text{trail-false-cls } \Gamma)$

proof (*rule iffI* ; *elim bexE*)

fix $C :: 'f$ *gclause*

assume *C-in*: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N$ **and** *C-false*: *trail-false-cls* Γ C

thus $fBex N (\text{trail-false-cls } \Gamma)$

by (*smt* (*verit*, *ccfv-SIG*) *fimage-iff iefac-def set-mset-efac trail-false-cls-def*)

next

fix $C :: 'f$ *gclause*

assume $C \mid \in \mid N$ **and** *trail-false-cls* Γ C

thus $fBex (\text{iefac } \mathcal{F} \mid \uparrow \mid N) (\text{trail-false-cls } \Gamma)$

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

begin

19 ORD-RES-4 (implicit factorization)

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

inductive *ord-res-4* **where**

factoring:

$NN = \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \implies$
 $\text{is-least-false-clause } NN \ C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C \ L \implies$
 $\text{is-pos } L \implies$
 $\mathcal{F}' = \text{finsert } C \ \mathcal{F} \implies$
 $\text{ord-res-4 } N \ (U_{er}, \mathcal{F}) \ (U_{er}, \mathcal{F}') \mid$

resolution:

$NN = \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \implies$
 $\text{is-least-false-clause } NN \ C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C \ L \implies$
 $\text{is-neg } L \implies$
 $D \mid \in \mid NN \implies$
 $D \prec_c C \implies$
 $\text{ord-res.production (fset } NN) \ D = \{\text{atm-of } L\} \implies$
 $U_{er}' = \text{finsert } (eres \ D \ C) \ U_{er} \implies$
 $\text{ord-res-4 } N \ (U_{er}, \mathcal{F}) \ (U_{er}', \mathcal{F})$

inductive *ord-res-4-step* **where**

$\text{ord-res-4 } N \ s \ s' \implies \text{ord-res-4-step } (N, s) \ (N, s')$

inductive *ord-res-4-final* **where**

$\text{ord-res-final } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \implies \text{ord-res-4-final } (N, U_{er}, \mathcal{F})$

sublocale *ord-res-4-semantics: semantics* **where**

$\text{step} = \text{ord-res-4-step}$ **and**

$\text{final} = \text{ord-res-4-final}$

proof *unfold-locales*

fix $S_4 :: 'f \text{ gterm clause fset} \times 'f \text{ gterm clause fset} \times 'f \text{ gterm clause fset}$

obtain $N \ U_{er} \ \mathcal{F} :: 'f \text{ gterm clause fset}$ **where**

$S_4\text{-def: } S_4 = (N, (U_{er}, \mathcal{F}))$

by (*metis prod.exhaust*)

assume *ord-res-4-final* S_4

hence $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \vee \neg \text{ex-false-clause } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))$

by (*simp add: S4-def ord-res-4-final.simps ord-res-final-def*)

thus *finished* *ord-res-4-step* S_4

proof (*elim disjE*)


```

assume {#} |∈| iefac  $\mathcal{F}$  |q (N |∪|  $U_{er}$ )
hence is-least-false-clause (iefac  $\mathcal{F}$  |q (N |∪|  $U_{er}$ )) {#}
using is-least-false-clause-mempty by metis
hence  $\nexists$  C L. is-least-false-clause (iefac  $\mathcal{F}$  |q (N |∪|  $U_{er}$ )) C ∧ 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-4 N ( $U_{er}$ ,  $\mathcal{F}$ ) 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  $\mathcal{F}$  |q (N |∪|  $U_{er}$ )))
hence  $\nexists$  C. is-least-false-clause (iefac  $\mathcal{F}$  |q (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  $\nexists$  x. ord-res-4 N ( $U_{er}$ ,  $\mathcal{F}$ ) 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  $\mathcal{F}$ 1  $U_{er}$ 1 C1 L1  $\mathcal{F}$ 1')
show ?thesis
using step2
proof (cases N s s'' rule: ord-res-4.cases)
case (factoring NN2  $\mathcal{F}$ 2  $U_{er}$ 2 C2 L2  $\mathcal{F}$ 2')
with hyps1 show ?thesis
by (metis Uniq-D Uniq-is-least-false-clause prod.inject)
next
case (resolution NN2  $\mathcal{F}$ 2  $U_{er}$ 2 C2 L2 D2  $U_{er}$ 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 ..
qed
next
case hyps1: (resolution NN1  $\mathcal{F}$ 1  $U_{er}$ 1 C1 L1 D1  $U_{er}$ 1')
show ?thesis
using step2

```

```

proof (cases  $N s s''$  rule: ord-res-4.cases)
  case (factoring  $NN \mathcal{F} U_{er} C L \mathcal{F}'$ )
  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  $NN \mathcal{F} U_{er} C L D U_{er}'$ )
  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 insertII 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  $x y \implies$  ord-res-4-step  $x z \implies y = z$ 
  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  $\neg$  ord-res-4-final  $S$ 
  shows  $\exists S'. \text{ord-res-4-step } S S'$ 
proof –

```

```

  from assms obtain  $N U_{er} \mathcal{F}$  where
     $S\text{-def}: S = (N, (U_{er}, \mathcal{F}))$  and
     $\{\#\} \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$  and
     $\text{ex-false-clause } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$ 
  by (metis ord-res-4-final.intros ord-res-final-def surj-pair)

```

```

obtain  $C$  where  $C\text{-least-false}: \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) C$ 
  using  $\langle \text{ex-false-clause } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) \rangle$ 
   $\text{obtains-least-false-clause-if-ex-false-clause}$ 
  by metis

```

```

hence  $C \neq \{\#\}$ 
  using  $\langle \{\#\} \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$ 
  unfolding is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff
  by metis

```

```

then obtain  $L$  where  $C\text{-max}: \text{linorder-lit.is-maximal-in-mset } C L$ 
  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 F |↑| (N |∪| Uer) and
  D <c C and
  ord-res.is-strictly-maximal-lit (Pos A) D and
  ord-res.production (fset (iefac F |↑| (N |∪| Uer))) 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 ∨ (∃ 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 M*) ⊨ *C* ⇒

C' = *The-optional* (*linorder-cls.is-least-in-fset* {*D* |∈| *iefac F* |↑| (*N* |∪| *U_{er}*).

C <_{*c*} *D* |}) ⇒

ord-res-5 N (*U_{er}*, *F*, *M*, *Some C*) (*U_{er}*, *F*, *M*, *C'*) |

production:

¬ (*dom M*) ⊨ *C* ⇒

$linorder-lit.is-maximal-in-mset\ C\ L \implies$
 $is-pos\ L \implies$
 $linorder-lit.is-greatest-in-mset\ C\ L \implies$
 $\mathcal{M}' = \mathcal{M}(atm-of\ L := Some\ C) \implies$
 $C' = The-optional\ (linorder-cls.is-least-in-fset\ \{|D|\in|\ iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er})\}.$
 $C \prec_c D\}) \implies$
 $ord-res-5\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}, \mathcal{M}', C')\ |$

factoring:

$\neg (dom\ \mathcal{M}) \models C \implies$
 $linorder-lit.is-maximal-in-mset\ C\ L \implies$
 $is-pos\ L \implies$
 $\neg linorder-lit.is-greatest-in-mset\ C\ L \implies$
 $\mathcal{F}' = finsert\ C\ \mathcal{F} \implies$
 $\mathcal{M}' = (\lambda-. None) \implies$
 $C' = The-optional\ (linorder-cls.is-least-in-fset\ (iefac\ \mathcal{F}'\ |\uparrow\ (N\ |\cup|\ U_{er}))) \implies$
 $ord-res-5\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}, \mathcal{F}', \mathcal{M}', C')\ |$

resolution:

$\neg (dom\ \mathcal{M}) \models C \implies$
 $linorder-lit.is-maximal-in-mset\ C\ L \implies$
 $is-neg\ L \implies$
 $\mathcal{M}\ (atm-of\ L) = Some\ D \implies$
 $U_{er}' = finsert\ (eres\ D\ C)\ U_{er} \implies$
 $\mathcal{M}' = (\lambda-. None) \implies$
 $C' = The-optional\ (linorder-cls.is-least-in-fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}')) \implies$
 $ord-res-5\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', C')$

inductive $ord-res-5-step :: 'f\ ord-res-5 \Rightarrow 'f\ ord-res-5 \Rightarrow bool$ **where**
 $ord-res-5\ N\ s\ s' \implies ord-res-5-step\ (N, s)\ (N, s')$

lemma $tranclp-ord-res-5-step-if-tranclp-ord-res-5:$
 $(ord-res-5\ N)^{++}\ s\ s' \implies ord-res-5-step^{++}\ (N, s)\ (N, s')$
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 $ord-res-5-semantic: semantic$ **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
   $N U_{er} \mathcal{F} :: 'f \text{ gterm clause fset and}$ 
   $\mathcal{M} :: 'f \text{ gterm} \Rightarrow 'f \text{ gclause option and}$ 
   $\mathcal{C} :: 'f \text{ gclause option where}$ 
   $S5\text{-def: } S5 = (N, (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}))$ 
  by (metis prod.exhaust)

assume ord-res-5-final S5
hence  $\mathcal{C} = \text{None} \vee \mathcal{C} = \text{Some } \{\#\}$ 
  by (simp add: S5-def ord-res-5-final.simps)
hence  $\nexists x. \text{ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) x$ 
  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 U1er)
      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 U1er)
      show ?thesis
      using step2
  proof (cases N s s'' rule: ord-res-5.cases)
    case (skip M2 C2 C2' F2 U2er)
      with hyps1 show ?thesis
      by simp
  next
    case hyps2: (production M2 C2 L2 M2' C2' F2 U2er)
      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' U2er)
      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 U2er' U2er 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' U1er)
show ?thesis
  using step2
proof (cases N s s'' rule: ord-res-5.cases)
  case (skip M2 C2 C2' F2 U2er)
  with hyps1 show ?thesis
    by simp
next
case hyps2: (production M2 C2 L2 M2' C2' F2 U2er)
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' U2er)
have C1 = C2
  using hyps1 hyps2 by simp
thus ?thesis
  using hyps1 hyps2 by simp
next
case hyps2: (resolution M2 C2 L2 D2 U2er' U2er 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 U1er' U1er M1' C1' F1)
show ?thesis
  using step2
proof (cases N s s'' rule: ord-res-5.cases)
  case (skip M2 C2 C2' F2 U2er)
  with hyps1 show ?thesis
    by simp
next
  case hyps2: (production M2 C2 L2 M2' C2' F2 U2er)
  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' U2er)
  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 arg0
  thus ?thesis ..
next
  case hyps2: (resolution M2 C2 L2 D2 U2er' U2er M2' C2' F2)
  have U1er = U2er F1 = F2 M1 = M2 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  $\langle M1 \text{ (atm-of } L1) = \text{Some } D1 \rangle \langle M2 \text{ (atm-of } L2) = \text{Some } D2 \rangle \langle M1 = M2 \rangle$ 
    by simp

  have U1er' = U2er'
    using  $\langle U1_{er}' = \text{finsert (eres } D1 \ C1) \ U1_{er} \rangle \langle U2_{er}' = \text{finsert (eres } D2 \ C2) \ U2_{er} \rangle$ 
     $\langle D1 = D2 \rangle \langle C1 = C2 \rangle \langle U1_{er} = U2_{er} \rangle$ 
    by arg0

  moreover have M1' = M2'
    using  $\langle M1' = (\lambda-. \text{None}) \rangle \langle M2' = (\lambda-. \text{None}) \rangle$ 
    by arg0

  moreover have C1' = C2'

```

```

using hyps1 hyps2  $\langle \mathcal{F}1 = \mathcal{F}2 \rangle \langle U1_{er}' = U2_{er}' \rangle$  by simp

ultimately show ?thesis
  using  $\langle s' = (U1_{er}', \mathcal{F}1, \mathcal{M}1', \mathcal{C}1') \rangle \langle s'' = (U2_{er}', \mathcal{F}2, \mathcal{M}2', \mathcal{C}2') \rangle$ 
     $\langle \mathcal{F}1 = \mathcal{F}2 \rangle$ 
  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  $\implies$  ord-res-5-step x z  $\implies$  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  $\longleftrightarrow$ 
     $(\forall U_{er} \mathcal{F} \mathcal{M} C. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \longrightarrow C \in |iefac \mathcal{F} | \uparrow (N \cup U_{er}))$ 

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 M C C' F Uer)
  thus ?thesis
    unfolding next-clause-in-factorized-clause-def
    by (metis Pair-inject The-optional-eq-SomeD linorder-clis.is-minimal-in-fset-eq-is-least-in-fset
      linorder-clis.is-minimal-in-fset-ffilter-iff)
next
  case (production M C L M' C' F Uer)
  thus ?thesis
    unfolding next-clause-in-factorized-clause-def
    by (metis Pair-inject The-optional-eq-SomeD linorder-clis.is-minimal-in-fset-eq-is-least-in-fset
      linorder-clis.is-minimal-in-fset-ffilter-iff)
next
  case (factoring M C L F' F M' C' Uer)
  thus ?thesis
    unfolding next-clause-in-factorized-clause-def
    by (metis Pair-inject The-optional-eq-SomeD linorder-clis.is-least-in-fset-iff)
next
  case (resolution M C L D Uer' Uer M' C' F)
  thus ?thesis
    unfolding next-clause-in-factorized-clause-def
    by (metis Pair-inject The-optional-eq-SomeD linorder-clis.is-least-in-fset-iff)
qed

```


definition *implicitly-factorized-clauses-subset* **where**
implicitly-factorized-clauses-subset $N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \mathcal{M} \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow \mathcal{F} \mid\subseteq N \mid\cup U_{er})$

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* $\mathcal{M} \mathcal{C} \mathcal{C}' \mathcal{F} U_{er}$)
thus *?thesis*
using *invars*
by (*simp add*: *implicitly-factorized-clauses-subset-def*)

next
case (*production* $\mathcal{M} \mathcal{C} L \mathcal{M}' \mathcal{C}' \mathcal{F} U_{er}$)
thus *?thesis*
using *invars*
by (*simp add*: *implicitly-factorized-clauses-subset-def*)

next
case (*factoring* $\mathcal{M} \mathcal{C} L \mathcal{F}' \mathcal{F} \mathcal{M}' \mathcal{C}' U_{er}$)
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* $\mathcal{M} \mathcal{C} L D U_{er}' U_{er} \mathcal{M}' \mathcal{C}' \mathcal{F}$)
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: $C \mid\in NN$ **and**
D-least-gt-C: *linorder-cls.is-least-in-fset* (*ffilter* (\prec_c) C) NN) D
shows *ord-res.interp* (*fset* NN) $D = \text{ord-res.interp}$ (*fset* NN) $C \cup \text{ord-res.production}$
(*fset* NN) C

proof –
have *nex-between-C-and-D*: $\neg (\exists CD \mid\in NN. C \prec_c CD \wedge CD \prec_c D)$
using *D-least-gt-C*
unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *auto*

have *ord-res.interp* (*fset* NN) $D = \text{ord-res.interp}$ (*fset* $\{B \mid\in NN. B \preceq_c C\}$)

D

proof (*rule ord-res.interp-swap-clause-set*)
 have $NN = \{|B \in| NN. B \preceq_c C|\} \cup \{|E \in| NN. D \preceq_c E|\}$
 using *nex-between-C-and-D by force*
 show $\{Da. Da \in| NN \wedge Da \prec_c D\} = \{Da. Da \in| \{|B \in| NN. (\prec_c)^{==} B C|\} \wedge Da \prec_c D\}$
using $\langle NN = \{|B \in| NN. (\prec_c)^{==} B C|\} \cup \text{ffilter } ((\prec_c)^{==} D) NN \rangle$
linorder-cls.leD by auto
qed

also have $\dots = \bigcup (\text{ord-res.production } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } \{Da. Da \in| \{|B \in| NN. (\prec_c)^{==} B C|\} \wedge Da \prec_c D\})$
unfolding *ord-res.interp-def ..*

also have $\dots = \bigcup (\text{ord-res.production } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } \{Da \in fset NN. (\prec_c)^{==} Da C \wedge Da \prec_c D\})$
by auto

also have $\dots = \bigcup (\text{ord-res.production } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } \{Da \in fset NN. (\prec_c)^{==} Da C\})$

proof –
 have $\{|Da \in| NN. Da \preceq_c C \wedge Da \prec_c D\} = \{|Da \in| NN. Da \preceq_c C\}$
 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 $\dots = \bigcup (\text{ord-res.production } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } \{Da \in fset NN. Da \prec_c C\}) \cup \text{ord-res.production } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } C$

proof –
 have $\{Da. Da \in| NN \wedge (\prec_c)^{==} Da C\} = \text{insert } C \{Da. Da \in| NN \wedge Da \prec_c C\}$

using *C-in by auto*

thus *?thesis*

by blast

qed

also have $\dots = \text{ord-res-Interp } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } C$
unfolding *ord-res.interp-def by auto*

also have $\dots = \text{ord-res-Interp } (fset NN) \text{ ‘ } C$

proof –

have $\text{ord-res.interp } (fset \{|B \in| NN. (\prec_c)^{==} B C|\}) \text{ ‘ } C = \text{ord-res.interp } (fset NN) \text{ ‘ } C$

using *ord-res.interp-swap-clause-set[of fset \{|B \in| NN. (\prec_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-clc.is-least-in-set N C

shows ord-res.interp N C = {}

using assms

unfolding ord-res.interp-def

unfolding linorder-clc.is-least-in-set-iff

by auto

definition model-eq-interp-upto-next-clause where

model-eq-interp-upto-next-clause N s \longleftrightarrow

($\forall U_{er} \mathcal{F} \mathcal{M} C. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \longrightarrow$

dom $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{ } U_{er})) C)$)

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 $\mathcal{M} C C' \mathcal{F} U_{er}$)

have dom $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{ } U_{er})) D$ if $C' = \text{Some } D$ for D

proof –

have dom $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{ } U_{er})) C$

using invars(1)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
OF $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle$].

also have ... = ord-res-Interp (fset (iefac \mathcal{F} | \uparrow (N | \cup U_{er}))) C

proof –

have ord-res.production (fset (iefac \mathcal{F} | \uparrow (N | \cup U_{er}))) C = {}

using $\langle \text{dom } \mathcal{M} \models C \rangle$

unfolding $invars(1)[unfolding\ model\ eq\ interp\ upto\ next\ clause\ def, rule\ format,$
 $OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C) \rangle]$
by (*simp add: ord-res.production-unfold*)
thus *?thesis*
by *simp*
qed

also have $\dots = ord\ res.\ interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ D$
proof (*rule interp-eq-Interp-if-least-greater[symmetric]*)
show $C \in |iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})$
using $invars(2)[unfolding\ next\ clause\ in\ factorized\ clause\ def, rule\ format,$
 $OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C) \rangle]$.
next
show $linorder\ cls.\ is\ least\ in\ fset\ (ffilter\ ((\prec_c)\ C)\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))$
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 $dom\ M' = ord\ res.\ interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ D$ **if** $C' = Some\ D$
for *D*
proof –
have $dom\ M' = dom\ M \cup \{atm\ of\ L\}$
unfolding $\langle M' = M(atm\ of\ L \mapsto C) \rangle$ **by** *simp*

also have $\dots = ord\ res.\ interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ C \cup \{atm\ of\ L\}$
unfolding $invars(1)[unfolding\ model\ eq\ interp\ upto\ next\ clause\ def, rule\ format,$
 $OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C) \rangle]$..

also have $\dots = ord\ res.\ interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ C \cup$
 $ord\ res.\ production\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ C$
proof –
have $\neg\ ord\ res.\ interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ C \models C$
using $\langle \neg\ dom\ M \models C \rangle$
unfolding $invars(1)[unfolding\ model\ eq\ interp\ upto\ next\ clause\ def, rule\ format,$
 $OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C) \rangle]$.
hence $atm\ of\ L \in ord\ res.\ production\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ C$
using $\langle is\ pos\ L \rangle \langle ord\ res.\ is\ strictly\ maximal\ lit\ L\ C \rangle$
using $invars(2)[unfolding\ next\ clause\ in\ factorized\ clause\ def, rule\ format,$
 $OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C) \rangle]$
unfolding *ord-res.production-unfold mem-Collect-eq*
by (*metis linorder-lit.is-greatest-in-mset-iff literal.collapse(1) multi-member-split*)

hence $\text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C = \{ \text{atm-of } L \}$
by $(\text{metis empty-iff insertE ord-res.production-eq-empty-or-singleton})$
thus $?thesis$
by $argo$
qed

also have $\dots = \text{ord-res.interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) D$
proof $(\text{rule interp-eq-Interp-if-least-greater[symmetric]})$
show $C \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using $\text{invars(2)[unfolded next-clause-in-factorized-clause-def, rule-format, OF } \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle]$.
next
show $\text{linorder-clc.is-least-in-fset (ffilter ((\prec_c) C) (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))}$
 D
using $\text{step-hyps(}\beta\text{-) that by (metis Some-eq-The-optionalD)}$
qed

finally show $?thesis$.
qed

thus $?thesis$
using $\text{step-hyps by (simp add: model-eq-interp-upto-next-clause-def)}$
next
case $\text{step-hyps: (factoring } \mathcal{M} C L \mathcal{F}' \mathcal{F} \mathcal{M}' C' U_{er})$

have $\text{dom } \mathcal{M}' = \text{ord-res.interp (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) D$ **if** $C' = \text{Some } D$ **for** D
proof –
have $\text{dom } \mathcal{M}' = \{ \}$
using $\text{step-hyps(}\beta\text{-) by simp}$
also have $\dots = \text{ord-res.interp (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) D$
proof $(\text{rule interp-eq-empty-if-least-in-set[symmetric]})$
show $\text{linorder-clc.is-least-in-set (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) D$
using $\text{step-hyps(}\beta\text{-) that}$
by $(\text{metis Some-eq-The-optionalD linorder-clc.is-least-in-fset-iff linorder-clc.is-least-in-set-iff})$
qed
finally show $?thesis$.
qed

thus $?thesis$
using $\text{step-hyps by (simp add: model-eq-interp-upto-next-clause-def)}$
next
case $\text{step-hyps: (resolution } \mathcal{M} C L D U_{er}' U_{er} \mathcal{M}' C' \mathcal{F})$

have $\text{dom } \mathcal{M}' = \text{ord-res.interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) D$ **if** $C' = \text{Some } D$ **for** D
proof –
have $\text{dom } \mathcal{M}' = \{ \}$

```

    using step-hyps(3-) by simp
  also have ... = ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ')) D
  proof (rule interp-eq-empty-if-least-in-set[symmetric])
    show linorder-cls.is-least-in-set (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ')) 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  $N s \longleftrightarrow$ 
  ( $\forall U_{er} \mathcal{F} \mathcal{M} \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow$ 
  ( $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$ 
  ord-res-Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )))  $C \models C$ ))

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  $\mathcal{M} D C' \mathcal{F} U_{er}$ )

  have D-true: ord-res-Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )))  $D \models D$ 
  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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )))  $C \models C$ 
  if  $C' = \text{Some } E$  and C-in:  $C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})$  and  $C \prec_c E$  for  $C E$ 
proof -
  have linorder-cls.is-least-in-fset (ffilter (( $\prec_c$ )  $D$ ) (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )))  $E$ 
  using step-hyps  $\langle C' = \text{Some } E \rangle$  by (metis Some-eq-The-optionalD)
  hence  $C \preceq_c D$ 
  using C-in  $\langle C \prec_c E \rangle$ 
  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,

```

$OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle C\text{-in}$ **by auto**
qed

moreover have $ord\text{-res}\text{-Interp} (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \models C$
if $C' = \text{None}$ **and** $C\text{-in}: C \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **for** C
proof –
have $\nexists x. \text{linorder}\text{-cls}.\text{is}\text{-least}\text{-in}\text{-fset} (ffilter ((\prec_c) D) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))$
 x
using $step\text{-hyps} \langle C' = \text{None} \rangle$
by ($metis (no\text{-types}, \text{opaque}\text{-lifting}) \text{None}\text{-eq}\text{-The}\text{-optional}D \text{linorder}\text{-cls}.\text{Uniq}\text{-is}\text{-least}\text{-in}\text{-fset} \text{the1}\text{-equality}'$)
hence $\neg (\exists x \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). D \prec_c x)$
by ($metis \text{empty}E \text{ffmember}\text{-filter} \text{linorder}\text{-cls}.\text{ex1}\text{-least}\text{-in}\text{-fset}$)
hence $C \prec_c D \vee C = D$
using $C\text{-in}$ **by force**
thus $?thesis$
proof ($\text{elim} \text{disj}E$)
assume $C \prec_c D$
then show $?thesis$
using $\text{invars}(1) \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle C\text{-in}$
by ($auto \text{simp}: \text{all}\text{-smaller}\text{-clauses}\text{-true}\text{-wrt}\text{-respective}\text{-Interp}\text{-def}$)
next
assume $C = D$
then show $?thesis$
using $D\text{-true}$ **by argo**
qed
qed

ultimately show $?thesis$
using $step\text{-hyps}$
by ($\text{smt} (\text{verit}, \text{best}) \text{all}\text{-smaller}\text{-clauses}\text{-true}\text{-wrt}\text{-respective}\text{-Interp}\text{-def} \text{old}.\text{prod}.\text{inject} \text{option}.\text{exhaust}$)
next
case $step\text{-hyps}: (\text{production } \mathcal{M} D K \mathcal{M}' C' \mathcal{F} U_{er})$

have $K \in \# D$
using $step\text{-hyps}(3-)$ **by** ($metis \text{linorder}\text{-lit}.\text{is}\text{-greatest}\text{-in}\text{-mset}\text{-iff}$)

have $\neg ord\text{-res}.\text{interp} (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D \models D$
using $\langle \neg \text{dom } \mathcal{M} \models D \rangle$
unfolding $\text{invars}(2)[\text{unfolded} \text{model}\text{-eq}\text{-interp}\text{-upto}\text{-next}\text{-clause}\text{-def}, \text{rule}\text{-format}, OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle]$.
hence $\text{atm}\text{-of } K \in ord\text{-res}.\text{production} (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D$
using $\langle \text{is}\text{-pos } K \rangle \langle ord\text{-res}.\text{is}\text{-strictly}\text{-maximal}\text{-lit } K D \rangle \langle K \in \# D \rangle$
using $\text{invars}(3)[\text{unfolded} \text{next}\text{-clause}\text{-in}\text{-factorized}\text{-clause}\text{-def}, \text{rule}\text{-format}, OF \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle]$
unfolding $ord\text{-res}.\text{production}\text{-unfold} \text{mem}\text{-Collect}\text{-eq}$
by ($metis \text{literal}.\text{collapse}(1) \text{multi}\text{-member}\text{-split}$)
hence $\text{prod}\text{-}D\text{-eq}: ord\text{-res}.\text{production} (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D = \{\text{atm}\text{-of}$

$K\}$
by (*metis empty-iff insertE ord-res.production-eq-empty-or-singleton*)
hence *ord-res-Interp* (*fset* (*iefac* \mathcal{F} | \uparrow ($N \cup U_{er}$))) $D \models K$
using $\langle is-pos\ K \rangle$ **by force**
hence *D-true*: *ord-res-Interp* (*fset* (*iefac* \mathcal{F} | \uparrow ($N \cup U_{er}$))) $D \models D$
using $\langle K \in\# D \rangle$ **by auto**

have $dom\ \mathcal{M}' = dom\ \mathcal{M} \cup \{atm-of\ K\}$
unfolding $\langle \mathcal{M}' = \mathcal{M}(atm-of\ K \mapsto D) \rangle$ **by simp**

also have $\dots = ord-res.interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})))\ D \cup \{atm-of\ K\}$
unfolding *invars*(2)[*unfolded model-eq-interp-upto-next-clause-def, rule-format*,
 $OF\ \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D) \rangle$] ..

also have $\dots = ord-res.interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})))\ D \cup$
 $ord-res.production\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})))\ D$
using *prod-D-eq* **by argo**

finally have *dom-M'-eq*: $dom\ \mathcal{M}' = ord-res-Interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})))\ D$.

have *ord-res-Interp* (*fset* (*iefac* \mathcal{F} | \uparrow ($N \cup U_{er}$))) $C \models C$
if $C' = Some\ E$ **and** *C-in*: $C \in\ |iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})$ **and** $C \prec_c E$ **for** $C\ E$
proof –
have *linorder-cls.is-least-in-fset* (*ffilter* ($(\prec_c)\ D$) (*iefac* \mathcal{F} | \uparrow ($N \cup U_{er}$))) E
using *step-hyps* $\langle C' = Some\ E \rangle$ **by** (*metis Some-eq-The-optionalD*)
hence $C \preceq_c D$
using *C-in* $\langle C \prec_c E \rangle$
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*,
 $OF\ \langle s = (U_{er}, \mathcal{F}, \mathcal{M}, Some\ D) \rangle\ C-in$] **by auto**

qed

moreover have *ord-res-Interp* (*fset* (*iefac* \mathcal{F} | \uparrow ($N \cup U_{er}$))) $C \models C$
if $C' = None$ **and** *C-in*: $C \in\ |iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})$ **for** C
proof –
have $\nexists x. linorder-cls.is-least-in-fset\ (ffilter\ ((\prec_c)\ D)\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er})))$
 x
using *step-hyps* $\langle C' = None \rangle$
by (*metis (no-types, opaque-lifting) None-eq-The-optionalD linorder-cls.Uniq-is-least-in-fset the1-equality'*)
hence $\neg (\exists x \in\ |iefac\ \mathcal{F}\ |\uparrow\ (N\ \cup\ U_{er}). D \prec_c x)$
by (*metis femptyE fmember-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)  $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle$  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  $\mathcal{M} D K \mathcal{F}' \mathcal{F} \mathcal{M}' C' U_{er}$ )
have  $D \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
  using invars(2-)  $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \rangle$ 
  by (metis next-clause-in-factorized-clause-def)
hence  $D \in | N \mid \cup \mid U_{er}$ 
  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  $\text{iefac } \mathcal{F}' D \in | \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ 
  unfolding  $\langle \mathcal{F}' = \text{finsert } D \mathcal{F} \rangle$  by simp
hence  $C' \neq \text{None}$ 
  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
   $C' = \text{Some } E$ 
  by auto

have  $\neg (\exists C \in | \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c E)$ 
  using  $\langle C' = \text{Some } E \rangle$  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  $\langle C' = \text{Some } E \rangle$  by simp
next
case step-hyps: (resolution  $\mathcal{M} C L D U_{er}' U_{er} \mathcal{M}' C' \mathcal{F}$ )
hence  $\text{eres } D C \in | N \mid \cup \mid U_{er}'$ 
  by simp
hence  $\text{iefac } \mathcal{F} (\text{eres } D C) \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ 
  by simp

```

hence $C' \neq \text{None}$
using $\text{step-hyps}(3-)$
by (*metis The-optional-eq-NoneD finsert-not-fempty linorder-cls.ex1-least-in-fset set-finsert*)
then obtain E **where**
 $C' = \text{Some } E$
by *auto*

have $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})). C \prec_c E$
using $\langle C' = \text{Some } E \rangle \text{step-hyps}(9)$
by (*metis The-optional-eq-SomeD linorder-cls.is-least-in-fset-iff linorder-cls.less-imp-not-less*)

thus *?thesis*
unfolding $\text{step-hyps}(1,2)$ *all-smaller-clauses-true-wrt-respective-Interp-def*
using $\langle C' = \text{Some } E \rangle$ **by** *simp*
qed

lemma *all-smaller-clauses-true-wrt-model:*

assumes
invars:
 $\text{all-smaller-clauses-true-wrt-respective-Interp } N \ s$
 $\text{model-eq-interp-upto-next-clause } N \ s$
shows $\forall U_{er} \ \mathcal{F} \ \mathcal{M} \ D. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{dom } \mathcal{M} \models C)$
proof (*intro allI impI ballI*)
fix $U_{er} \ \mathcal{F} \ \mathcal{M} \ D \ C$
assume
 $s\text{-def}: s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D)$ **and**
 $C\text{-in}: C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $C\text{-lt}: C \prec_c D$

hence $C\text{-true}: \text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C \models C$
using $\text{invars}(1)$ [*unfolded all-smaller-clauses-true-wrt-respective-Interp-def s-def*]
by *auto*

moreover have $\text{dom } \mathcal{M} = \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ D$
using $\text{invars}(2)$ $s\text{-def}$ **by** (*metis model-eq-interp-upto-next-clause-def*)

ultimately show $\text{dom } \mathcal{M} \models C$
using $\text{ord-res.entailed-clause-stays-entailed}' \ C\text{-lt}$ **by** *metis*
qed

definition *model-eq-sublocale* **where**

$\text{model-eq-sublocale } N \ s \longleftrightarrow$
 $(\forall U_{er} \ \mathcal{F} \ \mathcal{M}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{None}) \longrightarrow$
 $(\text{let } NN = \text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \text{ in } \text{dom } \mathcal{M} = \bigcup (\text{ord-res.production } NN \ ' \ NN)))$

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 $\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow$

$(\forall C\ |\in|\ \text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}). (\forall D. \mathcal{C} = \text{Some}\ D \longrightarrow C \prec_c D) \longrightarrow \text{dom}\ \mathcal{M} \models C)$

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: $C\ |\in|\ \text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})$ **and**

C-lt: $\forall D. \mathcal{C} = \text{Some}\ D \longrightarrow C \prec_c D$

hence *C-true*: *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})$)) $C \models C$

using *invars*(1) **by** (*metis all-smaller-clauses-true-wrt-respective-Interp-def*)

show $\text{dom}\ \mathcal{M} \models C$

proof (*cases* \mathcal{C})

case *C-def*: *None*

have *let* $NN = \text{fset}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}))$ *in* $\text{dom}\ \mathcal{M} = \bigcup (\text{ord-res.production}\ NN\ \text{'}\ NN)$

using *invars*(3) *s-def* *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 *C-def*: (*Some* D)

have $C \prec_c D$

using *C-lt* *C-def* **by** *metis*

moreover have $\text{dom}\ \mathcal{M} = \text{ord-res.interp}\ (\text{fset}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ D$

using *invars*(2) *s-def* *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 $\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some}\ C) \longrightarrow$

$(\forall D. \text{is-least-false-clause}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}))\ D \longrightarrow C \preceq_c D)$

```

proof (intro allI impI ballI)
  fix  $U_{er} \mathcal{F} \mathcal{M} C D$ 
  assume
     $s$ -def:  $s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C)$  and
     $D$ -least-false: is-least-false-clause (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ )  $D$ 
  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  $N s \longleftrightarrow$ 
  ( $\forall U_{er} \mathcal{F} \mathcal{M} C. s = (U_{er}, \mathcal{F}, \mathcal{M}, C) \longrightarrow (\forall A C. \mathcal{M} A = \text{Some } C \longrightarrow$ 
     $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C)$ )

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  $\mathcal{M} C C' \mathcal{F} U_{er}$ )
  thus ?thesis
    using invars(1) by (simp add: atoms-in-model-were-produced-def)
next
  case (production  $\mathcal{M} C L \mathcal{M}' C' \mathcal{F} U_{er}$ )
  obtain  $A$  where  $L = \text{Pos } A$ 
    using  $\langle \text{is-pos } L \rangle$  by (cases  $L$ ) simp-all

  have atm-of  $L \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C$ 
  unfolding ord-res.production-unfold mem-Collect-eq
proof (intro exI conjI)
  show atm-of  $L = A$ 
    unfolding  $\langle L = \text{Pos } A \rangle$  literal.sel ..
next
  show  $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
    using invars(3)[unfolded next-clause-in-factorized-clause-def, rule-format,
      OF  $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle$ ].
next
  have  $L \in \# C$ 
    using  $\langle \text{linorder-lit.is-maximal-in-mset } C L \rangle$ 
    unfolding linorder-lit.is-maximal-in-mset-iff ..

```

```

thus  $C = \text{add-mset } (\text{Pos } A) (C - \{\#\text{Pos } A\# \})$ 
  unfolding  $\langle L = \text{Pos } A \rangle$  by simp
next
  show ord-res.is-strictly-maximal-lit  $(\text{Pos } A) C$ 
  using  $\langle \text{ord-res.is-strictly-maximal-lit } L C \rangle$ 
  unfolding  $\langle L = \text{Pos } A \rangle$  .
next
  show  $\neg \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \models C$ 
  using  $\langle \neg \text{dom } \mathcal{M} \models C \rangle$ 
  unfolding invars(2)[unfolded model-eq-interp-upto-next-clause-def, rule-format,
    OF  $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle$ ].
qed simp-all

thus ?thesis
  using invars(1)
  by (simp add: atoms-in-model-were-produced-def
     $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle \langle s' = (U_{er}, \mathcal{F}, \mathcal{M}', C') \rangle \langle \mathcal{M}' = \mathcal{M}(\text{atm-of } L$ 
 $\mapsto C) \rangle$ )
qed (simp-all add: atoms-in-model-were-produced-def)

definition all-produced-atoms-in-model where
  all-produced-atoms-in-model  $N s \longleftrightarrow$ 
   $(\forall U_{er} \mathcal{F} \mathcal{M} D. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \longrightarrow (\forall C A. C \prec_c D \longrightarrow$ 
   $A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \longrightarrow \mathcal{M} A = \text{Some}$ 
   $C))$ 

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  $\mathcal{M} C C' \mathcal{F} U_{er}$ )
  have ord-res.production  $(\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C = \{\}$ 
  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)  $\langle s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \rangle$ 
  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-filter-iff linorder-cls.not-less-iff-gr-or-eq local.skip(2)
    local.skip(4) ord-res.mem-productionE)
next
  case step-hyps: (production  $\mathcal{M} C L \mathcal{M}' C' \mathcal{F} U_{er}$ )
  obtain  $A$  where  $L = \text{Pos } A$ 

```

```

using ⟨is-pos L⟩ by (cases L) simp-all

have atm-of L ∈ ord-res.production (fset (iefac  $\mathcal{F}$  | $\uparrow$  ( $N \cup U_{er}$ ))) 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  $C \models \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
    using invars ⟨s = (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some C)⟩ by (metis next-clause-in-factorized-clause-def)
next
  have  $L \in \# C$ 
    using ⟨linorder-lit.is-maximal-in-mset C L⟩
    unfolding linorder-lit.is-maximal-in-mset-iff ..
  thus  $C = \text{add-mset} (\text{Pos } A) (C - \{\# \text{Pos } A\})$ 
    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  $\neg \text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er}))) C \models C$ 
    using ⟨ $\neg \text{dom } \mathcal{M} \models C$ ⟩
    using invars ⟨s = (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some C)⟩ by (metis model-eq-interp-upto-next-clause-def)
qed simp-all

thus ?thesis
  using invars ⟨s = (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some C)⟩
  unfolding all-produced-atoms-in-model-def
  using ⟨s' = (Uer,  $\mathcal{F}$ ,  $\mathcal{M}'$ ,  $\mathcal{C}'$ )⟩ ⟨ $\mathcal{M}' = \mathcal{M}(\text{atm-of } L \mapsto C)$ ⟩
  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  $\mathcal{M} C L \mathcal{F}' \mathcal{F} \mathcal{M}' \mathcal{C}' U_{er}$ )
  have  $\neg (\exists C \models \text{iefac } \mathcal{F}' \uparrow (N \cup U_{er}). C \prec_c D)$  if  $\mathcal{C}' = \text{Some } D$  for D
  proof (rule nbex-less-than-least-in-fset)
    show linorder-cls.is-least-in-fset (iefac  $\mathcal{F}'$  | $\uparrow$  ( $N \cup U_{er}$ )) 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  $\mathcal{M} C L D U_{er}' U_{er} \mathcal{M}' \mathcal{C}' \mathcal{F}$ )
  have  $\neg (\exists C \models \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}'). C \prec_c D)$  if  $\mathcal{C}' = \text{Some } D$  for D
  proof (rule nbex-less-than-least-in-fset)

```

```

show linorder-cls.is-least-in-fset (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ )  $D$ 
  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
  F-subset:  $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$  and
  C-least: linorder-cls.is-least-in-fset (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ )  $C$ 
shows ord-res-5-invars  $N (U_{er}, \mathcal{F}, \text{Map.empty}, \text{Some } C)$ 
unfolding ord-res-5-invars-def
proof (intro conjI)
  show next-clause-in-factorized-clause  $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$ 
    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  $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$ 
    unfolding implicitly-factorized-clauses-subset-def
    using F-subset by simp
  next
  have ord-res.interp (iefac  $\mathcal{F} \mid \uparrow (fset\ N \cup fset\ U_{er})$ )  $C = \{\}$ 
    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  $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$ 
    unfolding model-eq-interp-upto-next-clause-def by simp
  next
  have  $\neg(\exists Ca \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). Ca \prec_c C)$ 
    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  $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$ 
    unfolding all-smaller-clauses-true-wrt-respective-Interp-def by simp
  next
  show atoms-in-model-were-produced  $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$ 
    unfolding atoms-in-model-were-produced-def by simp
  next

```

have $\forall Ca. Ca \prec_c C \longrightarrow Ca \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using *C-least[unfolding linorder-cls.is-least-in-fset-iff]*
by (*metis linorder-cls.order.asym*)
thus *all-produced-atoms-in-model* $N (U_{er}, \mathcal{F}, \lambda x. \text{None}, \text{Some } C)$
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 $U_{er} \mathcal{F} \mathcal{M} C$ **where** *s-def: s = (U_{er}, F, M, C)*
by (*metis prod.exhaust*)

then show *?thesis*
unfolding *ord-res-5-invars-def*
using *invars[unfolding 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: (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: (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}, F, M, Some C) **and**
D-least-false: is-least-false-clause (iefac F |[↑] (N |[∪] | U_{er})) D
shows $C \preceq_c D$
proof –
have *D-in: D |[∈] | iefac F |[↑] (N |[∪] | U_{er})*
using *D-least-false*


```

unfolding is-least-false-clause-def linorder-clis-least-in-filter-iff
by argo

show  $C \preceq_c D$ 
proof (rule next-clause-lt-least-false-clause[rule-format])
  show is-least-false-clause (iefac  $\mathcal{F}$  |q (N | $\cup$ |  $U_{er}$ )) D
  using D-least-false .
next
  show  $(U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) ..$ 
next
  show all-smaller-clauses-true-wrt-respective-Interp N (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some C)
  using invars by (metis s-def ord-res-5-invars-def)
next
  show model-eq-interp-upto-next-clause N (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , 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:  $\neg$  ord-res-5-final S and
  invars:  $\forall N s. S = (N, s) \longrightarrow$  ord-res-5-invars N s
shows  $\exists S'. \text{ord-res-5-step } S S'$ 
proof –
from not-final obtain  $N U_{er} \mathcal{F} \mathcal{M} C$  where
  S-def:  $S = (N, (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C))$  and  $C \neq \{\#\}$ 
unfolding ord-res-5-final.simps de-Morgan-disj not-ex
by (metis option.exhaust surj-pair)

note  $\text{invars}' = \text{invars}[\text{unfolded } \text{ord-res-5-invars-def}, \text{rule-format}, \text{OF } S\text{-def}]$ 

have  $\exists s'. \text{ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) s'$ 
proof (cases dom  $\mathcal{M} \models C$ )
  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  $\langle C \neq \{\#\} \rangle$ ] ..

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-strictly-max: False
  thus ?thesis
    using ord-res-5.factoring[OF C-false L-max L-pos L-not-strictly-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  $\mathcal{F}$  | $\uparrow$  (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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )
    using invars' by (metis next-clause-in-factorized-clause-def)
next
  have  $\neg$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (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  $\mathcal{F}$  | $\uparrow$  (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  $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) C  $\models$  C
    by simp
next
fix D
  assume D-in: D | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ) and D  $\neq$  C and
    C-false:  $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) D  $\models$  D
  have  $\neg$  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  $\langle D \neq C \rangle$  by order
qed
then obtain D where D | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) 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 insertII
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 (Uer, F, M, C)
  shows ∃ M' C'. (ord-res-5 N)** (Uer, F, M, C) (Uer, F, M', C') ∧
    (∄ M'' C''. ord-res-5 N (Uer, F, M', C') (Uer, F, M'', C''))

```

proof –

define R **where**

$$R = (\lambda(\mathcal{M}, \mathcal{C}) (\mathcal{M}', \mathcal{C}'). \text{ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}'))$$

define $f :: ('f \text{ gterm} \Rightarrow 'f \text{ gclause option}) \times 'f \text{ gclause option} \Rightarrow 'f \text{ gclause fset}$

where

$$f = (\lambda(\mathcal{M}, \mathcal{C}). \text{case } \mathcal{C} \text{ of None} \Rightarrow \{|\}\} \mid \text{Some } \mathcal{C} \Rightarrow \text{finsert } \mathcal{C} \{|\mathcal{D} \mid \in \mid \text{iefac } \mathcal{F} \mid \} (N \mid \cup \mid U_{er}). \mathcal{C} \preceq_c \mathcal{D} \mid \})$$

have $\text{Wellfounded.wfp-on } \{x'. R^{**} (\mathcal{M}, \mathcal{C}) x'\} R^{-1-1}$

proof (*rule wfp-on-if-convertible-to-wfp-on*)

have $\text{wfp } (|\subset|)$

by *auto*

thus $\text{Wellfounded.wfp-on } (f \text{ ' } \{x'. R^{**} (\mathcal{M}, \mathcal{C}) x'\}) (|\subset|)$

using $\text{Wellfounded.wfp-on-subset subset-UNIV}$ **by** *metis*

next

fix $x \ y :: ('f \text{ gterm} \Rightarrow 'f \text{ gclause option}) \times 'f \text{ gclause option}$

obtain $\mathcal{M}_x \ \mathcal{C}_x$ **where** $x\text{-def}: x = (\mathcal{M}_x, \mathcal{C}_x)$

by *force*

obtain $\mathcal{M}_y \ \mathcal{C}_y$ **where** $y\text{-def}: y = (\mathcal{M}_y, \mathcal{C}_y)$

by *force*

have $\text{rtranclp-with-constsD}: (\lambda(y, z) (y', z'). R (x, y, z) (x, y', z'))^{**} (y, z) (y', z') \Rightarrow$

$$R^{**} (x, y, z) (x, y', z') \text{ for } R \ x \ y \ z \ y' \ z'$$

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

$$(\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, \mathcal{M}_x, \mathcal{C}_x) \text{ and}$$

$$(\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, \mathcal{M}_y, \mathcal{C}_y)$$

unfolding *atomize-conj mem-Collect-eq R-def x-def y-def*

by (*auto intro: rtranclp-with-constsD*)

hence

$$x\text{-invars: ord-res-5-invars } N (U_{er}, \mathcal{F}, \mathcal{M}_x, \mathcal{C}_x) \text{ and}$$

$$y\text{-invars: ord-res-5-invars } N (U_{er}, \mathcal{F}, \mathcal{M}_y, \mathcal{C}_y)$$

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 $f y \mid\sqsubset f x$
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 $f y \mid\subseteq \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D\}$
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 $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))$* D
using *step-hyps \mathcal{C}_y -def* **by** (*metis The-optional-eq-SomeD*)
hence *f-y-eq*: $f y = \{|E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). D \preceq_c E\}$
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 $\dots \mid\sqsubset \text{finsert } C \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \preceq_c D\}$
by *auto*

also have $\dots = f x$
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 $f y \mid\subseteq \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D\}$
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 $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))$* D
using *step-hyps \mathcal{C}_y -def* **by** (*metis The-optional-eq-SomeD*)

hence $f\text{-}y\text{-eq}$: $f\ y = \{|E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \cdot D \preceq_c E \mid\}$
unfolding $f\text{-def } y\text{-def } \text{prod.case } C_y\text{-def } \text{option.case } \text{linorder-cls.is-least-in-ffilter-iff}$
by *auto*

show *?thesis*
unfolding $f\text{-}y\text{-eq } \text{subset-ffilter}$
using $D\text{-least}$
unfolding $\text{linorder-cls.is-least-in-ffilter-iff}$
by *auto*

qed

also have $\dots \mid C \mid \text{finsert } C \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \cdot C \preceq_c D \mid\}$
by *auto*

also have $\dots = f\ x$
unfolding $f\text{-def } x\text{-def } y\text{-def } \text{prod.case } \text{step-hyps } \text{option.case}$ **by** *metis*

finally show *?thesis* .

next

case step-hyps : (*factoring* $C\ L$)

have $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using $x\text{-invars}$ **unfolding** $\langle C_x = \text{Some } C \rangle$
by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)
hence $C \mid \notin \mid \mathcal{F}$
using $\text{step-hyps}(3,4,5)$
by (*smt (verit, ccfv-SIG) fimage-iff iefac-def literal.collapse(1) ex1-efac-eq-factoring-chain ex-ground-factoringI*)
moreover have $C \mid \in \mid \mathcal{F}$
using $\langle \mathcal{F} = \text{finsert } C\ \mathcal{F} \rangle$ **by** *auto*
ultimately have *False*
by *contradiction*
thus *?thesis ..*

next

case step-hyps : (*resolution* $C\ L\ D$)

have $D\text{-productive}$: $\text{atm-of } L \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})))\ D$
using $x\text{-invars } \text{step-hyps}$
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 $\text{eres } D\ C \neq C$

unfolding *eres-ident-iff* **by** *metis*

have $eres\ D\ C\ |\notin|\ U_{er}$
proof (*rule notI*)
 assume $eres\ D\ C\ |\in|\ U_{er}$
 hence $iefac\ \mathcal{F}\ (eres\ D\ C)\ |\in|\ iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})$
 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 |\in| F*)
 case *True*
 hence $iefac\ \mathcal{F}\ C = efac\ C$
 by (*simp add: iefac-def*)
 also have $\dots \preceq_c\ C$
 by (*metis efac-subset subset-implies-reflclp-multp*)
 also have $\dots \prec_c\ D$
 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 $\langle eres\ D\ C \neq C \rangle$ **by** *order*
qed

ultimately show *?thesis*
by *metis*
qed

ultimately have $ord\text{-}res\text{-}Interp\ (fset\ (iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ (iefac\ \mathcal{F}\ (eres\ D\ C)) \models iefac\ \mathcal{F}\ (eres\ D\ C)$
 using *x-invars* **unfolding** $\langle \mathcal{C}_x = Some\ C \rangle$
unfolding *ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def*
by *fast*
 hence $ord\text{-}res.\text{interp}\ (fset\ (iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ C \models iefac\ \mathcal{F}\ (eres\ D\ C)$
 using *ord-res.entailed-clause-stays-entailed'* $\langle iefac\ \mathcal{F}\ (eres\ D\ C)\ \prec_c\ C \rangle$ **by**
 metis
 hence $ord\text{-}res.\text{interp}\ (fset\ (iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ C \models eres\ D\ C$
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  $\mathcal{F}$  | $\uparrow$  ( $N$  | $\cup$ |  $U_{er}$ )))  $C$   $\models$   $C$ 
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  $A$   $\not\#$   $C'$  and
    eres- $D$ - $C$ -eq: eres  $D$   $C$  = repeat-mset (Suc  $m$ )  $D'$  +  $C'$ 
    using  $\langle$ eres  $D$   $C$   $\neq$   $C$  $\rangle$ [THEN eres-not-ident $D$ ] by metis

  hence
    atm-of  $L$  =  $A$  and
     $D$ -in:  $D$  | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  ( $N$  | $\cup$ |  $U_{er}$ ) and
     $D$ -false:  $\neg$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  ( $N$  | $\cup$ |  $U_{er}$ )))  $D$   $\models$   $D$ 
    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:  $\neg$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  ( $N$  | $\cup$ |  $U_{er}$ )))  $D$   $\models$   $D'$ 
    using  $D$ -false  $D$ -def by fastforce

  have  $D$   $\prec_c$   $C$ 
  using  $\langle$  $\exists$   $DC$ . ground-resolution  $D$   $C$   $DC$  $\rangle$  left-premise-lt-right-premise-if-ground-resolution
by blast

  let  $?I$  = ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  ( $N$  | $\cup$ |  $U_{er}$ )))  $C$ 

  assume  $?I$   $\models$  eres  $D$   $C$ 
  hence  $?I$   $\models$   $D' \vee ?I$   $\models$   $C'$ 
    unfolding eres- $D$ - $C$ -eq true-cls-union true-cls-repeat-mset-Suc .

  moreover have  $\neg$   $?I$   $\models$   $D'$ 
    using  $\langle$  $D$   $\prec_c$   $C$  $\rangle$ 
    by (smt (verit)  $D$ -def  $D$ -productive  $\langle$ ord-res.is-strictly-maximal-lit (Pos
A)  $D$  $\rangle$ 
diff-single-eq-union ord-res.mem-productionE linorder-lit.is-greatest-in-mset-iff
ord-res.Uniq-strictly-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  $?I$   $\models$   $C$ 
    by (simp add:  $C$ -def)
qed
hence dom  $\mathcal{M}_x$   $\models$   $C$ 
  using  $x$ -invars  $\langle$  $\mathcal{C}_x$  = Some  $C$  $\rangle$ 

```



```

    by (metis model-eq-interp-upto-next-clause-def ord-res-5-invars-def)
  thus False
    using ⟨ $\neg \text{dom } \mathcal{M}_x \models C$ ⟩ by contradiction
qed
hence False
  using ⟨ $U_{er} = \text{finsert } (\text{eres } D \ C) \ U_{er}$ ⟩ by auto
  thus ?thesis ..
qed
qed

then obtain  $\mathcal{M}' \ C'$  where  $R^{**} (\mathcal{M}, \mathcal{C}) (\mathcal{M}', \mathcal{C}')$  and  $\#z. R (\mathcal{M}', \mathcal{C}') z$ 
  using ex-terminating-rtranclp-strong by (metis surj-pair)

show ?thesis
proof (intro exI conjI)
  show (ord-res-5 N)**  $(U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}')$ 
    using ⟨ $R^{**} (\mathcal{M}, \mathcal{C}) (\mathcal{M}', \mathcal{C}')$ ⟩
    by (induction  $(\mathcal{M}, \mathcal{C})$  arbitrary:  $\mathcal{M} \ \mathcal{C}$  rule: converse-rtranclp-induct) (auto
simp: R-def)
  next
    show  $\# \mathcal{M}'' \ \mathcal{C}''.$  ord-res-5 N  $(U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}') (U_{er}, \mathcal{F}, \mathcal{M}'', \mathcal{C}'')$ 
      using ⟨ $\#z. R (\mathcal{M}', \mathcal{C}') z$ ⟩
      by (simp add: R-def)
    qed
  qed
qed

lemma MAGIC2:
  assumes invars: ord-res-5-invars N  $(U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C)$ 
  assumes  $C \neq \{\#\}$ 
  shows  $\exists s'. \text{ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) s'$ 
proof (cases  $(\text{dom } \mathcal{M}) \models C$ )
  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 ⟨ $C \neq \{\#\}$ ⟩ 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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )) C
    unfolding is-least-false-clause-def linorder-clis.is-least-in-ffilter-iff
proof (intro conjI ballI impI)
    show C | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )
    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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) 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  $\mathcal{F}$  | $\uparrow$  (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  $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) C  $\models$  C
    using C-false model-eq-interp-upto-next-clause-def by simp
next
fix D
assume
    D | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ) and
    D  $\neq$  C and
     $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) D  $\models$  D

moreover have  $\forall B$  | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ). B  $\prec_c$  C  $\longrightarrow$ 
    ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) B  $\models$  B
    using invars
unfolding ord-res-5-invars-def all-smaller-clauses-true-wrt-respective-Interp-def
    by simp

ultimately show C  $\prec_c$  D
    by force

```

qed
then obtain D **where** $D \in |\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $D \prec_c C$ **and**
ord-res.is-strictly-maximal-lit (*Pos* A) D **and**
 $D\text{-prod: ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D = \{A\}$
using *bx-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
 $L\text{-max}[\text{unfolded } Neg]$ **by** *metis*

have \mathcal{M} (*atm-of* L) = *Some* D
using *invars*
unfolding *ord-res-5-invars-def all-produced-atoms-in-model-def*
by (*metis* $Neg \langle D \prec_c C \rangle D\text{-prod insertI1 literal.sel(2)$)

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* $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$ **and**
*steps: (ord-res-5 N)*** ($U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}$) ($U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}'$) **and**
no-more-steps: ($\nexists \mathcal{M}'' \mathcal{C}''$. ord-res-5 N ($U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}'$) ($U_{er}, \mathcal{F}, \mathcal{M}'', \mathcal{C}''$))
shows ($\forall C. C' = \text{Some } C \longleftrightarrow \text{is-least-false-clause (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C$)
proof –
have *invars'*: *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}')$
using *steps invars rtranclp-ord-res-5-preserves-invars* **by** *metis*

show *?thesis*
proof (*cases* \mathcal{C}')
case *None*

moreover have $\nexists C. \text{is-least-false-clause (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C$
proof (*rule notI, elim exE*)
fix C

have *all-smaller-clauses-true-wrt-respective-Interp* $N (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}')$
using *invars'* **unfolding** *ord-res-5-invars-def* **by** *metis*
hence ($\forall C \in |\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \models C$)
by (*simp add: $\langle C' = \text{None} \rangle$ all-smaller-clauses-true-wrt-respective-Interp-def*)

moreover assume *is-least-false-clause (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid 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* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}$)) D
unfolding *is-least-false-clause-def linorder-cls.is-least-in-filter-iff*
proof (*intro conjI ballI impI*)
show $D \mid \in \mid$ *iefac* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}$)
using *invars' $\langle C' = \text{Some } D \rangle$*
unfolding *ord-res-5-invars-def next-clause-in-factorized-clause-def*
by *metis*
next
have $D \neq \{\#\} \implies \exists s'. \text{ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}', \text{Some } D) s'$
using *MAGIC2 invars' $\langle C' = \text{Some } D \rangle$ by metis*

thus $\neg \text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D \models D$
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 op-
tion.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
 $E \mid \in \mid$ *iefac* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}$) **and**
 $E \neq D$ **and**
 $\neg \text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) E \models E$

moreover have $\forall C \mid \in \mid$ *iefac* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}$). $C \prec_c D \implies$
 $\text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \models C$
using *invars' $\langle C' = \text{Some } D \rangle$*
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 $\exists \mathcal{M}' \mathcal{C}'. (\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, \mathcal{M}', \mathcal{C}') \wedge$
 $(\forall C. C' = \text{Some } C \iff \text{is-least-false-clause (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C)$

```

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:
 $(\text{dom } \mathcal{M}) \models C \implies$
 $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D| \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}).$
 $C \prec_c D\}) \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, C') |$

production:
 $\neg (\text{dom } \mathcal{M}) \models C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C L \implies$
 $\text{is-pos } L \implies$
 $\text{linorder-lit.is-greatest-in-mset } C L \implies$
 $\mathcal{M}' = \mathcal{M}(\text{atm-of } L := \text{Some } C) \implies$
 $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D| \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}).$
 $C \prec_c D\}) \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}', C') |$

factoring:
 $\neg (\text{dom } \mathcal{M}) \models C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C L \implies$
 $\text{is-pos } L \implies$
 $\neg \text{linorder-lit.is-greatest-in-mset } C L \implies$
 $\mathcal{F}' = \text{finsert } C \mathcal{F} \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (\text{efac } C)) |$

resolution-bot:
 $\neg (\text{dom } \mathcal{M}) \models C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C L \implies$
 $\text{is-neg } L \implies$
 $\mathcal{M} (\text{atm-of } L) = \text{Some } D \implies$
 $U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies$
 $\text{eres } D C = \{\#\} \implies$
 $\mathcal{M}' = (\lambda-. \text{None}) \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } \{\#\}) |$

resolution-pos:

$\neg (\text{dom } \mathcal{M}) \models C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C L \implies$
 $\text{is-neg } L \implies$
 $\mathcal{M} (\text{atm-of } L) = \text{Some } D \implies$
 $U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies$
 $\text{eres } D C \neq \{\#\} \implies$
 $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \implies$
 $\text{linorder-lit.is-maximal-in-mset } (\text{eres } D C) K \implies$
 $\text{is-pos } K \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } (\text{eres } D C)) \mid$

resolution-neg:

$\neg (\text{dom } \mathcal{M}) \models C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C L \implies$
 $\text{is-neg } L \implies$
 $\mathcal{M} (\text{atm-of } L) = \text{Some } D \implies$
 $U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies$
 $\text{eres } D C \neq \{\#\} \implies$
 $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \implies$
 $\text{linorder-lit.is-maximal-in-mset } (\text{eres } D C) K \implies$
 $\text{is-neg } K \implies$
 $\mathcal{M} (\text{atm-of } K) = \text{Some } E \implies$
 $\text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } E)$

inductive ord-res-6-step where

$\text{ord-res-6 } N s s' \implies \text{ord-res-6-step } (N, s) (N, s')$

lemma tranclp-ord-res-6-step-if-tranclp-ord-res-6:

$(\text{ord-res-6 } N)^{++} s s' \implies \text{ord-res-6-step}^{++} (N, s) (N, s')$

by (*induction s' rule: tranclp-induct*)

(*auto intro: ord-res-6-step.intros tranclp.intros*)

lemma right-unique-ord-res-6: right-unique ($\text{ord-res-6 } N$)

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 $\text{ord-res-6-step } x y \implies \text{ord-res-6-step } x z \implies y = z$

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 ($N, U_{er}, \mathcal{F}, \mathcal{M}, \text{None}$) |

contradiction-found:

ord-res-6-final ($N, U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } \{\#\}$)

sublocale *ord-res-6-antics: semantics* **where**

step = *ord-res-6-step* **and**

final = *ord-res-6-final*

proof *unfold-locales*

fix $S :: 'f \text{ gclause } fset \times 'f \text{ gclause } fset \times 'f \text{ gclause } fset \times$
 $('f \text{ gterm } \Rightarrow 'f \text{ gclause } option) \times 'f \text{ gclause } option$

obtain

$N U_{er} \mathcal{F} :: 'f \text{ gterm } clause \ fset$ **and**

$\mathcal{M} :: 'f \text{ gterm } \Rightarrow 'f \text{ gclause } option$ **and**

$\mathcal{C} :: 'f \text{ gclause } option$ **where**

$S\text{-def}: S = (N, (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}))$

by (*metis prod.exhaust*)

assume *ord-res-6-final* S

hence $\mathcal{C} = \text{None} \vee \mathcal{C} = \text{Some } \{\#\}$

by (*simp add: S-def ord-res-6-final.simps*)

hence $\nexists x. \text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) x$

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* $\mathcal{M} \mathcal{C} \mathcal{C}' \mathcal{F} U_{er}$)

thus *?thesis*

by (*metis invars ord-res-5-preserves-invars ord-res-5.skip*)

next

case (*production* $\mathcal{M} \mathcal{C} L \mathcal{M}' \mathcal{C}' \mathcal{F} U_{er}$)

thus *?thesis*

by (*metis invars ord-res-5.production ord-res-5-preserves-invars*)

next

case *step-hyps: (factoring* $\mathcal{M} \mathcal{C} L \mathcal{F}' \mathcal{F} U_{er}$)

have *efac* $C \neq C'$

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 $C \in |iefac\ \mathcal{F}| (N \cup U_{er})$

using *invars step-hyps*

by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)

hence $C \in |N \cup U_{er}|$

using $\langle efac\ C \neq C \rangle$

by (*smt (verit, best) fimage-iff iefac-def ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)

hence $efac\ C \in |iefac\ \mathcal{F}'| (N \cup U_{er})$

using *step-hyps(3-)*

using *iefac-def* **by** *auto*

thus *next-clause-in-factorized-clause* $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (efac\ C))$

unfolding *next-clause-in-factorized-clause-def* **by** *simp*

have $\mathcal{F} \subseteq |N \cup U_{er}|$

using *invars*

unfolding *step-hyps(1,2) ord-res-5-invars-def implicitly-factorized-clauses-subset-def*

by *metis*

hence $\mathcal{F}' \subseteq |N \cup U_{er}|$

using $\langle C \in |N \cup U_{er}| \langle \mathcal{F}' = \text{finsert } C\ \mathcal{F} \rangle$ **by** *simp*

thus *implicitly-factorized-clauses-subset* $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (efac\ C))$

unfolding *implicitly-factorized-clauses-subset-def* **by** *simp*

have *dom-M-eq*: $\text{dom } \mathcal{M} = \text{ord-res.interp } (\text{fset } (iefac\ \mathcal{F} | (N \cup U_{er})))\ C$

using *invars step-hyps*

by (*simp add: model-eq-interp-upto-next-clause-def ord-res-5-invars-def*)

have $efac\ C \notin |iefac\ \mathcal{F}| (N \cup U_{er})$

proof (*rule notI*)

assume $efac\ C \in |iefac\ \mathcal{F}| (N \cup U_{er})$

show *False*

proof (*cases atm-of L ∈ ord-res.production (fset (iefac F | (N ∪ U_{er}))) (efac C)*)

assume *atm-of L ∈ ord-res.production (fset (iefac F | (N ∪ U_{er}))) (efac C)*

C)

hence *atm-of L ∈ ord-res.interp (fset (iefac F | (N ∪ U_{er}))) C*

using $\langle efac\ C \prec_c\ C \rangle$ *ord-res.production-subset-if-less-cls* **by** *blast*

hence $\text{dom } \mathcal{M} \Vdash C$

using *step-hyps*

by (*metis dom-M-eq linorder-lit.is-maximal-in-mset-iff literal.collapse(1)*)

pos-literal-in-imp-true-cls)

thus *False*

using $\langle \neg \text{dom } \mathcal{M} \models C \rangle$ **by** *contradiction*

next

C) **assume** *atm-of* $L \notin \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) (efac C))}$

hence *ord-res.interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) (efac C) \models efac C*

unfolding *ord-res.production-unfold mem-Collect-eq*

using $\langle efac C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$

by (*metis Pos-atm-of-iff* $\langle efac C \neq C \rangle$ *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} \mid \uparrow (N \mid \cup U_{er})) C \models efac C*

using $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle \langle efac C \prec_c C \rangle \langle efac C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$

ord-res.lift-interp-entails **by** *metis*

hence *ord-res.interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) C \models C*

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

hence *dom } \mathcal{M} \models C*

using *dom-} \mathcal{M}-eq* **by** *argo*

thus *False*

using $\langle \neg \text{dom } \mathcal{M} \models C \rangle$ **by** *contradiction*

qed

qed

have *iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er}) = finsert (iefac } \mathcal{F}' C) (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))*

– $\{C\}$

proof (*intro fsubset-antisym fsubsetI*)

fix $x :: 'f \text{ gclause}$

assume $x \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$

thus $x \mid \in \mid \text{finsert (iefac } \mathcal{F}' C) (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \mid - \mid \{C\}$

by (*smt (verit) } \langle efac C \neq C \rangle* *factorizable-if-neq-efac fimage-iff finsert-iff*
fminusI *fsingletonE iefac-def ex1-efac-eq-factoring-chain step-hyps(7)*)

next

fix $x :: 'f \text{ gclause}$

assume *x-in: } $x \mid \in \mid \text{finsert (iefac } \mathcal{F}' C) (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \mid - \mid \{C\}$*

hence $x = \text{iefac } \mathcal{F}' C \vee x \mid \in \mid (iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \mid - \mid \{C\}$

by *blast*

thus $x \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$

proof (*elim disjE*)

assume $x = \text{iefac } \mathcal{F}' C$

thus $x \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$

using $\langle C \mid \in \mid N \mid \cup U_{er} \rangle$ **by** *blast*

next

assume $x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \mid - \mid \{C\}$

hence $x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$ **and** $x \neq C$

by *simp-all*

then obtain x' **where** $x' \in N \cup U_{er}$ **and** $x = \text{iefac } \mathcal{F} \ x'$
by *auto*
moreover have $x' \neq C$
using $\langle x \neq C \rangle \langle x = \text{iefac } \mathcal{F} \ x' \rangle$
by (*metis* $\langle \text{efac } C \notin \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}) \rangle \langle x \in \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}) \rangle$) *iefac-def*
ultimately show $x \in \text{iefac } \mathcal{F}' \mid^\dagger (N \cup U_{er})$
using *iefac-def step-hyps*(γ) **by** *simp*
qed
qed

have $\text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F}' \mid^\dagger (N \cup U_{er}))) (\text{efac } C) =$
 $\text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}))) (\text{efac } C)$
proof (*rule ord-res.interp-swap-clause-set*)
show $\{D. D \in \text{iefac } \mathcal{F}' \mid^\dagger (N \cup U_{er}) \wedge D \prec_c \text{efac } C\} =$
 $\{D. D \in \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}) \wedge D \prec_c \text{efac } C\}$
unfolding $\langle \text{iefac } \mathcal{F}' \mid^\dagger (N \cup U_{er}) = \text{finsert} (\text{iefac } \mathcal{F}' \ C) (\text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er})) - \{C\} \rangle$
using $\langle \text{efac } C \prec_c C \rangle$
using *iefac-def by force*
qed

also have $\dots = \text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}))) \ C$
proof –
have $\forall x \in \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}). x \prec_c C \longrightarrow$
 $\text{ord-res.Interp} (\text{fset} (\text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}))) \ x \models x$
using *invars[unfolded ord-res-5-invars-def step-hyps(1)*
all-smaller-clauses-true-wrt-respective-Interp-def, simplified]
by *simp*
then have $\text{ord-res.production} (\text{fset} (\text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er}))) \ x = \{\}$
if $x \in \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er})$ **and** $\text{efac } C \prec_c x$ **and** $x \prec_c C$ **for** x
proof –
have $x \preceq_l y \wedge y \preceq_l z$
if $X \preceq_c Y$ **and** $Y \preceq_c Z$ **and**
 $\text{linorder-lit.is-maximal-in-mset } X \ x$ **and**
 $\text{linorder-lit.is-maximal-in-mset } Y \ y$ **and**
 $\text{linorder-lit.is-maximal-in-mset } Z \ z$
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*
 $\text{linorder-lit.leI linorder-lit.mulp}_{HO}$ -*if-maximal-less-that-maximal*
 mulp-eq-mulp_{HO}
 $\text{that}(3) \ \text{that}(4) \ \text{that}(5)$)
hence $y = x$
if $X \preceq_c Y$ **and** $Y \preceq_c Z$ **and**
 $\text{linorder-lit.is-maximal-in-mset } X \ x$ **and**
 $\text{linorder-lit.is-maximal-in-mset } Y \ y$ **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 $X \prec_c Y$ **and** $Y \prec_c Z$ **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 \langle *ord-res.is-maximal-lit* L C \rangle
using \langle *efac* $C \prec_c x$ \rangle \langle $x \prec_c C$ \rangle *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 $\exists A. A \in$ *ord-res.production* (*fset* (*iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$))) x
proof (*rule notI* , *elim exE*)
fix $A :: 'f$ *gterm*
assume $A \in$ *ord-res.production* (*fset* (*iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$))) x
then obtain x' **where**
 $x \in |$ *iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$) **and**
 $x =$ *add-mset* (*Pos* A) x' **and**
ord-res.is-strictly-maximal-lit (*Pos* A) x **and**
 \neg *ord-res.interp* (*fset* (*iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$))) $x \models x$
by (*metis ord-res.mem-productionE*)

have $A \in$ *ord-res.interp* (*fset* (*iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$))) C
using \langle $A \in$ *ord-res.production* (*fset* (*iefac* \mathcal{F} $| \uparrow$ ($N \cup U_{er}$))) x \rangle
ord-res.production-subset-if-less-cls *that(3)* **by** *fastforce*

moreover have $L =$ *Pos* A

using $\langle \text{ord-res.is-maximal-lit } L \ x \rangle \langle \text{ord-res.is-strictly-maximal-lit } (Pos \ A) \ x \rangle$
by $(metis \ \text{Uniq-D linorder-lit.Uniq-is-maximal-in-mset linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset})$
moreover have $L \in \# \ C$
using $\text{step-hyps linorder-lit.is-maximal-in-mset-iff}$ **by** metis
ultimately have $\text{ord-res.interp } (fset \ (iefac \ \mathcal{F} \ |\!| \ (N \ |\cup| \ U_{er}))) \ C \models C$
by auto
hence $\text{dom } \mathcal{M} \models C$
using $\text{dom-}\mathcal{M}\text{-eq}$ **by** argo
thus False
using $\langle \neg \ \text{dom } \mathcal{M} \models C \rangle$ **by** contradiction
qed
thus $?thesis$
by simp
qed
moreover have $\{x. x \in |iefac \ \mathcal{F} \ |\!| \ (N \ |\cup| \ U_{er}) \wedge x \prec_c C\} =$
 $\{x. x \in |iefac \ \mathcal{F} \ |\!| \ (N \ |\cup| \ U_{er}) \wedge x \prec_c \text{efac } C\} \cup$
 $\{x. x \in |iefac \ \mathcal{F} \ |\!| \ (N \ |\cup| \ U_{er}) \wedge \text{efac } C \prec_c x \wedge x \prec_c C\}$
proof –
have $\{w \in NN. w \prec_c z\} = \{w \in NN. w \prec_c x\} \cup \{y \in NN. x \prec_c y \wedge y \prec_c z\}$
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 $\text{that}(2)$ **by** auto
also have $\dots = \{w \in NN. w \prec_c x \vee x \prec_c w \wedge w \prec_c z\}$
using $\text{that}(1)$ **by** auto
also have $\dots = \{w \in NN. w \prec_c x\} \cup \{y \in NN. x \prec_c y \wedge y \prec_c z\}$
by auto
finally show $?thesis$.
qed
thus $?thesis$
using $\langle \text{efac } C \prec_c C \rangle \langle \text{efac } C \notin |iefac \ \mathcal{F} \ |\!| \ (N \ |\cup| \ U_{er}) \rangle$ **by** (simp only:)
qed
ultimately show $?thesis$
unfolding $\text{ord-res.interp-def}$ **by** auto
qed
finally show $\text{model-eq-interp-upto-next-clause } N \ (U_{er}, \ \mathcal{F}', \ \mathcal{M}, \ \text{Some } (\text{efac } C))$
unfolding $\text{model-eq-interp-upto-next-clause-def}$
using $\text{dom-}\mathcal{M}\text{-eq}$

by simp

have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er})$)) $x \models x$
if $x \in \text{iefac } \mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er})$ **and** $x \prec_c \text{efac } C$ **for** x

proof –

have $x \in \text{iefac } \mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er})$
using *that*
by (*metis* $\langle \text{iefac } \mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er}) = \text{finsert} (\text{iefac } \mathcal{F}' C) (\text{iefac } \mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er})) \mid - \mid \{C\} \rangle$
finsert-iff fminusE iefac-def linorder-cls.neq-iff)

moreover have $x \prec_c C$
using $\langle x \prec_c \text{efac } C \rangle \langle \text{efac } C \prec_c C \rangle$ **by** *order*

ultimately have *ord-res-Interp* (*fset* (*iefac* $\mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er})$)) $x \models x$
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* $\mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er})$)) $x =$
ord-res-Interp (*fset* (*iefac* $\mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er})$)) x

proof (*rule* *ord-res.Interp-swap-clause-set*)
show $\{D. D \in \text{iefac } \mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er}) \wedge (\prec_c)^{==} D x\} =$
 $\{D. D \in \text{iefac } \mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er}) \wedge (\prec_c)^{==} D x\}$
unfolding $\langle \text{iefac } \mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er}) = \text{finsert} (\text{iefac } \mathcal{F}' C) (\text{iefac } \mathcal{F} \upharpoonright^{\uparrow} (N \upcup U_{er})) \mid - \mid \{C\} \rangle$
using $\langle x \prec_c \text{efac } C \rangle \langle x \prec_c C \rangle$
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* $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some} (\text{efac } C))$
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def* **by** *blast*

have *linorder-lit.is-greatest-in-mset* (*efac* C) L
using $\langle \text{linorder-lit.is-maximal-in-mset } C L \rangle$

by (*metis* $\langle \text{efac } C \neq C \rangle$ *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 $A \in \text{ord-res.production} (\text{fset} (\text{iefac } \mathcal{F}' \upharpoonright^{\uparrow} (N \upcup U_{er}))) D$

if $\mathcal{M} A = \text{Some } D$ **for** $A D$
proof –
have $A \in \text{dom } \mathcal{M}$
using $\langle \mathcal{M} A = \text{Some } D \rangle$ **by** *blast*

hence $A \in \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C$
using *dom-M-eq* **by** *argo*

have $A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D$
using *invars* $\langle \mathcal{M} A = \text{Some } D \rangle$
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* $A \prec_i L$
using $\langle A \in \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \rangle$
by (*smt (verit, del-insts) UN-E*) $\langle A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D \rangle$
calculation dom-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-irrefl
linorder-lit.mulp-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 $D \prec_c \text{efac } C$
using $\langle \text{linorder-lit.is-greatest-in-mset } (\text{efac } C) L \rangle$
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.mulp_{HO}-if-maximal-less-that-maximal mulp-eq-mulp_{HO})

have *ord-res.production* $(\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D =$
ord-res.production $(\text{fset } (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}))) D$
proof (*rule ord-res.production-swap-clause-set*)
have $D \prec_c C$
using $\langle D \prec_c \text{efac } C \rangle \langle \text{efac } C \prec_c C \rangle$ **by** *order*
thus $\{Da. Da \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} Da D\} =$
 $\{Da. Da \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} Da D\}$
using $\langle D \prec_c \text{efac } C \rangle$
unfolding $\langle \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) = \text{finsert } (\text{iefac } \mathcal{F}' C) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \mid - \mid \{C\} \rangle$
by (*metis (no-types, opaque-lifting) finsertE finsertI2 fminus-iff fsingleton-iff*
iefac-def linorder-cls.leD)
qed

```

thus ?thesis
  using ⟨ $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) ) D}$ ⟩ by argo
qed

thus atoms-in-model-were-produced  $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some (efac } C))$ 
  unfolding atoms-in-model-were-produced-def by simp

have  $\mathcal{M} A = \text{Some } x$ 
  if  $x \prec_c \text{efac } C$  and  $A \in \text{ord-res.production (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) ) x}$ 
  for  $x A$ 
proof –
  have  $x \prec_c C$ 
    using ⟨ $x \prec_c \text{efac } C$ ⟩ ⟨ $\text{efac } C \prec_c C$ ⟩ by order

moreover have  $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) ) x}$ 
proof –
  have ord-res.production (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) ) x =
    ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) ) x
  proof (rule ord-res.production-swap-clause-set)
    show  $\{D. D \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} D x\} = \{D. D \in \mid \text{iefac}$ 
 $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} D x\}$ 
    using ⟨ $x \prec_c \text{efac } C$ ⟩ ⟨ $x \prec_c C$ ⟩
    unfolding ⟨ $\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) = \text{finsert (iefac } \mathcal{F}' C) (\text{iefac } \mathcal{F} \mid \uparrow$ 
 $(N \mid \cup \mid U_{er})) \mid - \mid \{C\}$ ⟩
    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 ⟨ $A \in \text{ord-res.production (fset (iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})) ) x}$ ⟩ 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  $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some (efac } C))$ 
  unfolding all-produced-atoms-in-model-def by simp
qed
next
case step-hyps: (resolution-bot } \mathcal{M} D L C U_{er}' U_{er} \mathcal{M}' \mathcal{F})
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 *step-hyps* **by** *blast*

moreover have *linorder-cls.is-least-in-fset (iefac $\mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}')$) \{\#\}*
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: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$*
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def implicitly-factorized-clauses-subset-def*
by *metis*

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 $\mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er})$) 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 $\mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er})$) E*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-filter-iff*
proof (*intro conjI ballI impI*)
show *E \in iefac $\mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er})$*
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 $\mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er})$)) E \models E*
using *invars*

unfolding *ord-res-5-invars-def step-hyps(1,2)*
using $\langle \neg \text{dom } \mathcal{M} \models E \rangle$ **by** (*metis model-eq-interp-upto-next-clause-def*)

moreover have *ord-res.production (fset (iefac \mathcal{F} | \uparrow (N \cup U_{er}))) E = \{\}*
proof –
have $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L E$
using $\langle \text{ord-res.is-maximal-lit } L E \rangle \langle \text{is-neg } L \rangle$
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 $\neg \text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})) E \models E$
by *simp*

next
fix F
assume $F\text{-in}: F \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})$ **and** $F \neq E$ **and**
 $F\text{-false}: \neg \text{ord-res-Interp (fset (iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})) F \models F$

have $\neg F \prec_c E$
using *invars*
unfolding *ord-res-5-invars-def step-hyps(1,2)*
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def*
using $F\text{-in } F\text{-false}$
by (*metis option.inject*)

thus $E \prec_c F$
using $\langle F \neq E \rangle$ **by** *order*
qed

have $L\text{-prod-by-D}: \text{atm-of } L \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}))$
 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 $D\text{-prod}: \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})) D \neq \{\}$
by (*metis empty-iff*)

have *ord-res.is-maximal-lit (-L) D*
using $L\text{-prod-by-D } L\text{-neg}$
by (*metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset literal.collapse(2) ord-res.mem-productionE uminus-Neg*)

moreover have $-L \prec_l L$
using $L\text{-neg}$
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) umi-*)

nus-not-id)

ultimately have $D \prec_c E$

using *L-max linorder-lit.multip-if-maximal-less-that-maximal* **by** *metis*

have $\text{eres } D E \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$

proof (*rule notI*)

assume $\text{eres } D E \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$

moreover have $\neg (E \prec_c \text{eres } D E)$

using $\langle \text{eres } D E \prec_c E \rangle$ **by** *order*

ultimately have $\text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) (\text{eres } D E) \models \text{eres } D E$

using *E-least-false* $\langle \text{eres } D E \neq E \rangle$

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 $\text{efac } (\text{eres } D E) \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$

proof (*rule notI*)

have $\text{efac } (\text{eres } D E) \preceq_c \text{eres } D E$

by (*meson efac-subset subset-implies-reflclp-multip*)

hence $\neg (E \prec_c \text{efac } (\text{eres } D E))$

using $\langle \text{eres } D E \prec_c E \rangle$ **by** *order*

moreover assume $\text{efac } (\text{eres } D E) \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$

moreover have $\text{efac } (\text{eres } D E) \neq E$

by (*metis* $\langle \text{eres } D E \prec_c E \rangle$ *efac-properties-if-not-ident(1) linorder-cls.not-less-iff-gr-or-eq*)

ultimately have $\text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) (\text{efac } (\text{eres } D E)) \models \text{efac } (\text{eres } D E)$

using *E-least-false*

unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*

by *metis*

hence $\text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) (\text{eres } D E) \models \text{eres } D E$

using $\langle \text{efac } (\text{eres } D E) \preceq_c \text{eres } D E \rangle$ *ord-res.entaile-clause-stays-entailed* **by**

fastforce

hence $\text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) E \models E$

using *clause-true-if-resolved-true*

by (*smt (verit) D-prod ex1-eres-eq-full-run-ground-resolution full-run-def rtranclpD*)

moreover have $\neg \text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) E \models E$

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 $eres\ D\ E \not\subseteq N \cup U_{er}$
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 $iefac\ \mathcal{F} \upharpoonright (N \cup U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F} \upharpoonright (N \cup U_{er}))$
unfolding $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er} \rangle$ **by** *simp*

show *?thesis*
unfolding *ord-res-5-invars-def step-hyps(1,2)*
proof (*intro conjI*)
have $eres\ D\ E \in iefac\ \mathcal{F} \upharpoonright (N \cup U_{er}')$
unfolding $\langle iefac\ \mathcal{F} \upharpoonright (N \cup U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F} \upharpoonright (N \cup U_{er})) \rangle$ **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 $\mathcal{F} \subseteq N \cup U_{er}'$
unfolding $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er} \rangle$
using $\langle \mathcal{F} \subseteq N \cup U_{er} \rangle$ **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-M-eq*: $dom\ \mathcal{M} = ord-res.interp\ (fset\ (iefac\ \mathcal{F} \upharpoonright (N \cup U_{er})))\ E$
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def model-eq-interp-upto-next-clause-def*
by *metis*

have $\forall x \in \# E. \neg dom\ \mathcal{M} \models x$
using $\langle \neg dom\ \mathcal{M} \models E \rangle$ **by** (*simp add: true-cls-def*)

moreover have $\forall x \in \# D. x \neq -L \longrightarrow \neg dom\ \mathcal{M} \models x$
proof –
have $\forall x \in \# D. x \neq -L \longrightarrow \neg ord-res.interp\ (fset\ (iefac\ \mathcal{F} \upharpoonright (N \cup U_{er})))\ D \models x$
using *L-prod-by-D* **by** (*metis ord-res.mem-productionE true-cls-def*)
moreover have $\forall x \in \# D. x \neq -L \longrightarrow atm-of\ x \prec_t atm-of\ (-L)$
using $\langle ord-res.is-maximal-lit\ (-L)\ D \rangle$ *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 $\forall x \in \# D. x \neq -L \longrightarrow \neg ord-res.interp\ (fset\ (iefac\ \mathcal{F} \upharpoonright (N \cup U_{er})))\ E \models x$
using *ord-res.interp-fixed-for-smaller-literals*
 $OF\ \langle ord-res.is-maximal-lit\ (-L)\ D \rangle - \langle D \prec_c E \rangle$
by *fastforce*

then show *?thesis*
unfolding *dom-M-eq[symmetric]* .
qed

moreover have $K \in\# \text{eres } D \ E$
using $\langle \text{ord-res.is-maximal-lit } K \ (\text{eres } D \ E) \rangle$
using *linorder-lit.is-maximal-in-mset-iff* **by** *metis*

moreover have $\forall x \in\# \text{eres } D \ E. x \in\# D \vee x \in\# E$
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have $\forall x \in\# \text{eres } D \ E. x \neq -L$
proof (*intro ballI notI*)
fix x **assume** $x \in\# \text{eres } D \ E \ x = -L$
obtain $m \ A \ D' \ E'$ **where**
ord-res.is-strictly-maximal-lit (*Pos A*) D **and**
 $D = \text{add-mset } (\text{Pos } A) \ D'$ **and**
 $E = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + E'$ **and**
 $\text{Neg } A \notin\# E'$ **and**
 $\text{eres } D \ E = \text{repeat-mset } (\text{Suc } m) \ D' + E'$
using $\langle \text{eres } D \ E \neq E' \rangle$ [*THEN eres-not-identD*] **by** *metis*

have $L = \text{Neg } A$
using $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ D \rangle$
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 $\langle x \in\# \text{eres } D \ E \rangle$
unfolding $\langle \text{eres } D \ E = \text{repeat-mset } (\text{Suc } m) \ D' + E' \rangle$
by (*metis member-mset-repeat-mset-Suc union-iff*)
thus *False*
proof (*elim disjE*)
assume $x \in\# D'$
hence $\text{Pos } A \in\# D'$
unfolding $\langle x = -L \rangle \langle L = \text{Neg } A \rangle$ **by** *simp*
hence $\neg \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ D$
using $\langle D = \text{add-mset } (\text{Pos } A) \ D' \rangle$
using *linorder-lit.is-greatest-in-mset-iff* **by** *auto*
thus *False*
using $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ D \rangle$ **by** *contradiction*

next
assume $x \in\# E'$
hence $\text{Pos } A \in\# E'$
unfolding $\langle x = -L \rangle \langle L = \text{Neg } A \rangle$ **by** *simp*
hence $\text{Pos } A \in\# E$
unfolding $\langle E = \text{replicate-mset } (\text{Suc } m) \ (\text{Neg } A) + E' \rangle$ **by** *simp*
hence *ord-res.production (fset (iefac \mathcal{F} | \uparrow (N | \cup | U_{er})))* $D \models_l \text{Pos } A$

```

    using L-prod-by-D ⟨L = Neg A⟩ by auto
  hence ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) E  $\models$  Pos A
    by (metis ⟨L = 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 (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) E  $\models$  E
    using ⟨Pos A  $\in$  # E⟩ by blast
  hence dom  $\mathcal{M}$   $\models$  E
    using dom- $\mathcal{M}$ -eq by argo
  thus False
    using ⟨ $\neg$  dom  $\mathcal{M}$   $\models$  E⟩ by contradiction
qed
qed

ultimately have  $\neg$  dom  $\mathcal{M}$   $\models$  K
  by metis

have dom  $\mathcal{M}' = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{| } U_{er}')))$  (eres D E)
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-of K
    unfolding ⟨ $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{ atm-of } K\}$ ⟩ by simp-all

  have A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) E
    using ⟨A  $\in$  dom  $\mathcal{M}$ ⟩
  unfolding ⟨dom  $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{| } U_{er}')))$  E⟩ .
  hence A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) (eres D E)
    using ord-res.interp-fixed-for-smaller-literals ⟨ord-res.is-maximal-lit K (eres
D E)⟩
    ⟨A  $\prec_t$  atm-of K⟩ ⟨eres D E  $\prec_c$  E⟩
  by metis
  thus A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}')))$  (eres D E)
  unfolding ⟨iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}') = \text{finsert (eres D E) (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{| } U_{er}))}$ ⟩
    using ord-res.interp-insert-greater-clause-strong by simp
next
  fix A :: 'f gterm
  assume A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}')))$  (eres D E)
  hence A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) (eres D E)
    unfolding ⟨iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}') = \text{finsert (eres D E) (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{| } U_{er}))}$ ⟩
    using ord-res.interp-insert-greater-clause-strong by simp
  hence A  $\in$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) E
    using ⟨eres D E  $\prec_c$  E⟩ ord-res.interp-subset-if-less-cls by blast
  hence A  $\in$  dom  $\mathcal{M}$ 
    unfolding ⟨dom  $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \text{ |} \uparrow \text{ (N |} \cup \text{| } U_{er})))$  E⟩ .

moreover have A  $\prec_t$  atm-of K
proof -

```

obtain C **where**

$C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**

$C \prec_c \text{eres } D \ E$ **and**

A -*prod-by-C*: $A \in \text{ord-res.production } (fset (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C$

using $\langle A \in \text{ord-res.interp } (fset (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) (\text{eres } D \ E) \rangle$

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 $A \preceq_t \text{atm-of } K$

using $\langle \text{ord-res.is-maximal-lit } K (\text{eres } D \ E) \rangle \langle C \prec_c \text{eres } D \ E \rangle$

by $(\text{metis } \text{linorder-cls.dual-order.asym } \text{linorder-lit.multip-if-maximal-less-that-maximal}$
 $\text{linorder-trm.not-le-imp-less } \text{literal.collapse}(1) \ \text{ord-res.less-lit-simps}(1)$
 $\text{step-hyps}(11))$

moreover have $A \neq \text{atm-of } K$

using $\langle A \in \text{dom } \mathcal{M} \rangle \langle \neg \text{dom } \mathcal{M} \models K \rangle \langle \text{is-pos } K \rangle$ **by** *force*

ultimately show *?thesis*

by *order*

qed

ultimately show $A \in \text{dom } \mathcal{M}'$

unfolding $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \preceq_t \text{atm-of } K\} \rangle$ **by** *simp*

qed

thus *model-eq-interp-upto-next-clause* $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } (\text{eres } D \ E))$

unfolding *model-eq-interp-upto-next-clause-def* **by** *simp*

have $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \text{eres } D \ E \longrightarrow$

ord-res-Interp $(\text{iefac } \mathcal{F} \mid \uparrow (fset \ N \cup \ fset \ U_{er}')) \ C \models C$

by $(\text{smt } (\text{verit}, \text{cefv-threshold}) \ \text{E-least-false } \text{Uniq-def } \text{Uniq-is-least-false-clause}$

$\langle \text{eres } D \ E \prec_c E \rangle \langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F}$

$\mid \uparrow (N \mid \cup \mid U_{er})) \rangle$

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 $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}'). C \prec_c \text{eres } D \ E \longrightarrow$

ord-res-Interp $(\text{iefac } \mathcal{F} \mid \uparrow (fset \ N \cup \ fset \ U_{er}')) \ C \models C$

unfolding $\langle \text{eres } D \ E \prec_c E \rangle \langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E)$

$(\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$

by *simp*

thus *all-smaller-clauses-true-wrt-respective-Interp* $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } (\text{eres } D \ E))$

$D E))$
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def by simp*
have $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) C$
if $\mathcal{M}' A = \text{Some } C$ **for** $A C$
proof –
have $\mathcal{M} A = \text{Some } C$ **and** $A \prec_t \text{atm-of } K$
unfolding *atomize-conj*
using $\langle \mathcal{M}' A = \text{Some } C \rangle \langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \rangle$
by (*metis Int-iff domI dom-restrict mem-Collect-eq restrict-in*)

hence *A-prod-by-C*: $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) C$
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def atoms-in-model-were-produced-def*
by *metis*

moreover have $\text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) C =$
 $\text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) C$
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 $\text{Pos } A \prec_l K$
using $\langle A \prec_t \text{atm-of } K \rangle$
by (*metis Pos-atm-of-iff ord-res.less-lit-simps(1) step-hyps(11)*)
ultimately have $C \prec_c \text{eres } D E$
using *linorder-lit.mulp-if-maximal-less-that-maximal step-hyps(10) by*
blast
thus $\{D. D \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} D C\} =$
 $\{D. D \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') \wedge (\prec_c)^{==} D C\}$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') = \text{fininsert (eres } D E) (\text{iefac } \mathcal{F} \mid \uparrow (N$
 $\mid \cup \mid U_{er})) \rangle$
by *auto*
qed

ultimately show *?thesis*
by *argo*
qed

thus *atoms-in-model-were-produced* $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some (eres } D E))$
unfolding *atoms-in-model-were-produced-def by simp*

have $\mathcal{M}' A = \text{Some } C$
if $C \prec_c \text{eres } D E$ **and** *A-in*: $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid$
 $U_{er}')) C$
for $C A$
proof –
have $C \prec_c E$
using $\langle C \prec_c \text{eres } D E \rangle \langle \text{eres } D E \prec_c E \rangle$ **by** *order*

moreover have A -prod-by- C : $A \in \text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) C$
proof –
have $\text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) C = \text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')) C$
proof (*rule ord-res.production-swap-clause-set*)
show $\{D. D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \wedge (\prec_c)^{==} D C\} = \{D. D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') \wedge (\prec_c)^{==} D C\}$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') = \text{finsert } (eres D E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \rangle$
using $\langle C \prec_c eres D E \rangle$
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 $\mathcal{M} A = \text{Some } C$
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def all-produced-atoms-in-model-def*
by *metis*

moreover have $A \prec_t \text{atm-of } K$
proof –
have $A \in \text{dom } \mathcal{M}$
using $\langle \mathcal{M} A = \text{Some } C \rangle$ **by** *auto*

have $\text{ord-res.is-maximal-lit } (Pos A) C$
using A -prod-by- C $\text{ord-res.mem-productionE}$ **by** *blast*

hence $A \preceq_t \text{atm-of } K$
using $\langle \text{ord-res.is-maximal-lit } K (eres D E) \rangle \langle C \prec_c eres D E \rangle$
by (*metis linorder-cls.dual-order.asym linorder-lit.multip-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 $A \neq \text{atm-of } K$
using $\langle A \in \text{dom } \mathcal{M} \rangle \langle \neg \text{dom } \mathcal{M} \models_l K \rangle \langle \text{is-pos } K \rangle$ **by** *force*

ultimately show *?thesis*
by *order*
qed

ultimately show $\mathcal{M}' A = \text{Some } C$
unfolding $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \rangle$ **by** *simp*
qed

thus *all-produced-atoms-in-model* $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } (eres D E))$

unfolding *all-produced-atoms-in-model-def* **by** *simp*
qed
next
case *step-hyps*: (*resolution-neg* \mathcal{M} E L D U_{er}' U_{er} \mathcal{M}' K C \mathcal{F})

obtain A_L **where** *L-def*: $L = \text{Neg } A_L$
using $\langle \text{is-neg } L \rangle$ **by** (*cases* L) *simp-all*

have $A_L \in \text{ord-res.production}$ (*fset* (*iefac* \mathcal{F} $|^\dagger$ ($N \cup U_{er}$))) D
using *invars* $\langle \mathcal{M} (\text{atm-of } L) = \text{Some } D \rangle$
unfolding *step-hyps*(1,2) *ord-res-5-invars-def* *atoms-in-model-were-produced-def*
unfolding *L-def* *literal.sel*
by *metis*

hence $D \models \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er})$ **and**
ord-res.is-strictly-maximal-lit (*Pos* A_L) D **and**
D-false: $\neg \text{ord-res.interp}$ (*fset* (*iefac* \mathcal{F} $|^\dagger$ ($N \cup U_{er}$))) $D \models D$
unfolding *atomize-conj* **by** (*metis* *ord-res.mem-productionE*)

obtain A_K **where** *K-def*: $K = \text{Neg } A_K$
using $\langle \text{is-neg } K \rangle$ **by** (*cases* K) *simp-all*

have $A_K \in \text{ord-res.production}$ (*fset* (*iefac* \mathcal{F} $|^\dagger$ ($N \cup U_{er}$))) C
using *invars* $\langle \mathcal{M} (\text{atm-of } K) = \text{Some } C \rangle$
unfolding *step-hyps*(1,2) *ord-res-5-invars-def* *atoms-in-model-were-produced-def*
unfolding *K-def* *literal.sel*
by *metis*

hence $C \models \text{iefac } \mathcal{F} \mid^\dagger (N \cup U_{er})$ **and**
ord-res.is-strictly-maximal-lit (*Pos* A_K) C **and**
C-false: $\neg \text{ord-res.interp}$ (*fset* (*iefac* \mathcal{F} $|^\dagger$ ($N \cup U_{er}$))) $C \models C$
unfolding *atomize-conj* **by** (*metis* *ord-res.mem-productionE*)

have $D \prec_c E$
using $\langle \text{ord-res.is-strictly-maximal-lit} (\text{Pos } A_L) D \rangle$ $\langle \text{ord-res.is-maximal-lit } L E \rangle$ _[unfolded *L-def*]
using *linorder-lit.multip-if-maximal-less-that-maximal* *ord-res.less-lit-simps*(2)
by *blast*

have *eres* $D E \neq E$
using $\langle \text{ord-res.is-strictly-maximal-lit} (\text{Pos } A_L) D \rangle$ $\langle \text{ord-res.is-maximal-lit } L E \rangle$ _[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.multip-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 $iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}))$
unfolding $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er} \rangle$ **by** *simp*

hence $\{|C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}')\ .\ C\ \prec_c\ eres\ D\ E|\} = \{|C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}')\ .\ C\ \prec_c\ eres\ D\ E|\}$
by *auto*

show *?thesis*
unfolding *step-hyps(1,2)* *ord-res-5-invars-def*
proof (*intro* *conjI*)
have $C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}')$
using $\langle C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}') \rangle$
unfolding $\langle iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}') = finsert\ (eres\ D\ E)\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er})) \rangle$
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 $\mathcal{F}\ |\subseteq|\ N\ |\cup|\ U_{er}$
using *invars*
unfolding *step-hyps(1,2)* *ord-res-5-invars-def* *implicitly-factorized-clauses-subset-def*
by *metis*

hence $\mathcal{F}\ |\subseteq|\ N\ |\cup|\ U_{er}'$
unfolding $\langle U_{er}' = finsert\ (eres\ D\ E)\ U_{er} \rangle$ **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 $Pos\ A_K \prec_l Neg\ A_K$
by *simp*

hence $C \prec_c eres\ D\ E$
using $\langle ord-res.is-strictly-maximal-lit\ (Pos\ A_K)\ C \rangle$
 $\langle ord-res.is-maximal-lit\ K\ (eres\ D\ E) \rangle$ [*unfolded K-def*]
using *linorder-lit.mulp-if-maximal-less-that-maximal* **by** *blast*

have $C \prec_c E$
using $\langle C \prec_c eres\ D\ E \rangle \langle eres\ D\ E \prec_c E \rangle$ **by** *order*

have $\{ |x| \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \cdot x \prec_c C \} = \{ |x| \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \cdot x \prec_c C \}$
using $\langle C \prec_c \text{eres } D \ E \rangle$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) = \text{finsert} (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$
by $(\text{metis } \text{ffilter-eq-ffilter-minus-singleton } \text{finsert-absorb } \text{fminus-finsert-absorb } \text{linorder-cls.less-asym})$

hence $\text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C =$
 $\text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C$
using $\langle C \prec_c \text{eres } D \ E \rangle$
by $(\text{simp add: } \langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) = \text{finsert} (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$
 $\text{ord-res.interp-insert-greater-clause})$

have $\text{dom-}\mathcal{M}\text{-eq: } \text{dom } \mathcal{M} = \text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ E$
using invars
unfolding $\text{step-hyps}(1,2) \ \text{ord-res-5-invars-def } \text{model-eq-interp-upto-next-clause-def}$
by metis

have $\forall x \in \# \ E. \neg \text{dom } \mathcal{M} \models_l x$
using $\langle \neg \text{dom } \mathcal{M} \models E \rangle$ **by** $(\text{simp add: } \text{true-cls-def})$

moreover have $\forall x \in \# \ D. x \neq -L \longrightarrow \neg \text{dom } \mathcal{M} \models_l x$
proof –
have $\forall x \in \# \ D. x \neq -L \longrightarrow \neg \text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))$
 $D \models_l x$
using $D\text{-false by blast}$
moreover have $\forall x \in \# \ D. x \neq -L \longrightarrow \text{atm-of } x \prec_t \text{atm-of } (-L)$
unfolding $L\text{-def } \text{uminus-Neg } \text{literal.sel}$
using $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A_L) \ D \rangle$
by $(\text{metis } \text{Pos-atm-of-iff } \langle A_L \in \text{ord-res.production} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ D \rangle$
 $\text{ord-res.less-trm-if-neg } \text{ord-res.lesseq-trm-if-pos } \text{reflclp-iff})$
ultimately have $\forall x \in \# \ D. x \neq -L \longrightarrow \neg \text{ord-res.interp} (\text{fset} (\text{iefac } \mathcal{F} \mid \uparrow$
 $(N \mid \cup \mid U_{er}))) \ E \models_l x$
using $\text{ord-res.interp-fixed-for-smaller-literals}$
using $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A_L) \ D \rangle \langle D \prec_c E \rangle$
using $L\text{-def by fastforce}$
thus $?thesis$
unfolding $\text{dom-}\mathcal{M}\text{-eq}[\text{symmetric}] .$
qed

moreover have $K \in \# \ \text{eres } D \ E$
using $\langle \text{ord-res.is-maximal-lit } K (\text{eres } D \ E) \rangle$
using $\text{linorder-lit.is-maximal-in-mset-iff by metis}$

moreover have $\forall x \in \# \ \text{eres } D \ E. x \in \# \ D \vee x \in \# \ E$

using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have $\forall x \in \# \text{ eres } D \ E. \ x \neq - L$

proof (*intro ballI notI*)

fix x **assume** $x \in \# \text{ eres } D \ E \ x = - L$

obtain $m \ A \ D' \ E'$ **where**

ord-res.is-strictly-maximal-lit (*Pos A*) D **and**

$D = \text{add-mset}$ (*Pos A*) D' **and**

$E = \text{replicate-mset}$ (*Suc m*) (*Neg A*) + E' **and**

$\text{Neg } A \notin \# \ E'$ **and**

$\text{eres } D \ E = \text{repeat-mset}$ (*Suc m*) $D' + E'$

using $\langle \text{eres } D \ E \neq E' \rangle$ [*THEN eres-not-identD*] **by** *metis*

have $L = \text{Neg } A$

using $\langle \text{ord-res.is-strictly-maximal-lit} \ (\text{Pos } A) \ D \rangle$

by (*metis L-def Uniq-D* $\langle \text{ord-res.is-strictly-maximal-lit} \ (\text{Pos } A_L) \ D \rangle$ *linorder-lit.Uniq-is-greatest-in-mset literal.inject(1)*)

have $x \in \# \ D' \vee x \in \# \ E'$

using $\langle x \in \# \ \text{eres } D \ E \rangle$

unfolding $\langle \text{eres } D \ E = \text{repeat-mset} \ (\text{Suc } m) \ D' + E' \rangle$

by (*metis member-mset-repeat-mset-Suc union-iff*)

thus *False*

proof (*elim disjE*)

assume $x \in \# \ D'$

hence $\text{Pos } A \in \# \ D'$

unfolding $\langle x = - L \rangle \langle L = \text{Neg } A \rangle$ **by** *simp*

hence $\neg \text{ord-res.is-strictly-maximal-lit} \ (\text{Pos } A) \ D$

using $\langle D = \text{add-mset} \ (\text{Pos } A) \ D' \rangle$

using *linorder-lit.is-greatest-in-mset-iff* **by** *auto*

thus *False*

using $\langle \text{ord-res.is-strictly-maximal-lit} \ (\text{Pos } A) \ D \rangle$ **by** *contradiction*

next

assume $x \in \# \ E'$

hence $\text{Pos } A \in \# \ E'$

unfolding $\langle x = - L \rangle \langle L = \text{Neg } A \rangle$ **by** *simp*

hence $\text{Pos } A \in \# \ E$

unfolding $\langle E = \text{replicate-mset} \ (\text{Suc } m) \ (\text{Neg } A) + E' \rangle$ **by** *simp*

hence *ord-res.production* (*fset* (*iefac* $\mathcal{F} \mid \uparrow (N \mid \cup U_{er})$)) $D \Vdash_l \text{Pos } A$

using *L-def* $\langle A_L \in \text{ord-res.production} \ (\text{fset} \ (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) \ D \rangle$

$\langle L = \text{Neg } A \rangle$

by *blast*

hence *ord-res.interp* (*fset* (*iefac* $\mathcal{F} \mid \uparrow (N \mid \cup U_{er})$)) $E \Vdash_l \text{Pos } A$

by (*metis* $\langle L = \text{Neg } A \rangle$ *dom-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* (*fset* (*iefac* $\mathcal{F} \mid \uparrow (N \mid \cup U_{er})$)) $E \Vdash E$

using $\langle \text{Pos } A \in \# \ E \rangle$ **by** *blast*

hence $\text{dom } \mathcal{M} \Vdash E$

using *dom-M-eq* **by** *argo*

```

    thus False
      using  $\langle \neg \text{dom } \mathcal{M} \models E \rangle$  by contradiction
    qed
  qed

ultimately have  $\neg \text{dom } \mathcal{M} \models K$ 
  by metis

have  $\text{dom } \mathcal{M}' = \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')))$  C
proof (intro subset-antisym subsetI)
  fix A :: 'f gterm
  assume  $A \in \text{dom } \mathcal{M}'$ 
  hence  $A \in \text{dom } \mathcal{M}$  and  $A \prec_t \text{atm-of } K$ 
    unfolding  $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \rangle$  by simp-all

  have  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  E
    using  $\langle A \in \text{dom } \mathcal{M} \rangle$ 
  unfolding  $\langle \text{dom } \mathcal{M} = \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  E  $\rangle$  .
  hence  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  C
    using ord-res.interp-fixed-for-smaller-literals  $\langle \text{ord-res.is-maximal-lit } K$  (eres
D E)  $\rangle$ 
     $\langle A \prec_t \text{atm-of } K \rangle$   $\langle C \prec_c E \rangle$ 
  by (smt (verit, del-insts) K-def
     $\langle A_K \in \text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  C  $\rangle$   $\langle \text{eres } D E$ 
 $\prec_c E \rangle$ 
    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  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')))$  C
  unfolding  $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') = \text{finsert } (\text{eres } D E) (iefac \mathcal{F} \mid \uparrow (N \mid \cup$ 
Uer'))  $\rangle$ 
    using ord-res.interp-insert-greater-clause-strong
    by (simp add:  $\langle C \prec_c \text{eres } D E \rangle$ )
  next
  fix A :: 'f gterm
  assume  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')))$  C
  hence  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  C
    unfolding  $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') = \text{finsert } (\text{eres } D E) (iefac \mathcal{F} \mid \uparrow (N \mid \cup$ 
Uer'))  $\rangle$ 
    using ord-res.interp-insert-greater-clause-strong by (simp add:  $\langle C \prec_c \text{eres}$ 
D E)  $\rangle$ 
  hence  $A \in \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  E
    using  $\langle C \prec_c E \rangle$  ord-res.interp-subset-if-less-cls by blast
  hence  $A \in \text{dom } \mathcal{M}$ 
    unfolding  $\langle \text{dom } \mathcal{M} = \text{ord-res.interp } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$  E  $\rangle$  .

moreover have  $A \prec_t \text{atm-of } K$ 
proof -
  obtain B where
     $B \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$  and

```

$B \prec_c C$ and
A-prod-by-B: $A \in \text{ord-res.production } (fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ B$
using $\langle A \in \text{ord-res.interp } (fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ C \rangle$
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 $A \preceq_t \text{atm-of } K$
using *ord-res.is-maximal-lit* $K \ (eres \ D \ E) \ \langle C \prec_c \text{eres } D \ E \rangle$
by $(metis \ K\text{-def} \ \langle B \prec_c C \rangle \ \text{asymptD}$
linorder-cls.less-trans *linorder-lit.multip-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 $A \neq \text{atm-of } K$
using $\langle A \in \text{dom } \mathcal{M} \rangle \ \langle \neg \text{dom } \mathcal{M} \models K \rangle$
unfolding *K-def*
by $(metis \ \langle A \in \text{ord-res.interp } (fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ C \rangle$
 $\langle A_K \in \text{ord-res.production } (fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ C \rangle \ \text{atm-of-uminus}$
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 $A \in \text{dom } \mathcal{M}'$
unfolding $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \ \{A. \ A \prec_t \text{atm-of } K\} \rangle$ **by** *simp*
qed

thus *model-eq-interp-upto-next-clause* $N \ (U_{er}', \ \mathcal{F}, \ \mathcal{M}', \ \text{Some } C)$
unfolding *model-eq-interp-upto-next-clause-def* **by** *simp*

have $\forall x \in |iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}). \ x \prec_c E \longrightarrow$
ord-res-Interp $(fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ x \models x$
using *invars*
unfolding *step-hyps(1,2)* *ord-res-5-invars-def* *all-smaller-clauses-true-wrt-respective-Interp-def*
by *simp*

hence $\forall x \in |iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}). \ x \prec_c C \longrightarrow$
ord-res-Interp $(fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ x \models x$
using $\langle C \prec_c E \rangle$ **by** *simp*

moreover have $\forall x \in |iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}). \ x \prec_c C \longrightarrow$
ord-res-Interp $(fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}))) \ x =$
ord-res-Interp $(fset \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}')) \ x$
unfolding $\langle iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er}') = \text{finsert } (eres \ D \ E) \ (iefac \ \mathcal{F} \ | \ \uparrow \ (N \ | \cup \ U_{er})) \rangle$

by (metis (no-types, lifting) $\langle C \prec_c \text{eres } D \ E \rangle$ finite-fset finsert.rep-eq
linorder.cls.dual-order.strict-trans2 ord-res.Interp-insert-greater-clause
sup2CI)

ultimately have $\forall x \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) . x \prec_c C \longrightarrow$
ord-res-Interp (iefac \mathcal{F} ' (fset $N \cup$ fset U_{er}')) $x \models x$
by simp

hence $\forall x \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') . x \prec_c C \longrightarrow$
ord-res-Interp (iefac \mathcal{F} ' (fset $N \cup$ fset U_{er}')) $x \models x$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$
using $\langle C \prec_c \text{eres } D \ E \rangle$ by auto

thus all-smaller-clauses-true-wrt-respective-Interp $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } C)$
unfolding all-smaller-clauses-true-wrt-respective-Interp-def
by simp

have $A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')))$ C
if $\mathcal{M}' A = \text{Some } C$ for $A \ C$

proof –

have $\mathcal{M} A = \text{Some } C$ and $A \prec_t \text{atm-of } K$
unfolding atomize-conj
using $\langle \mathcal{M}' A = \text{Some } C \rangle \langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \rangle$
by (metis Int-iff domI dom-restrict mem-Collect-eq restrict-in)

hence $A\text{-prod-by-}C: A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C$

using invars

unfolding step-hyps(1,2) ord-res-5-invars-def atoms-in-model-were-produced-def
by metis

moreover have $\text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ C =$
 $\text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')))$ C

proof (rule ord-res.production-swap-clause-set)

have $\text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) \ C$

using $A\text{-prod-by-}C$ ord-res.mem-productionE by metis

moreover have $\text{Pos } A \prec_t K$

using $\langle A \prec_t \text{atm-of } K \rangle$

by (simp add: K-def)

ultimately have $C \prec_c \text{eres } D \ E$

using linorder-lit.mulp-if-maximal-less-than-maximal step-hyps(10) by

blast

thus $\{D. D \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \wedge (\prec_c)^{==} D \ C\} =$

$\{D. D \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') \wedge (\prec_c)^{==} D \ C\}$

unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$

by auto

qed

ultimately show *?thesis*
by *argo*
qed

thus *atoms-in-model-were-produced* $N (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } C)$
unfolding *atoms-in-model-were-produced-def* **by** *simp*

have $\mathcal{M}' A = \text{Some } B$
if $B \prec_c C$ **and** *A-in*: $A \in \text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')))$
B

for $B A$
proof –
have $B \prec_c \text{eres } D E$
using $\langle B \prec_c C \rangle \langle C \prec_c \text{eres } D E \rangle$ **by** *order*
hence $B \prec_c E$
using $\langle \text{eres } D E \prec_c E \rangle$ **by** *order*

moreover have *A-prod-by-B*: $A \in \text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) B$
proof –
have $\text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) B =$
 $\text{ord-res.production } (fset (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}')) B$
proof (*rule ord-res.production-swap-clause-set*)
show $\{D. D \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \wedge (\prec_c)^{==} D B\} =$
 $\{D. D \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') \wedge (\prec_c)^{==} D B\}$
unfolding $\langle iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}') = finsert (\text{eres } D E) (iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \rangle$
using $\langle B \prec_c \text{eres } D E \rangle$
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 $\mathcal{M} A = \text{Some } B$
using *invars*
unfolding *step-hyps(1,2) ord-res-5-invars-def all-produced-atoms-in-model-def*
by *metis*

moreover have $A \prec_t \text{atm-of } K$
proof –
have $A \in \text{dom } \mathcal{M}$
using $\langle \mathcal{M} A = \text{Some } B \rangle$ **by** *auto*

have *ord-res.is-maximal-lit* $(\text{Pos } A) B$
using *A-prod-by-B ord-res.mem-productionE* **by** *blast*

hence $A \preceq_t \text{atm-of } K$

using $\langle \text{ord-res.is-maximal-lit } K \text{ (eres } D \ E) \rangle \langle B \prec_c \text{ eres } D \ E \rangle$
by $(\text{metis } K\text{-def asympD linorder-lit.multp-if-maximal-less-that-maximal}$
 $\text{linorder-trm.le-less-linear literal.sel(2) ord-res.asymp-less-cls}$
 $\text{ord-res.less-lit-simps(4)})$

moreover have $A \neq \text{atm-of } K$
using $\langle A \in \text{dom } \mathcal{M} \rangle \langle \neg \text{dom } \mathcal{M} \models_l K \rangle$
using $\langle \mathcal{M} \ A = \text{Some } B \rangle \text{step-hyps(12) that(1) by force}$

ultimately show $?thesis$
by order
qed

ultimately show $\mathcal{M}' \ A = \text{Some } B$
unfolding $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \ \{A. \ A \prec_t \text{atm-of } K\} \rangle \text{ by simp}$
qed

thus $\text{all-produced-atoms-in-model } N \ (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } C)$
unfolding $\text{all-produced-atoms-in-model-def}$ **by** simp
qed
qed

lemma $\text{rtranclp-ord-res-6-preserves-invars}$:
assumes $\text{steps: (ord-res-6 } N)^{**} \ s \ s'$ **and** $\text{invars: ord-res-5-invars } N \ s$
shows $\text{ord-res-5-invars } N \ s'$
using steps invars
by $(\text{induction } s \text{ rule: converse-rtranclp-induct}) \ (\text{auto intro: ord-res-6-preserves-invars})$

lemma $\text{ex-ord-res-6-if-not-final}$:
assumes
 $\text{not-final: } \neg \text{ord-res-6-final } S$ **and**
 $\text{invars: } \forall N \ s. \ S = (N, s) \longrightarrow \text{ord-res-5-invars } N \ s$
shows $\exists S'. \ \text{ord-res-6-step } S \ S'$

proof –
from not-final **obtain** $N \ U_{er} \ \mathcal{F} \ \mathcal{M} \ C$ **where**
 $S\text{-def: } S = (N, (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C))$ **and** $C \neq \{\#\}$
unfolding $\text{ord-res-6-final.simps de-Morgan-disj not-ex}$
by $(\text{metis option.exhaust surj-pair})$

note $\text{invars}' = \text{invars}[\text{unfolded ord-res-5-invars-def, rule-format, OF } S\text{-def}]$

have $\exists s'. \ \text{ord-res-6 } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \ s'$

proof $(\text{cases } \text{dom } \mathcal{M} \models C)$

case True

thus $?thesis$

using ord-res-6.skip **by** metis

next

case $C\text{-false: False}$

obtain L **where** $L\text{-max: linorder-lit.is-maximal-in-mset } C \ L$

```

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-strictly-max: False
    thus ?thesis
    using ord-res-6.factoring[OF C-false L-max L-pos L-not-strictly-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  $\mathcal{F}$  | $\uparrow$  (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  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ )
    using invars' by (metis next-clause-in-factorized-clause-def)
  next
    have  $\neg$  ord-res.interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (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  $\mathcal{F}$  | $\uparrow$  (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  $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) C  $\models$  C
  by simp
next
  fix D
  assume D-in: D | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ) and D  $\neq$  C and
    C-false:  $\neg$  ord-res.Interp (fset (iefac  $\mathcal{F}$  | $\uparrow$  (N | $\cup$ |  $U_{er}$ ))) D  $\models$  D
  have  $\neg$  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 $\langle D \neq C \rangle$ **by** *order*
qed
then obtain D **where** $D \in |\text{iefac } \mathcal{F} \mid^\dagger (N \mid \cup \mid U_{er})$ **and**
 $D \prec_c C$ **and**
ord-res.is-strictly-maximal-lit (Pos A) D and
D-prod: ord-res.production (fset (iefac $\mathcal{F} \mid^\dagger (N \mid \cup \mid U_{er}))$) $D = \{A\}$
using *bx-smaller-productive-clause-if-least-false-clause-has-negative-max-lit*
L-max[unfolded Neg] by metis

hence $\exists DC$. *ground-resolution D C DC*
unfolding *ground-resolution-def*
using *L-max Neg ex-ground-resolutionI by blast*

hence *eres D C \neq C*
unfolding *eres-ident-iff by metis*

hence *eres D C \prec_c C*
using *eres-le[of D C] by order*

have \mathcal{M} (*atm-of L*) = *Some D*
using *invars'*
by (*metis Neg $\langle D \prec_c C \rangle$ 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 $\langle \mathcal{M}$ (atm-of L) =*
Some D \rangle] 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 $\langle \mathcal{M}$ (atm-of L) = Some D \rangle \langle eres D C \neq {# \rangle*
K-max by metis
next
case *K-def: (Neg A_K)*
hence *K-neg: is-neg K*

by *simp*

have $\neg \text{ord-res-Interp (fset (finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))}$
 $(eres D C) \Vdash eres D C$

by (*smt (verit) C-least-false D-prod Interp-insert-unproductive K-max*
K-neg Uniq-D
 $\langle eres D C \neq C \rangle$ *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*:
 $\text{is-least-false-clause (finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) (eres$
 $D C)$

using *C-least-false* $\langle eres D C \prec_c C \rangle$
by (*metis is-least-false-clause-finsert-smaller-false-clause*)

then obtain E **where** $E \mid \in \mid \text{finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))}$
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} \mid \uparrow (N \mid \cup \mid
 $U_{er})))) E = \{A_K\}$

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} \mid \uparrow (N \mid \cup \mid U_{er})) E =*
ord-res.production (fset (finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})))) E
proof (*rule ord-res.production-swap-clause-set*)

have $eres D C \notin \mid \text{iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})}$
proof (*rule notI*)

assume $eres D C \mid \in \mid \text{iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})}$
hence $\text{finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))} = \text{iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid$
 $U_{er})$

by *blast*
hence *is-least-false-clause (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) (eres D C)*
using *eres-least by argo*
thus *False*
using *C-least-false* $\langle eres D C \neq C \rangle$
by (*metis Uniq-D Uniq-is-least-false-clause*)

qed

thus $\{D. D \mid \in \mid \text{iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})} \wedge (\prec_c)^{==} D E\} =$
 $\{Da. Da \mid \in \mid \text{finsert (eres D C) (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))} \wedge (\prec_c)^{==} Da$
 $E\}$

by (*metis (mono-tags, lifting) \langle E \prec_c eres D C \rangle finsert-iff linorder-cls.leD*)
qed

```

also have ... = {AK}
  using E-prod .

finally have ord-res.production (fset (iefac  $\mathcal{F}$  |† (N | $\cup$ | Uer))) E = {AK}
.

hence  $\mathcal{M}$  (atm-of K) = Some E
  using invars'
  unfolding ord-res-5-invars-def all-produced-atoms-in-model-def
by (metis K-def  $\langle E \prec_c \text{eres } D \ C \rangle$  eres-le insertII linorder-cls.dual-order.strict-trans1
  literal.sel(2))

thus ?thesis
  using ord-res-6.resolution-neg
  using C-false L-max L-neg  $\langle \mathcal{M}$  (atm-of L) = Some D  $\rangle$   $\langle \text{eres } D \ C \neq \{\#\} \rangle$ 
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  $\neg$  ord-res-6-final S'  $\implies \exists S''$ . ord-res-6-step S' S''
  using ex-ord-res-6-if-not-final[of S']  $\langle S' = (N, s') \rangle$  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:  $\mathcal{F} \subseteq N \cup U_{er}$  and
  C-least: linorder-cls.is-least-in-fset (iefac  $\mathcal{F} \uparrow (N \cup U_{er})$ ) C and
  D-in:  $D \in \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$  and
  D-lt-least-false:  $\forall E. \text{is-least-false-clause } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) E \longrightarrow D \preceq_c E$ 
and
   $C \preceq_c D$ 
shows  $\exists \mathcal{M}. (\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \text{Map.empty}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D)$ 
using ord-res.wfP-less-cls D-in  $\langle C \preceq_c D \rangle$  D-lt-least-false
proof (induction D rule: wfp-induct-rule)
case (less D)

have invars: ord-res-5-invars N (Uer,  $\mathcal{F}$ , 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 =  $\{ | C \in \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D | \}$ 

show ?case
proof (cases clauses-lt-D =  $\{ | \}$ )
case True
hence C = D
  unfolding clauses-lt-D-def
  using C-least  $\langle C \preceq_c D \rangle$ 
  by (metis fempty-iff fmember-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  $x \in \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
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 fmember-filter linorder-cls.is-greatest-in-fset-iff
    linorder-cls.not-less nbx-less-than-least-in-fset)

```

moreover have $\bigwedge E. \text{ is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) E \implies x \prec_c$
E
using $\langle x \prec_c D \rangle \text{ less.premis by force}$

ultimately obtain \mathcal{M} **where**
IH: $(\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \text{Map.empty}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } x)$
using *less.IH by blast*

moreover have $\exists \mathcal{M}'. \text{ ord-res-5 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } x) (U_{er}, \mathcal{F}, \mathcal{M}', \text{Some } D)$
D)
proof –

have *linorder-clis.is-least-in-fset* $(\text{ffilter } ((\prec_c) x) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) D$
using *x-greatest[unfolded clauses-lt-D-def]*
by $(\text{smt } (\text{verit}) \text{ ffilter-filter less.premis } (1) \text{ linorder-clis.is-greatest-in-fset-iff } \text{ linorder-clis.is-least-in-ffilter-iff } \text{ linorder-clis.not-less-iff-gr-or-eq})$
hence *next-clause-eq: The-optional* $(\text{linorder-clis.is-least-in-fset } (\text{ffilter } ((\prec_c) x) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))) = \text{Some } D$
by $(\text{metis } \text{linorder-clis.Uniq-is-least-in-fset } \text{The-optional-eq-SomeI})$

have *x-true: ord-res-Interp* $(\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) x \models x$
using *less.premis*
unfolding *is-least-false-clause-def linorder-clis.is-least-in-ffilter-iff*

by $(\text{metis } \langle \bigwedge E. \text{ is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) E \implies x \prec_c$
E $\rangle \langle x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$
ex-false-clause-iff finsert-absorb is-least-false-clause-finsert-smaller-false-clause
linorder-clis.order.irrefl ex-false-clause-def)

show *?thesis*
proof $(\text{cases } \text{dom } \mathcal{M} \models x)$
case *True*
thus *?thesis*
using *ord-res-5.skip next-clause-eq by metis*

next
case *False*
hence $\neg \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) x \models x$
using *rtranclp-ord-res-5-preserves-invars[OF IH invars, unfolded ord-res-5-invars-def, simplified]*
by $(\text{simp add: model-eq-interp-upto-next-clause-def})$

thus *?thesis*
using *ord-res-5.production[OF False] next-clause-eq*
using *x-true*
by $(\text{metis } \text{Un-empty-right linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset } \text{unproductive-if-nex-strictly-maximal-pos-lit})$

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 $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } E) (U_{er}', \mathcal{F}', \mathcal{M}', \text{Some } D)$ **and**

invars: *ord-res-5-invars* $N (U_{er}', \mathcal{F}', \mathcal{M}', \text{Some } D)$ **and**

M'-def: $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. \exists K. \text{linorder-lit.is-maximal-in-mset } D \ K \ \wedge$
 $A \prec_t \text{atm-of } K\}$ **and**

C-least: *linorder-clc.is-least-in-fset* (*iefac* $\mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')$) C

shows (*ord-res-5* N)** ($U_{er}', \mathcal{F}', \text{Map.empty}, \text{Some } C$) ($U_{er}', \mathcal{F}', \mathcal{M}', \text{Some } D$)

proof –

have *D-in*: $D \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')$

using *invars*

by (*metis next-clause-in-factorized-clause-def ord-res-5-invars-def*)

have $\mathcal{F}' \mid \subseteq \mid N \mid \cup \mid U_{er}'$

using *invars*

by (*metis implicitly-factorized-clauses-subset-def ord-res-5-invars-def*)

moreover have $C \preceq_c D$

using *C-least D-in*

by (*metis linorder-clc.dual-order.strict-iff-order linorder-clc.is-least-in-fset-iff*
linorder-clc.le-cases)

moreover have $\forall F. \text{is-least-false-clause } (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')) \ F \longrightarrow D \preceq_c$
 F

using *invars le-least-false-clause by metis*

ultimately obtain \mathcal{M}'' **where**

steps: (*ord-res-5* N)** ($U_{er}', \mathcal{F}', \text{Map.empty}, \text{Some } C$) ($U_{er}', \mathcal{F}', \mathcal{M}'', \text{Some } D$)

using *C-least D-in*

by (*metis ex-model-build-from-least-clause-to-any-less-than-least-false*)

have *ord-res-5-invars* $N (U_{er}', \mathcal{F}', \text{Map.empty}, \text{Some } C)$

using $\langle \mathcal{F}' \mid \subseteq \mid N \mid \cup \mid U_{er}' \rangle$ *C-least ord-res-5-invars-initial-state* **by** *metis*

hence *ord-res-5-invars* $N (U_{er}', \mathcal{F}', \mathcal{M}'', \text{Some } D)$

using $\langle (\text{ord-res-5 } N)** (U_{er}', \mathcal{F}', \lambda x. \text{None}, \text{Some } C) (U_{er}', \mathcal{F}', \mathcal{M}'', \text{Some } D) \rangle$

rtranclp-ord-res-5-preserves-invars **by** *metis*

hence \mathcal{M}'' -*spec*:

dom $\mathcal{M}'' = \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')))$ D

$\forall A \ C. \mathcal{M}'' \ A = \text{Some } C \longrightarrow A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')))$ C

$\forall C \ A. C \prec_c D \longrightarrow A \in \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}')))$ C


```

→ 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 M' = M''
proof (cases D = {#})
  case True
  have M' = Map.empty
  proof -
    have  $\nexists K. \text{ord-res.is-maximal-lit } K D \wedge A \prec_t \text{atm-of } K$  for A
    unfolding  $\langle D = \{ \# \} \rangle$ 
    by (simp add: linorder-lit.is-maximal-in-mset-iff)
    hence  $\{ A. \exists K. \text{ord-res.is-maximal-lit } K D \wedge A \prec_t \text{atm-of } K \} = \{ \}$ 
    by simp
    thus ?thesis
    unfolding M'-def by auto
  qed

also have Map.empty = M''
proof -
  have ord-res.interp (fset (iefac F' | $\uparrow$  (N  $\cup$  Uer'))) D = {#}
  unfolding  $\langle D = \{ \# \} \rangle$  by simp
  thus ?thesis
  using 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  $\{ A. \exists K. \text{ord-res.is-maximal-lit } K D \wedge A \prec_t \text{atm-of } K \} = \{ A. A \prec_t$ 
atm-of K  $\}$ 
  by (metis (no-types, lifting) linorder-lit.Uniq-is-maximal-in-mset the1-equality')
  hence M'-def': M' = restrict-map M {A. A  $\prec_t$  atm-of K}
  unfolding M'-def by argo

show ?thesis
proof (intro ext)
  fix x
  have M'-spec:
    dom M' = ord-res.interp (fset (iefac F' | $\uparrow$  (N  $\cup$  Uer'))) D
     $\forall A C. M' A = \text{Some } C \longrightarrow A \in \text{ord-res.production (fset (iefac F' | $\uparrow$  (N$ 
 $\cup$  Uer'))) } C
     $\forall C A. C \prec_c D \longrightarrow A \in \text{ord-res.production (fset (iefac F' | $\uparrow$  (N  $\cup$  Uer')))$ 
C  $\longrightarrow M' A = \text{Some } C$ 
  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 M' = dom M''
  using M'-spec(1) M''-spec(1) by argo

moreover have  $\forall A C. M' A = \text{Some } C \longleftrightarrow M'' A = \text{Some } C$ 
  using M'-spec(2) 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  $M' x = M'' x$ 
  by (metis domD domIff)
qed
qed

thus ?thesis
  using  $\langle (ord-res-5 N)^{**} (U_{er}', \mathcal{F}', \text{Map.empty}, \text{Some } C) (U_{er}', \mathcal{F}', M'', \text{Some } D) \rangle$  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:

```

```

   $\neg \text{trail-false-cls } \Gamma C \implies$ 

```

```

   $\text{linorder-lit.is-maximal-in-mset } C L \implies$ 

```

```

   $\text{linorder-trm.is-least-in-fset } \{|A| \in | \text{atms-of-cls } (N \cup U_{er})\}.$ 

```

```

   $A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma \implies$ 

```

```

   $\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \implies$ 

```

```

   $\text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', \text{Some } C) |$ 

```

skip-defined:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg(\exists A | \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A | \notin | \text{trail-atms } \Gamma) \implies \\
& \text{trail-defined-lit } \Gamma L \implies \\
& C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D | \in | \text{iefac } \mathcal{F} | \uparrow (N \mid \cup \mid U_{er}). \\
C \prec_c D\}) \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma, C') |
\end{aligned}$$

skip-undefined-neg:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg(\exists A | \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A | \notin | \text{trail-atms } \Gamma) \implies \\
& \neg \text{trail-defined-lit } \Gamma L \implies \\
& \text{is-neg } L \implies \\
& \Gamma' = (L, \text{None}) \# \Gamma \implies \\
& C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D | \in | \text{iefac } \mathcal{F} | \uparrow (N \mid \cup \mid U_{er}). \\
C \prec_c D\}) \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', C') |
\end{aligned}$$

skip-undefined-pos:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg(\exists A | \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A | \notin | \text{trail-atms } \Gamma) \implies \\
& \neg \text{trail-defined-lit } \Gamma L \implies \\
& \text{is-pos } L \implies \\
& \neg \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq L\# \} \implies \\
& \text{linorder-cls.is-least-in-fset } \{|D | \in | \text{iefac } \mathcal{F} | \uparrow (N \mid \cup \mid U_{er}). C \prec_c D\} D \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) |
\end{aligned}$$

skip-undefined-pos-ultimate:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg(\exists A | \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A | \notin | \text{trail-atms } \Gamma) \implies \\
& \neg \text{trail-defined-lit } \Gamma L \implies \\
& \text{is-pos } L \implies \\
& \neg \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq L\# \} \implies \\
& \Gamma' = (-L, \text{None}) \# \Gamma \implies \\
& \neg(\exists D | \in | \text{iefac } \mathcal{F} | \uparrow (N \mid \cup \mid U_{er}). C \prec_c D) \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', \text{None}) |
\end{aligned}$$

production:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg(\exists A | \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A | \notin | \text{trail-atms } \Gamma) \implies \\
& \neg \text{trail-defined-lit } \Gamma L \implies \\
& \text{is-pos } L \implies \\
& \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq L\# \} \implies \\
& \text{linorder-lit.is-greatest-in-mset } C L \implies
\end{aligned}$$

$$\begin{aligned}
& \Gamma' = (L, \text{Some } C) \# \Gamma \implies \\
& C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D| \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})\}. \\
C \prec_c D \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', C') \mid
\end{aligned}$$

factoring:

$$\begin{aligned}
& \neg \text{trail-false-cls } \Gamma C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \neg (\exists A \mid \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma) \implies \\
& \neg \text{trail-defined-lit } \Gamma L \implies \\
& \text{is-pos } L \implies \\
& \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq L\# \} \implies \\
& \neg \text{linorder-lit.is-greatest-in-mset } C L \implies \\
& \mathcal{F}' = \text{finsert } C \mathcal{F} \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}', \Gamma, \text{Some } (\text{efac } C)) \mid
\end{aligned}$$

resolution-bot:

$$\begin{aligned}
& \text{trail-false-cls } \Gamma E \implies \\
& \text{linorder-lit.is-maximal-in-mset } E L \implies \\
& \text{is-neg } L \implies \\
& \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \implies \\
& U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \implies \\
& \text{eres } D E = \{\#\} \implies \\
& \Gamma' = [] \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) (U_{er}', \mathcal{F}, \Gamma', \text{Some } \{\#\}) \mid
\end{aligned}$$

resolution-pos:

$$\begin{aligned}
& \text{trail-false-cls } \Gamma E \implies \\
& \text{linorder-lit.is-maximal-in-mset } E L \implies \\
& \text{is-neg } L \implies \\
& \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \implies \\
& U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \implies \\
& \text{eres } D E \neq \{\#\} \implies \\
& \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \implies \\
& \text{linorder-lit.is-maximal-in-mset } (\text{eres } D E) K \implies \\
& \text{is-pos } K \implies \\
& \text{ord-res-}\gamma N (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) (U_{er}', \mathcal{F}, \Gamma', \text{Some } (\text{eres } D E)) \mid
\end{aligned}$$

resolution-neg:

$$\begin{aligned}
& \text{trail-false-cls } \Gamma E \implies \\
& \text{linorder-lit.is-maximal-in-mset } E L \implies \\
& \text{is-neg } L \implies \\
& \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \implies \\
& U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \implies \\
& \text{eres } D E \neq \{\#\} \implies \\
& \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \implies \\
& \text{linorder-lit.is-maximal-in-mset } (\text{eres } D E) K \implies \\
& \text{is-neg } K \implies \\
& \text{map-of } \Gamma (- K) = \text{Some } (\text{Some } C) \implies
\end{aligned}$$

ord-res-7 $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) (U_{er}', \mathcal{F}, \Gamma', \text{Some } C)$

```

lemma right-unique-ord-res-7:
  fixes  $N :: 'f \text{ gclause } fset$ 
  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  $\Gamma \ C \ L \ U_{er} \ A \ \Gamma' \ \mathcal{F}$ )
      show ?thesis
        using step2
        unfolding step-hyps1(1)
  proof (cases  $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) \ z$  rule: ord-res-7.cases)
    case step-hyps2: (decide-neg  $L2 \ A2 \ \Gamma2'$ )
      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  $\langle L2 = L \rangle$ 
        using linorder-trm.Uniq-is-least-in-fset[THEN Uniq-D] by presburger
      have  $\Gamma2' = \Gamma'$ 
        using step-hyps1 step-hyps2
        unfolding  $\langle L2 = L \rangle \langle A2 = A \rangle$  by metis
      show ?thesis
        unfolding step-hyps1(2) step-hyps2(1)
        unfolding  $\langle \Gamma2' = \Gamma' \rangle ..$ 
  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  $\langle L2 = L \rangle$ 
        unfolding linorder-trm.is-least-in-filter-iff by metis
      then show ?thesis ..
  next
    case step-hyps2: (skip-undefined-neg  $L2 \ \Gamma2' \ C2'$ )
      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  $\langle L2 = L \rangle$ 

```

```

    unfolding linorder-trm.is-least-in-filter-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-filter-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-filter-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-filter-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-filter-iff by metis
then show ?thesis ..
next
case step-hyps2: (resolution-bot L2 D2 Uer2′ Γ2′)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2′ Γ2′ K2)

```

```

    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
  next
    case step-hyps2: (resolution-neg L2 D2 Uer2' 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 Uer C' F)
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases N (Uer, 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-filter-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-filter-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 Uer2′ Γ2′)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2′ Γ2′ K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2′ Γ2′ K2 C2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
qed

```



```

next
case step-hyps1: (skip-undefined-neg  $\Gamma$   $C$   $L$   $U_{er}$   $\Gamma'$   $C'$   $\mathcal{F}$ )
show ?thesis
  using step2 unfolding step-hyps1(1)
proof (cases  $N$  ( $U_{er}$ ,  $\mathcal{F}$ ,  $\Gamma$ , Some  $C$ )  $z$  rule: ord-res-7.cases)
case step-hyps2: (decide-neg  $L2$   $A2$   $\Gamma2'$ )
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  $\langle L2 = L \rangle$ 
  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  $\langle L2 = L \rangle$  by argo
then show ?thesis ..
next
case step-hyps2: (skip-undefined-neg  $L2$   $\Gamma2'$   $C2'$ )
have  $L2 = L$ 
  using step-hyps1 step-hyps2
  using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
have  $\Gamma2' = \Gamma'$ 
  using step-hyps1 step-hyps2
  unfolding  $\langle L2 = L \rangle$  by argo
have  $C2' = C'$ 
  using step-hyps1 step-hyps2 by argo
show ?thesis
  unfolding step-hyps1(2) step-hyps2(1)
  unfolding  $\langle C2' = C' \rangle$   $\langle \Gamma2' = \Gamma' \rangle$  ..
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  $\langle L2 = L \rangle$ 
  unfolding linorder-trm.is-least-in-ffilter-iff by metis
then show ?thesis ..
next
case step-hyps2: (skip-undefined-pos-ultimate  $L2$   $\Gamma2'$ )
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-filter-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 Uer2' Γ2')
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2' Γ2' K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' 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 Uer F D)
show ?thesis
  using step2 unfolding step-hyps1(1)
proof (cases N (Uer, 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 Uer2' Γ2')
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2' Γ2' K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' 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 Uer Γ' F)
show ?thesis
  using step2 unfolding step-hyps1(1)
proof (cases N (Uer, 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  $\Gamma 2'$  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  $\langle L2 = L \rangle$  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  $\Gamma 2'$ )
have L2 = L
  using step-hyps1 step-hyps2
  using linorder-lit.Uniq-is-maximal-in-mset[THEN Uniq-D] by metis
have  $\Gamma 2' = \Gamma'$ 
  using step-hyps1 step-hyps2
  unfolding  $\langle L2 = L \rangle$  by argo
show ?thesis
  unfolding step-hyps1(2) step-hyps2(1)
  unfolding  $\langle \Gamma 2' = \Gamma' \rangle$  ..
next
case step-hyps2: (production L2  $\Gamma 2'$  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  $\langle L2 = L \rangle$  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  $\langle L2 = L \rangle$  by argo
then show ?thesis ..
next
case step-hyps2: (resolution-bot L2 D2  $U_{er} 2' \Gamma 2'$ )
have False
  using step-hyps1 step-hyps2 by argo

```

```

    then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2' Γ2' K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' Γ2' K2 C2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
qed
next
case step-hyps1: (production Γ C L Uer Γ' C' F)
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases N (Uer, 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 Uer2′ Γ2′)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2′ Γ2′ K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2′ Γ2′ K2 C2)

```

```

    have False
      using step-hyps1 step-hyps2 by argo
    then show ?thesis ..
qed
next
case step-hyps1: (factoring  $\Gamma$  C L  $U_{er}$   $\mathcal{F}'$   $\mathcal{F}$ )
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases N ( $U_{er}$ ,  $\mathcal{F}$ ,  $\Gamma$ , Some C) z rule: ord-res-7.cases)
case step-hyps2: (decide-neg L2 A2  $\Gamma 2'$ )
  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  $\langle L2 = L \rangle$ 
    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  $\langle L2 = L \rangle$  by argo
  then show ?thesis ..
next
case step-hyps2: (skip-undefined-neg L2  $\Gamma 2'$  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  $\langle L2 = L \rangle$  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  $\langle L2 = L \rangle$  by argo
  then show ?thesis ..
next
case step-hyps2: (skip-undefined-pos-ultimate L2  $\Gamma 2'$ )
  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-filter-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 Uer2' Γ2')
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-pos L2 D2 Uer2' Γ2' K2)
have False
  using step-hyps1 step-hyps2 by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' 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 Uer' Uer Γ' F)
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases N (Uer, 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 Uer2' Γ2')

```

```

have  $U_{er}2' = U_{er}'$ 
  using step-hyps1 step-hyps2
  by argo
have  $\Gamma 2' = \Gamma'$ 
  using step-hyps1 step-hyps2
  by argo
show ?thesis
  unfolding step-hyps1(2) step-hyps2(1)
  unfolding  $\langle U_{er}2' = U_{er}' \rangle \langle \Gamma 2' = \Gamma' \rangle ..$ 
next
case step-hyps2: (resolution-pos L2 D2 Uer2' 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  $\langle L2 = L \rangle$ 
  by (metis option.inject)
have False
  using step-hyps1 step-hyps2
  unfolding  $\langle D2 = D \rangle$ 
  by argo
then show ?thesis ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' 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  $\langle L2 = L \rangle$ 
  by (metis option.inject)
have False
  using step-hyps1 step-hyps2
  unfolding  $\langle D2 = D \rangle$ 
  by argo
then show ?thesis ..
qed
next
case step-hyps1: (resolution-pos Γ E L D Uer' Uer Γ' K F)
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases  $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } E)$  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 Uer2' 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 Uer2' 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 Uer2' = Uer'
    using step-hyps1 step-hyps2
    unfolding ⟨D2 = D⟩
    by argo
  have Γ2' = Γ'
    using step-hyps1 step-hyps2
    unfolding ⟨K2 = K⟩
    by argo
  show ?thesis
    unfolding step-hyps1(2) step-hyps2(1)
    unfolding ⟨Uer2' = Uer'⟩ ⟨Γ2' = Γ'⟩ ⟨D2 = D⟩ ..
next
case step-hyps2: (resolution-neg L2 D2 Uer2' 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 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 Uer' Uer Γ' K C F)
show ?thesis
  using step2
  unfolding step-hyps1(1)
proof (cases N (Uer, 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 Uer2' 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 Uer2' 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 Uer2' 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 Uer2' = Uer'
    using step-hyps1 step-hyps2
    unfolding ⟨D2 = D⟩
    by argo
  have Γ2' = Γ'
    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 ⟨Uer2' = Uer'⟩ ⟨Γ2' = Γ'⟩ ⟨C2 = C⟩ ..
qed
qed
qed

```

inductive *ord-res-7-final* **where**

model-found:

ord-res-7-final ($N, U_{er}, \mathcal{F}, \Gamma, None$) |

contradiction-found:

ord-res-7-final ($N, U_{er}, \mathcal{F}, \Gamma, Some \{\#\}$)

sublocale *ord-res-7-antics: semantics* **where**

step = *constant-context ord-res-7* **and**

final = *ord-res-7-final*

proof *unfold-locals*

fix $S :: 'f\ gclause\ fset \times 'f\ gclause\ fset \times 'f\ gclause\ fset \times$
 $('f\ gterm\ literal \times 'f\ gterm\ literal\ multiset\ option)\ list \times$
 $'f\ gterm\ literal\ multiset\ option$

obtain

$N\ U_{er}\ \mathcal{F} :: 'f\ gterm\ clause\ fset$ **and**

$\Gamma :: ('f\ gterm\ literal \times 'f\ gterm\ literal\ multiset\ option)\ list$ **and**

$\mathcal{C} :: 'f\ gclause\ option$

where
S-def: $S = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$
by (*metis prod.exhaust*)

assume *ord-res-7-final S*

hence $\nexists x. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) x$
unfolding *S-def*
proof (*cases (N, U_{er}, F, Γ, 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}, F, Γ, C) if
 $\mathcal{F} \mid\subseteq N \mid\cup U_{er}$ **and**
 $(\forall C. \mathcal{C} = \text{Some } C \longrightarrow C \mid\in \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er}))$ **and**
 $(\forall D. \mathcal{C} = \text{Some } D \longrightarrow$
 $(\forall C \mid\in \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in\# C. K \neq L_C\#})$
 $\wedge \text{is-pos } L_C)))$ **and**
 $(\forall C \mid\in \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $\text{trail-true-cls } \Gamma C)$ **and**
 $(\forall C \mid\in \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \mid\in \text{atms-of-clss } (N \mid\cup U_{er}). A \prec_t \text{atm-of } K \wedge A \not\in \text{trail-atms}$
 $\Gamma)))$ **and**
 $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
 $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid\cup U_{er}))$ **and**
 $(\mathcal{C} = \text{None} \longrightarrow \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid\cup U_{er}))$ **and**
 $(\forall C. \mathcal{C} = \text{Some } C \longrightarrow$
 $(\forall A \mid\in \text{trail-atms } \Gamma. \exists L. \text{linorder-lit.is-maximal-in-mset } C L \wedge A \preceq_t \text{atm-of}$
 $L))$ **and**
 $(\forall Ln \in \text{set } \Gamma. \text{is-neg } (fst Ln) \longleftrightarrow \text{snd } Ln = \text{None})$ **and**
 $(\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid\in \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup U_{er}). C \prec_c \{\#fst Ln\#} \longrightarrow \text{trail-true-cls } \Gamma C))$

and
 $(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
 $(fst Ln))$ **and**

$(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))$ **and**
 $(\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \#$
 $C. K \neq L \# \})$ **and**
 $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$

lemma *clause-almost-defined-if-lt-next-clause:*

assumes *ord-res-7-invars* $N (U_{er}, \mathcal{F}, \Gamma, C)$

shows $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). (\forall D. C = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{ \#L \in \# C. L \neq K \# \})$

proof (*intro ballI impI allI*)

fix $C :: 'f \text{ gclause}$ **and** $K :: 'f \text{ gliteral}$

assume

C-in: $C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**

C-lt: $\forall D. C = \text{Some } D \longrightarrow C \prec_c D$ **and**

C-max-lit: *ord-res.is-maximal-lit* $K C$

show *trail-defined-cls* $\Gamma \{ \#L \in \# C. L \neq K \# \}$

using *assms*

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma, C)$ *rule: ord-res-7-invars.cases*)

case *invars: 1*

have $\neg (\exists A \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma)$
using *invars C-in C-lt C-max-lit by metis*

hence *C-almost-almost-defined: trail-defined-cls* $\Gamma \{ \#L \in \# C. L \neq K \wedge L \neq$
 $- K \# \}$

using *clause-almost-almost-definedI[OF C-in C-max-lit] by blast*

show *?thesis*

proof (*cases* *trail-defined-lit* ΓK)

case *True*

hence *trail-defined-lit* $\Gamma (- K)$

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* $\Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er})$

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 $C \prec_c D$
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
 $\langle \neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms}$
 $\Gamma) \rangle$
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 \longleftrightarrow
 $(\forall U_{er} \mathcal{F} \Gamma \mathcal{C}. s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \longrightarrow$
 $\mathcal{F} \subseteq | N \cup U_{er} \wedge$
 $(\forall C. \mathcal{C} = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})) \wedge$
 $(\forall D. \mathcal{C} = \text{Some } D \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{ \#K \in \# C. K \neq L_C \# \}$
 $\wedge \text{is-pos } L_C)))) \wedge$
 $(\forall C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $\text{trail-true-cls } \Gamma C) \wedge$
 $(\forall C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms}$
 $\Gamma))) \wedge$
 $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma \wedge$
 $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \cup U_{er})) \wedge$
 $(\mathcal{C} = \text{None} \longrightarrow \text{trail-atms } \Gamma = \text{atms-of-clss } (N \cup U_{er})) \wedge$
 $(\forall C. \mathcal{C} = \text{Some } C \longrightarrow$
 $(\forall A \in | \text{trail-atms } \Gamma. \exists L. \text{linorder-lit.is-maximal-in-mset } C L \wedge A \preceq_t \text{atm-of}$
 $L)) \wedge$
 $(\forall Ln \in \text{set } \Gamma. \text{is-neg } (fst Ln) \longleftrightarrow \text{snd } Ln = \text{None}) \wedge$
 $(\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er}). C \prec_c \{ \#fst Ln \# \} \longrightarrow \text{trail-true-cls } \Gamma C))$
 \wedge
 $(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
 $(fst Ln)) \wedge$
 $(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})) \wedge$

$(\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C. K \neq L \# \}) \wedge$
 $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} | ^\uparrow (N \cup | U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
 (is ?NICE $N s \longleftrightarrow$?UGLY $N s$)
proof (rule iffI)
 show ?NICE $N s \implies$?UGLY $N s$
 apply (intro allI impI)
 subgoal premises *prems* for $U_{er} \mathcal{F} \Gamma C$
 using *prems*(1) unfolding *prems*(2)
 by (cases $N (U_{er}, \mathcal{F}, \Gamma, C)$ rule: ord-res-7-invars.cases) (simp only:)
 done
next
 assume ?UGLY $N s$
 obtain $U_{er} \mathcal{F} \Gamma C$ where *s-def*: $s = (U_{er}, \mathcal{F}, \Gamma, C)$
 by (metis prod.exhaust)
 show ?NICE $N s$
 using $\langle ?UGLY N s \rangle$ [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* $N (U_{er}, \mathcal{F}, \Gamma, C)$
 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* $N (U_{er}, \mathcal{F}, \Gamma, C)$ and $(L, \text{Some } C) \in \text{set } \Gamma$
 shows *trail-false-cls* $\Gamma \{ \#K \in \# C. K \neq L \# \}$
proof –
 obtain $\Gamma_1 \Gamma_0$ where Γ -eq: $\Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0$
 using $\langle (L, \text{Some } C) \in \text{set } \Gamma \rangle$ by (metis split-list)

 hence *trail-false-cls* $\Gamma_0 \{ \#K \in \# C. K \neq L \# \}$
 using *invars* by (simp-all add: *ord-res-7-invars-def*)

 moreover have *suffix* $\Gamma_0 \Gamma$
 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 $\Gamma D L U_{er} A \Gamma' \mathcal{F}$)

note $D\text{-max-lit} = \langle \text{ord-res.is-maximal-lit } L D \rangle$

have

$A \in | \text{atms-of-clss } (N \cup U_{er})$ **and**

$A \prec_t \text{atm-of } L$ **and**

$A \notin | \text{trail-atms } \Gamma$

using *step-hyps unfolding* *atomize-conj linorder-trm.is-least-in-filter-iff* **by**
argo

have *suffix* $\Gamma \Gamma'$

using *step-hyps unfolding suffix-def* **by** *simp*

have

$\mathcal{F}\text{-subset}$: $\mathcal{F} \subseteq | N \cup U_{er}$ **and**

$D\text{-in}$: $D \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ **and**

clauses-lt-D-seldomly-have-undef-max-lit:

$\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow$

$(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in \# C. K \neq L_C\# \} \wedge$

is-pos } L_C)) **and**

clauses-lt-D-true: $\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma$

C **and**

no-undef-atm-lt-max-lit-if-lt-D: $\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$

$\neg (\exists A \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma))$

and

$\Gamma\text{-sorted}$: *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**

$\Gamma\text{-lower}$: *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \cup U_{er}))$

and

trail-atms-le0: $\forall A \in | \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L D \wedge (A \preceq_t$
atm-of } L) **and**

$\Gamma\text{-deci-iff-neg}$: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$ **and**

$\Gamma\text{-deci-ball-lt-true}$: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c \{\#fst Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma C)$

and

$\Gamma\text{-prop-in}$: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \uparrow (N \cup$
 $U_{er})$ and

$\Gamma\text{-prop-greatest}$:

$\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$

$(fst Ln)$ **and**

$\Gamma\text{-prop-almost-false}$:

$\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C.$

$K \neq L\#\}$ **and**

$\Gamma\text{-prop-ball-lt-true}$: $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$

using *invar* **by** (*simp-all add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$ *ord-res-7-invars-def*)

have *trail-atms-le*: $\forall A \mid \in \mid \text{trail-atms } \Gamma. A \preceq_t \text{atm-of } L$
using *trail-atms-le0* $\langle \text{ord-res.is-maximal-lit } L \ D \rangle$
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have *clauses-lt-D-almost-defined*: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow \text{trail-defined-cls } \Gamma \ \{\#L \in \# \ C. L$
 $\neq K \# \})$
using *invar*[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$] *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 $\forall C'. \text{Some } D = \text{Some } C' \longrightarrow C' \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using *D-in* **by** *simp*

moreover have *trail-true-cls* $\Gamma' \ \{\#K \in \# \ C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
if *Some* $D = \text{Some } E$ **and**
C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C \ L_C$ **and**
L_C-undef: $\neg \text{trail-defined-lit } \Gamma' \ L_C$
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 $\neg \text{trail-defined-lit } \Gamma \ L_C$
using *L_C-undef* $\langle \text{suffix } \Gamma \ \Gamma' \rangle$
using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

ultimately have *trail-true-cls* $\Gamma \ \{\#K \in \# \ C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format, OF C-in - C-max-lit*]
by *metis*

thus *?thesis*
using $\langle \text{suffix } \Gamma \ \Gamma' \rangle$ *trail-true-cls-if-trail-true-suffix* **by** *blast*
qed

moreover have
trail-true-cls $\Gamma' \ C$
 $\bigwedge K. \text{linorder-lit.is-maximal-in-mset } C \ K \implies$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma')$
if *C-lt*: $\bigwedge D'. \text{Some } D = \text{Some } D' \implies C \prec_c D'$ **and** $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **for** C
proof –

have $C \prec_c D$
using $C\text{-lt}$ **by** metis

hence $\text{trail-true-clss } \Gamma \ C$
using $\text{clauses-lt-D-true } \langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$ **by** metis

thus $\text{trail-true-clss } \Gamma' \ C$
using $\langle \text{suffix } \Gamma \ \Gamma' \rangle$
by $(\text{metis } \text{trail-true-clss-if-trail-true-suffix})$

fix K
assume $C\text{-max-lit: linorder-lit.is-maximal-in-mset } C \ K$

have $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \not\subseteq \text{trail-atms } \Gamma)$
using $\text{no-undef-atm-lt-max-lit-if-lt-D}$
using $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle \langle C \prec_c D \rangle$ $C\text{-max-lit}$ **by** metis

thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \not\subseteq \text{trail-atms } \Gamma')$
using $\text{step-hyps}(6)$ **by** auto
qed

moreover have $\text{sorted-wrt } (\lambda x \ y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma'$
proof –
have $\forall x \mid \in \mid \text{trail-atms } \Gamma. x \prec_t A$
using $\text{step-hyps}(5)[\text{unfolded linorder-trm.is-least-in-ffilter-iff, simplified}]$
by $(\text{metis } \Gamma\text{-lower linorder-trm.antisym-conv3 linorder-trm.is-lower-set-iff})$

hence $\forall x \in \text{set } \Gamma. \text{atm-of } (fst \ x) \prec_t A$
by $(\text{simp add: fset-trail-atms})$

thus $?thesis$
using $\Gamma\text{-sorted}$ **by** $(\text{simp add: } \langle \Gamma' = (Neg \ A, \ None) \# \ \Gamma \rangle)$
qed

moreover have $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma') \ (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
proof –
have $\text{linorder-trm.is-lower-set } (\text{insert } A \ (\text{fset } (\text{trail-atms } \Gamma)))$
 $(\text{fset } (\text{atms-of-clss } (N \mid \cup \mid U_{er})))$
proof $(\text{rule linorder-trm.is-lower-set-insertI})$
show $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using $\text{step-hyps}(5)[\text{unfolded linorder-trm.is-least-in-ffilter-iff, simplified}]$
by argo

next
show $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). w \prec_t A \longrightarrow w \mid \in \mid \text{trail-atms } \Gamma$
using $\text{step-hyps}(5)[\text{unfolded linorder-trm.is-least-in-ffilter-iff, simplified}]$
by fastforce

```

next
  show linorder-trm.is-lower-fset (trail-atms  $\Gamma$ ) (atms-of-cls ( $N \mid \cup \mid U_{er}$ ))
  using  $\Gamma$ -lower .
qed

moreover have trail-atms  $\Gamma' = \text{finsert } A \text{ (trail-atms } \Gamma)$ 
  by (simp add:  $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ )

ultimately show ?thesis
  by simp
qed

moreover have  $\forall A \mid \in \mid \text{trail-atms } \Gamma'. \exists L. \text{ord-res.is-maximal-lit } L \ D \wedge (A \preceq_t$ 
atm-of } L)
proof (intro ballI exI conjI)
  show ord-res.is-maximal-lit  $L \ D$ 
  using  $\langle \text{ord-res.is-maximal-lit } L \ D \rangle$  .
next
fix  $x$  assume  $x \mid \in \mid \text{trail-atms } \Gamma'$ 

hence  $x = A \vee x \mid \in \mid \text{trail-atms } \Gamma$ 
  unfolding  $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$  by simp

thus  $x \preceq_t \text{atm-of } L$ 
proof (elim disjE)
  assume  $x = A$ 
  thus  $x \preceq_t \text{atm-of } L$ 
  using step-hyps(5) by (simp add: linorder-trm.is-least-in-ffilter-iff)
next
  assume  $x \mid \in \mid \text{trail-atms } \Gamma$ 
  thus  $x \preceq_t \text{atm-of } L$ 
  using trail-atms-le by metis
qed
qed

moreover have  $\forall Ln \in \text{set } \Gamma'. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$ 
  unfolding  $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ 
  using  $\Gamma$ -deci-iff-neg by simp

moreover have trail-true-cls  $\Gamma' \ C$ 
  if  $Ln \in \text{set } \Gamma'$  and  $\text{snd } Ln = \text{None}$  and  $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$  and  $C$ 
 $\prec_c \{ \# \text{fst } Ln \# \}$ 
  for  $Ln \ C$ 
proof –
  have  $Ln = (\text{Neg } A, \text{None}) \vee Ln \in \text{set } \Gamma$ 
  using  $\langle Ln \in \text{set } \Gamma' \rangle$  unfolding  $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$  by simp

hence trail-true-cls  $\Gamma \ C$ 
proof (elim disjE)

```


assume $Ln = (Neg\ A, None)$

hence $fst\ Ln \prec_l L$
by (*metis* $\langle A \prec_t atm\ of\ L \rangle$ *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.mulp-if-maximal-less-that-maximal* **by** *metis*

hence $C \prec_c D$
using $\langle C \prec_c \{\#fst\ Ln\#\} \rangle$ **by** *order*

thus *trail-true-cls* $\Gamma\ C$
using $\langle C \in |iefac\ \mathcal{F}\ |\ \uparrow (N\ |\cup\ U_{er}) \rangle$
using *clauses-lt-D-true* **by** *blast*

next
assume $Ln \in set\ \Gamma$

thus *trail-true-cls* $\Gamma\ C$
using Γ -*deci-ball-lt-true* $\langle snd\ Ln = None \rangle$ $\langle C \in |iefac\ \mathcal{F}\ |\ \uparrow (N\ |\cup\ U_{er}) \rangle$
 $\langle C \prec_c \{\#fst\ Ln\#\} \rangle$
by *metis*

qed

thus *trail-true-cls* $\Gamma'\ C$
using $\langle suffix\ \Gamma\ \Gamma' \rangle$ **by** (*metis* *trail-true-cls-if-trail-true-suffix*)

qed

moreover have $\forall Ln \in set\ \Gamma'. \forall C. snd\ Ln = Some\ C \longrightarrow C \in |iefac\ \mathcal{F}\ |\ \uparrow (N\ |\cup\ U_{er})$
unfolding $\langle \Gamma' = (Neg\ A, None) \# \Gamma \rangle$ **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 $\langle \Gamma' = (Neg\ A, None) \# \Gamma \rangle$ **using** Γ -*prop-greatest* **by** *simp*

moreover have $\forall \Gamma_1\ L\ C\ \Gamma_0. \Gamma' = \Gamma_1\ @\ (L, Some\ C) \# \Gamma_0 \longrightarrow trail-false-cls\ \Gamma_0\ \{\#K \in\# C. K \neq L\#\}$
unfolding $\langle \Gamma' = (Neg\ A, None) \# \Gamma \rangle$ **using** Γ -*prop-almost-false*
by (*metis* (*no-types, lifting*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have $(\forall \Gamma_1\ L\ D\ \Gamma_0. \Gamma' = \Gamma_1\ @\ (L, Some\ D) \# \Gamma_0 \longrightarrow (\forall C \in |iefac\ \mathcal{F}\ |\ \uparrow (N\ |\cup\ U_{er}). C \prec_c D \longrightarrow trail-true-cls\ \Gamma_0\ C))$
using Γ -*prop-ball-lt-true*
unfolding $\langle \Gamma' = (Neg\ A, None) \# \Gamma \rangle$

by (smt (verit, best) append-eq-Cons-conv list.inject option.discI snd-conv)

ultimately show *?thesis*

by (auto simp add: ‹s' = (U_{er}, F, Γ', Some D)› ord-res-7-invars-def)

next

case *step-hyps*: (skip-defined Γ D K U_{er} C' 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: $D \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er})$ **and**

clauses-lt-D-seldomly-have-undef-max-lit:

$\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$

$(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in\# C. K \neq L_C\# \} \wedge$

is-pos L_C)) **and**

clauses-lt-D-true: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma$

C **and**

no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$

$\neg (\exists A \mid\in\mid \text{atms-of-clss } (N \mid\cup\mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \text{trail-atms } \Gamma))$

and

Γ-sorted: *sorted-wrt* ($\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$) Γ **and**

Γ-lower: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N |∪| U_{er}))

and

trail-atms-le0: $\forall A \mid\in\mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L D \wedge (A \preceq_t$

atm-of L) **and**

Γ-deci-iff-neg: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$ **and**

Γ-deci-ball-lt-true: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$

$(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c \{\#fst Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma C)$

and

Γ-prop-in: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid$

U_{er}) **and**

Γ-prop-greatest:

$\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$

(*fst* Ln) **and**

Γ-prop-almost-false:

$\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in\# C.$

K ≠ *L*#} **and**

Γ-prop-ball-lt-true: $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$

$(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$

using invar by (*simp-all* add: ‹s = (U_{er}, F, Γ, Some D)› ord-res-7-invars-def)

have *clauses-lt-D-almost-defined*: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{\#L \in\# C. L$

≠ *K*#})

using invar[*unfolded* ‹s = (U_{er}, F, Γ, 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: $\bigwedge L. L \in\# D \implies L \preceq_l K$
using $\langle \text{ord-res.is-maximal-lit } K D \rangle$
unfolding atomize-imp atomize-all atomize-conj linorder-lit.is-maximal-in-mset-iff
using linorder-lit.leI **by** blast

hence atm-of $K \mid\in\mid$ atms-of-cls $(N \mid\cup\mid U_{er})$
using D-in atm-of-in-atms-of-clsI **by** metis

have trail-atms-le: $\forall A \mid\in\mid$ trail-atms $\Gamma. A \preceq_t \text{atm-of } K$
using trail-atms-le0 $\langle \text{ord-res.is-maximal-lit } K D \rangle$
by (metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset)

have trail-defined-cls $\Gamma \{ \#Ka \in\# D. Ka \neq K\# \}$
using step-hyps(4,5,6) D-in **by** (metis clause-almost-definedI)

show ?thesis
unfolding $\langle s' = (U_{er}, \mathcal{F}, \Gamma, C') \rangle$
proof (intro ord-res- γ -invars.intros)
show $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$
using \mathcal{F} -subset .

show $\forall C'. C' = \text{Some } C' \longrightarrow C' \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$
using step-hyps(γ) **by** (metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD)

have trail-true-cls $\Gamma \{ \#K \in\# C. K \neq L_C\# \} \wedge \text{is-pos } L_C$
if $C' = \text{Some } E$ and
 C -in: $C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$ and
 C -lt: $C \prec_c E$ and
 C -max-lit: linorder-lit.is-maximal-in-mset $C L_C$ and
 L_C -undef: $\neg \text{trail-defined-lit } \Gamma L_C$
for $E C L_C$

proof –
have linorder-cls.is-least-in-fset (ffilter ($(\prec_c) D$) (iefac $\mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$)) E
using $\langle C' = \text{Some } E \rangle$ 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 $C\text{-max-lit } D\text{-max-lit}$
by $(\text{metis } \text{Uniq-}D \text{ linorder-lit. } \text{Uniq-is-maximal-in-mset})$
hence False
using $L_C\text{-undef } \langle \text{trail-defined-lit } \Gamma K \rangle$ **by** argo
thus $?thesis ..$
qed
qed

thus $\forall D. C' = \text{Some } D \longrightarrow (\forall C \in |\text{iefac } \mathcal{F} |^\dagger (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{ord-res.is-maximal-lit } L_C C \longrightarrow \neg \text{trail-defined-lit } \Gamma L_C \longrightarrow$
 $\text{trail-true-cls } \Gamma \{\#K \in\# C. K \neq L_C\# \} \wedge \text{is-pos } L_C))$
by metis

have
 $\text{trail-true-cls } \Gamma C$
 $\wedge K. \text{linorder-lit.is-maximal-in-mset } C K \implies$
 $\neg (\exists A \in |\text{atms-of-cls } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \text{trail-atms } \Gamma)$
if $C\text{-lt}$: $\wedge E. C' = \text{Some } E \implies C \prec_c E$ **and** $C\text{-in}$: $C \in |\text{iefac } \mathcal{F} |^\dagger (N \mid \cup \mid$
 $U_{er})$ **for** C

proof –
have $C \preceq_c D$
using $\text{step-hyps that by } (\text{metis } \text{clause-le-if-lt-least-greater})$

hence $C \prec_c D \vee C = D$
by simp

thus $\text{trail-true-cls } \Gamma C$

proof $(\text{elim } \text{disjE})$

assume $C \prec_c D$

thus $\text{trail-true-cls } \Gamma C$

using $\text{clauses-lt-D-true } \langle C \in |\text{iefac } \mathcal{F} |^\dagger (N \mid \cup \mid U_{er}) \rangle$ **by** metis

next

assume $C = D$

have $\text{trail-defined-cls } \Gamma D$

using $\langle \text{trail-defined-lit } \Gamma K \rangle \langle \text{trail-defined-cls } \Gamma \{\#Ka \in\# D. Ka \neq K\# \} \rangle$

unfolding $\text{trail-defined-cls-def}$ **by** auto

hence $\text{trail-true-cls } \Gamma D$

using $\Gamma\text{-consistent } \langle \neg \text{trail-false-cls } \Gamma D \rangle$

by $(\text{metis } \text{trail-true-or-false-cls-if-defined})$

thus $\text{trail-true-cls } \Gamma C$

using $\langle C = D \rangle$ **by** simp

qed

fix K_c

assume C -max-lit: $\text{linorder-lit.is-maximal-in-mset } C K_c$
thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_c \wedge A \notin \mid \text{trail-atms } \Gamma)$
using $\langle C \prec_c D \vee C = D \rangle$
proof (elim disjE)
assume $C \prec_c D$
thus ?thesis
using $\text{no-undef-atm-lt-max-lit-if-lt-D } \langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$
 C -max-lit **by** metis
next
assume $C = D$
thus ?thesis
by ($\text{metis } C$ -max-lit D -max-lit $\text{Uniq-D linorder-lit.Uniq-is-maximal-in-mset step-hyps(5)}$)
qed
qed

thus
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow \text{trail-true-cls } \Gamma C$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \mid \text{trail-atms } \Gamma))$
unfolding $\text{atomize-conj by metis}$

show $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
using Γ -sorted .

show $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
using Γ -lower .

have $\text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **if** $C' = \text{None}$
proof ($\text{intro fsubset-antisym}$)
show $\text{trail-atms } \Gamma \mid \subseteq \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using Γ -lower **unfolding** $\text{linorder-trm.is-lower-set-iff by blast}$
next
have $\text{nbex-gt-D}: \neg (\exists E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). D \prec_c E)$
using $\text{step-hyps } \langle C' = \text{None} \rangle$
by ($\text{metis clause-le-if-lt-least-greater linorder-cls.leD option.simps(3)}$)

have $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \text{atm-of } K \prec_t A)$
proof ($\text{intro notI , elim bexE}$)
fix $A :: 'f gterm$
assume $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and** $\text{atm-of } K \prec_t A$

hence $A \mid \in \mid \text{atms-of-clss } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}))$
by simp

then obtain $E L$ **where** E -in: $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and** $L \in \# E$

and $A = atm\text{-of } L$
unfolding $atms\text{-of-clss-def } atm\text{-of-clss-def}$
by ($smt (verit, del\text{-insts}) fimage\text{-iff } fmember\text{-ffUnion-iff } fset\text{-fset-mset}$)

have $K \prec_l L$
using $\langle atm\text{-of } K \prec_t A \rangle \langle A = atm\text{-of } L \rangle$
by ($cases K; cases L$) $simp\text{-all}$

hence $D \prec_c E$
using $D\text{-max-lit } \langle L \in\# E \rangle$
by ($metis empty\text{-iff } linorder\text{-lit.ex-maximal-in-mset } linorder\text{-lit.is-maximal-in-mset-iff } linorder\text{-lit.less-linear } linorder\text{-lit.less-trans } linorder\text{-lit.mulp-if-maximal-less-that-maximal } set\text{-mset-empty}$)

thus $False$
using $E\text{-in } nbex\text{-gt-}D$ **by** $metis$

qed

hence $\neg (\exists A | \in | atms\text{-of-clss } (N \cup U_{er}). A \notin | trail\text{-atms } \Gamma)$
using $step\text{-hyps}(5) step\text{-hyps}(6)$
by ($metis linorder\text{-trm.linorder-cases } trail\text{-defined-lit-iff-trail-defined-atm}$)

then show $atms\text{-of-clss } (N \cup U_{er}) \subseteq | trail\text{-atms } \Gamma$
by $blast$

qed

thus $C' = None \longrightarrow trail\text{-atms } \Gamma = atms\text{-of-clss } (N \cup U_{er})$
by $metis$

show $\forall D. C' = Some D \longrightarrow (\forall A | \in | trail\text{-atms } \Gamma. \exists L. ord\text{-res.is-maximal-lit } L D \wedge A \preceq_t atm\text{-of } L)$
proof ($intro allI impI ballI$)
fix $E :: 'f gterm literal multiset$ **and** $A :: 'f gterm$
assume $C' = Some E$ **and** $A | \in | trail\text{-atms } \Gamma$

have $linorder\text{-cls.is-least-in-fset } (ffilter ((\prec_c) D) (iefac \mathcal{F} | \uparrow (N \cup U_{er}))) E$
using $step\text{-hyps}(7) \langle C' = Some E \rangle$ **by** ($metis Some\text{-eq-The-optionalD}$)

hence $D \prec_c E$
unfolding $linorder\text{-cls.is-least-in-ffilter-iff}$ **by** $argo$

obtain L **where** $linorder\text{-lit.is-maximal-in-mset } E L$
by ($metis \langle D \prec_c E \rangle linorder\text{-cls.leD } linorder\text{-lit.ex-maximal-in-mset } mempty\text{-lesseq-clss}$)

show $\exists L. ord\text{-res.is-maximal-lit } L E \wedge A \preceq_t atm\text{-of } L$
proof ($intro exI conjI$)
show $ord\text{-res.is-maximal-lit } L E$
using $\langle ord\text{-res.is-maximal-lit } L E \rangle$.

```

next
  have  $K \preceq_l L$ 
    using step-hyps(4)  $\langle \text{ord-res.is-maximal-lit } L \ E \rangle$ 
    by (metis  $\langle D \prec_c E \rangle$  linorder-cls.less-asym linorder-lit.leI
      linorder-lit.mulp-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  $\langle A \mid \in \mid \text{trail-atms } \Gamma \rangle$  trail-atms-le by blast

  ultimately show  $A \preceq_t$  atm-of  $L$ 
    by order
  qed
qed

show  $\forall Ln \in \text{set } \Gamma. \text{is-neg } (\text{fst } Ln) = (\text{snd } Ln = \text{None})$ 
  using  $\Gamma$ -deci-iff-neg by metis

show  $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$ 
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \{\#\text{fst } Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma \ C)$ 
  using  $\Gamma$ -deci-ball-lt-true .

show  $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
  using  $\Gamma$ -prop-in by simp

show  $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset}$ 
 $C \ (\text{fst } Ln)$ 
  using  $\Gamma$ -prop-greatest by simp

show  $\forall \Gamma_1 \ L \ C \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } C) \ \# \ \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \ \{\#K$ 
 $\in \# \ C. K \neq L\#\}$ 
  using  $\Gamma$ -prop-almost-false .

show  $(\forall \Gamma_1 \ L \ D \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } D) \ \# \ \Gamma_0 \longrightarrow$ 
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 \ C))$ 
  using  $\Gamma$ -prop-ball-lt-true .
qed
next
case step-hyps: (skip-undefined-neg  $\Gamma \ D \ K \ U_{er} \ \Gamma' \ C' \ \mathcal{F}$ )

note  $D$ -max-lit =  $\langle \text{ord-res.is-maximal-lit } K \ D \rangle$ 

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: $D \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
 $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge$
***is-pos } L_C)) **and**
clauses-lt-D-true: $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma$
C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma))$
and
Γ-sorted: $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
Γ-lower: $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
and
trail-atms-le0: $\forall A \in | \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L D \wedge (A \preceq_t$
***atm-of } L) **and**
Γ-deci-iff-neg: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$ **and**
Γ-deci-ball-lt-true: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \{ \#fst Ln \# \} \longrightarrow \text{trail-true-cls } \Gamma C)$
and
Γ-prop-in: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid$
U_{er}) **and**
Γ-prop-greatest:
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
(fst Ln) **and**
Γ-prop-almost-false:
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C.$
 $K \neq L \# \}$ **and**
Γ-prop-ball-lt-true: $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
using invar by $(\text{simp-all add: } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle \text{ord-res-}\gamma\text{-invars-def})$

have ***clauses-lt-D-almost-defined***: $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{ \#L \in \# C. L$
 $\neq K \# \})$
using invar $[\text{unfolded } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle]$ ***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 $\langle \text{ord-res.is-maximal-lit } K D \rangle$
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of* $K \in | \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using *D-in atm-of-in-atms-of-clssI* **by** *metis*******

have *trail-atms-le*: $\forall A \mid \in \mid \text{trail-atms } \Gamma. A \preceq_t \text{atm-of } K$
using *trail-atms-le0* $\langle \text{ord-res.is-maximal-lit } K \ D \rangle$
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

have $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$
using *F-subset* .

moreover have $\forall C'. C' = \text{Some } C' \longrightarrow C' \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using *step-hyps(9)* **by** (*metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD*)

moreover have *trail-true-cls* $\Gamma' \{ \#K \in \# C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
if $C' = \text{Some } E$ **and**
C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
C-lt: $C \prec_c E$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C \ L_C$ **and**
L_C-undef: $\neg \text{trail-defined-lit } \Gamma' \ L_C$
for $E \ C \ L_C$

proof –
have *linorder-cls.is-least-in-fset* (*ffilter* ($(\prec_c) \ D$) (*iefac* $\mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$)) E
using $\langle C' = \text{Some } E \rangle$ *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 $\neg \text{trail-defined-lit } \Gamma \ L_C$
using *L_C-undef* $\langle \text{suffix } \Gamma \ \Gamma' \rangle$
using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

ultimately have *trail-true-cls* $\Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format, OF C-in - C-max-lit*] **by** *metis*

thus *?thesis*
using $\langle \text{suffix } \Gamma \ \Gamma' \rangle$ *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 $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$
proof –
have $\text{atm-of } K \mid \notin \mid \text{trail-atms } \Gamma$
using $\langle \neg \text{trail-defined-lit } \Gamma K \rangle$
by (*simp add: trail-defined-lit-iff-trail-defined-atm*)

have $x \prec_t \text{atm-of } K$ **if** $x\text{-in: } x \mid \in \mid \text{trail-atms } \Gamma$ **for** x
proof –
have $x \preceq_t \text{atm-of } K$
using $x\text{-in trail-atms-le}$ **by** *metis*

moreover have $x \neq \text{atm-of } K$
using $x\text{-in } \langle \text{atm-of } K \mid \notin \mid \text{trail-atms } \Gamma \rangle$ **by** *metis*

ultimately show *?thesis*
by *order*
qed

hence $\forall x \in \text{set } \Gamma. \text{atm-of } (fst x) \prec_t \text{atm-of } K$
by (*simp add: fset-trail-atms*)

thus *?thesis*
using $\Gamma\text{-sorted } \langle \Gamma' = (K, None) \# \Gamma \rangle$ **by** *simp*
qed

moreover have $\Gamma'\text{-lower: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma') (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
proof –
have $\text{linorder-trm.is-lower-set } (\text{insert } (\text{atm-of } K) (\text{fset } (\text{trail-atms } \Gamma)))$
 $(\text{fset } (\text{atms-of-clss } (N \mid \cup \mid U_{er})))$
proof (*rule linorder-trm.is-lower-set-insertI*)
show $\text{atm-of } K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using $\langle \text{atm-of } K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$.
next
show $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). w \prec_t (\text{atm-of } K) \longrightarrow w \mid \in \mid \text{trail-atms } \Gamma$
using *step-hyps(5)[unfolded linorder-trm.is-least-in-filter-iff, simplified]*
by *fastforce*
next
show $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
using $\Gamma\text{-lower}$.
qed

moreover have $\text{trail-atms } \Gamma' = \text{finsert } (\text{atm-of } K) (\text{trail-atms } \Gamma)$
by (*simp add: } \Gamma' = (K, None) \# \Gamma)

ultimately show *?thesis*
by *simp**

qed

moreover have *trail-atms* $\Gamma' = \text{atms-of-clss } (N \cup U_{er})$ if $C' = \text{None}$

proof (intro *fsubset-antisym*)

show *trail-atms* $\Gamma' \subseteq \text{atms-of-clss } (N \cup U_{er})$

using Γ' -lower unfolding *linorder-trm.is-lower-set-iff* by blast

next

have *nbex-gt-D*: $\neg (\exists E \in \text{iefac } \mathcal{F} \mid (N \cup U_{er}). D \prec_c E)$

using *step-hyps* $\langle C' = \text{None} \rangle$

by (*metis clause-le-if-lt-least-greater linorder-clss.leD option.simps(3)*)

have $\neg (\exists A \in \text{atms-of-clss } (N \cup U_{er}). \text{atm-of } K \prec_t A)$

proof (intro *notI*, *elim bexE*)

fix $A :: 'f \text{ gterm}$

assume $A \in \text{atms-of-clss } (N \cup U_{er})$ and $\text{atm-of } K \prec_t A$

hence $A \in \text{atms-of-clss } (\text{iefac } \mathcal{F} \mid (N \cup U_{er}))$

by *simp*

then obtain $E L$ where *E-in*: $E \in \text{iefac } \mathcal{F} \mid (N \cup U_{er})$ and $L \in \# E$

and $A = \text{atm-of } L$

unfolding *atms-of-clss-def atms-of-clss-def*

by (*smt (verit, del-insts) fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have $K \prec_l L$

using $\langle \text{atm-of } K \prec_t A \rangle \langle A = \text{atm-of } L \rangle$

by (*cases K; cases L simp-all*)

hence $D \prec_c E$

using *D-max-lit* $\langle L \in \# E \rangle$

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.mulp-if-maximal-less-that-maximal set-mset-empty)

thus *False*

using *E-in nbex-gt-D* by *metis*

qed

hence $\neg (\exists A \in \text{atms-of-clss } (N \cup U_{er}). A \notin \text{trail-atms } \Gamma')$

using *step-hyps*

by (*metis finsert-iff linorder-trm.antisym-conv3 prod.sel(1) trail-atms.simps(2)*)

then show *atms-of-clss* $(N \cup U_{er}) \subseteq \text{trail-atms } \Gamma'$

by *blast*

qed

moreover have $\forall D. C' = \text{Some } D \longrightarrow (\forall A \in \text{trail-atms } \Gamma'. \exists L. \text{ord-res.is-maximal-lit } L D \wedge A \preceq_t \text{atm-of } L)$

proof (intro *allI impI ballI*)

fix $E :: 'f\ gterm\ literal\ multiset$ **and** $A :: 'f\ gterm$
assume $C' = Some\ E$ **and** $A \in | trail-atms\ \Gamma'$

have $linorder-cls.is-least-in-fset\ (ffilter\ ((<_c)\ D)\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})))\ E$
using $step-hyps(9)\ \langle C' = Some\ E \rangle$ **by** $(metis\ Some-eq-The-optionalD)$

hence $D <_c\ E$
unfolding $linorder-cls.is-least-in-filter-iff$ **by** $argo$

obtain L **where** $linorder-lit.is-maximal-in-mset\ E\ L$
by $(metis\ \langle D <_c\ E \rangle\ linorder-cls.leD\ linorder-lit.ex-maximal-in-mset\ mempty-lesseq-cls)$

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 $\langle ord-res.is-maximal-lit\ L\ E \rangle$.

next

have $K \preceq_l\ L$
using $step-hyps(4)\ \langle ord-res.is-maximal-lit\ L\ E \rangle$
by $(metis\ \langle D <_c\ E \rangle\ linorder-cls.less-asym\ linorder-lit.leI\ linorder-lit.mulp-if-maximal-less-that-maximal)$

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 $A = atm-of\ K \vee A \in | trail-atms\ \Gamma$
using $\langle A \in | trail-atms\ \Gamma' \rangle$
unfolding $\langle \Gamma' = (K, None) \# \Gamma \rangle$
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 $\forall Ln \in set\ \Gamma'. snd\ Ln = None \iff is-neg\ (fst\ Ln)$

unfolding $\langle \Gamma' = (K, None) \# \Gamma \rangle$
using $\Gamma-deci-iff-neg\ \langle is-neg\ K \rangle$ **by** $simp$

moreover have $trail-true-cls\ \Gamma'\ C$

if $Ln \in set\ \Gamma'$ **and** $snd\ Ln = None$ **and** $C \in | iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er})$ **and** $C <_c\ \{\#fst\ Ln\# \}$
for $Ln\ C$

proof –
have $Ln = (K, None) \vee Ln \in \text{set } \Gamma$
using $\langle Ln \in \text{set } \Gamma' \rangle$ **unfolding** $\langle \Gamma' = (K, None) \# \Gamma \rangle$ **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 $\langle C \prec_c \{\#fst Ln\# \}$
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $C \prec_c D$
using *D-max-lit*
by (*metis* $\langle K \in \# D \rangle$ *empty-iff ord-res.multp-if-all-left-smaller set-mset-empty*)

thus *trail-true-cls* Γ C
using $\langle C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$
using *clauses-lt-D-true* **by** *blast*

next
assume $Ln \in \text{set } \Gamma$

thus *trail-true-cls* Γ C
using Γ -*deci-ball-lt-true* $\langle \text{snd } Ln = None \rangle$ $\langle C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$
 $\langle C \prec_c \{\#fst Ln\# \}$
by *metis*
qed

thus *trail-true-cls* Γ' C
using $\langle \text{suffix } \Gamma \Gamma' \rangle$ **by** (*metis* *trail-true-cls-if-trail-true-suffix*)
qed

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using Γ -*prop-in* $\langle \Gamma' = (K, None) \# \Gamma \rangle$ **by** *simp*

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$ (*fst* Ln)
using Γ -*prop-greatest* $\langle \Gamma' = (K, None) \# \Gamma \rangle$ **by** *simp*

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$
unfolding $\langle \Gamma' = (K, None) \# \Gamma \rangle$ **using** Γ -*prop-almost-false*
by (*metis* (*no-types, lifting*) *append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have $(\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow (\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
unfolding $\langle \Gamma' = (K, None) \# \Gamma \rangle$ **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 $\langle s' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$

proof (intro ord-res- γ -invars.intros)

have trail-true-cls $\Gamma' C$

$\wedge K_C$. linorder-lit.is-maximal-in-mset $C K_C \implies$

$\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \mid \notin \mid \text{trail-atms } \Gamma')$

if C-lt: $\wedge E$. $C' = \text{Some } E \implies C \prec_c E$ and C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 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 $\Gamma' C$

proof (elim disjE)

assume $C \prec_c D$

hence trail-true-cls ΓC

using clauses-lt-D-true $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$ by metis

thus trail-true-cls $\Gamma' C$

using $\langle \text{suffix } \Gamma \Gamma' \rangle$ by (metis trail-true-cls-if-trail-true-suffix)

next

assume $C = D$

have trail-true-lit $\Gamma' K$

unfolding $\langle \Gamma' = (K, \text{None}) \# \Gamma \rangle$ trail-true-lit-def by simp

thus trail-true-cls $\Gamma' C$

unfolding $\langle C = D \rangle$ trail-true-cls-def

using $\langle K \in \# D \rangle$ by metis

qed

fix K_C

assume C-max-lit: linorder-lit.is-maximal-in-mset $C K_C$

show $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \mid \notin \mid \text{trail-atms } \Gamma')$

using $\langle C \prec_c D \vee C = D \rangle$

proof (elim disjE)

assume $C \prec_c D$

hence $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \mid \notin \mid \text{trail-atms } \Gamma')$

using no-undef-atm-lt-max-lit-if-lt-D $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$ C-max-lit by metis

thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \mid \text{trail-atms}$
 $\Gamma')$
by (*metis finsert-iff step-hyps(8) trail-atms.simps(2)*)
next
assume $C = D$
thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \mid \text{trail-atms}$
 $\Gamma')$
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
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow \text{trail-true-cls}$
 $\Gamma' C$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \mid \text{trail-atms } \Gamma'))$
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps: (skip-undefined-pos $\Gamma D K U_{er} \mathcal{F} E$)*

note $D\text{-max-lit} = \langle \text{ord-res.is-maximal-lit } K D \rangle$
note $E\text{-least} = \langle \text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) D) (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}))) E \rangle$

have $D \prec_c E$
using $E\text{-least}$ **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *argo*

have
 $\mathcal{F}\text{-subset: } \mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$ **and**
 $D\text{-in: } D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
 $\text{clauses-lt-D-seldomly-have-undef-max-lit:}$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge$
 $\text{is-pos } L_C))$ **and**
 $\text{clauses-lt-D-true: } \forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma$
 C **and**
 $\text{no-undef-atm-lt-max-lit-if-lt-D: } \forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \mid \text{trail-atms } \Gamma))$
and
 $\Gamma\text{-sorted: sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
 $\Gamma\text{-lower: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$

and
trail-atms-le0: $\forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L \ D \wedge (A \preceq_t \text{atm-of } L)$ **and**
 $\Gamma\text{-deci-iff-neg}$: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$ **and**
 $\Gamma\text{-deci-ball-lt-true}$: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \{\#\text{fst } Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma \ C)$
and
 $\Gamma\text{-prop-in}$: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $\Gamma\text{-prop-greatest}$:
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
 $(\text{fst } Ln)$ **and**
 $\Gamma\text{-prop-almost-false}$:
 $\forall \Gamma_1 \ L \ C \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } C) \ # \ \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \ \{\#K \in \# \ C. K \neq L\# \}$ **and**
 $\Gamma\text{-prop-ball-lt-true}$: $(\forall \Gamma_1 \ L \ D \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } D) \ # \ \Gamma_0 \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 \ C))$
using invar by $(\text{simp-all add: } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle \text{ord-res-}\gamma\text{-invars-def})$

have *clauses-lt-D-almost-defined*: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow \text{trail-defined-cls } \Gamma \ \{\#L \in \# \ C. L \neq K\# \})$
using invar $[\text{unfolded } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle]$ *clause-almost-defined-if-lt-next-clause*
by *simp*

have $\Gamma\text{-consistent}$: *trail-consistent* Γ
using *trail-consistent-if-sorted-wrt-atoms* $\Gamma\text{-sorted}$ **by** *metis*

have $K \in \# \ D$
using $\langle \text{ord-res.is-maximal-lit } K \ D \rangle$
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of* $K \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er})$
using *D-in atm-of-in-atms-of-clsI* **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 $\langle \text{is-pos } K \rangle$ *D-max-lit*
by $(\text{metis } (\text{no-types, opaque-lifting}) \text{is-pos-def } \text{linorder-lit.antisym-conv3}$
 $\text{linorder-lit.is-maximal-in-mset-iff } \text{linorder-trm.less-imp-not-eq } \text{ord-res.less-lit-simps}(4)$
 $\text{uminus-Pos } \text{uminus-not-id})$

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 $\langle \neg \text{trail-false-cls } \Gamma \ \{\#L \in \# \ D. L \neq K\# \} \rangle$

using *trail-true-or-false-cls-if-defined* **by** *metis*

hence $D\text{-true}$: *trail-true-cls* ΓD
unfolding *trail-true-cls-def* **by** *auto*

have *trail-atms-le*: $\forall A \mid \in \mid$ *trail-atms* Γ . $A \preceq_t$ *atm-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* $\langle D \prec_c E \rangle$ *linorder-cls.leD* *linorder-lit.ex-maximal-in-mset* *mempty-lesseq-cls*)

have $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$
using *F-subset* .

moreover have $\forall C'$. *Some* $E = \text{Some } C' \longrightarrow C' \mid \in \mid$ *iefac* $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using *step-hyps* **unfolding** *linorder-cls.is-least-in-ffilter-iff* **by** *simp*

moreover have *trail-true-cls* $\Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge$ *is-pos* L_C
if *Some* $E = \text{Some } E'$ **and**
C-in: $C \mid \in \mid$ *iefac* $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **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* ΓL_C
for $E' C L_C$

proof –
have $E' = E$
using *that* **by** *simp*
hence *linorder-cls.is-least-in-fset* (*ffilter* ($(\prec_c) D$) (*iefac* $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$)) 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* $\langle E' = E \rangle$ **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* $\Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge$ *is-pos* L_C
using *clauses-lt-D-seldomly-have-undef-max-lit*[*rule-format*, *OF C-in - C-max-lit 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 $\langle C = D \rangle$ \langle *trail-true-cls* $\Gamma \{ \#L \in \# D. L \neq K \# \} \rangle$ \langle *is-pos* $K \rangle$ **by** *metis*

qed
qed

moreover have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
using Γ -sorted .

moreover have *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma) (\text{atms-of-cls } (N \mid \cup U_{er}))$
using Γ -lower .

moreover have $\forall D. \text{Some } E = \text{Some } D \longrightarrow (\forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L D \wedge A \preceq_t \text{atm-of } L)$
proof –
have $\forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L E \wedge A \preceq_t \text{atm-of } L$
proof (*intro ballI*)
fix $A :: 'f \text{ gterm}$
assume $A \mid \in \mid \text{trail-atms } \Gamma$
show $\exists L. \text{ord-res.is-maximal-lit } L E \wedge A \preceq_t \text{atm-of } L$
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* $\langle D \prec_c E \rangle$ *linorder-cls.less-asm linorder-lit.leI linorder-lit.mulp-if-maximal-less-that-maximal*)

hence *atm-of* $K \preceq_t \text{atm-of } L$
by (*cases K; cases L simp-all*)

moreover have $A \preceq_t \text{atm-of } K$
using $\langle A \mid \in \mid \text{trail-atms } \Gamma \rangle$ *trail-atms-le* **by** *metis*

ultimately show $A \preceq_t \text{atm-of } L$
by *order*

qed
qed

thus *?thesis*
by *simp*

qed

moreover have $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$
using Γ -*deci-iff-neg* .

moreover have $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow (\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). C \prec_c \{\#fst Ln\} \longrightarrow \text{trail-true-cls } \Gamma C)$
using Γ -*deci-ball-lt-true* .

moreover have $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$

using Γ -prop-in .

moreover have $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C \text{ (fst } Ln)$
using Γ -prop-greatest .

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C. K \neq L \# \}$
using Γ -prop-almost-false .

moreover have $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow (\forall C | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
using Γ -prop-ball-lt-true .

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle$
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* ΓC
 $\wedge K_C. \text{linorder-lit.is-maximal-in-mset } C K_C \implies$
 $\neg (\exists A | \in | \text{atms-of-cls } (N | \cup | U_{er}). A \prec_t \text{atm-of } K_C \wedge A | \notin | \text{trail-atms } \Gamma)$
if *C-lt*: $\wedge E'. \text{Some } E = \text{Some } E' \implies C \prec_c E'$ **and** *C-in*: $C | \in | \text{iefac } \mathcal{F} | \uparrow$
 $(N | \cup | U_{er})$ **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* ΓC
proof (*elim disjE*)
assume $C \prec_c D$
thus *trail-true-cls* ΓC
using *clauses-lt-D-true* $\langle C | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}) \rangle$ **by** *metis*
next
assume $C = D$
thus *trail-true-cls* ΓC
using *D-true* **by** *argo*
qed

fix K_C
assume *C-max-lit*: *linorder-lit.is-maximal-in-mset* $C K_C$
show $\neg (\exists A | \in | \text{atms-of-cls } (N | \cup | U_{er}). A \prec_t \text{atm-of } K_C \wedge A | \notin | \text{trail-atms } \Gamma)$
using $\langle C \prec_c D \vee C = D \rangle$
proof (*elim disjE*)
assume $C \prec_c D$

thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \mid \text{trail-atms } \Gamma)$
using *no-undef-atm-lt-max-lit-if-lt-D* $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$
C-max-lit **by** *metis*
next
assume $C = D$
thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \mid \text{trail-atms } \Gamma)$
by (*metis C-max-lit D-max-lit Uniq-D linorder-lit.Uniq-is-maximal-in-mset step-hyps(5)*)
qed
qed

thus
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. \text{Some } E = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
trail-true-cls $\Gamma \ C$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall D. \text{Some } E = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K \ C \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \mid \text{trail-atms } \Gamma))$
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps*: (*skip-undefined-pos-ultimate* $\Gamma \ D \ K \ U_{er} \ \Gamma' \ \mathcal{F}$)

note *D-max-lit* = $\langle \text{ord-res.is-maximal-lit } K \ D \rangle$

have *suffix* $\Gamma \ \Gamma'$
using $\langle \Gamma' = (- \ K, \text{None}) \# \Gamma \rangle$ **by** (*simp add: suffix-def*)

have
F-subset: $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$ **and**
D-in: $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C \ L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma \ L_C \longrightarrow (\text{trail-true-cls } \Gamma \ \{\#K \in \# \ C. \ K \neq L_C \# \} \wedge$
*is-pos } L_C)) **and**
clauses-lt-D-true: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma \ C$ **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \mid \text{trail-atms } \Gamma))$
and
 Γ -*sorted*: *sorted-wrt* $(\lambda x \ y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma$ **and**
 Γ -*lower*: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* $(N \mid \cup \mid U_{er})$)
and
trail-atms-le0: $\forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L \ D \wedge (A \preceq_t \text{atm-of } L)$ **and**
 Γ -*deci-iff-neg*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst \ Ln)$ **and***

Γ -*deci-ball-lt-true*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \{\#fst Ln\# \} \longrightarrow \text{trail-true-clcs } \Gamma C)$
and
 Γ -*prop-in*: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 Γ -*prop-greatest*:
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
(fst Ln) **and**
 Γ -*prop-almost-false*:
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-clcs } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$ **and**
 Γ -*prop-ball-lt-true*: $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-clcs } \Gamma_0 C))$
using invar by (*simp-all add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle \text{ord-res-}\gamma\text{-invars-def}$)

have *clauses-lt-D-almost-defined*: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-clcs } \Gamma \{\#L \in \# C. L \neq K\# \})$
using invar[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$] *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 $\langle \text{ord-res.is-maximal-lit } K D \rangle$
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

hence *atm-of* $K \mid \in \mid \text{atms-of-clcs } (N \mid \cup \mid U_{er})$
using *D-in atm-of-in-atms-of-clcsI* **by** *metis*

have *trail-defined-clcs* $\Gamma \{\#L \in \# D. L \neq K \wedge L \neq -K\# \}$
using *clause-almost-almost-definedI*[*OF D-in step-hyps(4,5)*] .

moreover have $-K \notin \# D$
using $\langle \text{is-pos } K \rangle$ *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-clcs* $\Gamma \{\#L \in \# D. L \neq K\# \}$
unfolding *trail-defined-clcs-def* **by** *auto*

hence *trail-true-clcs* $\Gamma \{\#L \in \# D. L \neq K\# \}$
using $\langle \neg \text{trail-false-clcs } \Gamma \{\#L \in \# D. L \neq K\# \} \rangle$
using *trail-true-or-false-clcs-if-defined* **by** *metis*

hence *D-true*: *trail-true-clcs* ΓD
unfolding *trail-true-clcs-def* **by** *auto*

have *trail-atms-le*: $\forall A \mid \in \mid \text{trail-atms } \Gamma. A \preceq_t \text{atm-of } K$
using *trail-atms-le0 D-max-lit*
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 $\forall C'. \text{None} = \text{Some } C' \longrightarrow C' \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
by *simp*

moreover have *trail-true-cls* $\Gamma' \{ \#K \in \# C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
if *None = Some E' and*
C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
C-lt: $C \prec_c E'$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C L_C$ **and**
L_C-undef: $\neg \text{trail-defined-lit } \Gamma' L_C$
for $E' C L_C$
using *that by simp*

moreover have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$
proof –
have $\forall x \in \text{set } \Gamma. \text{atm-of } (fst x) \prec_t \text{atm-of } K$
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 $\langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle$
using *Γ -sorted by simp*
qed

moreover have *Γ' -lower*: *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma') (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
proof –
have *atm-of* $K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using $\langle \text{atm-of } K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$.

moreover have $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). w \prec_t \text{atm-of } K \longrightarrow w \mid \in \mid \text{trail-atms } \Gamma$
using *step-hyps(5) by blast*

ultimately show *?thesis*
using *Γ -lower by (simp add: $\langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle$ linorder-trm.is-lower-set-insertI)*
qed

moreover have *trail-atms* $\Gamma' = \text{atms-of-clss } (N \mid \cup \mid U_{er})$
proof (*intro fsubset-antisym*)
show *trail-atms* $\Gamma' \mid \subseteq \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$

using Γ' -lower **unfolding** *linorder-trm.is-lower-set-iff* **by** *blast*
next
have $nbex\text{-}gt\text{-}D: \neg (\exists E \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). D \prec_c E)$
using *step-hyps* **by** *argo*

have $\neg (\exists A \mid \in \mid atm\text{-}of\text{-}cls (N \mid \cup \mid U_{er}). atm\text{-}of K \prec_t A)$
proof (*intro notI* , *elim bexE*)
fix $A :: 'f\ gterm$
assume $A \mid \in \mid atm\text{-}of\text{-}cls (N \mid \cup \mid U_{er})$ **and** $atm\text{-}of K \prec_t A$

hence $A \mid \in \mid atm\text{-}of\text{-}cls (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))$
by *simp*

then obtain $E L$ **where** $E\text{-}in: E \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $L \in\# E$
and $A = atm\text{-}of L$
unfolding *atms-of-cls-def atm-of-cls-def*
by (*smt (verit, del-insts) fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have $K \prec_l L$
using $\langle atm\text{-}of K \prec_t A \rangle \langle A = atm\text{-}of L \rangle$
by (*cases K; cases L*) *simp-all*

hence $D \prec_c E$
using *D-max-lit* $\langle L \in\# E \rangle$
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.mulp-if-maximal-less-that-maximal set-mset-empty)

thus *False*
using $E\text{-}in\ nbex\text{-}gt\text{-}D$ **by** *metis*
qed

hence $\neg (\exists A \mid \in \mid atm\text{-}of\text{-}cls (N \mid \cup \mid U_{er}). A \mid \notin \mid trail\text{-}atms \Gamma')$
using *step-hyps*
by (*metis atm-of-uminus finsert-iff fst-conv linorder-trm.antisym-conv3 trail-atms.simps(2)*)

then show $atms\text{-}of\text{-}cls (N \mid \cup \mid U_{er}) \mid \subseteq \mid trail\text{-}atms \Gamma'$
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 $\Gamma\text{-}deci\text{-}iff\text{-}neg \langle is\text{-}pos K \rangle$
by (*simp add:* $\langle \Gamma' = (- K, None) \# \Gamma \rangle is\text{-}pos\text{-}neg\text{-}not\text{-}is\text{-}pos$)

moreover have $trail\text{-}true\text{-}cls \Gamma' C$

if $L_n \in \text{set } \Gamma'$ **and** $\text{snd } L_n = \text{None}$ **and** $C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $C \prec_c \{\#fst L_n\# \}$
for $L_n C$
proof –
have $L_n = (- K, \text{None}) \vee L_n \in \text{set } \Gamma$
using $\langle L_n \in \text{set } \Gamma' \rangle$ **unfolding** $\langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle$ **by** *simp*

hence *trail-true-cls* ΓC
proof (*elim disjE*)
assume $L_n = (- K, \text{None})$

hence $\forall x \in \# C. x \prec_l - K$
using $\langle C \prec_c \{\#fst L_n\# \} \rangle$
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $C \preceq_c D$
using *D-max-lit step-hyps*
using *linorder-cls.leI that(3)* **by** *blast*

thus *trail-true-cls* ΓC
using $\langle C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$ *D-true*
using *clauses-lt-D-true* **by** *blast*
next
assume $L_n \in \text{set } \Gamma$

thus *trail-true-cls* ΓC
using Γ -*deci-ball-lt-true* $\langle \text{snd } L_n = \text{None} \rangle$ $\langle C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$
 $\langle C \prec_c \{\#fst L_n\# \} \rangle$
by *metis*
qed

thus *trail-true-cls* $\Gamma' C$
using $\langle \text{suffix } \Gamma \Gamma' \rangle$ **by** (*metis trail-true-cls-if-trail-true-suffix*)
qed

moreover have $\forall L_n \in \text{set } \Gamma'. \forall C. \text{snd } L_n = \text{Some } C \longrightarrow C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using Γ -*prop-in* **by** (*simp add: \langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle*)

moreover have $\forall L_n \in \text{set } \Gamma'. \forall C. \text{snd } L_n = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C (\text{fst } L_n)$
using Γ -*prop-greatest* **by** (*simp add: \langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle*)

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$
using Γ -*prop-almost-false*
unfolding $\langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle$
by (*metis (no-types, lifting) append-eq-Cons-conv list.inject option.discI prod.inject*)

moreover have $(\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C |\in| \text{iefac } \mathcal{F} \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
using $\Gamma\text{-prop-ball-lt-true}$
unfolding $\langle \Gamma' = (- K, \text{None}) \# \Gamma \rangle$
by $(\text{metis } D\text{-true clauses-lt-}D\text{-true linorder-cls.neq-iff list.inject step-hyps}(10))$
suffix-Cons
suffix-def)

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}, \mathcal{F}, \Gamma', \text{None}) \rangle$
proof $(\text{intro ord-res-}\gamma\text{-invars.intros})$
have $\text{trail-true-cls } \Gamma' C$
 $\wedge K_C. \text{linorder-lit.is-maximal-in-mset } C K_C \implies$
 $\neg (\exists A |\in| \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \text{trail-atms } \Gamma')$
if $C\text{-lt: } \wedge E. \text{None} = \text{Some } E \implies C \prec_c E$ **and** $C\text{-in: } C |\in| \text{iefac } \mathcal{F} \uparrow (N \mid \cup \mid U_{er})$ **for** C
proof –
have $\text{None} = \text{The-optional } (\text{linorder-cls.is-least-in-fset}$
 $(\text{ffilter } ((\prec_c) D) (\text{iefac } \mathcal{F} \uparrow (N \mid \cup \mid U_{er}))))$
using step-hyps
by $(\text{metis } (\text{no-types, opaque-lifting}) \text{Some-eq-The-optional}D$
 $\text{linorder-cls.is-least-in-ffilter-iff not-Some-eq})$

hence $C \preceq_c D$
using $\text{step-hyps that by } (\text{metis clause-le-if-lt-least-greater})$

hence $C \prec_c D \vee C = D$
by simp

hence $\text{trail-true-cls } \Gamma C$
proof (elim disjE)
assume $C \prec_c D$
thus $\text{trail-true-cls } \Gamma C$
using $\text{clauses-lt-}D\text{-true } \langle C |\in| \text{iefac } \mathcal{F} \uparrow (N \mid \cup \mid U_{er}) \rangle$ **by** metis

next
assume $C = D$
thus $\text{trail-true-cls } \Gamma C$
using $D\text{-true by argo}$

qed

thus $\text{trail-true-cls } \Gamma' C$
using $\langle \text{suffix } \Gamma \Gamma' \rangle$ **by** $(\text{metis trail-true-cls-if-trail-true-suffix})$

fix K_C
assume $C\text{-max-lit: linorder-lit.is-maximal-in-mset } C K_C$
thus $\neg (\exists A |\in| \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin \text{trail-atms } \Gamma')$
using $\langle C \prec_c D \vee C = D \rangle$

proof (*elim disjE*)
assume $C \prec_c D$
hence $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin | \text{trail-atms } \Gamma)$
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 finset-iff linorder-cls.order.irrefl linorder-lit.antisym-conv3 linorder-lit.multip-if-maximal-less-that-maximal step-hyps(5) step-hyps(9) trail-atms.simps(2)*)
qed
qed

thus
 $\forall C | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). (\forall D. \text{None} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow \text{trail-true-clss } \Gamma' C$
 $\forall C | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). (\forall D. \text{None} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$
 $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma))$
unfolding *atomize-conj* **by** *metis*
qed *simp-all*
next
case *step-hyps: (production $\Gamma D K U_{er} \Gamma' C' \mathcal{F}$)*

note $D\text{-max-lit} = \langle \text{ord-res.is-maximal-lit } K D \rangle$

have *suffix $\Gamma \Gamma'$*
using *step-hyps* **by** (*simp add: suffix-def*)

have
 $\mathcal{F}\text{-subset: } \mathcal{F} \subseteq N \cup U_{er}$ **and**
 $D\text{-in: } D \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
 $\forall C \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-clss } \Gamma \{ \#K \in \# C. K \neq L_C \# \} \wedge$
*is-pos } L_C)) **and**
clauses-lt-D-true: $\forall C \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow \text{trail-true-clss } \Gamma C$ **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma))$*

and
 $\Gamma\text{-sorted: sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
 $\Gamma\text{-lower: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \cup U_{er}))$

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**
 $\Gamma-deci-iff-neg: \forall Ln \in set \Gamma. snd Ln = None \longleftrightarrow is-neg (fst Ln)$ **and**
 $\Gamma-deci-ball-lt-true: \forall Ln \in set \Gamma. snd Ln = None \longrightarrow$
 $(\forall C \in | iefac \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c \{\#fst Ln\} \longrightarrow trail-true-cls \Gamma C)$
and
 $\Gamma-prop-in: \forall Ln \in set \Gamma. \forall C. snd Ln = Some C \longrightarrow C \in | iefac \mathcal{F} \uparrow (N \cup U_{er})$ **and**
 $\Gamma-prop-greatest:$
 $\forall Ln \in set \Gamma. \forall C. snd Ln = Some C \longrightarrow linorder-lit.is-greatest-in-mset C (fst Ln)$ **and**
 $\Gamma-prop-almost-false:$
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, Some C) \# \Gamma_0 \longrightarrow trail-false-cls \Gamma_0 \{\#K \in \# C. K \neq L\# \}$ **and**
 $\Gamma-prop-ball-lt-true: (\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, Some D) \# \Gamma_0 \longrightarrow$
 $(\forall C \in | iefac \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow trail-true-cls \Gamma_0 C))$
using invar by ($simp-all add: \langle s = (U_{er}, \mathcal{F}, \Gamma, Some D) \rangle ord-res-7-invars-def$)
have clauses-lt-D-almost-defined: $\forall C \in | iefac \mathcal{F} \uparrow (N \cup U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. linorder-lit.is-maximal-in-mset C K \longrightarrow trail-defined-cls \Gamma \{\#L \in \# C. L \neq K\# \})$
using invar[$unfolded \langle s = (U_{er}, \mathcal{F}, \Gamma, Some D) \rangle$] $clause-almost-defined-if-lt-next-clause$
by simp

have $K \in \# D$
using $\langle ord-res.is-maximal-lit K D \rangle$
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 $atm-of K \notin | trail-atms \Gamma$
using $\langle \neg trail-defined-lit \Gamma K \rangle$
by ($metis trail-defined-lit-iff-trail-defined-atm$)

hence $trail-atms-lt: \forall A \in | trail-atms \Gamma. A \prec_t atm-of K$
using $trail-atms-le0 \langle ord-res.is-maximal-lit K D \rangle$
by ($metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset linorder-trm.antisym-conv1$)

have $\mathcal{F} \subseteq | N \cup U_{er}$
using $\mathcal{F}-subset$.

moreover have $\forall C'. C' = Some C' \longrightarrow C' \in | iefac \mathcal{F} \uparrow (N \cup U_{er})$
using $step-hyps(11)$ **by** ($metis linorder-cls.is-least-in-ffilter-iff Some-eq-The-optionalD$)

moreover have $trail-true-cls \Gamma' \{\#K \in \# C. K \neq L_C\# \} \wedge is-pos L_C$
if $C' = Some E$ **and**
 $C-in: C \in | iefac \mathcal{F} \uparrow (N \cup U_{er})$ **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-clc.is-least-in-fset* (*ffilter* (\prec_c) D) (*iefac* \mathcal{F} $|^{\dagger}$ (N $| \cup |$ U_{er}))) E
using $\langle C' = \text{Some } E \rangle$ *step-hyps* **by** (*metis Some-eq-The-optionalD*)
hence $C \prec_c D \vee C = D$
unfolding *linorder-clc.is-least-in-ffilter-iff*
using *C-lt* **by** (*metis C-in linorder-clc.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* $\langle \text{suffix } \Gamma \Gamma' \rangle$
using *trail-defined-lit-if-trail-defined-suffix* **by** *blast*

ultimately have *trail-true-clc* 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 $\langle \text{suffix } \Gamma \Gamma' \rangle$ *trail-true-clc-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* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x))$ Γ'
proof –
have $x \prec_t \text{atm-of } K$ **if** x -in: $x \in |$ *trail-atms* Γ **for** x
using x -in *trail-atms-lt* **by** *metis*

hence $\forall x \in \text{set } \Gamma. \text{atm-of } (fst x) \prec_t \text{atm-of } K$
by (*simp add: fset-trail-atms*)

thus *?thesis*
using Γ -sorted
by (*simp add:* $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$)
qed

moreover have Γ' -lower: *linorder-trm.is-lower-fset* (*trail-atms* Γ') (*atms-of-clss* ($N \mid \cup \mid U_{er}$))
proof –
have *linorder-trm.is-lower-set* (*insert* (*atm-of* K) (*fset* (*trail-atms* Γ)))
(*fset* (*atms-of-clss* ($N \mid \cup \mid U_{er}$)))
proof (*rule linorder-trm.is-lower-set-insertI*)
show *atm-of* $K \mid \in \mid$ *atms-of-clss* ($N \mid \cup \mid U_{er}$)
using $\langle \text{atm-of } K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$.
next
show $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). w \prec_t (\text{atm-of } K) \longrightarrow w \mid \in \mid \text{trail-atms}$
 Γ
using *step-hyps(5)[unfolded linorder-trm.is-least-in-filter-iff, simplified]*
by *fastforce*
next
show *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* ($N \mid \cup \mid U_{er}$))
using Γ -lower .
qed

moreover have *trail-atms* $\Gamma' = \text{finsert}$ (*atm-of* K) (*trail-atms* Γ)
by (*simp add: $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$*)

ultimately show *?thesis*
by *simp*
qed

moreover have *trail-atms* $\Gamma' = \text{atms-of-clss}$ ($N \mid \cup \mid U_{er}$) **if** $C' = \text{None}$
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: $\neg (\exists E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). D \prec_c E)$*
using *step-hyps $\langle C' = \text{None} \rangle$*
by (*metis clause-le-if-lt-least-greater linorder-clss.leD option.simps(3)*)

have $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \text{atm-of } K \prec_t A)$
proof (*intro notI , elim bexE*)
fix $A :: 'f \text{ gterm}$
assume $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and** *atm-of* $K \prec_t A$

hence $A \mid \in \mid \text{atms-of-clss } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}))$
by *simp*

then obtain $E \ L$ **where** *E-in: $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$* **and** $L \in \# E$
and $A = \text{atm-of } L$
unfolding *atms-of-clss-def atms-of-clss-def*
by (*smt (verit, del-insts) fimage-iff fmember-ffUnion-iff fset-fset-mset*)

have $K \prec_l L$
using $\langle \text{atm-of } K \prec_t A \rangle \langle A = \text{atm-of } L \rangle$

by (*cases K; cases L*) *simp-all*
hence $D \prec_c E$
using *D-max-lit* $\langle L \in\# E \rangle$
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.mulp-if-maximal-less-that-maximal set-mset-empty)
thus *False*
using *E-in nbex-gt-D* **by** *metis*
qed
hence $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A | \notin | \text{trail-atms } \Gamma')$
using *step-hyps*
by (*metis finsert-iff fst-conv linorder-trm.antisym-conv3 trail-atms.simps(2)*)
then show $\text{atms-of-clss } (N \cup U_{er}) \subseteq | \text{trail-atms } \Gamma'$
by *blast*
qed
moreover have $\forall D. C' = \text{Some } D \longrightarrow (\forall A | \in | \text{trail-atms } \Gamma').$
 $\exists L. \text{ord-res.is-maximal-lit } L D \wedge A \preceq_t \text{atm-of } L$
proof (*intro allI impI ballI*)
fix $E :: 'f \text{ gterm literal multiset}$ **and** $A :: 'f \text{ gterm}$
assume $C' = \text{Some } E$ **and** $A | \in | \text{trail-atms } \Gamma'$
have $\text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) D) (\text{iefac } \mathcal{F} | \uparrow | (N \cup U_{er}))) E$
using *step-hyps(11)* $\langle C' = \text{Some } E \rangle$ **by** (*metis Some-eq-The-optionalD*)
hence $D \prec_c E$
unfolding *linorder-cls.is-least-in-ffilter-iff* **by** *argo*
obtain L **where** $\text{linorder-lit.is-maximal-in-mset } E L$
by (*metis* $\langle D \prec_c E \rangle$ *linorder-cls.leD linorder-lit.ex-maximal-in-mset mempty-lesseq-cls*)
show $\exists L. \text{ord-res.is-maximal-lit } L E \wedge A \preceq_t \text{atm-of } L$
proof (*intro exI conjI*)
show $\text{ord-res.is-maximal-lit } L E$
using $\langle \text{ord-res.is-maximal-lit } L E \rangle$.
next
have $K \preceq_l L$
using *step-hyps(4)* $\langle \text{ord-res.is-maximal-lit } L E \rangle$
by (*metis* $\langle D \prec_c E \rangle$ *linorder-cls.less-asymlinorder-lit.leI*
linorder-lit.mulp-if-maximal-less-that-maximal)
hence $\text{atm-of } K \preceq_t \text{atm-of } L$
by (*cases K; cases L*) *simp-all*
moreover have $A \preceq_t \text{atm-of } K$

proof –
have $A = \text{atm-of } K \vee A \mid \in \mid \text{ trail-atms } \Gamma$
using $\langle A \mid \in \mid \text{ trail-atms } \Gamma' \rangle$
unfolding $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$
by *simp*

thus $A \preceq_t \text{atm-of } K$
using *trail-atms-lt* **by** *blast*

qed

ultimately show $A \preceq_t \text{atm-of } L$
by *order*

qed

moreover have $\forall Ln \in \text{set } \Gamma'. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$
unfolding $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$
using $\Gamma\text{-deci-iff-neg } \langle \text{is-pos } K \rangle$ **by** *simp*

moreover have *trail-true-cls* $\Gamma' C$
if $Ln \in \text{set } \Gamma'$ **and** $\text{snd } Ln = \text{None}$ **and** $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and** $C \prec_c \{ \# \text{fst } Ln \# \}$
for $Ln C$

proof –
have $Ln = (K, \text{Some } D) \vee Ln \in \text{set } \Gamma$
using $\langle Ln \in \text{set } \Gamma' \rangle$ **unfolding** $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$ **by** *simp*

hence *trail-true-cls* ΓC

proof (*elim disjE*)
assume $Ln = (K, \text{Some } D)$
hence *False*
using $\langle \text{snd } Ln = \text{None} \rangle$ **by** *simp*
thus *?thesis ..*

next

assume $Ln \in \text{set } \Gamma$

thus *trail-true-cls* ΓC
using $\Gamma\text{-deci-ball-lt-true } \langle \text{snd } Ln = \text{None} \rangle \langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$
 $\langle C \prec_c \{ \# \text{fst } Ln \# \} \rangle$
by *metis*

qed

thus *trail-true-cls* $\Gamma' C$
using $\langle \text{suffix } \Gamma \Gamma' \rangle$ **by** (*metis trail-true-cls-if-trail-true-suffix*)

qed

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using $\Gamma\text{-prop-in step-hyps}(10) \langle D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$ **by** *simp*

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C \text{ (fst } Ln)$

using $\Gamma\text{-prop-greatest step-hyps}(8,9,10)$ **by** *simp*

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C. K \neq L \# \}$

unfolding $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$

using $\Gamma\text{-prop-almost-false } \langle \text{trail-false-cls } \Gamma \{ \#x \in \# D. x \neq K \# \} \rangle$

by (*metis (no-types, lifting) append-eq-Cons-conv list.inject option.inject prod.inject*)

moreover have $(\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow (\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$

proof –

have $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma C$

proof (*intro ballI impI*)

fix $C :: 'f \text{ gterm literal multiset}$

assume $C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $C \prec_c D$

thus $\text{trail-true-cls } \Gamma C$

using *clauses-lt-D-true by metis*

qed

thus *?thesis*

unfolding $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$

by (*smt (verit, ccfv-SIG) $\Gamma\text{-prop-ball-lt-true append-eq-Cons-conv list.inject option.inject prod.inject}$*)

qed

ultimately show *?thesis*

unfolding $\langle s' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$

proof (*intro ord-res-7-invars.intros*)

have $\text{trail-true-cls } \Gamma' C$

$\wedge K_C. \text{linorder-lit.is-maximal-in-mset } C K_C \implies$

$\neg (\exists A \in | \text{atms-of-cls } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin | \text{trail-atms } \Gamma')$

if $C\text{-lt}: \wedge E. C' = \text{Some } E \implies C \prec_c E$ **and** $C\text{-in}: C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **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 $\text{trail-true-cls } \Gamma' C$

proof (*elim disjE*)

assume $C \prec_c D$

hence $\text{trail-true-cls } \Gamma C$


```

using clauses-lt-D-true  $\langle C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$  by metis

thus trail-true-cls  $\Gamma' C$ 
  using  $\langle \text{suffix } \Gamma \Gamma' \rangle$  by (metis trail-true-cls-if-trail-true-suffix)
next
assume  $C = D$ 

have trail-true-lit  $\Gamma' K$ 
  using  $\langle \Gamma' = (K, \text{Some } D) \# \Gamma \rangle$   $\langle \text{is-pos } K \rangle$ 
  unfolding trail-true-lit-def by (cases K) simp-all

thus trail-true-cls  $\Gamma' C$ 
  unfolding  $\langle C = D \rangle$  trail-true-cls-def
  using  $\langle K \in \# D \rangle$  by metis
qed

fix  $K_C$ 
assume C-max-lit: linorder-lit.is-maximal-in-mset  $C K_C$ 
show  $\neg (\exists A \in | \text{atms-of-cls} (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin | \text{trail-atms}$ 
 $\Gamma')$ 
  using  $\langle C \prec_c D \vee C = D \rangle$ 
proof (elim disjE)
  assume  $C \prec_c D$ 
hence  $\neg (\exists A \in | \text{atms-of-cls} (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \notin | \text{trail-atms}$ 
 $\Gamma)$ 
  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  $\langle \text{suffix } \Gamma \Gamma' \rangle$  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
 $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow \text{trail-true-cls}$ 
 $\Gamma' C$ 
 $\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). (\forall D. C' = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$ 
 $(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$ 
 $\neg (\exists A \in | \text{atms-of-cls} (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma'))$ 
unfolding atomize-conj by metis
qed simp-all
next
case step-hyps: (factoring  $\Gamma D K U_{er} \mathcal{F}' \mathcal{F})$ 

note D-max-lit =  $\langle \text{ord-res.is-maximal-lit } K D \rangle$ 

```

have
F-subset: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**
D-in: $D \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
 $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in\# C. K \neq L_C\} \wedge$
*is-pos } L_C)) **and**
clauses-lt-D-true: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma$
C **and**
no-undef-atm-lt-max-lit-if-lt-D: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$
 $\neg (\exists A \mid\in\mid \text{atms-of-clss } (N \mid\cup\mid U_{er}). A \prec_t \text{atm-of } K \wedge A \mid\notin\mid \text{trail-atms } \Gamma))$
and
 Γ -sorted: *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
 Γ -lower: *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid\cup\mid U_{er}))$
and
trail-atms-le0: $\forall A \mid\in\mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L D \wedge (A \preceq_t$
*atm-of } L) **and**
 Γ -*deci-iff-neg*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$ **and**
 Γ -*deci-ball-lt-true*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c \{\#fst Ln\} \longrightarrow \text{trail-true-cls } \Gamma C)$
and
 Γ -*prop-in*: $\forall Ln \in \text{set } \Gamma. \forall D. \text{snd } Ln = \text{Some } D \longrightarrow D \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid$
U_{er}) **and**
 Γ -*prop-greatest*:
 $\forall Ln \in \text{set } \Gamma. \forall D. \text{snd } Ln = \text{Some } D \longrightarrow \text{linorder-lit.is-greatest-in-mset } D$
(fst Ln) **and**
 Γ -*prop-almost-false*:
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in\# C.$
K $\neq L\#\}$ **and**
 Γ -*prop-ball-lt-true*: $(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$
using invar by (*simp-all add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle \text{ord-res-}\gamma\text{-invars-def}$)

have *clauses-lt-D-almost-defined*: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{\#L \in\# C. L$
 $\neq K\#\})$
using invar[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$] *clause-almost-defined-if-lt-next-clause*
by *simp*

have *atm-of K* $\mid\notin\mid \text{trail-atms } \Gamma$
using $\langle \neg \text{trail-defined-lit } \Gamma K \rangle$
by (*metis trail-defined-lit-iff-trail-defined-atm*)

hence *trail-atms-lt*: $\forall A \mid\in\mid \text{trail-atms } \Gamma. A \prec_t \text{atm-of } K$
using *trail-atms-le0* $\langle \text{ord-res.is-maximal-lit } K D \rangle$
by (*metis Uniq-D linorder-lit.Uniq-is-maximal-in-mset linorder-trm.antisym-conv1*)**

have $efac\ D \neq D$
using $\langle \neg\ ord\text{-res.is-strictly-maximal-lit}\ K\ D \rangle\ D\text{-max-lit}\ \langle is\text{-pos}\ K \rangle$
by $(metis\ ex1\text{-efac-eq-factoring-chain}\ ex\text{-ground-factoringI}\ is\text{-pos-def})$

hence $efac\ D \prec_c\ D$
by $(metis\ efac\text{-properties-if-not-ident}(1))$

hence $D\text{-in-strong}: D \in\ N \cup\ U_{er}$ **and** $D \notin\ \mathcal{F}$
using $D\text{-in}\ \langle efac\ D \neq D \rangle$
unfolding $atomize\text{-conj}\ iefac\text{-def}$
by $(smt\ (verit)\ factorizable\text{-if-neq-efac}\ fimage\text{-iff}\ iefac\text{-def}\ ex1\text{-efac-eq-factoring-chain})$

have $\mathcal{F}' \subseteq\ N \cup\ U_{er}$
unfolding $\langle \mathcal{F}' = finsert\ D\ \mathcal{F} \rangle$
using $\mathcal{F}\text{-subset}\ D\text{-in-strong}\ \text{by}\ simp$

moreover have $\forall C'. Some\ (efac\ D) = Some\ C' \longrightarrow C' \in\ iefac\ \mathcal{F}' \mid\ (N \cup\ U_{er})$
proof –
have $efac\ D \in\ iefac\ \mathcal{F}' \mid\ (N \cup\ U_{er})$
using $D\text{-in-strong}\ \text{by}\ (simp\ add:\ iefac\text{-def}\ \langle \mathcal{F}' = finsert\ D\ \mathcal{F} \rangle)$

thus $?thesis$
by $simp$
qed

moreover have $trail\text{-true-cl}\ \Gamma\ \{\#K \in\# C. K \neq L_C\# \} \wedge is\text{-pos}\ L_C$
if $Some\ (efac\ D) = Some\ E$ **and**
 $C\text{-in}: C \in\ iefac\ \mathcal{F}' \mid\ (N \cup\ U_{er})$ **and**
 $C\text{-lt}: C \prec_c\ E$ **and**
 $C\text{-max-lit}: linorder\text{-lit.is-maximal-in-mset}\ C\ L_C$ **and**
 $L_C\text{-undef}: \neg\ trail\text{-defined-lit}\ \Gamma\ L_C$
for $E\ C\ L_C$

proof –
have $E = efac\ D$
using $that\ \text{by}\ simp$
hence $C \prec_c\ efac\ D$
using $C\text{-lt}\ \text{by}\ order$
hence $C \neq efac\ D$
by $order$
hence $C \in\ iefac\ \mathcal{F}' \mid\ (N \cup\ U_{er})$
using $C\text{-in}\ iefac\text{-def}\ step\text{-hyps}(10)\ \text{by}\ auto$
moreover have $C \prec_c\ D$
using $\langle C \prec_c\ efac\ D \rangle\ \langle efac\ D \prec_c\ D \rangle\ \text{by}\ order$
ultimately show $trail\text{-true-cl}\ \Gamma\ \{\#K \in\# C. K \neq L_C\# \} \wedge is\text{-pos}\ L_C$
using $clauses\text{-lt-D-seldomly-have-undef-max-lit}\ C\text{-max-lit}\ L_C\text{-undef}\ \text{by}\ metis$
qed

moreover have *trail-true-cls* ΓC
 $\bigwedge K_C. \text{linorder-lit.is-maximal-in-mset } C K_C \implies \text{trail-defined-cls } \Gamma \{\#L \in \# C. L \neq K_C\# \}$
if *C-lt*: $\bigwedge E. \text{Some } (efac D) = \text{Some } E \implies C \prec_c E$ **and** *C-in*: $C \in |iefac \mathcal{F}'| \uparrow (N \cup U_{er})$ **for** C
proof –
have $C \prec_c efac D$
using *C-lt* **by** *metis*

hence $C \neq efac D$
by *order*

hence $C \in |iefac \mathcal{F}'| \uparrow (N \cup U_{er})$
using *C-in* **by** (*auto simp*: $\langle \mathcal{F}' = \text{finsert } D \mathcal{F} \rangle$ *iefac-def*)

moreover have $C \prec_c D$
using $\langle C \prec_c efac D \rangle \langle efac D \prec_c D \rangle$ **by** *order*

ultimately show *trail-true-cls* ΓC
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* $\langle C \in |iefac \mathcal{F}'| \uparrow (N \cup U_{er}) \rangle \langle C \prec_c D \rangle$
C-max-lit
by *metis*
qed

moreover have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
using Γ -*sorted* .

moreover have *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma) (\text{atms-of-cls } (N \cup U_{er}))$
using Γ -*lower* .

moreover have $\forall D'. \text{Some } (efac D) = \text{Some } D' \longrightarrow (\forall A \in |trail-atms \Gamma|. \exists L. \text{ord-res.is-maximal-lit } L D' \wedge A \preceq_t \text{atm-of } L)$
proof –
have $\forall A \in |trail-atms \Gamma|. \exists L. \text{ord-res.is-maximal-lit } L (efac D) \wedge A \preceq_t \text{atm-of } L$
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 $\forall Ln \in set \Gamma. snd Ln = None \longleftrightarrow is-neg (fst Ln)$
using Γ -*deci-iff-neg* .

moreover have *trail-true-cls* ΓC
if $Ln \in set \Gamma$ **and** $snd Ln = None$ **and** $C \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er})$ **and** $C \prec_c \{\#fst Ln\# \}$
for $Ln C$
proof –
have $C = efac D \vee C \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er})$
using $\langle C \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er}) \rangle$ **by** (*auto simp: iefac-def* $\langle \mathcal{F}' = finsert D \mathcal{F} \rangle$)

thus *trail-true-cls* ΓC
proof (*elim disjE*)
assume $C = efac D$

hence *linorder-lit.is-greatest-in-mset* $C K$
using *D-max-lit* $\langle is-pos K \rangle$
by (*metis greatest-literal-in-efacI*)

hence $K \prec_l fst Ln$
using $\langle C \prec_c \{\#fst Ln\# \} \rangle$
by (*simp add: linorder-lit.is-greatest-in-mset-iff*)

hence *atm-of* $K \preceq_t$ *atm-of* $(fst Ln)$
by (*cases K; cases fst Ln*) *simp-all*

moreover have *atm-of* $(fst Ln) \in |trail-atms \Gamma$
using $\langle Ln \in set \Gamma \rangle$ **by** (*simp add: fset-trail-atms*)

moreover have *atm-of* $K \in |atms-of-cls (N \cup U_{er})$
by (*meson D-in D-max-lit atm-of-in-atms-of-clsI linorder-lit.is-maximal-in-mset-iff*)

ultimately have *atm-of* $K \in |trail-atms \Gamma$
using Γ -*lower*
unfolding *linorder-trm.is-lower-set-iff*
by *fastforce*

hence *False*
using $\langle atm-of K \notin |trail-atms \Gamma \rangle$ **by** *contradiction*

thus *trail-true-cls* ΓC ..

next

assume $C \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er})$

thus *trail-true-cls* ΓC

using Γ -*deci-ball-lt-true* $\langle Ln \in set \Gamma \rangle$ $\langle snd Ln = None \rangle$ $\langle C \prec_c \{\#fst Ln\# \} \rangle$
by *metis*

qed

qed

moreover have $C \in | \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ **if** $Ln \in \text{set } \Gamma$ **and** $\text{snd } Ln = \text{Some } C$ **for** $Ln \ C$

proof –

have $\text{atm-of } (fst \ Ln) \prec_t \text{atm-of } K$
using $\text{trail-atms-lit}[\text{unfolded } \text{fset-trail-atms}, \text{simplified}] \langle Ln \in \text{set } \Gamma \rangle$ **by** metis

hence $\text{atm-of } (fst \ Ln) \neq \text{atm-of } K$
by order

hence $\text{fst } Ln \neq K$
by $(\text{cases } \text{fst } Ln; \text{cases } K) \text{ simp-all}$

moreover have $\text{ord-res.is-maximal-lit } (fst \ Ln) \ C$
using $\Gamma\text{-prop-greatest } \langle Ln \in \text{set } \Gamma \rangle \langle \text{snd } Ln = \text{Some } C \rangle$ **by** blast

ultimately have $C \neq D$
using $\langle \text{ord-res.is-maximal-lit } K \ D \rangle$ **by** $(\text{metis } \text{Uniq-D linorder-lit.Uniq-is-maximal-in-mset})$

moreover have $C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using $\Gamma\text{-prop-in } \langle Ln \in \text{set } \Gamma \rangle \langle \text{snd } Ln = \text{Some } C \rangle$ **by** metis

ultimately show $?thesis$
by $(\text{auto simp: } \langle \mathcal{F}' = \text{finsert } D \ \mathcal{F} \rangle \text{iefac-def})$

qed

moreover have $\forall Ln \in \text{set } \Gamma. \forall D. \text{snd } Ln = \text{Some } D \longrightarrow \text{linorder-lit.is-greatest-in-mset } D \ (fst \ Ln)$
using $\Gamma\text{-prop-greatest}$ **by** simp

moreover have $\forall \Gamma_1 \ L \ C \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } C) \ # \ \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \ \{\#K \in \# \ C. \ K \neq L\# \}$
using $\Gamma\text{-prop-almost-false}$.

moreover have $(\forall \Gamma_1 \ L \ D \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \text{Some } D) \ # \ \Gamma_0 \longrightarrow (\forall C \in | \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \ C \prec_c \ D \longrightarrow \text{trail-true-cls } \Gamma_0 \ C))$

proof $(\text{intro allI impI ballI})$

fix

$\Gamma_1 \ \Gamma_0 :: ('f \ \text{gliteral} \times 'f \ \text{gclause option}) \ \text{list}$ **and**
 $L :: 'f \ \text{gliteral}$ **and**
 $C_0 \ C_1 :: 'f \ \text{gclause}$

assume

$\Gamma\text{-eq: } \Gamma = \Gamma_1 \ @ \ (L, \text{Some } C_1) \ # \ \Gamma_0$ **and**
 $C_0\text{-in: } C_0 \in | \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $C_0 \prec_c \ C_1$

have $C_0 = \text{efac } D \vee C_0 \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using $C_0\text{-in}$ **by** $(\text{auto simp: } \text{iefac-def } \langle \mathcal{F}' = \text{finsert } D \ \mathcal{F} \rangle)$

thus *trail-true-cls* Γ_0 C_0
proof (*elim disjE*)
assume $C_0 = \text{efac } D$

have *atm-of* $L \mid \in \mid$ *trail-atms* Γ
using $\Gamma\text{-eq}$ **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 $\Gamma\text{-eq}$ $\Gamma\text{-prop-greatest}$ **by** *simp*

moreover have *linorder-lit.is-greatest-in-mset* (*efac D*) K
using $\langle \text{is-pos } K \rangle$ *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.mulp-if-maximal-less-that-maximal)

hence *False*
using $\langle C_0 = \text{efac } D \rangle$ $\langle C_0 \prec_c C_1 \rangle$ **by** *order*

thus *?thesis ..*

next
assume $C_0 \mid \in \mid$ *iefac* \mathcal{F} $\mid \uparrow$ ($N \mid \cup \mid$ U_{er})
thus *?thesis*
using $\Gamma\text{-prop-ball-lt-true}$ $\Gamma\text{-eq}$ $\langle C_0 \prec_c C_1 \rangle$ **by** *metis*

qed
qed

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}, \mathcal{F}', \Gamma, \text{Some}(\text{efac } D)) \rangle$
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* Γ C
 $\bigwedge K_C. \text{linorder-lit.is-maximal-in-mset } C \ K_C \implies$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \not\in \mid \text{trail-atms } \Gamma)$
if *C-lt*: $\bigwedge E. \text{Some}(\text{efac } D) = \text{Some } E \implies C \prec_c E$ **and** *C-in*: $C \mid \in \mid$ *iefac* \mathcal{F}'
 $\mid \uparrow$ ($N \mid \cup \mid$ U_{er})
for C
proof –
have $C \prec_c$ *efac D*
using *C-lt* **by** *metis*

hence $C \neq \text{efac } D$
by *order*

hence $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using $C\text{-in}$ **by** (*auto simp: $\langle \mathcal{F}' = \text{finsert } D \ \mathcal{F} \rangle \text{iefac-def}$*)

moreover have $C \prec_c D$
using $\langle C \prec_c \text{efac } D \rangle \langle \text{efac } D \prec_c D \rangle$ **by order**

ultimately show $\text{trail-true-clcs } \Gamma \ C$
using clauses-lt-D-true **by metis**

fix K_C
assume $C\text{-max-lit: linorder-lit.is-maximal-in-mset } C \ K_C$
thus $\neg (\exists A \mid \in \mid \text{atms-of-clcs } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K_C \wedge A \mid \notin \mid \text{trail-atms } \Gamma)$
using $\text{no-undef-atm-lt-max-lit-if-lt-D}$ $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle \langle C \prec_c D \rangle$ **by metis**
qed

thus
 $\forall C \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall Da. \text{Some } (\text{efac } D) = \text{Some } Da \longrightarrow C \prec_c Da) \longrightarrow$
 $\text{trail-true-clcs } \Gamma \ C$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er}). (\forall Da. \text{Some } (\text{efac } D) = \text{Some } Da \longrightarrow C \prec_c Da) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K \ C \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clcs } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma))$
unfolding atomize-conj **by metis**
qed simp-all

next
case $\text{step-hyps: } (\text{resolution-bot } \Gamma \ E \ K \ D \ U_{er}' \ U_{er} \ \Gamma' \ \mathcal{F})$

have
 $\mathcal{F}\text{-subset: } \mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$ **and**
 $E\text{-in: } E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
 $\text{clauses-lt-E-seldomly-have-undef-max-lit:}$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C \ L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma \ L_C \longrightarrow (\text{trail-true-clcs } \Gamma \ \{\#K \in \# \ C. K \neq L_C\# \} \wedge$
 $\text{is-pos } L_C))$ **and**
 $\text{clauses-lt-E-true: } \forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow \text{trail-true-clcs } \Gamma \ C$ **and**
 $\text{no-undef-atm-lt-max-lit-if-lt-E: } \forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clcs } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma))$

and
 $\Gamma\text{-sorted: sorted-wrt } (\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma$ **and**
 $\Gamma\text{-lower: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) \ (\text{atms-of-clcs } (N \mid \cup \mid U_{er}))$

and
 $\text{trail-atms-le0: } \forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L \ E \wedge (A \preceq_t$

atm-of L) **and**
 Γ -*deci-iff-neg*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$ **and**
 Γ -*deci-ball-lt-true*: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c \{\#\text{fst } Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma C)$
and
 Γ -*prop-in*: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 Γ -*prop-greatest*:
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
(*fst Ln*) **and**
 Γ -*prop-almost-false*:
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$ **and**
 Γ -*prop-ball-lt-true*: $\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C)$
using invar by (*simp-all add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle$ *ord-res-7-invars-def*)

have *clauses-lt-E-almost-defined*: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{\#L \in \# C. L \neq K\# \})$
using invar[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle$] *clause-almost-defined-if-lt-next-clause*
by *simp*

have $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}'$
unfolding $\langle U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \rangle$
using \mathcal{F} -*subset* **by** *blast*

moreover have $\forall C'. \text{Some } \{\#\} = \text{Some } C' \longrightarrow C' \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$
by (*simp add*: $\langle \text{eres } D E = \{\#\} \rangle \langle U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \rangle$)

moreover have *trail-true-cls* $\Gamma' \{\#K \in \# C. K \neq L_C\# \} \wedge \text{is-pos } L_C$
if *Some* $\{\#\} = \text{Some } E$ **and**
 C -*in*: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ **and**
 C -*lt*: $C \prec_c E$ **and**
 C -*max-lit*: *linorder-lit.is-maximal-in-mset* $C L_C$ **and**
 L_C -*undef*: $\neg \text{trail-defined-lit } \Gamma' L_C$
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-defined-cls* $\Gamma' \{\#L \in \# x. L \neq K_x\# \}$
if *Some* $\{\#\} = \text{Some } y$ **and** x -*in*: $x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ **and** $x \prec_c y$
and
 x -*max-lit*: *linorder-lit.is-maximal-in-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-wrt ($\lambda x y. atm-of (fst y) \prec_t atm-of (fst x)$) Γ'*
unfolding $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have *linorder-trm.is-lower-fset (trail-atms Γ') (atms-of-cls (N | \cup | U_{er}'))*
unfolding $\langle \Gamma' = [] \rangle$
by (*simp add: linorder-trm.is-lower-set-iff*)

moreover have $\forall D. Some \{\#\} = Some D \longrightarrow (\forall A | \in | trail-atms \Gamma'. \exists L. ord-res.is-maximal-lit L D \wedge A \preceq_t atm-of L)$
unfolding $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall Ln \in set \Gamma'. snd Ln = None \longleftrightarrow is-neg (fst Ln)$
unfolding $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall Ln \in set \Gamma'. snd Ln = None \longrightarrow (\forall C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}'). C \prec_c \{\#fst Ln\# \} \longrightarrow trail-true-cls \Gamma' C)$
using $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall Ln \in set \Gamma'. snd Ln = None \longrightarrow \neg(\exists C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}'). linorder-lit.is-maximal-in-mset C (- (fst Ln)))$
using $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall Ln \in set \Gamma'. \forall D. snd Ln = Some D \longrightarrow \neg(\exists C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}'). C \prec_c D \wedge fst Ln \in \# C)$
using $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall Ln \in set \Gamma'. \forall C. snd Ln = Some C \longrightarrow C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}')$
using $\langle \Gamma' = [] \rangle$ **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 $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, Some C) \# \Gamma_0 \longrightarrow trail-false-cls \Gamma_0 \{\#K \in \# C. K \neq L\# \}$
using $\langle \Gamma' = [] \rangle$ **by** *simp*

moreover have $\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, Some D) \# \Gamma_0 \longrightarrow (\forall C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}'). C \prec_c D \longrightarrow trail-true-cls \Gamma_0 C)$
using $\langle \Gamma' = [] \rangle$ **by** *simp*

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } \{\#\}) \rangle$
proof (*intro ord-res-7-invars.intros*)
have *trail-true-cls* $\Gamma' x$
 $\wedge K_x. \text{linorder-lit.is-maximal-in-mset } x K_x \implies$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}'). A \prec_t \text{atm-of } K_x \wedge A \mid \notin \mid \text{trail-atms } \Gamma')$
if *C-lt*: $\wedge E. \text{Some } \{\#\} = \text{Some } E \implies x \prec_c E$ **and** *C-in*: $x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N$
 $\mid \cup \mid U_{er}')$
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 $\Gamma' x$
 $\wedge K_x. \text{linorder-lit.is-maximal-in-mset } x K_x \implies$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}'). A \prec_t \text{atm-of } K_x \wedge A \mid \notin \mid \text{trail-atms}$
 $\Gamma')$
by *argo+*
qed

thus
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}'). (\forall D. \text{Some } \{\#\} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
trail-true-cls $\Gamma' C$
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}'). (\forall D. \text{Some } \{\#\} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}'). A \prec_t \text{atm-of } K \wedge A \mid \notin \mid \text{trail-atms } \Gamma'))$
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* = $\langle \text{ord-res.is-maximal-lit } L E \rangle$
note *eres-max-lit* = $\langle \text{ord-res.is-maximal-lit } K (\text{eres } D E) \rangle$

have
 \mathcal{F} -*subset*: $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}$ **and**
 E -*in*: $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
clauses-lt-D-seldomly-have-undef-max-lit:
 $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow$
 $(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in \# C. K \neq L_C \#\} \wedge$
 $\text{is-pos } L_C))$ **and**
clauses-lt-E-true: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow \text{trail-true-cls } \Gamma$
 C **and**
no-undef-atm-lt-max-lit-if-lt-E: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow$
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$

$\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin \text{trail-atms } \Gamma)$
and
 $\Gamma\text{-sorted: sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
 $\Gamma\text{-lower: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
and
 $\text{trail-atms-le0: } \forall A \mid \in \mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L E \wedge (A \preceq_t \text{atm-of } L)$ **and**
 $\Gamma\text{-deci-iff-neg: } \forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst Ln)$ **and**
 $\Gamma\text{-deci-ball-lt-true: } \forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c \{\#fst Ln\} \longrightarrow \text{trail-true-cls } \Gamma C)$
and
 $\Gamma\text{-prop-in: } \forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ **and**
 $\Gamma\text{-prop-greatest:}$
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
 $(fst Ln)$ **and**
 $\Gamma\text{-prop-almost-false:}$
 $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$ **and**
 $\Gamma\text{-prop-ball-lt-true: } \forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C)$
using invar by $(\text{simp-all add: } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle \text{ord-res-}\gamma\text{-invars-def})$

**have clauses-lt-E-almost-defined: } \forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c E \longrightarrow
 $(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{\#L \in \# C. L \neq K\# \})$
using invar $[\text{unfolded } \langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle]$ *clause-almost-defined-if-lt-next-clause*
by simp

have $\Gamma\text{-consistent: trail-consistent } \Gamma$
using *trail-consistent-if-sorted-wrt-atoms* $\Gamma\text{-sorted}$ **by metis**

have *trail-atms-le*: $\forall A \mid \in \mid \text{trail-atms } \Gamma. A \preceq_t \text{atm-of } L$
using *trail-atms-le0 E-max-lit*
by $(\text{metis } \text{Uniq-D linorder-lit.Uniq-is-maximal-in-mset})$

have $(- L, \text{Some } D) \in \text{set } \Gamma$
using $\langle \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \rangle$ **by** $(\text{metis } \text{map-of-SomeD})$

hence *D-in*: $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using $\Gamma\text{-prop-in}$ **by simp**

have *D-max-lit*: $\text{linorder-lit.is-greatest-in-mset } D (- L)$
using $\Gamma\text{-prop-greatest } \langle (- L, \text{Some } D) \in \text{set } \Gamma \rangle$ **by fastforce**

have *suffix* $\Gamma' \Gamma$
using *step-hyps(9) suffix-dropWhile* **by metis**

hence *atms-of-cls* $(\text{eres } D E) \mid \subseteq \mid \text{atms-of-cls } D \mid \cup \mid \text{atms-of-cls } E$**

using *lit-in-one-of-resolvents-if-in-eres*
unfolding *atms-of-cls-def* **by** *fastforce*

moreover have *atms-of-cls* $D \sqsubseteq$ *atms-of-cls* $(N \cup U_{er})$
using $\langle D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \cup U_{er}) \rangle$
by (*metis atms-of-cls-fimage-iefac atms-of-cls-finsert finsert-absorb funion-upper1*)

moreover have *atms-of-cls* $E \sqsubseteq$ *atms-of-cls* $(N \cup U_{er})$
using $\langle E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \cup U_{er}) \rangle$
by (*metis atms-of-cls-fimage-iefac atms-of-cls-finsert finsert-absorb funion-upper1*)

ultimately have *atms-of-cls* $(N \cup U_{er}) =$ *atms-of-cls* $(N \cup U_{er}')$
unfolding $\langle U_{er}' = \text{finsert } (eres D E) U_{er} \rangle$ *atms-of-cls-def* **by** *auto*

obtain A_L **where** *L-def*: $L = \text{Neg } A_L$
using $\langle \text{is-neg } L \rangle$ **by** (*cases L*) *simp-all*

have $D \prec_c E$
using *clause-lt-clause-if-max-lit-comp*
using *E-max-lit* $\langle \text{is-neg } L \rangle$ *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* $\langle \text{is-neg } L \rangle$ *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 $\neg L \notin\# E$
using *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*
using Γ -*consistent* $\langle \text{trail-false-cls } \Gamma E \rangle$ **by** *metis*

hence $\forall K \in\# \text{eres } D E. \text{atm-of } K \prec_t \text{atm-of } L$
using *lit-in-eres-lt-greatest-lit-in-greatest-resolvent*[*OF* $\langle \text{eres } D E \neq E \rangle$ *E-max-lit*]
by *metis*

hence $\forall K \in\# \text{eres } D E. K \neq L \wedge K \neq \neg L$
by *fastforce*

moreover have $\forall L \in\# \text{eres } D E. L \in\# D \vee L \in\# E$
using *lit-in-one-of-resolvents-if-in-eres* **by** *metis*

moreover have *D-almost-false*: *trail-false-cls* $\Gamma \{ \#K \in\# D. K \neq \neg L \# \}$
using *ord-res-7-invars-implies-propagated-clause-almost-false*
using $\langle (\neg L, \text{Some } D) \in \text{set } \Gamma \rangle$ *invar*[*unfolded* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } E) \rangle$]

by *metis*

ultimately have *trail-false-cls* Γ (*eres D E*)
using $\langle \text{trail-false-cls } \Gamma \ E \rangle$ **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* $\langle \text{eres D E} \prec_c E \rangle$ $\langle \text{trail-false-cls } \Gamma \ (\text{eres D E}) \rangle$ by *metis*

hence *eres D E* $|\notin|$ \mathcal{F}
using *F-subset* by *blast*

hence *iefac* \mathcal{F} (*eres D E*) = *eres D E*
by (*simp add: iefac-def*)

hence *iefac* \mathcal{F} $|\uparrow|$ (*N* $|\cup|$ *U_{er}'*) = *fininsert* (*eres D E*) (*iefac* \mathcal{F} $|\uparrow|$ (*N* $|\cup|$ *U_{er}*))
unfolding $\langle U_{er}' = \text{fininsert } (\text{eres D E}) \ U_{er} \rangle$ by *simp*

have *trail-false-lit* Γ *K*
by (*meson* $\langle \text{trail-false-cls } \Gamma \ (\text{eres D E}) \rangle$ *linorder-lit.is-maximal-in-mset-iff*
step-hyps(10)
trail-false-cls-def)

have *mem-set- Γ' -iff*: (*Ln* \in *set* Γ') = (\neg *atm-of K* \preceq_t *atm-of (fst Ln)*) \wedge *Ln* \in *set* Γ) for *Ln*

unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \ \Gamma \rangle$

proof (*rule mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*)

show *sorted-wrt* ($\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)$) Γ

using Γ -sorted .

next

show *monotone-on* (*set* Γ) ($\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)$) ($\lambda x y. y \leq x$)

($\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$)

by (*rule monotone-onI*) *auto*

qed

hence *atms-in- Γ' -lt-atm-K*: $\forall x |\in|$ *trail-atms* Γ' . $x \prec_t \text{atm-of } K$
by (*auto simp add: fset-trail-atms*)

have $\mathcal{F} |\subseteq|$ *N* $|\cup|$ *U_{er}'*
unfolding $\langle U_{er}' = \text{fininsert } (\text{eres D E}) \ U_{er} \rangle$
using *F-subset* by *blast*

moreover have $\forall C'. \text{Some } (\text{eres D E}) = \text{Some } C' \longrightarrow C' |\in|$ *iefac* \mathcal{F} $|\uparrow|$ (*N* $|\cup|$)

U_{er}')
proof –
have $eres\ D\ E\ |\in|\ iefac\ \mathcal{F}\ |^q\ (N\ |\cup|\ U_{er}')$
using $\langle iefac\ \mathcal{F}\ (eres\ D\ E) = eres\ D\ E \rangle$
by $(simp\ add: \langle U_{er}' = fininsert\ (eres\ D\ E)\ U_{er} \rangle)$

thus *?thesis*
by *simp*
qed

moreover have $sorted-wrt\ (\lambda x\ y.\ atm-of\ (fst\ y)\ \prec_t\ atm-of\ (fst\ x))\ \Gamma'$
using $\Gamma\text{-sorted}\ step-hyps(9)$ **by** $(metis\ sorted-wrt-dropWhile)$

moreover have $\Gamma'\text{-lower:}\ linorder-trm.is-lower-fset\ (trail-atms\ \Gamma')\ (atms-of-cls\ (N\ |\cup|\ U_{er}'))$
proof –
have $linorder-trm.is-lower-fset\ (trail-atms\ \Gamma')\ (trail-atms\ \Gamma)$
unfolding $linorder-trm.is-lower-set-iff$
proof $(intro\ conjI\ ballI\ impI)$
show $fset\ (trail-atms\ \Gamma') \subseteq fset\ (trail-atms\ \Gamma)$
unfolding $fset-trail-atms$ **using** $\langle suffix\ \Gamma'\ \Gamma \rangle$ **by** $(metis\ image-mono\ set-mono-suffix)$

next
obtain Γ'' **where** $\Gamma = \Gamma''\ @\ \Gamma'$
using $\langle suffix\ \Gamma'\ \Gamma \rangle$ **unfolding** $suffix-def$ **by** *metis*

fix $l\ x$
assume $l\ |\in|\ trail-atms\ \Gamma'$ **and** $x\ |\in|\ trail-atms\ \Gamma$ **and** $x\ \prec_t\ l$

have $x\ |\in|\ trail-atms\ \Gamma'' \vee x\ |\in|\ trail-atms\ \Gamma'$
using $\langle x\ |\in|\ trail-atms\ \Gamma \rangle$ **unfolding** $\langle \Gamma = \Gamma''\ @\ \Gamma' \rangle$ $fset-trail-atms$ **by** *auto*

thus $x\ |\in|\ trail-atms\ \Gamma'$
proof $(elim\ disjE)$
assume $x\ |\in|\ trail-atms\ \Gamma''$

hence $l\ \prec_t\ x$
using $\Gamma\text{-sorted}\ \langle l\ |\in|\ trail-atms\ \Gamma' \rangle$
unfolding $\langle \Gamma = \Gamma''\ @\ \Gamma' \rangle$ $sorted-wrt-append\ fset-trail-atms$ **by** *blast*

hence *False*
using $\langle x\ \prec_t\ l \rangle$ **by** *order*

thus $x\ |\in|\ trail-atms\ \Gamma' ..$
next
assume $x\ |\in|\ trail-atms\ \Gamma'$
thus $x\ |\in|\ trail-atms\ \Gamma'$.
qed
qed

thus *?thesis*
using Γ -lower
unfolding $\langle \text{atms-of-clss } (N \mid \cup \mid U_{er}) = \text{atms-of-clss } (N \mid \cup \mid U_{er}') \rangle$
by order
qed

moreover have $\forall DE. \text{Some } (\text{eres } D \ E) = \text{Some } DE \longrightarrow (\forall A \mid \in \mid \text{trail-atms } \Gamma'. \exists L. \text{ord-res.is-maximal-lit } L \ DE \wedge A \preceq_t \text{atm-of } L)$
using *atms-in- Γ' -lt-atm-K eres-max-lit* **by blast**

moreover have $\forall Ln \in \text{set } \Gamma'. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$
using Γ -*deci-iff-neg* $\langle \text{suffix } \Gamma' \ \Gamma \rangle$
by (*metis* (*no-types*, *opaque-lifting*) *in-set-conv-decomp suffixE suffix-appendD*)

moreover have *trail-true-clss* $\Gamma' \ C$
if $Ln \in \text{set } \Gamma'$ **and** $\text{snd } Ln = \text{None}$ **and** $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')$ **and** $C \prec_c \{\#\text{fst } Ln\#\}$
for $Ln \ C$
proof –
have $Ln \in \text{set } \Gamma$
using $\langle Ln \in \text{set } \Gamma' \rangle \langle \text{suffix } \Gamma' \ \Gamma \rangle$ *set-mono-suffix* **by blast**

have $C = \text{eres } D \ E \vee C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using $\langle C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}') \rangle$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})) \rangle$
by simp

thus *trail-true-clss* $\Gamma' \ C$
proof (*elim disjE*)
assume $C = \text{eres } D \ E$

hence $K \prec_l \text{fst } Ln$
using $\langle C \prec_c \{\#\text{fst } Ln\#\} \rangle$ *eres-max-lit*
by (*simp add: linorder-lit.is-maximal-in-mset-iff*)

hence $\text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$
by (*cases K; cases fst Ln*) *simp-all*

moreover have $\text{atm-of } (\text{fst } Ln) \mid \in \mid \text{trail-atms } \Gamma'$
using $\langle Ln \in \text{set } \Gamma' \rangle$ **by** (*simp add: fset-trail-atms*)

moreover have $\text{atm-of } K \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}')$
by (*metis* $\langle \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})) \rangle$)
atm-of-in-atms-of-clssI eres-max-lit finsert-iff linorder-lit.is-maximal-in-mset-iff)

ultimately have $\text{atm-of } K \mid \in \mid \text{trail-atms } \Gamma'$

using Γ' -lower
unfolding *linorder-trm.is-lower-set-iff*
by *fastforce*

moreover have *atm-of* $K \notin \text{trail-atms } \Gamma'$
using *atms-in- Γ' -lt-atm-K* **by** *blast*

ultimately show *trail-true-cls* $\Gamma' C$
by *contradiction*

next
assume $C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$

hence *trail-true-cls* ΓC
using Γ -*deci-ball-lt-true* $\langle Ln \in \text{set } \Gamma \rangle \langle \text{snd } Ln = \text{None} \rangle \langle C \prec_c \{\#fst Ln\# \}$
by *metis*

then obtain L_C **where** $L_C \in \# C$ **and** *trail-true-lit* ΓL_C
unfolding *trail-true-cls-def* **by** *auto*

hence $\forall x \in \# C. x \prec_l \text{fst } Ln$
using $\langle C \prec_c \{\#fst Ln\# \}$
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $L_C \prec_l \text{fst } Ln$
using $\langle L_C \in \# C \rangle$ **by** *metis*

hence *atm-of* $L_C \preceq_t \text{atm-of } (\text{fst } Ln)$
by (*cases* L_C ; *cases* $\text{fst } Ln$) *simp-all*

moreover have *atm-of* $(\text{fst } Ln) \prec_t \text{atm-of } K$
using *atms-in- Γ' -lt-atm-K*
by (*simp add: fset-trail-atms that(1)*)

ultimately have *atm-of* $L_C \prec_t \text{atm-of } K$
by *order*

have $L_C \in \text{fst ' set } \Gamma$
using $\langle \text{trail-true-lit } \Gamma L_C \rangle$
unfolding *trail-true-lit-def* .

hence $L_C \in \text{fst ' set } \Gamma'$
using *mem-set- Γ' -iff*
using $\langle \text{atm-of } L_C \prec_t \text{atm-of } K \rangle$ *linorder-trm.not-le* **by** *auto*

hence *trail-true-lit* $\Gamma' L_C$
unfolding *trail-true-lit-def* .

thus *trail-true-cls* $\Gamma' C$

using $\langle L_C \in \# C \rangle$
unfolding *trail-true-cls-def* **by** *auto*
qed
qed

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er} \wedge)$
using $\Gamma\text{-prop-in } \langle \text{suffix } \Gamma' \Gamma \rangle \text{ set-mono-suffix } \langle U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \rangle$
by *blast*

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C (\text{fst } Ln)$
using $\Gamma\text{-prop-greatest } \langle \text{suffix } \Gamma' \Gamma \rangle \text{ set-mono-suffix}$ **by** *blast*

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C. K \neq L \# \}$
using $\Gamma\text{-prop-almost-false } \langle \text{suffix } \Gamma' \Gamma \rangle$
by (*metis (no-types, lifting) append.assoc suffix-def*)

moreover have $\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow (\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er} \wedge). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C)$
proof (*intro allI impI ballI*)
fix
 $\Gamma_1 \Gamma_0 :: ('f \text{ gliteral } \times 'f \text{ gclause option}) \text{ list}$ **and**
 $L :: 'f \text{ gliteral}$ **and**
 $C_0 C_1 :: 'f \text{ gclause}$
assume
 $\Gamma'\text{-eq: } \Gamma' = \Gamma_1 @ (L, \text{Some } C_1) \# \Gamma_0$ **and**
 $C_0\text{-in: } C_0 \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er} \wedge)$ **and**
 $C_0 \prec_c C_1$

have $C_0 = \text{eres } D E \vee C_0 \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using $C_0\text{-in}$
unfolding $\langle \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er} \wedge) = \text{finsert } (\text{eres } D E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$
by *simp*

thus *trail-true-cls* $\Gamma_0 C_0$
proof (*elim disjE*)
assume $C_0 = \text{eres } D E$

have $\neg \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } (L, \text{Some } C_1))$ **and** $(L, \text{Some } C_1) \in \text{set } \Gamma$
unfolding *atomize-conj*
using $\Gamma'\text{-eq } \langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone[OF $\Gamma\text{-sorted mono-atms-lt}$]*
by (*metis in-set-conv-decomp*)

then have $\neg \text{atm-of } K \preceq_t \text{atm-of } L$

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* $\langle L, \text{Some } C_1 \rangle \in \text{set } \Gamma$ **by fastforce**
ultimately have *False*
using $\langle C_0 \prec_c C_1 \rangle$
by (*metis Neg-atm-of-iff* $\langle C_0 = \text{eres } D \ E \rangle$ *asymptD eres-max-lit*
linorder-lit.is-greatest-in-mset-iff-is-maximal-and-count-eq-one
linorder-lit.multip-if-maximal-less-than-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 $C_0 \in | \text{iefac } \mathcal{F} |^{\uparrow} (N \cup U_{er})$
thus trail-true-cls Γ_0 C_0
using Γ -*prop-ball-lt-true* Γ' -*eq* $\langle C_0 \prec_c C_1 \rangle$
by (*metis (no-types, opaque-lifting)* $\langle \text{suffix } \Gamma' \ \Gamma \rangle$ *append-assoc suffixE*)
qed
qed
ultimately show *?thesis*
unfolding $\langle s' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } (\text{eres } D \ E)) \rangle$
proof (*intro ord-res-7-invars.intros*)
have *clause-true-in- Γ' -if: trail-true-cls* $\Gamma' \ x$ **and**
 $\bigwedge K_x. \text{linorder-lit.is-maximal-in-mset } x \ K_x \implies$
 $\neg (\exists A \in | \text{atms-of-cls } (N \cup U_{er}') . A \prec_t \text{atm-of } K_x \wedge A \notin | \text{trail-atms } \Gamma' |)$
if x -*lt*: $\bigwedge DE. \text{Some } (\text{eres } D \ E) = \text{Some } DE \implies x \prec_c DE$ **and** x -*in*: $x \in |$
 $\text{iefac } \mathcal{F} |^{\uparrow} (N \cup U_{er}')$
for x
proof –
have $x \prec_c \text{eres } D \ E$
using x -*lt* **by** *metis*
hence $x \neq \text{eres } D \ E$
by *order*
hence x -*in'*: $x \in | \text{iefac } \mathcal{F} |^{\uparrow} (N \cup U_{er})$
using x -*in* $\langle \text{iefac } \mathcal{F} (\text{eres } D \ E) = \text{eres } D \ E \rangle$
by (*simp add*: $\langle U_{er}' = \text{finsert } (\text{eres } D \ E) \ U_{er} \rangle$)
have $x \prec_c E$
using $\langle x \prec_c \text{eres } D \ E \rangle \langle \text{eres } D \ E \prec_c E \rangle$ **by** *order*
have x -*true*: *trail-true-cls* $\Gamma \ x$

using *clauses-lt-E-true x-in'* $\langle x \prec_c E \rangle$ **by** *metis*

have $(- K, None) \in \text{set } \Gamma$
using $\langle \text{trail-false-lit } \Gamma K \rangle$
using $\langle \text{is-pos } K \rangle$
using $\Gamma\text{-deci-iff-neg}$
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** $L_x\text{-true}$: *trail-true-lit* ΓL_x
using $x\text{-true}$ **unfolding** *trail-true-cls-def* **by** *metis*

moreover have $L_x \neq K$
using $\Gamma\text{-consistent}$ $\langle \text{trail-false-cls } \Gamma (\text{eres } D E) \rangle$ $L_x\text{-true}$ *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* $\langle x \prec_c \text{eres } D E \rangle$ $\langle L_x \neq K \rangle$ $\langle L_x \in\# x \rangle$
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.multip-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 $\text{atm-of } L_x \neq \text{atm-of } K$
by (*simp add: atm-of-eq-atm-of*)

moreover have $L_x \preceq_l K$
proof (*rule linorder-lit.less-than-maximal-if-multip_{HO}[OF eres-max-lit - $\langle L_x \in\# x \rangle$]*)
show *multip_{HO}* $(\preceq_l) x (\text{eres } D E)$
using $\langle x \prec_c \text{eres } D E \rangle$
by (*simp add: multip-imp-multip_{HO}*)
qed

ultimately have $\text{atm-of } L_x \prec_t \text{atm-of } K$
by (*cases L_x; cases K*) *simp-all*

hence *trail-true-lit* $\Gamma' L_x$
unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$
using $L_x\text{-true}$
unfolding *trail-true-lit-def*
using *mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone[OF $\Gamma\text{-sorted}$*
mono-atms-lt]
by (*metis (no-types, lifting) image-iff linorder-trm.not-le*)

thus $trail\text{-}true\text{-}cls\ \Gamma' x$
using $\langle L_x \in\# x \rangle$ **unfolding** $trail\text{-}true\text{-}cls\text{-}def$ **by** $metis$

fix K_x
assume $x\text{-}max\text{-}lit: ord\text{-}res.is\text{-}maximal\text{-}lit\ K_x x$

hence $K_x \preceq_l K$
using $\langle x \prec_c eres\ D\ E \rangle$ $eres\text{-}max\text{-}lit$
using $linorder\text{-}lit.mul\text{-}t\text{-}p\text{-}if\text{-}maximal\text{-}less\text{-}that\text{-}maximal$ **by** $fastforce$

hence $atm\text{-}of\ K_x \preceq_t atm\text{-}of\ K$
by $(cases\ K_x; cases\ K)\ simp\text{-}all$

have $A \in | trail\text{-}atms\ \Gamma'$
if $A\text{-}lt: A \prec_t atm\text{-}of\ K_x$ **and** $A\text{-}in: A \in | atm\text{-}of\text{-}clss\ (N \cup U_{er})$ **for** A
unfolding $\langle \Gamma' = dropWhile\ (\lambda Ln. atm\text{-}of\ K \preceq_t atm\text{-}of\ (fst\ Ln))\ \Gamma \rangle$
proof $(rule\ in\text{-}trail\text{-}atms\text{-}dropWhileI)$
show $sorted\text{-}wrt\ (\lambda x y. atm\text{-}of\ (fst\ y) \prec_t atm\text{-}of\ (fst\ x))\ \Gamma$
using $\Gamma\text{-}sorted$.

next
show $monotone\text{-}on\ (set\ \Gamma)\ (\lambda x y. atm\text{-}of\ (fst\ y) \prec_t atm\text{-}of\ (fst\ x))\ (\lambda x y. y \leq x)\ (\lambda x. atm\text{-}of\ K \preceq_t atm\text{-}of\ (fst\ x))$
using $mono\text{-}atms\text{-}lt$.

next
show $\neg atm\text{-}of\ K \preceq_t A$
using $A\text{-}lt\ \langle atm\text{-}of\ K_x \preceq_t atm\text{-}of\ K \rangle$ **by** $order$

next
have $\neg (\exists A \in | atm\text{-}of\text{-}clss\ (N \cup U_{er}). A \prec_t atm\text{-}of\ K_x \wedge A \notin | trail\text{-}atms\ \Gamma)$
using $no\text{-}undef\text{-}atm\text{-}lt\text{-}max\text{-}lit\text{-}if\text{-}lt\text{-}E\ \langle x \in | iefac\ \mathcal{F} \mid \uparrow (N \cup U_{er}) \rangle\ \langle x \prec_c E \rangle\ x\text{-}max\text{-}lit$
by $metis$

thus $A \in | trail\text{-}atms\ \Gamma$
using $A\text{-}in\ A\text{-}lt$ **by** $metis$

qed

thus $\neg (\exists A \in | atm\text{-}of\text{-}clss\ (N \cup U_{er}'). A \prec_t atm\text{-}of\ K_x \wedge A \notin | trail\text{-}atms\ \Gamma')$
using $\langle atm\text{-}of\text{-}clss\ (N \cup U_{er}) = atm\text{-}of\text{-}clss\ (N \cup U_{er}') \rangle$ **by** $metis$

qed

thus
 $\forall C \in | iefac\ \mathcal{F} \mid \uparrow (N \cup U_{er}'). (\forall DE. Some\ (eres\ D\ E) = Some\ DE \longrightarrow C \prec_c DE) \longrightarrow$
 $trail\text{-}true\text{-}cls\ \Gamma' C$
 $\forall C \in | iefac\ \mathcal{F} \mid \uparrow (N \cup U_{er}'). (\forall DE. Some\ (eres\ D\ E) = Some\ DE \longrightarrow C \prec_c DE) \longrightarrow$

$(\forall K. \text{ord-res.is-maximal-lit } K \ C \longrightarrow$
 $\neg (\exists A \in |\text{atms-of-clss } (N \ \cup \ U_{er})|. A \prec_t \text{atm-of } K \wedge A \notin |\text{trail-atms } \Gamma'|)$
unfolding *atomize-conj* **by** *metis*

have *trail-true-cls* $\Gamma' \{ \#K \in \# \ C. K \neq L_C \# \} \wedge \text{is-pos } L_C$
if *Some* $(\text{eres } D \ E) = \text{Some } DE$ **and**
C-in: $C \in |\text{iefac } \mathcal{F} \ |^{\dagger} (N \ \cup \ U_{er})'$ **and**
C-lt: $C \prec_c DE$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C \ L_C$ **and**
L_C-undef: $\neg \text{trail-defined-lit } \Gamma' \ L_C$
for $DE \ C \ L_C$

proof –
have $DE = \text{eres } D \ E$
using *that* **by** *simp*
hence $C \prec_c \text{eres } D \ E$
using *C-lt* **by** *order*
hence $C \neq \text{eres } D \ E$
by *order*
hence $C \in |\text{iefac } \mathcal{F} \ |^{\dagger} (N \ \cup \ U_{er})$
using *C-in* $\langle \text{iefac } \mathcal{F} \ |^{\dagger} (N \ \cup \ U_{er})' = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \ |^{\dagger} (N \ \cup \ U_{er})) \rangle$
by *simp*
moreover **have** $C \prec_c E$
using $\langle C \prec_c \text{eres } D \ E \rangle \langle \text{eres } D \ E \prec_c E \rangle$ **by** *order*

have $L_C \preceq_l K$
using $\langle C \prec_c \text{eres } D \ E \rangle$ *C-max-lit* *eres-max-lit*
by (*meson linorder-cls.dual-order.asym linorder-lit.leI*
linorder-lit.mulp-if-maximal-less-that-maximal)
hence $L_C \prec_l K \vee L_C = K$
by *simp*

thus *?thesis*
proof (*elim disjE*)
assume $L_C \prec_l K$
hence *atm-of* $L_C \prec_t \text{atm-of } K$
using $\langle \text{is-pos } K \rangle$
by (*cases* L_C ; *cases* K) *simp-all*

have *trail-defined-lit* $\Gamma' \ L_C$
unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \ \Gamma \rangle$
proof (*intro trail-defined-lit-dropWhileI ballI*)
show *sorted-wrt* $(\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma$
using Γ -*sorted* .
next
show *monotone-on* $(\text{set } \Gamma) (\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) (\lambda x \ y. y \leq x)$
 $(\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } x))$
using *mono-atms-lt* .

```

next
  have  $\neg \text{atm-of } K \preceq_t \text{atm-of } L_C$ 
    using  $\langle \text{atm-of } L_C \prec_t \text{atm-of } K \rangle$  by order
  thus  $\neg \text{atm-of } K \preceq_t \text{atm-of } L_C \wedge \neg \text{atm-of } K \preceq_t \text{atm-of } (- L_C)$ 
    by simp
next
  have trail-defined-lit  $\Gamma K$ 
    using  $\langle \text{trail-false-lit } \Gamma K \rangle$ 
    using trail-defined-lit-iff-true-or-false by blast
  moreover have atm-of  $K \in | \text{atms-of-clss } (N \cup U_{er})$ 
    by (metis  $\langle \text{atms-of-clss } (N \cup U_{er}) = \text{atms-of-clss } (N \cup U_{er}') \rangle$ 
       $\langle \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}') = \text{finsert } (\text{eres } D E) (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) \rangle$ 
      atm-of-in-atms-of-clssI eres-max-lit finsertI1 linorder-lit.is-maximal-in-mset-iff)
  moreover have atm-of  $L_C \in | \text{atms-of-clss } (N \cup U_{er})$ 
    using  $\langle C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}') \rangle$  C-max-lit atm-of-in-atms-of-clssI
    unfolding linorder-lit.is-maximal-in-mset-iff
    by metis
  ultimately show trail-defined-lit  $\Gamma L_C$ 
    using  $\langle \text{atm-of } L_C \prec_t \text{atm-of } K \rangle$   $\Gamma$ -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  $\Gamma' C$ 
    using clause-true-in- $\Gamma'$ -if C-in  $\langle C \prec_c \text{eres } D E \rangle$  by blast
  hence trail-true-cls  $\Gamma' \{ \#K \in \# C. K \neq L_C \# \}$ 
    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  $\Gamma' \{ \#K \in \# C. K \neq L_C \# \} \wedge \text{is-pos } L_C$ 
    using  $\langle \text{is-pos } K \rangle$  by metis
qed
qed

thus  $\forall Da. \text{Some } (\text{eres } D E) = \text{Some } Da \longrightarrow$ 
   $(\forall C \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}'). C \prec_c Da \longrightarrow (\forall L_C. \text{ord-res.is-maximal-lit}$ 
   $L_C C \longrightarrow$ 
   $\neg \text{trail-defined-lit } \Gamma' L_C \longrightarrow \text{trail-true-cls } \Gamma' \{ \#K \in \# C. K \neq L_C \# \} \wedge$ 
   $\text{is-pos } L_C))$ 
  by metis
qed simp-all
next

```

case *step-hyps*: $\langle \text{resolution-neg } \Gamma \ E \ L \ D \ U_{er}' \ U_{er} \ \Gamma' \ K \ C \ \mathcal{F} \rangle$

note *E-max-lit* = $\langle \text{ord-res.is-maximal-lit } L \ E \rangle$

note *eres-max-lit* = $\langle \text{ord-res.is-maximal-lit } K \ (\text{eres } D \ E) \rangle$

have

F-subset: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**

E-in: $E \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$ **and**

clauses-lt-D-seldomly-have-undef-max-lit:

$\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c E \longrightarrow$

$(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C \ L_C \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma \ L_C \longrightarrow (\text{trail-true-cls } \Gamma \ \{\#K \in\# \ C. \ K \neq L_C\# \} \wedge$

is-pos $L_C))$ **and**

clauses-lt-E-true: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c E \longrightarrow \text{trail-true-cls } \Gamma$

C **and**

no-undef-atm-lt-max-lit-if-lt-E: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c E \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow$

$\neg (\exists A \mid\in\mid \text{atms-of-clss } (N \mid\cup\mid U_{er}). A \prec_t \text{atm-of } K \wedge A \mid\notin\mid \text{trail-atms } \Gamma))$

and

Γ-sorted: $\text{sorted-wrt } (\lambda x \ y. \ \text{atm-of } (fst \ y) \prec_t \ \text{atm-of } (fst \ x)) \ \Gamma$ **and**

Γ-lower: $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) \ (\text{atms-of-clss } (N \mid\cup\mid U_{er}))$

and

trail-atms-le0: $\forall A \mid\in\mid \text{trail-atms } \Gamma. \exists L. \text{ord-res.is-maximal-lit } L \ E \wedge (A \preceq_t \text{atm-of } L)$ **and**

Γ-deci-iff-neg: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (fst \ Ln)$ **and**

Γ-deci-ball-lt-true: $\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$

$(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c \{\#fst \ Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma \ C)$

and

Γ-prop-in: $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$ **and**

Γ-prop-greatest:

$\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$

$(fst \ Ln)$ **and**

Γ-prop-almost-false:

$\forall \Gamma_1 \ L \ C \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \ \text{Some } C) \ \# \ \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \ \{\#K \in\# \ C.$

$K \neq L\#\}$ **and**

Γ-prop-ball-lt-true: $(\forall \Gamma_1 \ L \ D \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \ \text{Some } D) \ \# \ \Gamma_0 \longrightarrow$

$(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 \ C))$ **and**

Γ-prop-ball-lt-true: $\forall \Gamma_1 \ L \ D \ \Gamma_0. \Gamma = \Gamma_1 \ @ \ (L, \ \text{Some } D) \ \# \ \Gamma_0 \longrightarrow$

$(\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 \ C)$

using invar by (*simp-all add*: $\langle s = (U_{er}, \ \mathcal{F}, \ \Gamma, \ \text{Some } E) \rangle \text{ord-res-7-invars-def}$)

have *clauses-lt-E-almost-defined*: $\forall C \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). C \prec_c E \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C \ K \longrightarrow \text{trail-defined-cls } \Gamma \ \{\#L \in\# \ C.$

$L \neq K\#\})$

using invar[*unfolded* $\langle s = (U_{er}, \ \mathcal{F}, \ \Gamma, \ \text{Some } E) \rangle$] *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* Γ . $A \preceq_t$ *atm-of* L
using *trail-atms-le0* *E-max-lit*
by (*metis* *Uniq-D* *linorder-lit*.*Uniq-is-maximal-in-mset*)

have $(- L, \text{Some } D) \in \text{set } \Gamma$
using $\langle \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \rangle$ **by** (*metis* *map-of-SomeD*)

hence *D-in*: $D \mid \in \mid$ *iefac* $\mathcal{F} \mid \uparrow \mid$ ($N \mid \cup \mid U_{er}$)
using Γ -*prop-in* **by** *simp*

have *D-max-lit*: *linorder-lit.is-greatest-in-mset* $D (- L)$
using Γ -*prop-greatest* $\langle (- L, \text{Some } D) \in \text{set } \Gamma \rangle$ **by** *fastforce*

have $D \prec_c E$
using *clause-lt-clause-if-max-lit-comp*
using *E-max-lit* $\langle \text{is-neg } L \rangle$ *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* $\langle \text{is-neg } L \rangle$ *D-max-lit* **by** *metis*

hence *eres* $D E \neq E$
by *order*

have $(- K, \text{Some } C) \in \text{set } \Gamma$
using $\langle \text{map-of } \Gamma (- K) = \text{Some } (\text{Some } C) \rangle$ **by** (*metis* *map-of-SomeD*)

hence *C-in*: $C \mid \in \mid$ *iefac* $\mathcal{F} \mid \uparrow \mid$ ($N \mid \cup \mid U_{er}$)
using Γ -*prop-in* **by** *simp*

have *C-max-lit*: *linorder-lit.is-greatest-in-mset* $C (- K)$
using Γ -*prop-greatest* $\langle (- K, \text{Some } C) \in \text{set } \Gamma \rangle$ **by** *fastforce*

hence $C \prec_c \text{eres } D E$
using $\langle \text{ord-res.is-maximal-lit } K (\text{eres } D E) \rangle$ $\langle \text{is-neg } L \rangle$
by (*metis* *Neg-atm-of-iff* *Pos-atm-of-iff* *linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset*
linorder-lit.mulp-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* $\Gamma' \Gamma$
using *step-hyps(9)* *suffix-dropWhile* **by** *metis*

hence *atms-of-cls* $(\text{eres } D E) \mid \subseteq \mid$ *atms-of-cls* $D \mid \cup \mid$ *atms-of-cls* E
using *lit-in-one-of-resolvents-if-in-eres*
unfolding *atms-of-cls-def* **by** *fastforce*

moreover have $atms\text{-of}\text{-cls } D \sqsubseteq | atms\text{-of}\text{-class } (N \cup | U_{er})$
using $D\text{-in}$
by (*metis atms-of-class-fimage-iefac atms-of-class-finsert finsert-absorb funion-upper1*)

moreover have $atms\text{-of}\text{-cls } E \sqsubseteq | atms\text{-of}\text{-class } (N \cup | U_{er})$
using $E\text{-in}$
by (*metis atms-of-class-fimage-iefac atms-of-class-finsert finsert-absorb funion-upper1*)

ultimately have $atms\text{-of}\text{-class } (N \cup | U_{er}) = atms\text{-of}\text{-class } (N \cup | U_{er}')$
unfolding $\langle U_{er}' = finsert (eres D E) U_{er} \rangle$ **atms-of-class-def** **by** *auto*

have $L \in \# E$
using $E\text{-max-lit}$ **unfolding** $linorder\text{-lit.is-maximal-in-mset-iff}$ **by** *metis*

hence $- L \notin \# E$
using $not\text{-both-lit-and-comp-lit-in-false-clause-if-trail-consistent}$
using $\Gamma\text{-consistent}$ $\langle trail\text{-false-cls } \Gamma E \rangle$ **by** *metis*

hence $\forall K \in \# eres D E. atm\text{-of } K \prec_t atm\text{-of } L$
using $lit\text{-in-eres-lt-greatest-lit-in-greatest-resolvent}[OF \langle eres D E \neq E \rangle E\text{-max-lit}]$
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 $lit\text{-in-one-of-resolvents-if-in-eres}$ **by** *metis*

moreover have $D\text{-almost-false: } trail\text{-false-cls } \Gamma \{ \#K \in \# D. K \neq - L \# \}$
using $ord\text{-res-7-invars-implies-propagated-clause-almost-false}$
using $\langle (- L, Some D) \in set \Gamma \rangle invar[unfolding \langle s = (U_{er}, \mathcal{F}, \Gamma, Some E) \rangle]$
by *metis*

ultimately have $trail\text{-false-cls } \Gamma (eres D E)$
using $\langle trail\text{-false-cls } \Gamma E \rangle$ **unfolding** $trail\text{-false-cls-def}$ **by** *fastforce*

have $eres D E \not\subseteq | N \cup | U_{er}$
using $eres\text{-not-in-known-clauses-if-trail-false-cls}$
using $\Gamma\text{-consistent clauses-lt-E-true}$ $\langle eres D E \prec_c E \rangle \langle trail\text{-false-cls } \Gamma (eres D E) \rangle$ **by** *metis*

hence $eres D E \not\subseteq | \mathcal{F}$
using $\mathcal{F}\text{-subset}$ **by** *blast*

hence $iefac \mathcal{F} (eres D E) = eres D E$
by (*simp add: iefac-def*)

hence $iefac \mathcal{F} |' (N \cup | U_{er}') = finsert (eres D E) (iefac \mathcal{F} |' (N \cup | U_{er}))$
by (*simp add: \langle U_{er}' = finsert (eres D E) U_{er} \rangle*)

have *mem-set- Γ' -iff*: $(Ln \in \text{set } \Gamma') = (\neg \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \wedge Ln \in \text{set } \Gamma)$ **for** Ln
unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$
proof (*rule mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone*)
show *sorted-wrt* $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$
using Γ -sorted .
next
show *monotone-on* $(\text{set } \Gamma) (\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) (\lambda x y. y \leq x)$
 $(\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln))$
by (*rule monotone-onI*) *auto*
qed

have *atms-in- Γ' -lt-atm-K*: $\forall A \mid \in \mid \text{trail-atms } \Gamma'. A \prec_t \text{atm-of } K$
proof –
have $\exists L. \text{ord-res.is-maximal-lit } L \ C \wedge x \prec_t \text{atm-of } L$ **if** $x \mid \in \mid \text{trail-atms } \Gamma'$ **for** x
proof (*intro exI conjI*)
show *ord-res.is-maximal-lit* $(- K) \ C$
using *C-max-lit* **by** *blast*
next
show $x \prec_t \text{atm-of } (- K)$
using $\langle x \mid \in \mid \text{trail-atms } \Gamma' \rangle$ *mem-set- Γ' -iff* **unfolding** *fset-trail-atms* **by** *fastforce*
qed

hence $\forall A \mid \in \mid \text{trail-atms } \Gamma'. A \prec_t \text{atm-of } (- K)$
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 $\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er}'$
unfolding $\langle U_{er}' = \text{finsert } (\text{eres } D \ E) \ U_{er} \rangle$
using *\mathcal{F} -subset* **by** *blast*

moreover have $\forall C'. \text{Some } C = \text{Some } C' \longrightarrow C' \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$
using *C-in* **by** (*simp add*: $\langle U_{er}' = \text{finsert } (\text{eres } D \ E) \ U_{er} \rangle$)

moreover have
 $\bigwedge K_x. \text{linorder-lit.is-maximal-in-mset } x \ K_x \implies \text{trail-defined-cls } \Gamma' \ \{ \#L \in \# \ x. L \neq K_x \# \}$ **and**
clause-true-in- Γ' -if: *trail-true-cls* $\Gamma' \ x$
if *x-lt*: $\bigwedge y. \text{Some } C = \text{Some } y \implies x \prec_c y$ **and** *x-in*: $x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ **for** x

proof –
have $x \prec_c C$
using $x\text{-lt}$ **by** *metis*

hence $x \prec_c \text{eres } D E$
using $\langle C \prec_c \text{eres } D E \rangle$ **by** *order*

hence $x \neq \text{eres } D E$
by *order*

have $x \prec_c E$
using $\langle x \prec_c \text{eres } D E \rangle \langle \text{eres } D E \prec_c E \rangle$ **by** *order*

moreover have $x\text{-in}'$: $x \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$
using $x\text{-in}$ $\langle x \neq \text{eres } D E \rangle$
unfolding $\langle \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}') = \text{finsert } (\text{eres } D E) (\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})) \rangle$
by *simp*

ultimately have $x\text{-true}$: *trail-true-cls* Γx
using *clauses-lt-E-true* **by** *metis*

then obtain L_x **where** $L_x \in \# x$ **and** *trail-true-lit* ΓL_x
unfolding *trail-true-cls-def* **by** *metis*

have $L_x \preceq_l - K$
using *C-max-lit* $\langle x \prec_c C \rangle \langle L_x \in \# x \rangle$
by (*smt* (*verit*, *ccfv-threshold*) *asypD 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.mulp-if-all-left-smaller transpE)

have *mono-atms-lt*: *monotone-on* (*set* Γ) $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x))$
 $(\lambda x y. y \leq x)$
 $(\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (fst x))$

proof (*intro monotone-onI*, *unfold le-bool-def*, *intro impI*)
fix $x y$
assume $\text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ **and** $\text{atm-of } K \preceq_t \text{atm-of } (fst y)$
thus $\text{atm-of } K \preceq_t \text{atm-of } (fst x)$
by *order*

qed

obtain $\Gamma_1 \Gamma_0$ **where** $\Gamma\text{-eq}$: $\Gamma = \Gamma_1 @ (- K, \text{Some } C) \# \Gamma_0$
using $\langle (- K, \text{Some } C) \in \text{set } \Gamma \rangle$ **by** (*metis split-list*)

hence *trail-true-cls* $\Gamma_0 x$
using $\Gamma\text{-prop-ball-lt-true } x\text{-in}' \langle x \prec_c C \rangle$ **by** *metis*

then obtain L_x where $L_x \in \# x$ and L_x -true: trail-true-lit $\Gamma_0 L_x$
 unfolding trail-true-cls-def by auto

moreover have $\Gamma' = \Gamma_0$

proof –

have $\Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) ((\Gamma_1 @ [(- K, \text{Some } C)]) @ \Gamma_0)$

unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle \Gamma$ -eq
 by simp

also have $\dots = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma_0$

proof (rule dropWhile-append2)

fix $Ln :: 'f \text{ gterm literal} \times 'f \text{ gclause option}$

assume $Ln \in \text{set } (\Gamma_1 @ [(- K, \text{Some } C)])$

moreover have $\forall x \in \text{set } \Gamma_1. \text{atm-of } K \prec_t \text{atm-of } (\text{fst } x)$
 using Γ -sorted by (simp add: Γ -eq sorted-wrt-append)

ultimately show $\text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$
 using $\langle \text{is-neg } K \rangle$ by auto

qed

also have $\dots = \Gamma_0$

proof (cases Γ_0)

case Nil

thus ?thesis

by (simp add: dropWhile-eq-self-iff)

next

case (Cons $Ln \Gamma_0'$)

hence $\text{atm-of } (\text{fst } Ln) \prec_t \text{atm-of } K$
 using Γ -sorted by (simp add: Γ -eq sorted-wrt-append)

hence $\neg \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$
 by order

hence $\neg \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } (\text{hd } \Gamma_0))$
 by (simp add: $\langle \Gamma_0 = Ln \# \Gamma_0' \rangle$)

thus ?thesis

by (simp add: dropWhile-eq-self-iff)

qed

finally show ?thesis .

qed

ultimately have trail-true-lit $\Gamma' L_x$
 by argo

```

thus trail-true-cls  $\Gamma' x$ 
  using  $\langle L_x \in\# x \rangle$  unfolding trail-true-cls-def by metis

fix  $K_x$  assume x-max-lit: ord-res.is-maximal-lit  $K_x x$ 
show trail-defined-cls  $\Gamma' \{\#L \in\# x. L \neq K_x\# \}$ 
  unfolding  $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$ 
proof (intro trail-defined-cls-dropWhileI ballI)
  show sorted-wrt  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$ 
    using  $\Gamma$ -sorted .
next
  show monotone-on (set  $\Gamma$ )  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) (\lambda x y. y$ 
 $\leq x)$ 
     $(\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } x))$ 
    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  $\langle L_x \in\# x \rangle \langle L_x \neq K_x \rangle$  unfolding linorder-lit.is-maximal-in-mset-iff
  by fastforce

moreover have  $K_x \preceq_l K$ 
  using  $\langle x \prec_c \text{eres } D E \rangle$ 
using linorder-lit.mulp-if-maximal-less-that-maximal[OF eres-max-lit x-max-lit]
  by fastforce

ultimately have atm-of  $L_x \prec_t \text{atm-of } K$ 
  using  $\langle \text{is-neg } K \rangle$ 
  by (metis C-max-lit Pos-atm-of-iff  $\langle x \prec_c C \rangle$  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.mulp-if-maximal-less-that-maximal linorder-trm.linorder-cases
literal.collapse(2) ord-res.less-lit-simps(3) ord-res.less-lit-simps(4) umi-
nus-Neg
x-max-lit)

hence  $\neg \text{atm-of } K \preceq_t \text{atm-of } L_x$ 
  by order

thus  $\neg \text{atm-of } K \preceq_t \text{atm-of } L_x \wedge \neg \text{atm-of } K \preceq_t \text{atm-of } (- L_x)$ 
  unfolding atm-of-uminus conj-absorb .
next
show trail-defined-cls  $\Gamma \{\#L \in\# x. L \neq K_x\# \}$ 
  using clauses-lt-E-almost-defined x-in'  $\langle x \prec_c E \rangle$  x-max-lit by metis

```

```

qed
qed

moreover have sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma'$ 
  using  $\Gamma$ -sorted step-hyps(9) by (metis sorted-wrt-dropWhile)

moreover have  $\Gamma'$ -lower: linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-clss
( $N \cup U_{er}$ '))
proof -
  have linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (trail-atms  $\Gamma$ )
    unfolding linorder-trm.is-lower-set-iff
  proof (intro conjI ballI impI)
    show fset (trail-atms  $\Gamma'$ )  $\subseteq$  fset (trail-atms  $\Gamma$ )
      unfolding fset-trail-atms using  $\langle \text{suffix } \Gamma' \Gamma \rangle$  by (metis image-mono
set-mono-suffix)
  next
    obtain  $\Gamma''$  where  $\Gamma = \Gamma'' @ \Gamma'$ 
      using  $\langle \text{suffix } \Gamma' \Gamma \rangle$  unfolding suffix-def by metis

  fix l x
  assume l  $\in$  trail-atms  $\Gamma'$  and x  $\in$  trail-atms  $\Gamma$  and  $x \prec_t l$ 

  have x  $\in$  trail-atms  $\Gamma'' \vee x \in$  trail-atms  $\Gamma'$ 
    using  $\langle x \in$  trail-atms  $\Gamma \rangle$  unfolding  $\langle \Gamma = \Gamma'' @ \Gamma' \rangle$  fset-trail-atms by auto

  thus x  $\in$  trail-atms  $\Gamma'$ 
  proof (elim disjE)
    assume x  $\in$  trail-atms  $\Gamma''$ 

    hence l  $\prec_t x$ 
      using  $\Gamma$ -sorted  $\langle l \in$  trail-atms  $\Gamma' \rangle$ 
      unfolding  $\langle \Gamma = \Gamma'' @ \Gamma' \rangle$  sorted-wrt-append fset-trail-atms by blast

    hence False
      using  $\langle x \prec_t l \rangle$  by order

  thus x  $\in$  trail-atms  $\Gamma'$  ..
  next
    assume x  $\in$  trail-atms  $\Gamma'$ 
    thus x  $\in$  trail-atms  $\Gamma'$  .
  qed
qed

thus ?thesis
  using  $\Gamma$ -lower
  unfolding  $\langle \text{atms-of-clss } (N \cup U_{er}) = \text{atms-of-clss } (N \cup U_{er}') \rangle$ 
  by order
qed

```

moreover have $\forall DE. \text{Some } C = \text{Some } DE \longrightarrow (\forall A \mid \in \mid \text{trail-atms } \Gamma'. \\ \exists L. \text{ord-res.is-maximal-lit } L \text{ } DE \wedge A \preceq_t \text{atm-of } L)$
using *atms-in- Γ' -lt-atm-K C-max-lit* **by** *fastforce*

moreover have $\forall Ln \in \text{set } \Gamma'. \text{snd } Ln = \text{None} \longleftrightarrow \text{is-neg } (\text{fst } Ln)$
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 $Ln \in \text{set } \Gamma'$ **and** $\text{snd } Ln = \text{None}$ **and** $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})'$ **and** $C \prec_c \{\#fst Ln\# \}$
for $Ln \ C$
proof –
have $Ln \in \text{set } \Gamma$
using *⟨ $Ln \in \text{set } \Gamma' \rangle \langle \text{suffix } \Gamma' \Gamma \rangle \text{set-mono-suffix}$* **by** *blast*

have $C = \text{eres } D \ E \vee C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using *⟨ $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})' \rangle$*
unfolding *⟨iefac $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})' = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$*
by *simp*

thus *trail-true-cls Γ' C*
proof (*elim disjE*)
assume $C = \text{eres } D \ E$

hence $K \prec_l \text{fst } Ln$
using *⟨ $C \prec_c \{\#fst Ln\# \} \rangle \text{eres-max-lit}$*
by (*simp add: linorder-lit.is-maximal-in-mset-iff*)

hence $\text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$
by (*cases K; cases fst Ln*) *simp-all*

moreover have $\text{atm-of } (\text{fst } Ln) \mid \in \mid \text{trail-atms } \Gamma'$
using *⟨ $Ln \in \text{set } \Gamma' \rangle \text{by } (\text{simp add: fset-trail-atms})$*

moreover have $\text{atm-of } K \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er})'$
by (*metis* *⟨iefac $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})' = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \rangle$*)
atm-of-in-atms-of-clsI eres-max-lit finsert-iff linorder-lit.is-maximal-in-mset-iff)

ultimately have $\text{atm-of } K \mid \in \mid \text{trail-atms } \Gamma'$
using *Γ' -lower*
unfolding *linorder-trm.is-lower-set-iff*
by *fastforce*

moreover have $\text{atm-of } K \not\mid \in \mid \text{trail-atms } \Gamma'$
using *atms-in- Γ' -lt-atm-K* **by** *blast*

ultimately show *trail-true-cls* $\Gamma' C$
by contradiction
next
assume $C \in | \text{iefac } \mathcal{F} |^\dagger (N \cup U_{er})$

hence *trail-true-cls* ΓC
using $\Gamma\text{-deci-ball-lt-true } \langle Ln \in \text{set } \Gamma \rangle \langle \text{snd } Ln = \text{None} \rangle \langle C \prec_c \{\#fst Ln\# \}$
by *metis*

then obtain L_C **where** $L_C \in \# C$ **and** *trail-true-lit* ΓL_C
unfolding *trail-true-cls-def* **by** *auto*

hence $\forall x \in \# C. x \prec_l fst Ln$
using $\langle C \prec_c \{\#fst Ln\# \}$
unfolding *multp-singleton-right*[*OF linorder-lit.transp-on-less*]
by *simp*

hence $L_C \prec_l fst Ln$
using $\langle L_C \in \# C \rangle$ **by** *metis*

hence *atm-of* $L_C \preceq_t \text{atm-of } (fst Ln)$
by (*cases* L_C ; *cases* $fst Ln$) *simp-all*

moreover have *atm-of* $(fst Ln) \prec_t \text{atm-of } K$
using *atms-in- Γ' -lt-atm-K*
by (*simp add: fset-trail-atms that(1)*)

ultimately have *atm-of* $L_C \prec_t \text{atm-of } K$
by *order*

have $L_C \in \text{fst } ' \text{set } \Gamma$
using $\langle \text{trail-true-lit } \Gamma L_C \rangle$
unfolding *trail-true-lit-def* .

hence $L_C \in \text{fst } ' \text{set } \Gamma'$
using *mem-set- Γ' -iff*
using $\langle \text{atm-of } L_C \prec_t \text{atm-of } K \rangle$ *linorder-trm.not-le* **by** *auto*

hence *trail-true-lit* $\Gamma' L_C$
unfolding *trail-true-lit-def* .

thus *trail-true-cls* $\Gamma' C$
using $\langle L_C \in \# C \rangle$
unfolding *trail-true-cls-def* **by** *auto*

qed
qed

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} |^\dagger (N \cup U_{er})$

using Γ -prop-in $\langle \text{suffix } \Gamma' \Gamma \rangle$ set-mono-suffix $\langle U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \rangle$
by *blast*

moreover have $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C \text{ (fst } Ln)$
using Γ -prop-greatest $\langle \text{suffix } \Gamma' \Gamma \rangle$ set-mono-suffix **by** *blast*

moreover have $\forall \Gamma_1 L C \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{ \#K \in \# C. K \neq L \# \}$
using Γ -prop-almost-false $\langle \text{suffix } \Gamma' \Gamma \rangle$
by (*metis* (*no-types*, *lifting*) *append.assoc suffix-def*)

moreover have $\forall \Gamma_1 L D \Gamma_0. \Gamma' = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$
 $(\forall C \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}'). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C)$
proof (*intro allI impI ballI*)

fix
 $\Gamma_1 \Gamma_0 :: ('f \text{ gliteral } \times 'f \text{ gclause option}) \text{ list and}$
 $L :: 'f \text{ gliteral and}$
 $C_0 C_1 :: 'f \text{ gclause}$

assume
 Γ' -eq: $\Gamma' = \Gamma_1 @ (L, \text{Some } C_1) \# \Gamma_0$ **and**
 C_0 -in: $C_0 \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}')$ **and**
 $C_0 \prec_c C_1$

have $C_0 = \text{eres } D E \vee C_0 \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$
using C_0 -in
unfolding $\langle \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}') = \text{finsert } (\text{eres } D E) (\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})) \rangle$
by *simp*

thus *trail-true-cls* $\Gamma_0 C_0$
proof (*elim disjE*)
assume $C_0 = \text{eres } D E$

have $\neg \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } (L, \text{Some } C_1))$ **and** $(L, \text{Some } C_1) \in \text{set } \Gamma$
unfolding *atomize-conj*
using Γ' -eq $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$
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 $\neg \text{atm-of } K \preceq_t \text{atm-of } L$
by *simp*

hence *atm-of* $L \prec_t \text{atm-of } K$
by *order*

moreover have *linorder-lit.is-greatest-in-mset* $C_1 L$
using Γ -prop-greatest $\langle (L, \text{Some } C_1) \in \text{set } \Gamma \rangle$ **by** *fastforce*

ultimately have *False*
using $\langle C_0 \prec_c C_1 \rangle$
by (*smt* (*verit*) $\langle C_0 = \text{eres } D \ E \rangle$ *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.multip-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* $\Gamma_0 \ C_0 \ ..$
next
assume $C_0 \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})$
thus *trail-true-cls* $\Gamma_0 \ C_0$
using $\Gamma\text{-prop-ball-lt-true}$ $\Gamma'\text{-eq}$ $\langle C_0 \prec_c C_1 \rangle$
by (*metis* (*no-types*, *opaque-lifting*) $\langle \text{suffix } \Gamma' \ \Gamma \rangle$ *append-assoc* *suffixE*)

qed
qed

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } C) \rangle$
proof (*intro* *ord-res- γ -invars.intros*)
have *clause-true-in- Γ' -if*: *trail-true-cls* $\Gamma' \ x$ **and**
 $\bigwedge K_x. \text{linorder-lit.is-maximal-in-mset } x \ K_x \implies$
 $\neg (\exists A \in | \text{atms-of-cls } (N \ | \cup \ | \ U_{er}') . A \prec_t \text{atm-of } K_x \wedge A \notin | \text{trail-atms } \Gamma')$
if *x-lt*: $\bigwedge DE. \text{Some } C = \text{Some } DE \implies x \prec_c DE$ **and** *x-in*: $x \in | \text{iefac } \mathcal{F} \ |^{\uparrow}$
 $(N \ | \cup \ | \ U_{er}')$
for x
proof –
have $x \prec_c C$
using *x-lt* **by** *metis*

hence $x \prec_c \text{eres } D \ E$
using $\langle C \prec_c \text{eres } D \ E \rangle$ **by** *order*

hence $x \neq \text{eres } D \ E$
by *order*

have $x \prec_c E$
using $\langle x \prec_c \text{eres } D \ E \rangle \langle \text{eres } D \ E \prec_c E \rangle$ **by** *order*

moreover have *x-in'*: $x \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})$
using *x-in* $\langle x \neq \text{eres } D \ E \rangle$
unfolding $\langle \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er}') = \text{finsert } (\text{eres } D \ E) (\text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})) \rangle$
by *simp*

ultimately have *x-true*: *trail-true-cls* $\Gamma \ x$
using *clauses-lt-E-true* **by** *metis*

then obtain L_x **where** $L_x \in \# x$ **and** *trail-true-lit* ΓL_x
unfolding *trail-true-cls-def* **by** *metis*

have $L_x \preceq_l - K$
using *C-max-lit* $\langle x \prec_c C \rangle \langle L_x \in \# x \rangle$
by (*smt* (*verit*, *ccfv-threshold*) *asymptD* *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.mulp-if-all-left-smaller *transpE*)

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

qed

obtain $\Gamma_1 \Gamma_0$ **where** $\Gamma\text{-eq}$: $\Gamma = \Gamma_1 @ (- K, \text{Some } C) \# \Gamma_0$
using $\langle (- K, \text{Some } C) \in \text{set } \Gamma \rangle$ **by** (*metis* *split-list*)

hence *trail-true-cls* $\Gamma_0 x$
using $\Gamma\text{-prop-ball-lt-true}$ $x\text{-in}' \langle x \prec_c C \rangle$ **by** *metis*

then obtain L_x **where** $L_x \in \# x$ **and** $L_x\text{-true}$: *trail-true-lit* $\Gamma_0 L_x$
unfolding *trail-true-cls-def* **by** *auto*

moreover have $\Gamma' = \Gamma_0$
proof –
have $\Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (fst Ln)) ((\Gamma_1 @ [(- K, \text{Some } C)]) @ \Gamma_0)$
Some C) $@ \Gamma_0$
unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (fst Ln)) \Gamma \rangle \Gamma\text{-eq}$
by *simp*

also have $\dots = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (fst Ln)) \Gamma_0$
proof (*rule* *dropWhile-append2*)
fix $Ln :: 'f \text{ gterm literal} \times 'f \text{ gclause option}$
assume $Ln \in \text{set } (\Gamma_1 @ [(- K, \text{Some } C)])$

moreover have $\forall x \in \text{set } \Gamma_1. \text{atm-of } K \prec_t \text{atm-of } (fst x)$
using $\Gamma\text{-sorted}$ **by** (*simp* *add*: $\Gamma\text{-eq}$ *sorted-wrt-append*)

ultimately show $\text{atm-of } K \preceq_t \text{atm-of } (fst Ln)$
using $\langle \text{is-neg } K \rangle$ **by** *auto*

qed

also have $\dots = \Gamma_0$

proof (cases Γ_0)

case Nil

thus ?thesis

by (simp add: dropWhile-eq-self-iff)

next

case (Cons Ln Γ_0')

hence atm-of (fst Ln) \prec_t atm-of K

using Γ -sorted by (simp add: Γ -eq sorted-wrt-append)

hence \neg atm-of K \preceq_t atm-of (fst Ln)

by order

hence \neg atm-of K \preceq_t atm-of (fst (hd Γ_0))

by (simp add: $\langle \Gamma_0 = Ln \# \Gamma_0' \rangle$)

thus ?thesis

by (simp add: dropWhile-eq-self-iff)

qed

finally show ?thesis .

qed

ultimately have trail-true-lit $\Gamma' L_x$

by argo

thus trail-true-cls $\Gamma' x$

using $\langle L_x \in \# x \rangle$ 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 $\langle x \prec_c \text{eres } D E \rangle$ eres-max-lit

using linorder-lit.multip-if-maximal-less-than-maximal by fastforce

hence atm-of $K_x \preceq_t$ atm-of K

by (cases K_x ; cases K) simp-all

have $A \in |$ trail-atms Γ'

if A-lt: $A \prec_t$ atm-of K_x and A-in: $A \in |$ atms-of-cls (N \cup U_{er}) for A

unfolding $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \rangle$

proof (rule in-trail-atms-dropWhileI)

show sorted-wrt $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$

using Γ -sorted .

next
show *monotone-on* (*set* Γ) ($\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$) ($\lambda x y. y \leq x$) ($\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (fst x)$)
using *mono-atms-lt* .
next
show $\neg \text{atm-of } K \preceq_t A$
using *A-lt* $\langle \text{atm-of } K_x \preceq_t \text{atm-of } K \rangle$ **by order**
next
have $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K_x \wedge A \notin | \text{trail-atms } \Gamma)$
using *no-undef-atm-lt-max-lit-if-lt-E* $\langle x | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}) \rangle \langle x \prec_c E \rangle$ *x-max-lit*
by metis

thus $A | \in | \text{trail-atms } \Gamma$
using *A-in A-lt* **by metis**
qed

thus $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}'). A \prec_t \text{atm-of } K_x \wedge A \notin | \text{trail-atms } \Gamma')$
using $\langle \text{atms-of-clss } (N \cup U_{er}) = \text{atms-of-clss } (N \cup U_{er}') \rangle$ **by metis**
qed

thus
 $\forall x | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}'). (\forall DE. \text{Some } C = \text{Some } DE \longrightarrow x \prec_c DE)$
 \longrightarrow
trail-true-cls $\Gamma' x$
 $\forall x | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}'). (\forall DE. \text{Some } C = \text{Some } DE \longrightarrow x \prec_c DE)$
 \longrightarrow
 $(\forall K. \text{ord-res.is-maximal-lit } K x \longrightarrow$
 $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}'). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma'))$
unfolding *atomize-conj* **by metis**

have *trail-true-cls* $\Gamma' \{ \#K \in \# B. K \neq L_B \# \} \wedge \text{is-pos } L_B$
if *Some* $C = \text{Some } C'$ **and**
B-in: $B | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}')$ **and**
B-lt: $B \prec_c C'$ **and**
B-max-lit: *linorder-lit.is-maximal-in-mset* $B L_B$ **and**
L_B-undef: $\neg \text{trail-defined-lit } \Gamma' L_B$
for $C' B L_B$
proof –
have $C' = C$
using *that* **by simp**
hence $B \prec_c C$
using *B-lt* **by order**
hence $B \prec_c \text{eres } D E$
using $\langle C \prec_c \text{eres } D E \rangle$ **by order**
hence $B \neq \text{eres } D E$
by order

hence $B \in |iefac \mathcal{F} |^{\uparrow} (N \cup U_{er})$
using $B\text{-in } \langle iefac \mathcal{F} |^{\uparrow} (N \cup U_{er}') = finsert (eres D E) (iefac \mathcal{F} |^{\uparrow} (N \cup U_{er})) \rangle$
by *simp*
moreover have $B \prec_c E$
using $\langle B \prec_c eres D E \rangle \langle eres D E \prec_c E \rangle$ **by** *order*

have $L_B \preceq_l - K$
using $\langle B \prec_c C \rangle B\text{-max-lit } C\text{-max-lit}$
by $(metis \text{ asympD linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.leI linorder-lit.mulp-if-maximal-less-that-maximal ord-res.asymp-less-cl})$
hence $L_B \prec_l - K \vee L_B = - K$
by *simp*

thus *?thesis*
proof $(elim \text{ disjE})$
assume $L_B \prec_l - K$
hence $atm\text{-of } L_B \prec_t atm\text{-of } K$
using $\langle is\text{-neg } K \rangle$
by $(cases L_B; cases K) \text{ simp-all}$

have *trail-defined-lit* $\Gamma' L_B$
unfolding $\langle \Gamma' = dropWhile (\lambda Ln. atm\text{-of } K \preceq_t atm\text{-of } (fst Ln)) \Gamma \rangle$
proof $(intro \text{ trail-defined-lit-dropWhileI ballI})$
show *sorted-wrt* $(\lambda x y. atm\text{-of } (fst y) \prec_t atm\text{-of } (fst x)) \Gamma$
using $\Gamma\text{-sorted}$.
next
show *monotone-on* $(set \Gamma) (\lambda x y. atm\text{-of } (fst y) \prec_t atm\text{-of } (fst x)) (\lambda x y. y \leq x)$
 $(\lambda x. atm\text{-of } K \preceq_t atm\text{-of } (fst x))$
using *mono-atms-lt* .
next
have $\neg atm\text{-of } K \preceq_t atm\text{-of } L_B$
using $\langle atm\text{-of } L_B \prec_t atm\text{-of } K \rangle$ **by** *order*
thus $\neg atm\text{-of } K \preceq_t atm\text{-of } L_B \wedge \neg atm\text{-of } K \preceq_t atm\text{-of } (- L_B)$
by *simp*
next
have *trail-defined-lit* ΓK
by $(metis \text{ map-of-eq-None-iff option.discI step-hyps(12) trail-defined-lit-def})$
moreover have $atm\text{-of } K \in |atms\text{-of-clss } (N \cup U_{er})$
by $(metis \langle atms\text{-of-clss } (N \cup U_{er}) = atms\text{-of-clss } (N \cup U_{er}') \rangle \langle iefac \mathcal{F} |^{\uparrow} (N \cup U_{er}') = finsert (eres D E) (iefac \mathcal{F} |^{\uparrow} (N \cup U_{er})) \rangle$
 $atm\text{-of-in-atms-of-clssI eres-max-lit finsertI1 linorder-lit.is-maximal-in-mset-iff})$
moreover have $atm\text{-of } L_B \in |atms\text{-of-clss } (N \cup U_{er})$
using $\langle B \in |iefac \mathcal{F} |^{\uparrow} (N \cup U_{er}) \rangle B\text{-max-lit } atm\text{-of-in-atms-of-clssI}$
unfolding *linorder-lit.is-maximal-in-mset-iff*
by *metis*

```

ultimately show trail-defined-lit  $\Gamma$   $L_B$ 
  using  $\langle atm\text{-of } L_B \prec_t atm\text{-of } K \rangle \Gamma\text{-lower}$ 
  unfolding trail-defined-lit-iff-trail-defined-atm
  by (meson linorder-trm.is-lower-set-iff)
qed

hence False
  using  $L_B\text{-undef}$  by contradiction

thus ?thesis ..
next
  have trail-true-cls  $\Gamma'$   $B$ 
    using clause-true-in- $\Gamma'$ -if  $B$ -in  $\langle B \prec_c C \rangle$  by blast
  hence trail-true-cls  $\Gamma'$   $\{\#K \in\# B. K \neq L_B\#\}$ 
    using  $L_B\text{-undef}$ 
  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  $\Gamma'$   $\{\#K \in\# B. K \neq L_B\#\} \wedge is\text{-pos } L_B$ 
    using  $\langle is\text{-neg } K \rangle$ 
    by (cases  $K$ ) simp-all
  qed
qed

thus  $\forall Da. Some C = Some Da \longrightarrow$ 
  ( $\forall x \in |iefac \mathcal{F} \mid \mid (N \mid \cup \mid U_{er} \wedge). x \prec_c Da \longrightarrow (\forall L_C. ord\text{-res.is-maximal-lit } L_C$ 
 $x \longrightarrow$ 
   $\neg trail\text{-defined-lit } \Gamma' L_C \longrightarrow trail\text{-true-cls } \Gamma' \{\#K \in\# x. K \neq L_C\#\} \wedge is\text{-pos}$ 
 $L_C))$ 
  by metis
  qed simp-all
qed

lemma rtranclp-ord-res-7-preserves-ord-res-7-invars:
  assumes
    step:  $(ord\text{-res-7 } N)^{**} s s'$  and
    invars:  $ord\text{-res-7-invars } N s$ 
  shows  $ord\text{-res-7-invars } N s'$ 
  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:  $(ord\text{-res-7 } N)^{++} s s'$  and
    invars:  $ord\text{-res-7-invars } N s$ 
  shows  $ord\text{-res-7-invars } N s'$ 
  using step invars ord-res-7-preserves-invars
  by (smt (verit, del-insts) tranclp-induct)

```


lemma *propagating-clause-almost-false*:
assumes *invars*: *ord-res- γ -invars* $N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ **and** $(L, \text{Some } C) \in \text{set } \Gamma$
shows *trail-false-cls* $\Gamma \{\#K \in \# C. K \neq L\# \}$
proof –
have $\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$
using *invars* **unfolding** *ord-res- γ -invars-def* **by** *metis*

hence $\exists \Gamma_1 \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \wedge \text{trail-false-cls } \Gamma_0 \{\#K \in \# C. K \neq L\# \}$
using $\langle (L, \text{Some } C) \in \text{set } \Gamma \rangle$ **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- γ -if-not-final*:
assumes
not-final: $\neg \text{ord-res-}\gamma\text{-final } (N, s)$ **and**
invars: *ord-res- γ -invars* $N s$
shows $\exists s'. \text{ord-res-}\gamma \ N \ s \ s'$
proof –
obtain $U_{er} \ \mathcal{F} \ \Gamma \ \mathcal{C}$ **where** $s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$
by (*metis surj-pair*)

note $\text{invars}' = \text{invars}[\text{unfolded } \text{ord-res-}\gamma\text{-invars-def}, \text{rule-format}, OF \ \langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle]$

have $\Gamma\text{-sorted}$: *sorted-wrt* $(\lambda x y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma$
using *invars'* **by** *argo*

have $\Gamma\text{-consistent}$: *trail-consistent* Γ
using $\Gamma\text{-sorted}$ *trail-consistent-if-sorted-wrt-atoms* **by** *metis*

have *distinct* $(\text{map } fst \ \Gamma)$
proof (*rule distinct-if-sorted-wrt-asymp*)
have *sorted-wrt* $(\lambda x y. fst \ y \prec_l \text{fst } x) \ \Gamma$
proof (*rule sorted-wrt-mono-rel*)
show *sorted-wrt* $(\lambda x y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma$
using $\Gamma\text{-sorted}$.

next
fix $x \ y :: 'f \ \text{gterm } \text{literal} \times 'f \ \text{gterm } \text{literal } \text{multiset } \text{option}$
assume $\text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)$
thus $\text{fst } y \prec_l \text{fst } x$
by (*cases fst x; cases fst y*) (*simp-all only: literal.sel ord-res.less-lit-simps*)
qed

thus *sorted-wrt* $(\lambda x y. y \prec_l x) \ (\text{map } fst \ \Gamma)$
unfolding *sorted-wrt-map* .

next
show *asympt-on* (set (map fst Γ)) ($\lambda x y. y \prec_l x$)
using *linorder-lit.asympt-on-greater* .
qed

hence *map-of- Γ -eq-SomeI*: $\bigwedge x y. (x, y) \in \text{set } \Gamma \implies \text{map-of } \Gamma x = \text{Some } y$
by (*metis map-of-is-SomeI*)

have Γ -lower: *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-cls* ($N \cup U_{er}$))
using *invars'* **by** *argo*

obtain E **where** $C = \text{Some } E$ **and** $E \neq \{\#\}$
using *not-final[unfolded <math>s = (U_{er}, \mathcal{F}, \Gamma, C)>* *ord-res-7-final.simps, simplified*
by *metis*

have E -in: $E \in | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$
using *invars'* $\langle C = \text{Some } E \rangle$ **by** *metis*

obtain L **where** E -max-lit: *ord-res.is-maximal-lit* $L E$
using $\langle E \neq \{\#\} \rangle$ *linorder-lit.ex-maximal-in-mset* **by** *metis*

show *?thesis*
proof (*cases trail-false-cls* ΓE)
case E -false: *True*

hence L -false: *trail-false-lit* ΓL
using E -max-lit **unfolding** *linorder-lit.is-maximal-in-mset-iff trail-false-cls-def*
by *metis*

hence $\exists \text{opt. } (- L, \text{opt}) \in \text{set } \Gamma$
by (*auto simp: trail-false-lit-def*)

have $\neg \text{is-pos } L$
proof (*rule notI*)
assume $\text{is-pos } L$

hence $\text{is-neg } (- L)$
by (*metis $\langle \text{is-pos } L \rangle \text{is-pos-neg-not-is-pos}$*)

hence $(- L, \text{None}) \in \text{set } \Gamma$
using *invars'* $\langle \exists \text{opt. } (- L, \text{opt}) \in \text{set } \Gamma \rangle$ **by** (*metis prod.sel*)

moreover have $E \prec_c \{\# - L\# \}$
using E -max-lit $\langle \text{is-pos } L \rangle$
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.multip-if-maximal-less-than-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 \neg *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** $(- L, \text{Some } D) \in \text{set } \Gamma$
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* Γ $(- L) = \text{Some } (\text{Some } D)$
using *map-of-\Gamma-eq-SomeI* **by** *metis*

have $\exists \Gamma_1 \Gamma_0. \Gamma = \Gamma_1 @ (- L, \text{Some } D) \# \Gamma_0 \wedge \text{trail-false-cls } \Gamma_0 \{ \#K \in \# D. K \neq - L \# \}$
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* Γ $\{ \#K \in \# D. K \neq - L \# \}$
using *invars <s = (U_{er}, \mathcal{F} , Γ , \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 $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle C = \text{Some } E \rangle$
using *E-false E-max-lit L-neg <map-of (- 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 $K \in \# \text{eres } D \ E$
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* Γ *K*
using *D-almost-false E-false unfolding trail-false-cls-def* **by** *auto*

hence $\exists \text{opt. } (- K, \text{opt}) \in \text{set } \Gamma$
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 $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } E \rangle$
using *E-false E-max-lit L-neg* $\langle \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \rangle$ *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** $(- K, \text{Some } C) \in \text{set } \Gamma$
using *invars' is-neg L* $\langle \exists \text{opt. } (- K, \text{opt}) \in \text{set } \Gamma \rangle$
by (*metis prod.sel is-pos-neg-not-is-pos not-Some-eq*)

hence $\text{map-of } \Gamma (- K) = \text{Some } (\text{Some } C)$
using *map-of-Γ-eq-SomeI* **by** *metis*

thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } E \rangle$
using *E-false E-max-lit L-neg* $\langle \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \rangle$ *False*
eres-max-lit
 $\langle \text{is-neg } K \rangle$
using *ord-res-7.resolution-neg* **by** *metis*

qed
qed

next
case *E-not-false: False*
show *?thesis*
proof (*cases* $\exists A \in |\text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin |\text{trail-atms } \Gamma|$)
case *True*

then obtain *A* **where** *A-least: linorder-trm.is-least-in-fset*
 $\{ | A \in |\text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin |\text{trail-atms } \Gamma| \} A$

```

by (metis (no-types, lifting) bot-fset.rep-eq empty-iff ffmembership-filter
linorder-trm.ex-least-in-fset)

show ?thesis
  unfolding ⟨s = (Uer, 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 = (Uer, 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 AL)

  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 = (Uer, 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 = (Uer, 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  $\exists D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). E \prec_c D$ )
  case True

    then obtain  $F$  where linorder-cls.is-least-in-fset
      (ffilter ( $(\prec_c) E$ ) (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ ))  $F$ 
    by (metis bot-fset.rep-eq empty-iff ffilter-filter linorder-cls.ex1-least-in-fset)

    thus ?thesis
      unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } E \rangle$ 
    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  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } E \rangle$ 
        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 AL)

    hence is-neg L
      by simp

    thus ?thesis
      unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } E \rangle$ 
    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
  qed
qed

lemma ord-res-7.safe-state-if-invars:
  fixes  $N :: \text{'f gclause fset and } s$ 
  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
     $\mathcal{R} = \text{ord-res-7}$  and  $\mathcal{F} = \text{ord-res-7-final}$  and  $\mathcal{I} = \text{ord-res-7-invars}$ ]
  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  $\Gamma C L \longleftrightarrow \neg \text{trail-defined-lit } \Gamma L \wedge$ 
  linorder-lit.is-maximal-in-mset  $C L \wedge \text{trail-false-cls } \Gamma \{\#K \in\# C. K \neq L\# \}$ 

lemma trail-false-if-could-have-propagated:
  clause-could-propagate  $\Gamma C L \implies \text{trail-false-cls } ((- L, n) \# \Gamma) C$ 
  unfolding clause-could-propagate-def trail-false-cls-def trail-false-lit-def by auto

lemma atoms-of-trail-lit-atom-of-propagatable-literal:
  assumes
     $\Gamma$ -lower: linorder-trm.is-lower-set (fset (trail-atms  $\Gamma$ ))  $\mathcal{A}$  and
    C-prop: clause-could-propagate  $\Gamma C L$  and
    atm-of  $L \in \mathcal{A}$ 
  shows  $\forall A \in \text{trail-atms } \Gamma. A \prec_t \text{atm-of } L$ 
proof -
  have atm-of  $L \notin \text{trail-atms } \Gamma$ 
  using C-prop
  unfolding clause-could-propagate-def trail-defined-lit-iff-trail-defined-atm
  by argo

  then show ?thesis
  using  $\Gamma$ -lower  $\langle \text{atm-of } L \in \mathcal{A} \rangle$ 
  by (metis linorder-trm.is-lower-set-iff linorder-trm.neqE)
qed

lemma trail-false-cls-filter-mset-iff:
  trail-false-cls  $\Gamma \{\#Ka \in\# C. Ka \neq K\# \} \longleftrightarrow (\forall L \in\# C. L \neq K \longrightarrow \text{trail-false-lit } \Gamma L)$ 
  unfolding trail-false-cls-def by auto

lemma clause-could-propagate-iff: clause-could-propagate  $\Gamma C K \longleftrightarrow$ 
   $\neg \text{trail-defined-lit } \Gamma K \wedge \text{ord-res.is-maximal-lit } K C \wedge (\forall L \in\# C. L \neq K \longrightarrow \text{trail-false-lit } \Gamma L)$ 
  unfolding clause-could-propagate-def trail-false-cls-filter-mset-iff ..

lemma clause-could-propagate-efac: clause-could-propagate  $\Gamma (\text{efac } C) = \text{clause-could-propagate } \Gamma C$ 
proof (rule ext)
  fix  $L$ 
  show clause-could-propagate  $\Gamma (\text{efac } C) L \longleftrightarrow \text{clause-could-propagate } \Gamma C L$ 

```

```

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  $\mathcal{F} N \Gamma L$ 
  shows  $fBex (iefac \mathcal{F} \mid \uparrow N) (\lambda C. \text{clause-could-propagate } \Gamma C L) \longleftrightarrow$ 
     $fBex N (\lambda C. \text{clause-could-propagate } \Gamma C L)$ 
  sketch (rule iffI; elim bexE)
proof (rule iffI; elim bexE)
  fix  $C :: 'f gclause$ 
  assume  $C \mid \in \mid iefac \mathcal{F} \mid \uparrow N$  and clause-could-propagate  $\Gamma C L$ 
  thus  $\exists C \mid \in \mid N. \text{clause-could-propagate } \Gamma C L$ 
    by (metis clause-could-propagate-efac fimageE iefac-def)
next
  fix  $C :: 'f gclause$ 
  assume  $C \mid \in \mid N$  and clause-could-propagate  $\Gamma C L$ 
  thus  $\exists C \mid \in \mid iefac \mathcal{F} \mid \uparrow N. \text{clause-could-propagate } \Gamma C L$ 
    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  $\times$  '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-8 where

```

```

  decide-neg:
     $\neg (\exists C \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies$ 
    linorder-trm.is-least-in-fset  $\{|A_2 \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er}).$ 
       $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\}$   $A \implies$ 

```


$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ clause-could-propagate } \Gamma C (Pos A)) \implies$
 $\Gamma' = (Neg A, None) \# \Gamma \implies$
 $\text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid$

propagate:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{ clause-could-propagate } \Gamma C (Pos A)\} C \implies$
 $\text{linorder-lit.is-greatest-in-mset } C (Pos A) \implies$
 $\Gamma' = (Pos A, Some C) \# \Gamma \implies$
 $\text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid$

factorize:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{ clause-could-propagate } \Gamma C (Pos A)\} C \implies$
 $\neg \text{linorder-lit.is-greatest-in-mset } C (Pos A) \implies$
 $\mathcal{F}' = \text{finsert } C \mathcal{F} \implies$
 $\text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma) \mid$

resolution:

$\text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma D\}$
 $D \implies$
 $\text{linorder-lit.is-maximal-in-mset } D (Neg A) \implies$
 $\text{map-of } \Gamma (Pos A) = \text{Some } (Some C) \implies$
 $U_{er}' = \text{finsert } (eres C D) U_{er} \implies$
 $\Gamma' = \text{dropWhile } (\lambda Ln. \forall K.$
 $\text{linorder-lit.is-maximal-in-mset } (eres C D) K \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (fst$
 $Ln)) \Gamma \implies$
 $\text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

lemma *right-unique-ord-res-8:*

fixes $N :: 'f \text{ gclause fset}$

shows *right-unique* ($\text{ord-res-8 } N$)

proof (*rule right-uniqueI*)

fix $x y z$

assume *step1*: $\text{ord-res-8 } N x y$ **and** *step2*: $\text{ord-res-8 } N x z$

show $y = z$

using *step1*

proof (*cases* $N x y$ *rule*: ord-res-8.cases)

case *step1-hyps*: ($\text{decide-neg } \mathcal{F} U_{er} \Gamma A \Gamma'$)

show *?thesis*

using *step2* **unfolding** $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma) z$ *rule*: ord-res-8.cases)

case ($\text{decide-neg } A \Gamma'$)

```

    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  $\mathcal{F} 2'$ )
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 (resolution D2 A2 C2  $U_{er} 2' \Gamma 2'$ )
with step1-hyps have False
  by (metis linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
qed
next
case step1-hyps: (propagate  $\mathcal{F} U_{er} \Gamma A C \Gamma'$ )
show ?thesis
  using step2 unfolding  $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
proof (cases  $N (U_{er}, \mathcal{F}, \Gamma) z$  rule: ord-res-8.cases)
case (decide-neg A2  $\Gamma 2'$ )
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 (propagate A2 C2  $\Gamma 2'$ )
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  $\mathcal{F} 2'$ )
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 (resolution D2 A2 C2  $U_{er} 2' \Gamma 2'$ )
with step1-hyps have False
  by (metis linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..

```

```

qed
next
case step1-hyps: (factorize  $\mathcal{F}$   $U_{er}$   $\Gamma$   $A$   $C$   $\mathcal{F}'$ )
show ?thesis
  using step2 unfolding  $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
proof (cases  $N (U_{er}, \mathcal{F}, \Gamma)$   $z$  rule: ord-res-8.cases)
  case (decide-neg  $A2$   $\Gamma2'$ )
  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  $A2$   $C2$   $\Gamma2'$ )
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  $D2$   $A2$   $C2$   $U_{er}2'$   $\Gamma2'$ )
with step1-hyps have False
  by (metis linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
qed
next
case step1-hyps: (resolution  $\Gamma$   $\mathcal{F}$   $U_{er}$   $D$   $A$   $C$   $U_{er}'$   $\Gamma'$ )
show ?thesis
  using step2 unfolding  $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
proof (cases  $N (U_{er}, \mathcal{F}, \Gamma)$   $z$  rule: ord-res-8.cases)
  case (decide-neg  $A$   $\Gamma'$ )
  with step1-hyps have False
  by (metis (no-types, opaque-lifting) linorder-cls.is-least-in-ffilter-iff)
  thus ?thesis ..
next
case (propagate  $A$   $C$   $\Gamma'$ )
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 Uer2' Γ2')
have D2 = D
  using ⟨linorder-cls.is-least-in-fset (ffilter (trail-false-cls  $\Gamma$ ) (iefac  $\mathcal{F}$  | $\uparrow$  ( $N$ 
| $\cup$ |  $U_{er}$ ))) D⟩
  using ⟨linorder-cls.is-least-in-fset (ffilter (trail-false-cls  $\Gamma$ ) (iefac  $\mathcal{F}$  | $\uparrow$  ( $N$ 
| $\cup$ |  $U_{er}$ ))) 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 ⟨ $y = (U_{er}', \mathcal{F}, \Gamma)$ ⟩ ⟨ $z = (U_{er}2', \mathcal{F}, \Gamma2')$ ⟩ prod.inject
proof (intro conjI)
  show  $U_{er}' = U_{er}2'$ 
  unfolding ⟨ $U_{er}' = \text{finsert}(\text{eres } C \ D) \ U_{er}$ ⟩ ⟨ $U_{er}2' = \text{finsert}(\text{eres } C2 \ D2)$ 
 $U_{er}$ ⟩
  unfolding ⟨D2 = D⟩ ⟨C2 = C⟩ ..
next
  show  $\mathcal{F} = \mathcal{F}$  ..
next
  show  $\Gamma' = \Gamma2'$ 
  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* \Rightarrow *bool* **where**
model-found:
 $\neg (\exists A \ | \in \ | \ \text{atms-of-cls} \ (N \ | \cup \ | \ U_{er}). \ A \ | \notin \ \text{trail-atms} \ \Gamma) \Longrightarrow$
 $\neg (\exists C \ | \in \ | \ \text{iefac} \ \mathcal{F} \ | \uparrow \ (N \ | \cup \ | \ U_{er}). \ \text{trail-false-cls} \ \Gamma \ C) \Longrightarrow$
ord-res-8-final ($N, U_{er}, \mathcal{F}, \Gamma$) |

contradiction-found:
 $\{\#\} \ | \in \ | \ \text{iefac} \ \mathcal{F} \ | \uparrow \ (N \ | \cup \ | \ U_{er}) \Longrightarrow$
ord-res-8-final ($N, U_{er}, \mathcal{F}, \Gamma$)

sublocale *ord-res-8-antics*: *semantics* **where**

step = constant-context ord-res-8 and
final = ord-res-8-final
proof *unfold-locales*
fix $S :: 'f \text{ ord-res-8-state}$

obtain
 $N \ U_{er} \ \mathcal{F} :: 'f \text{ gterm clause fset and}$
 $\Gamma :: ('f \text{ gterm literal} \times 'f \text{ gterm literal multiset option}) \text{ list where}$
 $S\text{-def: } S = (N, U_{er}, \mathcal{F}, \Gamma)$
by *(metis prod.exhaust)*

assume *ord-res-8-final S*

hence $\nexists x. \text{ord-res-8 } N \ (U_{er}, \mathcal{F}, \Gamma) \ x$
unfolding *S-def*
proof *(cases (N, U_{er}, F, Γ) rule: ord-res-8-final.cases)*
case *model-found*

have $\nexists A. \text{linorder-trm.is-least-in-fset } \{ |A_2 | \in | \text{atms-of-clss } (N \ | \cup | \ U_{er}).$
 $\forall A_1 | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} \ A$
using $\langle \neg (\exists A | \in | \text{atms-of-clss } (N \ | \cup | \ U_{er}). A | \notin | \text{trail-atms } \Gamma) \rangle$
using *linorder-trm.is-least-in-ffilter-iff by fastforce*

moreover have $\nexists C. \text{linorder-cls.is-least-in-fset}$
 $(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er}))) \ C$
using $\langle \neg \text{fBex } (\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er})) (\text{trail-false-cls } \Gamma) \rangle$
by *(metis linorder-cls.is-least-in-ffilter-iff)*

ultimately show *?thesis*
unfolding *ord-res-8.simps by blast*
next
case *contradiction-found*

hence $\exists C | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er}). \text{trail-false-cls } \Gamma \ C$
using *trail-false-cls-empty by metis*

moreover have $\nexists D \ A. \text{linorder-cls.is-least-in-fset } (\text{ffilter } (\text{trail-false-cls } \Gamma)$
 $(\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er}))) \ D \wedge \text{ord-res.is-maximal-lit } (\text{Neg } A) \ D$
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-empty)

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

$$\begin{aligned} \text{trail-is-sorted } N s &\longleftrightarrow \\ (\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) &\longrightarrow \\ \text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) &\prec_t \text{atm-of } (fst x)) \Gamma) \end{aligned}$$

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* $\mathcal{F} U_{er} \Gamma A \Gamma'$)

have $\forall x \in | \text{trail-atms } \Gamma. x \prec_t A$

using *step-hyps* **unfolding** *linorder-trm.is-least-in-filter-iff* **by** *simp*

hence $\forall x \in \text{set } \Gamma. \text{atm-of } (fst x) \prec_t A$

by (*simp add*: *fset-trail-atms*)

moreover **have** *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$

using *invar* **by** (*simp add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *trail-is-sorted-def*)

ultimately **have** *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$

by (*simp add*: $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$)

thus *?thesis*

by (*simp add*: $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$ *trail-is-sorted-def*)

next

case *step-hyps*: (*propagate* $\mathcal{F} U_{er} \Gamma A C \Gamma'$)

have $\forall x \in | \text{trail-atms } \Gamma. x \prec_t A$

using *step-hyps* **unfolding** *linorder-trm.is-least-in-filter-iff* **by** *simp*

hence $\forall x \in \text{set } \Gamma. \text{atm-of } (fst x) \prec_t A$

by (*simp add*: *fset-trail-atms*)

moreover **have** *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$

using *invar* **by** (*simp add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *trail-is-sorted-def*)

ultimately **have** *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$

by (*simp add*: $\langle \Gamma' = (\text{Pos } A, \text{Some } C) \# \Gamma \rangle$)

thus *?thesis*

by (*simp add*: $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$ *trail-is-sorted-def*)

next

case (*factorize* $\mathcal{F} U_{er} \Gamma A C \mathcal{F}'$)

have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
using invar by $(\text{simp add: } \langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle \text{ trail-is-sorted-def})$

thus *?thesis*
by $(\text{simp add: } \langle s' = (U_{er}, \mathcal{F}', \Gamma) \rangle \text{ trail-is-sorted-def})$

next
case *step-hyps*: $(\text{resolution } \Gamma \mathcal{F} U_{er} D A C U_{er}' \Gamma')$

have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$
using invar by $(\text{simp add: } \langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle \text{ trail-is-sorted-def})$

hence *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$
unfolding *step-hyps*(γ) **by** $(\text{rule sorted-wrt-dropWhile})$

thus *?thesis*
by $(\text{simp add: } \langle s' = (U_{er}', \mathcal{F}, \Gamma') \rangle \text{ trail-is-sorted-def})$

qed

inductive *trail-annotations-invars*
for $N :: 'f \text{ gterm literal multiset fset}$
where

Nil:
trail-annotations-invars $N (U_{er}, \mathcal{F}, [])$ |

Cons-None:
trail-annotations-invars $N (U_{er}, \mathcal{F}, (L, None) \# \Gamma)$
if *trail-annotations-invars* $N (U_{er}, \mathcal{F}, \Gamma)$ |

Cons-Some:
trail-annotations-invars $N (U_{er}, \mathcal{F}, (L, Some D) \# \Gamma)$
if *linorder-lit.is-greatest-in-mset* $D L$ **and**
 $D \in \text{iefac } \mathcal{F} \mid (N \cup U_{er})$ **and**
trail-false-cls $\Gamma \{ \#K \in \# D. K \neq L \# \}$ **and**
linorder-cls.is-least-in-fset
 $\{ \mid D \in \text{iefac } \mathcal{F} \mid (N \cup U_{er}). \text{ clause-could-propagate } \Gamma D L \}$ D **and**
trail-annotations-invars $N (U_{er}, \mathcal{F}, \Gamma)$

lemma
assumes

linorder-lit.is-greatest-in-mset $C L$ **and**
trail-false-cls $\Gamma \{ \#K \in \# C. K \neq L \# \}$ **and**
 $\neg \text{trail-defined-cls } \Gamma C$

shows *clause-could-propagate* $\Gamma C L$
unfolding *clause-could-propagate-iff*

proof (intro conjI)
show *linorder-lit.is-maximal-in-mset* $C L$
using $\langle \text{linorder-lit.is-greatest-in-mset } C L \rangle$
by $(\text{metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset})$

hence $L \in \# C$
unfolding *linorder-lit.is-maximal-in-mset-iff* ..

```

show  $\forall K \in \# C. K \neq L \longrightarrow \text{trail-false-lit } \Gamma K$ 
  using  $\langle \text{trail-false-cls } \Gamma \{ \#K \in \# C. K \neq L \# \} \rangle$ 
  unfolding trail-false-cls-def
  by simp

hence  $\forall K \in \# C. K \neq L \longrightarrow \text{trail-defined-lit } \Gamma K$ 
  using trail-defined-lit-iff-true-or-false by metis

thus  $\neg \text{trail-defined-lit } \Gamma L$ 
  using  $\langle \neg \text{trail-defined-cls } \Gamma C \rangle \langle L \in \# C \rangle$ 
  unfolding trail-defined-cls-def
  by metis
qed

lemma propagating-clause-in-clauses:
  assumes trail-annotations-invars  $N (U_{er}, \mathcal{F}, \Gamma)$  and map-of  $\Gamma L = \text{Some } (\text{Some } C)$ 
  shows  $C \in | \text{iefac } \mathcal{F} |^1 (N \cup U_{er})$ 
  using assms
proof (induction  $(U_{er}, \mathcal{F}, \Gamma)$  arbitrary:  $\Gamma$  rule: trail-annotations-invars.induct)
  case Nil
  hence False
  by simp
  thus ?case ..
next
  case (Cons-None  $\Gamma K$ )
  thus ?case
  by (metis map-of-Cons-code(2) option.discI option.inject)
next
  case (Cons-Some  $D K \Gamma$ )
  thus ?case
  by (metis map-of-Cons-code(2) option.inject)
qed

lemma trail-annotations-invars-mono-wrt-trail-suffix:
  assumes suffix  $\Gamma' \Gamma$  trail-annotations-invars  $N (U_{er}, \mathcal{F}, \Gamma)$ 
  shows trail-annotations-invars  $N (U_{er}, \mathcal{F}, \Gamma')$ 
  using assms(2,1)
proof (induction  $(U_{er}, \mathcal{F}, \Gamma)$  arbitrary:  $\Gamma \Gamma'$  rule: trail-annotations-invars.induct)
  case Nil
  thus ?case
  by (simp add: trail-annotations-invars.Nil)
next
  case (Cons-None  $\Gamma L$ )
  have  $\Gamma' = (L, \text{None}) \# \Gamma \vee \text{suffix } \Gamma' \Gamma$ 
  using Cons-None.premis 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.premis 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 Uer Γ A Γ')
show ?thesis
  unfolding ⟨s' = (Uer, F, Γ')⟩ ⟨Γ' = (Neg A, None) # Γ⟩
proof (rule trail-annotations-invars.Cons-None)
  show trail-annotations-invars N (Uer, F, Γ)
    using invars(1) by (simp add: ⟨s = (Uer, F, Γ)⟩)
qed
next
case step-hyps: (propagate F Uer Γ A C Γ')
show ?thesis
  unfolding ⟨s' = (Uer, 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 |∪| Uer)
  using step-hyps(5)
  by (simp add: linorder-cls.is-least-in-fset-iff)
next
show trail-false-cl 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 |∪| Uer). clause-could-propagate Γ C (Pos A)|} C
  using step-hyps by argo
next
show trail-annotations-invars N (Uer, F, Γ)
  using invars(1) by (simp add: ⟨s = (Uer, F, Γ)⟩)
qed

```

next
case *step-hyps*: (*factorize* \mathcal{F} U_{er} Γ A E \mathcal{F}')

hence *efac* $E \neq E$
by (*metis* (*no-types*, *lifting*) *ex1-efac-eq-factoring-chain* *ex-ground-factoringI* *linorder-clc.is-least-in-ffilter-iff* *clause-could-propagate-iff*)

moreover have *clause-could-propagate* Γ E (*Pos* A)
using *step-hyps* **unfolding** *linorder-clc.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 $\mathcal{F} \sqsubseteq \mathcal{F}'$
unfolding $\langle \mathcal{F}' = \text{finsert } E \ \mathcal{F} \rangle$ **by** *auto*

have *trail-annotations-invars* N (U_{er} , \mathcal{F} , Γ)
using *invars(1)* **by** (*simp add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$)

moreover have $\forall A_1 \in \text{trail-atms } \Gamma. A_1 \prec_t A$
using *step-hyps* **unfolding** *linorder-trm.is-least-in-ffilter-iff* **by** *blast*

ultimately show *?thesis*
unfolding $\langle s' = (U_{er}, \mathcal{F}', \Gamma) \rangle$
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 $\forall A_1 \in \text{trail-atms } \Gamma. A_1 \prec_t A$
using *Cons-None.prem*s **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: $\forall y \in \text{iefac } \mathcal{F} \mid \uparrow (N \cup U_{er}). y \neq C \longrightarrow \text{clause-could-propagate } \Gamma$ y
 $L \longrightarrow C \prec_c y$
using *Cons-Some.hyps(4)* **unfolding** *linorder-clc.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 $\langle \text{ord-res.is-strictly-maximal-lit L C} \rangle$.

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 $C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
using $\langle C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle \langle \mathcal{F} \subseteq \mathcal{F}' \rangle$
by (*smt (verit, best) assms fimage-iff fsubsetD iefac-def*)

show $C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
using $\langle C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$.

show *trail-false-cls $\Gamma \{\#x \in \# C. x \neq L\# \}$*
using $\langle \text{trail-false-cls } \Gamma \{\#x \in \# C. x \neq L\# \} \rangle$.

show *linorder-cls.is-least-in-fset*
 $\{ \mid C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C L \} C$
unfolding *linorder-cls.is-least-in-filter-iff*
proof (*intro conjI ballI impI*)
show $C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
using $\langle C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) \rangle$.

next
show *clause-could-propagate $\Gamma C L$*
using $\langle \text{clause-could-propagate } \Gamma C L \rangle$.

next
fix $D :: 'f \text{ gterm literal multiset}$
assume $D \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $D \neq C$ **and** *clause-could-propagate*
 $\Gamma D L$

have *atm-of L $\prec_t A$*
using *Cons-Some.prem* **by** (*simp add: fset-trail-atms*)

hence $L \prec_l \text{Pos } A$
by (*cases L simp-all*)

moreover have *ord-res.is-maximal-lit L D*
using $\langle \text{clause-could-propagate } \Gamma D L \rangle$ **unfolding** *clause-could-propagate-iff*
by *metis*

```

ultimately have  $D \prec_c \text{efac } E$ 
  using  $\langle \text{ord-res.is-strictly-maximal-lit } (Pos\ A)\ (\text{efac } E) \rangle$ 
  by  $(\text{metis linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset}$ 
     $\text{linorder-lit.mulp-if-maximal-less-that-maximal})$ 

hence  $D \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er})$ 
  using  $\langle D \in | \text{iefac } \mathcal{F}' \uparrow | (N \cup U_{er}) \rangle$ 
  unfolding  $\langle \mathcal{F}' = \text{finsert } E\ \mathcal{F} \rangle$ 
  using  $\text{iefac-def}$  by force

thus  $C \prec_c D$ 
  using  $C\text{-least } \langle D \neq C \rangle \langle \text{clause-could-propagate } \Gamma\ D\ L \rangle$  by  $\text{metis}$ 
qed

have  $\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A$ 
  using  $\text{Cons-Some.premis}$  by  $(\text{simp add: fset-trail-atms})$ 

thus  $\text{trail-annotations-invars } N\ (U_{er}, \mathcal{F}', \Gamma)$ 
  using  $\text{Cons-Some.hyps}$  by  $\text{argo}$ 
qed
qed
next
case  $\text{step-hyps: } (\text{resolution } \Gamma\ \mathcal{F}\ U_{er}\ E\ A\ D\ U_{er}'\ \Gamma')$ 

show  $?thesis$ 
proof  $(\text{cases eres } D\ E = \{\#\})$ 
  case  $\text{True}$ 
  hence  $\nexists K. \text{ord-res.is-maximal-lit } K\ (\text{eres } D\ E)$ 
    by  $(\text{simp add: linorder-lit.is-maximal-in-mset-iff})$ 
  hence  $\Gamma' = []$ 
  unfolding  $\text{step-hyps}$  by  $\text{simp}$ 
  thus  $?thesis$ 
  unfolding  $\langle s' = (U_{er}', \mathcal{F}, \Gamma') \rangle$ 
  using  $\text{trail-annotations-invars.Nil}$  by  $\text{metis}$ 
next
case  $\text{False}$ 

then obtain  $K$  where  $\text{eres-max-lit: linorder-lit.is-maximal-in-mset } (\text{eres } D\ E)$ 
 $K$ 
  using  $\text{linorder-lit.ex-maximal-in-mset}$  by  $\text{metis}$ 

have  $\Gamma'\text{-eq: } \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln))\ \Gamma$ 
  unfolding  $\text{step-hyps}(?)$ 
proof  $(\text{rule dropWhile-cong})$ 
  show  $\Gamma = \Gamma ..$ 
next
fix  $x :: 'f\ \text{gterm literal} \times 'f\ \text{gterm literal multiset option}$ 
assume  $x \in \text{set } \Gamma$ 
show  $(\forall K. \text{ord-res.is-maximal-lit } K\ (\text{eres } D\ E) \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst}$ 

```

$x)) =$
 (atm-of $K \preceq_t$ 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: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$)

moreover have suffix $\Gamma' \Gamma$
unfolding step-hyps
using suffix-dropWhile **by** metis

moreover have $\forall Ln \in \text{set } \Gamma'. \neg (\text{atm-of } K \preceq_t \text{ atm-of } (\text{fst } Ln))$
unfolding Γ' -eq
proof (rule ball-set-dropWhile-if-sorted-wrt-and-monotone-on)
have sorted-wrt ($\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{ atm-of } (\text{fst } x)$) Γ
using invars(2)
by (simp add: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ trail-is-sorted-def sorted-wrt-map)

thus sorted-wrt ($\lambda x y. \text{fst } y \prec_l \text{fst } x$) Γ
proof (rule sorted-wrt-mono-rel[rotated])
show $\bigwedge x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x) \implies \text{fst } y \prec_l \text{fst } x$
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) ($\lambda x y. \text{fst } y \prec_l \text{fst } x$) ($\lambda Ln y. y \leq Ln$)
 ($\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$)
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 $\langle s' = (U_{er}', \mathcal{F}, \Gamma') \rangle$
proof (induction (U_{er}, \mathcal{F} , Γ) arbitrary: $\Gamma \Gamma'$ rule: trail-annotations-invars.induct)
case Nil
thus ?case
by (simp add: trail-annotations-invars.Nil)

next
case (Cons-None ΓL)
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:  $\forall y | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). y \neq C \longrightarrow \text{clause-could-propagate } \Gamma$ 
  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, Some C) # Γ = Γ'' @ Γ'
  using Cons-Some.premis by (auto elim: suffixE)

show ?case
proof (cases Γ'')
  case Nil
  hence Γ' = (L, Some C) # Γ
  using ⟨(L, Some C) # Γ = Γ'' @ Γ'⟩ by simp

show ?thesis
  unfolding ⟨Γ' = (L, Some C) # Γ⟩
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 | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ 
  using ⟨C | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \rangle \langle U_{er}' = \text{finsert } (\text{eres } D \ E) \ U_{er} \rangle$ 
by simp

show trail-false-cl Γ {#K ∈# C. K ≠ L#}
  using ⟨trail-false-cl Γ {#K ∈# C. K ≠ L#}⟩ .

  show linorder-cls.is-least-in-fset {C | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ .
  clause-could-propagate Γ C L} C
  unfolding linorder-cls.is-least-in-ffilter-iff
proof (intro conjI ballI impI)
  show C | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ 
  using ⟨C | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}') \rangle$  .
  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 | ∈ | iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')$ 
  and x ≠ C
  and clause-could-propagate Γ x L

```

have *linorder-lit.is-maximal-in-mset* $x L$
using $\langle \text{clause-could-propagate } \Gamma x L \rangle$ **unfolding** *clause-could-propagate-def*
by *argo*

moreover have $L \prec_t K$
using $\langle \forall Ln \in \text{set } \Gamma'. \neg (\text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \rangle$
unfolding $\langle \Gamma' = (L, \text{Some } C) \# \Gamma \rangle$
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 $\text{set-mset } x \neq \text{set-mset } (\text{eres } D E)$
using *eres-max-lit*
by (*metis* *linorder-lit.is-maximal-in-mset-iff* *linorder-lit.neq-iff*)

hence $x \neq \text{iefac } \mathcal{F} (\text{eres } D E)$
using *iefac-def* **by** *auto*

hence $x \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$
using $\langle x \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}) \rangle$
unfolding $\langle U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \rangle$
by *simp*

then show $C \prec_c x$
using *C-least* $\langle x \neq C \rangle$ $\langle \text{clause-could-propagate } \Gamma x L \rangle$ **by** *metis*
qed

show *trail-annotations-invars* $N (U_{er}', \mathcal{F}, \Gamma)$
using *Cons-Some*
by (*simp* *add*: $\langle \Gamma' = (L, \text{Some } C) \# \Gamma \rangle$)

qed

next

case (*Cons a list*)

then show *?thesis*

by (*metis* *Cons-Some.hyps(6)* *Cons-Some.prem*s $\langle (L, \text{Some } C) \# \Gamma = \Gamma''$

@ $\Gamma' \rangle$ *empty-iff*

list.set(1) *list.set-intros(1)* *self-append-conv2* *suffix-Cons*)

qed

qed

qed

qed

definition *trail-is-lower-set* **where**

trail-is-lower-set $N s \longleftrightarrow$

$(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow$

linorder-trm.is-lower-fset (*trail-atms* Γ) (*atms-of-cls*s $(N \cup U_{er}))$)

lemma *atoms-not-in-clause-set-undefined-if-trail-is-sorted-lower-set:*
assumes *invar: trail-is-lower-set* $N (U_{er}, \mathcal{F}, \Gamma)$
shows $\forall A. A \notin \text{atms-of-clss } (N \cup U_{er}) \longrightarrow A \notin \text{trail-atms } \Gamma$
using *invar[unfolding 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* $\mathcal{F} U_{er} \Gamma A \Gamma')$

have

A-in: $A \in \text{atms-of-clss } (N \cup U_{er})$ **and**

A-gt: $\forall x \in \text{trail-atms } \Gamma. x \prec_t A$ **and**

A-least: $\forall x \in \text{atms-of-clss } (N \cup U_{er}). (\forall w \in \text{trail-atms } \Gamma. w \prec_t x) \longrightarrow x \neq A \longrightarrow A \prec_t x$

using *step-hyps unfolding linorder-trm.is-least-in-filter-iff by simp-all*

have *trail-atms* $\Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$

using *step-hyps by simp*

have

inv1: *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**

inv2: *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \cup U_{er}))$

using *invars(1,3)*

by (*simp-all only: <s = (U_{er}, F, Γ)> trail-is-lower-set-def trail-is-sorted-def*)

have *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$

unfolding $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle \text{list.map sorted-wrt.simps}$

proof (*intro conjI ballI*)

fix x

assume $x \in \text{set } \Gamma$

hence *atm-of* $(fst x) \in \text{trail-atms } \Gamma$

by (*auto simp: fset-trail-atms*)

hence *atm-of* $(fst x) \prec_t \text{atm-of } (\text{Neg } A)$

using *A-gt by simp*

thus *atm-of* $(fst x) \prec_t \text{atm-of } (fst (\text{Neg } A, \text{None}))$

unfolding *comp-def prod.sel* .

next

show *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$

using *inv1* .

qed

moreover have $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma') (\text{atms-of-clss } (N \mid \cup U_{er}))$
unfolding $\langle \text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma) \rangle \text{finsert.rep-eq}$
proof (*rule linorder-trm.is-lower-set-insertI*)
show $A \mid \in \mid \text{atms-of-clss } (N \mid \cup U_{er})$
using $A\text{-in}$.
next
show $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup U_{er}). w \prec_t A \longrightarrow w \mid \in \mid \text{trail-atms } \Gamma$
using $A\text{-least inv2}$
by (*metis linorder-trm.is-lower-set-iff linorder-trm.not-less-iff-gr-or-eq*)
next
show $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma)$
 $(\text{atms-of-clss } (N \mid \cup U_{er}))$
using inv2 .
qed

ultimately show $?thesis$
by (*simp add: $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle \text{trail-is-lower-set-def}$*)
next
case $\text{step-hyps: } (\text{propagate } \mathcal{F} U_{er} \Gamma A C \Gamma')$

have
 $A\text{-in: } A \mid \in \mid \text{atms-of-clss } (N \mid \cup U_{er})$ **and**
 $A\text{-gt: } \forall x \mid \in \mid \text{trail-atms } \Gamma. x \prec_t A$ **and**
 $A\text{-least: } \forall x \mid \in \mid \text{atms-of-clss } (N \mid \cup U_{er}). (\forall w \mid \in \mid \text{trail-atms } \Gamma. w \prec_t x) \longrightarrow x$
 $\neq A \longrightarrow A \prec_t x$
using $\text{step-hyps unfolding linorder-trm.is-least-in-filter-iff by simp-all}$

have $\text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$
using step-hyps by simp

have
 $\text{inv1: sorted-wrt } (\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$ **and**
 $\text{inv2: linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup U_{er}))$
using $\text{invars}(1,3)$
by (*simp-all only: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle \text{trail-is-lower-set-def trail-is-sorted-def}$*)

have $\text{sorted-wrt } (\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma'$
unfolding $\langle \Gamma' = (\text{Pos } A, \text{Some } C) \# \Gamma \rangle \text{list.map sorted-wrt.simps}$
proof (*intro conjI ballI*)
fix x
assume $x \in \text{set } \Gamma$
hence $\text{atm-of } (\text{fst } x) \mid \in \mid \text{trail-atms } \Gamma$
by (*auto simp: fset-trail-atms*)
hence $\text{atm-of } (\text{fst } x) \prec_t \text{atm-of } (\text{Pos } A)$
using $A\text{-gt by simp}$
thus $\text{atm-of } (\text{fst } x) \prec_t \text{atm-of } (\text{fst } (\text{Pos } A, \text{Some } C))$
unfolding comp-def prod.sel .

```

next
  show sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma$ 
    using inv1 .
qed

moreover have linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-clss ( $N \mid \cup \mid U_{er}$ ))
  unfolding  $\langle \text{trail-atms } \Gamma' = \text{finsert } A \text{ (trail-atms } \Gamma) \rangle$  finsert.rep-eq
proof (rule linorder-trm.is-lower-set-insertI)
  show  $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ 
    using A-in .
next
  show  $\forall w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). w \prec_t A \longrightarrow w \mid \in \mid \text{trail-atms } \Gamma$ 
    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  $\Gamma$ ) (atms-of-clss ( $N \mid \cup \mid U_{er}$ ))
    using inv2 .
qed

ultimately show ?thesis
  by (simp add:  $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$  trail-is-lower-set-def)
next
  case (factorize  $\mathcal{F} U_{er} \Gamma A C \mathcal{F}'$ )
  thus ?thesis
    using invars(1) by (simp add: trail-is-lower-set-def fset-trail-atms)
next
  case step-hyps: (resolution  $\Gamma \mathcal{F} U_{er} D A C U_{er}' \Gamma'$ )

  have suffix  $\Gamma' \Gamma$ 
    using step-hyps suffix-dropWhile by blast

  then obtain  $\Gamma''$  where  $\Gamma = \Gamma'' @ \Gamma'$ 
    unfolding suffix-def by metis

  have atms-of-clss ( $N \mid \cup \mid U_{er}'$ ) = atms-of-clss (finsert (eres  $C D$ ) ( $N \mid \cup \mid U_{er}$ ))
    unfolding  $\langle U_{er}' = \text{finsert } (eres C D) U_{er} \rangle$  by simp
  also have  $\dots = \text{atms-of-clss } (eres C D) \mid \cup \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ 
    unfolding atms-of-clss-finsert ..
  also have  $\dots = \text{atms-of-clss } (N \mid \cup \mid U_{er})$ 
proof -
  have  $\forall A \mid \in \mid \text{atms-of-clss } (eres C D). A \mid \in \mid \text{atms-of-clss } C \vee A \mid \in \mid \text{atms-of-clss } D$ 
    using lit-in-one-of-resolvents-if-in-eres
    by (smt (verit, best) atms-of-clss-def fimage-iff fset-fset-mset)

  moreover have  $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 
    using invars(2)[unfolded  $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ ] step-hyps(5)
    by (metis propagating-clause-in-clauses)

```

moreover have $D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using *linorder-cls.is-least-in-filter-iff step-hyps(3)* **by** *blast*

ultimately have $\forall A \in \text{atms-of-cls } (eres C D). A \in \text{atms-of-clss } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))$
by (*metis atms-of-clss-finsert funion-iff mk-disjoint-finsert*)

hence $\forall A \in \text{atms-of-cls } (eres C D). A \in \text{atms-of-clss } (N \mid \cup \mid U_{er})$
unfolding *atms-of-clss-fimage-iefac* .

thus *?thesis*
by *blast*

qed

finally have $\text{atms-of-clss } (N \mid \cup \mid U_{er}') = \text{atms-of-clss } (N \mid \cup \mid U_{er})$.

have
inv1: sorted-wrt $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ **and**
inv2: linorder-trm.is-lower-fset $(\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$
using *invars(1,3)*
by (*simp-all only: \langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle trail-is-lower-set-def trail-is-sorted-def*)

have *sorted-wrt* $(\lambda x y. y \prec_t x) (\text{map } (\text{atm-of} \circ \text{fst}) \Gamma)$
using *inv1* **by** (*simp add: sorted-wrt-map*)

hence *sorted-wrt* $(\lambda x y. y \prec_t x) (\text{map } (\text{atm-of} \circ \text{fst}) \Gamma'' @ \text{map } (\text{atm-of} \circ \text{fst}) \Gamma')$
by (*simp add: \langle \Gamma = \Gamma'' @ \Gamma' \rangle*)

moreover have *linorder-trm.is-lower-set* $(\text{set } (\text{map } (\text{atm-of} \circ \text{fst}) \Gamma'' @ \text{map } (\text{atm-of} \circ \text{fst}) \Gamma'))$
 $(\text{fset } (\text{atms-of-clss } (N \mid \cup \mid U_{er})))$
using *inv2* $\langle \Gamma = \Gamma'' @ \Gamma' \rangle$
by (*metis image-comp list.set-map map-append fset-trail-atms*)

ultimately have *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma') (\text{atms-of-clss } (N \mid \cup \mid U_{er}'))$
using *linorder-trm.sorted-and-lower-set-appendD-right*
using $\langle \text{atms-of-clss } (N \mid \cup \mid U_{er}') = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$
by (*metis (no-types, lifting) image-comp list.set-map fset-trail-atms*)

thus *?thesis*
by (*simp add: \langle s' = (U_{er}', \mathcal{F}, \Gamma') \rangle trail-is-lower-set-def*)

qed

definition *false-cls-is-empty-or-has-neg-max-lit* **where**
false-cls-is-empty-or-has-neg-max-lit $N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow (\forall C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{trail-false-cls } \Gamma C \longrightarrow C = \{\#\} \vee (\exists A. \text{linorder-lit.is-maximal-in-mset } C$
 $(\text{Neg } A))))$

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-cls* ($N \mid \cup \mid U_{er}$))

using \langle *trail-is-lower-set* N s \rangle [*unfolded trail-is-lower-set-def*,

rule-format, *OF* $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$].

have Γ -*sorted*: *sorted-wrt* ($\lambda x y. \text{atm-of } (fst\ y) \prec_t \text{atm-of } (fst\ x)$) Γ

using \langle *trail-is-sorted* N s \rangle [*unfolded trail-is-sorted-def*,

rule-format, *OF* $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$].

have *trail-consistent* Γ

using *trail-consistent-if-sorted-wrt-atoms*[*OF* Γ -*sorted*].

hence *trail-consistent* Γ'

unfolding $\langle \Gamma' = (Neg\ A, None) \# \Gamma \rangle$

proof (*rule* *trail-consistent.Cons* [*rotated*])

show \neg *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*: $x \mid \in \mid \text{trail-atms } \Gamma \longleftrightarrow x \prec_t A$

if x -*in*: $x \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er})$ **for** x

proof (*rule* *iffI*)

assume $x \mid \in \mid \text{trail-atms } \Gamma$

thus $x \prec_t A$

using *step-hyps*(4)

unfolding *linorder-trm.is-least-in-ffilter-iff*

by *blast*

next

assume $x \prec_t A$

thus $x \mid \in \mid \text{trail-atms } \Gamma$

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.neg-iff x-in*)

qed

have \neg *trail-false-cls* $\Gamma' C$ **if** *C-in*: $C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **for** C
proof –
have \neg *trail-false-cls* ΓC
using *C-in step-hyps*(3) **by** *metis*
hence *trail-true-cls* $\Gamma C \vee \neg$ *trail-defined-cls* ΓC
using *trail-true-or-false-cls-if-defined* **by** *metis*
thus \neg *trail-false-cls* $\Gamma' C$
proof (*elim disjE*)
assume *trail-true-cls* ΓC
hence *trail-true-cls* $\Gamma' C$
unfolding $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ *trail-true-cls-def*
by (*metis image-insert insert-iff list.set*(2) *trail-true-lit-def*)
thus \neg *trail-false-cls* $\Gamma' C$
using \langle *trail-consistent* $\Gamma' \rangle$
by (*metis not-trail-true-cls-and-trail-false-cls*)
next
assume \neg *trail-defined-cls* ΓC
then obtain L **where** *L-max*: *ord-res.is-maximal-lit* $L C$
by (*metis* \langle \neg *trail-false-cls* $\Gamma C \rangle$ *linorder-lit.ex-maximal-in-mset trail-false-cls-empty*)

have $L \in \# C$
using *L-max linorder-lit.is-maximal-in-mset-iff* **by** *metis*

have *atm-of* $L \in | \text{atms-of-cls} (N \mid \cup \mid U_{er})$
using *C-in* $\langle L \in \# C \rangle$ **by** (*metis atm-of-in-atms-of-clsI*)

show \neg *trail-false-cls* $\Gamma' C$
proof (*cases atm-of* $L = A$)
case *True*

have \neg *trail-defined-lit* $\Gamma (Pos A)$
using *step-hyps*(4)
unfolding *trail-defined-lit-iff-trail-defined-atm linorder-trm.is-least-in-ffilter-iff*
by *auto*

hence $(\exists K \in \# C. K \neq Pos A \wedge \neg$ *trail-false-lit* $\Gamma K) \vee$
 \neg *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 $K \in \# C$ **and** $K \neq Pos A$ **and** \neg *trail-false-lit* ΓK
thus \neg *trail-false-cls* $\Gamma' C$
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  $\neg \text{ord-res.is-maximal-lit } (Pos\ A)\ C$ 

hence  $L = Neg\ A$ 
  using  $\langle atm\text{-of } L = A \rangle L\text{-max}$  by  $(metis\ literal.exhaust\text{-}sel)$ 

thus  $\neg \text{trail-false-cl}\ \Gamma'\ C$ 
  unfolding  $\langle \Gamma' = (Neg\ A, None)\ \#\ \Gamma \rangle$ 
  unfolding  $\text{trail-false-cl}\text{-}def\ \text{trail-false-lit}\text{-}def$ 
  using  $\langle L \in\ \#\ C \rangle [unfolding\ \langle L = Neg\ A \rangle]$ 
  by  $(metis\ \langle \neg \text{trail-defined-cl}\ \Gamma\ C \rangle\ fst\text{-}conv\ image\text{-}insert\ insertE$ 
 $list.simps(15)$ 
 $\text{trail-defined-cl}\text{-}def\ \text{trail-defined-lit}\text{-}def\ uminus\text{-}lit\text{-}swap\ uminus\text{-}not\text{-}id')$ 
qed
next
case  $False$ 

moreover have  $\neg atm\text{-of } L \prec_t\ A$ 
proof  $(rule\ notI)$ 
  assume  $atm\text{-of } L \prec_t\ A$ 
  moreover have  $\forall K \in\ \#\ C. atm\text{-of } K \preceq_t\ atm\text{-of } L$ 
    using  $L\text{-max}\ linorder\text{-}lit.is\text{-}maximal\text{-}in\text{-}mset\text{-}iff$ 
    by  $(metis\ Neg\text{-}atm\text{-of}\text{-}iff\ linorder\text{-}trm.le\text{-}less\text{-}linear\ linorder\text{-}trm.linear$ 
 $literal.collapse(1)\ ord\text{-}res.less\text{-}lit\text{-}simps(1)\ ord\text{-}res.less\text{-}lit\text{-}simps(2)$ 
 $ord\text{-}res.less\text{-}lit\text{-}simps(3)\ ord\text{-}res.less\text{-}lit\text{-}simps(4))$ 
  ultimately have  $\forall K \in\ \#\ C. atm\text{-of } K \prec_t\ A$ 
  by  $(metis\ linorder\text{-}trm.antisym\text{-}conv1\ linorder\text{-}trm.dual\text{-}order.strict\text{-}trans)$ 
  moreover have  $\forall K \in\ \#\ C. atm\text{-of } K \in\ | \text{atms-of-cl}\text{ss } (N\ \cup\ U_{er})$ 
    using  $C\text{-in}$  by  $(metis\ atm\text{-of}\text{-}in\text{-}atms\text{-of}\text{-}cl}\text{ss}I)$ 
  ultimately have  $\forall K \in\ \#\ C. atm\text{-of } K \in\ | \text{trail-atms } \Gamma$ 
    using  $atm\text{-defined}\text{-}iff\text{-}lt\text{-}A$  by  $metis$ 
  hence  $\forall K \in\ \#\ C. \text{trail-defined-lit } \Gamma\ K$ 
    using  $\text{trail-defined-lit}\text{-}iff\text{-}trail\text{-defined}\text{-}atm$  by  $metis$ 
  hence  $\text{trail-defined-cl}\ \Gamma\ C$ 
    unfolding  $\text{trail-defined-cl}\text{-}def$  by  $argo$ 
  thus  $False$ 
    using  $\langle \neg \text{trail-defined-cl}\ \Gamma\ C \rangle$  by  $contradiction$ 
qed

ultimately have  $A \prec_t\ atm\text{-of } L$ 
  by  $order$ 

hence  $atm\text{-of } L \notin | \text{trail-atms } \Gamma'$ 
  unfolding  $\langle \Gamma' = (Neg\ A, None)\ \#\ \Gamma \rangle$ 
  unfolding  $\text{trail-atms.simps}\ prod.sel\ literal.sel$ 
  using  $atm\text{-defined}\text{-}iff\text{-}lt\text{-}A[OF\ \langle atm\text{-of } L \in\ | \text{atms-of-cl}\text{ss } (N\ \cup\ U_{er}) \rangle]$ 
  using  $\langle \neg atm\text{-of } L \prec_t\ A \rangle$  by  $simp$ 

hence  $\neg \text{trail-defined-lit } \Gamma'\ L$ 
  using  $\text{trail-defined-lit}\text{-}iff\text{-}trail\text{-defined}\text{-}atm$  by  $blast$ 

```

hence \neg *trail-false-lit* $\Gamma' L$
using *trail-defined-lit-iff-true-or-false* **by** *blast*

thus \neg *trail-false-cls* $\Gamma' C$
unfolding *trail-false-cls-def*
using $\langle L \in\# C \rangle$ **by** *metis*

qed
qed
qed

hence $\forall C \in |iefac \mathcal{F} |^{\uparrow} (N \cup U_{er}).$ *trail-false-cls* $\Gamma' C \longrightarrow$
 $C = \{\#\} \vee (\exists A. ord-res.is-maximal-lit (Neg A) C)$
by *metis*

thus *?thesis*
unfolding *false-cls-is-empty-or-has-neg-max-lit-def* $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$ **by**
simp

next
case *step-hyps*: (*propagate* $\mathcal{F} U_{er} \Gamma A C \Gamma'$)

have $E = \{\#\} \vee (\exists A. ord-res.is-maximal-lit (Neg A) E)$
if *E-in*: $E \in |iefac \mathcal{F} |^{\uparrow} (N \cup U_{er})$ **and** *E-false*: *trail-false-cls* $\Gamma' E$ **for** E
proof (*cases* $A \in atm-of \text{ ' set-mset } E$)

case *True*
have \neg *trail-false-cls* ΓE
using $\langle \neg fBex (iefac \mathcal{F} |^{\uparrow} (N \cup U_{er})) (trail-false-cls \Gamma) \rangle$ *E-in* **by** *metis*

hence $Neg A \in\# E$
using *E-false*[*unfolded* $\langle \Gamma' = (Pos A, Some C) \# \Gamma \rangle$]
by (*metis subtrail-falseI uminus-Pos*)

have *atm-of* $L \in | trail-atms \Gamma'$ **if** $L \in\# E$ **for** L
using *E-false* $\langle L \in\# E \rangle$
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** $x \prec_t A$ **if** $x \in | trail-atms \Gamma$ **for** x
using *step-hyps*(4) *that*
by (*simp add: linorder-trm.is-least-in-ffilter-iff*)

ultimately **have** *atm-of* $L \preceq_t A$ **if** $L \in\# E$ **for** L
unfolding $\langle \Gamma' = (Pos A, Some C) \# \Gamma \rangle$ *trail-atms.simps prod.sel literal.sel*
using $\langle L \in\# E \rangle$ **by** *blast*

hence $L \preceq_l Neg A$ **if** $L \in\# E$ **for** L
using $\langle L \in\# E \rangle$
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 $\exists A. \text{ord-res.is-maximal-lit } (\text{Neg } A) E$
using $\langle \text{Neg } A \in \# E \rangle$
by $(\text{metis } \langle \neg \text{trail-false-cls } \Gamma E \rangle \text{linorder-lit.ex-maximal-in-mset}$
 $\text{linorder-lit.is-maximal-in-mset-iff reflclp-iff trail-false-cls-mempty})$

thus *?thesis ..*
next
case *False*
hence $\text{trail-false-cls } \Gamma E$
using $E\text{-false}[\text{unfolded } \langle \Gamma' = (\text{Pos } A, \text{Some } C) \# \Gamma \rangle]$
by $(\text{metis } \text{atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel}(1) \text{ sub-trail-falseI})$

moreover have $\neg \text{trail-false-cls } \Gamma E$
using $\langle \neg fBex (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) (\text{trail-false-cls } \Gamma) \rangle E\text{-in}$ **by** *metis*

ultimately have *False*
by *contradiction*

thus *?thesis ..*
qed

thus *?thesis*
unfolding $\text{false-cls-is-mempty-or-has-neg-max-lit-def } \langle s' = (U_{er}, \mathcal{F}, \Gamma) \rangle$ **by**
simp
next
case $\text{step-hyps: } (\text{factorize } \mathcal{F} U_{er} \Gamma A C \mathcal{F}')$

have $\text{trail-false-cls } \Gamma (\text{iefac } \mathcal{F} C) \longleftrightarrow \text{trail-false-cls } \Gamma C$ **if** $C \in \mid N \mid \cup \mid U_{er}$ **for**
 $\mathcal{F} C$
using *that* **by** $(\text{simp add: iefac-def trail-false-cls-def})$

hence $\forall C \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{trail-false-cls } \Gamma C \longrightarrow C = \{\#\} \vee (\exists A. \text{ord-res.is-maximal-lit } (\text{Neg } A) C)$
using $\text{step-hyps}(3)$ **by** *force*

thus *?thesis*
by $(\text{simp add: } \langle s' = (U_{er}, \mathcal{F}', \Gamma) \rangle \text{false-cls-is-mempty-or-has-neg-max-lit-def})$
next
case $\text{step-hyps: } (\text{resolution } \Gamma \mathcal{F} U_{er} D A C U_{er}' \Gamma')$

have $\text{false-wrt-}\Gamma\text{-if-false-wrt-}\Gamma'$: $\text{trail-false-cls } \Gamma E$ **if** $\text{trail-false-cls } \Gamma' E$ **for** E
using *that*
unfolding $\text{step-hyps}(7) \text{ trail-false-cls-def trail-false-lit-def}$
using $\text{image-iff set-dropWhileD}$ **by** *fastforce*

have $\text{iefac } \mathcal{F} E = \{\#\} \vee (\exists A. \text{ord-res.is-maximal-lit } (\text{Neg } A) (\text{iefac } \mathcal{F} E))$
if $E \in \mid N \mid \cup \mid U_{er}' \text{ trail-false-cls } \Gamma' (\text{iefac } \mathcal{F} E)$ **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 = (Uer, 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 = (Uer, 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 $\forall Ln \in \text{set } \Gamma'. \text{ atm-of } (fst Ln) \prec_t \text{ atm-of } K$
 by auto

hence $\text{ atm-of } K \not\subseteq \text{ trail-atms } \Gamma'$
 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 $\neg \text{ trail-false-cls } \Gamma' (\text{eres } C D)$
 using trail-defined-lit-iff-true-or-false trail-false-cls-def by metis

hence $\neg \text{ trail-false-cls } \Gamma' E$
 unfolding $\langle E = \text{eres } C D \rangle$.

hence $\neg \text{ trail-false-cls } \Gamma' (\text{iefac } \mathcal{F} E)$
 unfolding trail-false-cls-def by (metis iefac-def set-mset-efac)

thus ?thesis
 using $\langle \text{trail-false-cls } \Gamma' (\text{iefac } \mathcal{F} E) \rangle$
 by contradiction
 qed

next

case False

hence $E \in N \cup U_{er}$
 using step-hyps(6) that(1) by force

moreover hence $\text{iefac } \mathcal{F} E \neq \{\#\}$
 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 $\text{trail-false-cls } \Gamma (\text{iefac } \mathcal{F} E)$
 using $\langle \text{trail-false-cls } \Gamma' (\text{iefac } \mathcal{F} E) \rangle$ false-wrt- Γ -if-false-wrt- Γ' by metis

ultimately have $\exists A. \text{ord-res.is-maximal-lit } (Neg A) (\text{iefac } \mathcal{F} E)$
 using invars(1)[unfolded $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ false-cls-is-mempty-or-has-neg-max-lit-def]
 by auto

thus ?thesis ..
 qed

thus ?thesis

by (simp add: $\langle s' = (U_{er}', \mathcal{F}, \Gamma') \rangle$ false-cls-is-mempty-or-has-neg-max-lit-def)
qed

definition *decided-literals-all-neg* **where**

decided-literals-all-neg $N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow$
 $(\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L))$

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* $\mathcal{F} U_{er} \Gamma A \Gamma'$)

have $\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L$

using *invar* **by** (simp add: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *decided-literals-all-neg-def*)

hence $\forall Ln \in set \Gamma'. \forall L. Ln = (L, None) \longrightarrow is-neg L$

unfolding $\langle \Gamma' = (Neg A, None) \# \Gamma \rangle$ **by** *simp*

thus *?thesis*

by (simp add: $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$ *decided-literals-all-neg-def*)

next

case (*propagate* $\mathcal{F} U_{er} \Gamma A C \Gamma'$)

have $\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L$

using *invar* **by** (simp add: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *decided-literals-all-neg-def*)

hence $\forall Ln \in set \Gamma'. \forall L. Ln = (L, None) \longrightarrow is-neg L$

unfolding $\langle \Gamma' = (Pos A, Some C) \# \Gamma \rangle$ **by** *simp*

thus *?thesis*

by (simp add: $\langle s' = (U_{er}, \mathcal{F}, \Gamma') \rangle$ *decided-literals-all-neg-def*)

next

case (*factorize* $\mathcal{F} U_{er} \Gamma A C \mathcal{F}'$)

have $\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L$

using *invar* **by** (simp add: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *decided-literals-all-neg-def*)

thus *?thesis*

by (simp add: $\langle s' = (U_{er}, \mathcal{F}', \Gamma) \rangle$ *decided-literals-all-neg-def*)

next

case *step-hyps*: (*resolution* $\Gamma \mathcal{F} U_{er} D A C U_{er}' \Gamma'$)

have $\forall Ln \in set \Gamma. \forall L. Ln = (L, None) \longrightarrow is-neg L$

using *invar* **by** (*simp add*: $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ *decided-literals-all-neg-def*)

moreover have *set* $\Gamma' \subseteq$ *set* Γ

unfolding $\langle \Gamma' = \text{dropWhile} - \Gamma \rangle$

by (*meson set-mono-suffix suffix-dropWhile*)

ultimately have $\forall Ln \in \text{set } \Gamma'. \forall L. Ln = (L, \text{None}) \longrightarrow \text{is-neg } L$

by *blast*

thus *?thesis*

by (*simp add*: $\langle s' = (U_{er'}, \mathcal{F}, \Gamma') \rangle$ *decided-literals-all-neg-def*)

qed

definition *ord-res-8-invars* **where**

ord-res-8-invars $N s \longleftrightarrow$

trail-is-sorted $N s \wedge$

trail-is-lower-set $N s \wedge$

false-cls-is-mempty-or-has-neg-max-lit $N s \wedge$

trail-annotations-invars $N s \wedge$

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: $\neg \text{ord-res-8-final } (N, s)$ **and**

invars: *ord-res-8-invars* $N\ s$

shows $\exists s'. \text{ord-res-8 } N\ s\ s'$

proof –

obtain $U_{er}\ \mathcal{F}\ \Gamma$ **where** $s = (U_{er}, \mathcal{F}, \Gamma)$

by (*metis surj-pair*)

note $\text{invars}' = \text{invars}[\text{unfolded } \langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle \text{ord-res-8-invars-def}]$

have

undef-atm-or-false-cls:

$(\exists x \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). x \mid \notin \mid \text{trail-atms } \Gamma) \wedge$
 $\neg (\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma\ x) \vee$

$(\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma\ x)$ **and**

$\{\#\} \mid \notin \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$

unfolding *atomize-conj*

using *not-final[unfolded ord-res-8-final.simps]* $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ **by** *metis*

show *?thesis*

using *undef-atm-or-false-cls*

proof (*elim disjE conjE*)

assume

ex-undef-atm: $\exists x \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). x \mid \notin \mid \text{trail-atms } \Gamma$ **and**

no-conflict: $\neg (\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma\ x)$

moreover have $\{ \mid A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1$
 $\prec_t A_2 \mid \} \neq \{ \mid \}$

proof –

obtain $A_2 :: 'f\ gterm$ **where**

$A_2\text{-in}$: $A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and**

$A_2\text{-undef}$: $A_2 \mid \notin \mid \text{trail-atms } \Gamma$

using *ex-undef-atm* **by** *metis*

have $A_1 \prec_t A_2$ **if** $A_1\text{-in}$: $A_1 \mid \in \mid \text{trail-atms } \Gamma$ **for** $A_1 :: 'f\ gterm$

proof –

have $A_1 \neq A_2$

using $A_1\text{-in } A_2\text{-undef}$ **by** *metis*

moreover have *linorder-trm.is-lower-fset* (*trail-atms* Γ) (*atms-of-clss* (N
 $\mid \cup \mid U_{er}$))

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* $\{|A_2| \in | \text{atms-of-cls } (N \cup U_{er})$.

$\forall A_1 | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2| \}$ A

using *ex-undef-atm* *linorder-trm.ex-least-in-fset* by *presburger*

show $\exists s'. \text{ord-res-8 } N \ s \ s'$

proof (*cases* $\exists C | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er}). \text{clause-could-propagate } \Gamma \ C \ (\text{Pos } A)$)

case *True*

hence $\{|C| \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er}). \text{clause-could-propagate } \Gamma \ C \ (\text{Pos } A)|\} \neq \{\emptyset\}$

by *force*

then obtain C where

C -least: *linorder-cls.is-least-in-fset*

$\{|C| \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er}). \text{clause-could-propagate } \Gamma \ C \ (\text{Pos } A)|\} \ C$

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 $\exists x | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er}). \text{trail-false-cls } \Gamma \ x$

then obtain $D :: 'f$ gclause where

D -least: *linorder-cls.is-least-in-fset*

$(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er}))) \ D$

by (*metis* *femptyE* *ffmember-filter* *linorder-cls.ex-least-in-fset*)

hence $D | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup U_{er})$ and *trail-false-cls* $\Gamma \ D$

unfolding *atomize-conj* *linorder-cls.is-least-in-filter-iff* by *argo*

moreover have $D \neq \{\#\}$

using $\langle D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle \langle \{ \# \} \mid \notin \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}) \rangle$ **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 $\langle \text{trail-false-cls } \Gamma \ D \rangle$
by *(metis linorder-lit.is-maximal-in-mset-iff trail-false-cls-def)*

hence $Pos\ A \in \text{fst } \text{' set } \Gamma$
unfolding *trail-false-lit-def* **by** *simp*

hence $\exists C. (Pos\ A, Some\ C) \in \text{set } \Gamma$
using *invars'[unfolded decided-literals-all-neg-def, simplified]*
by *fastforce*

then obtain $C :: \text{' gclause where}$
 $\text{map-of } \Gamma (Pos\ A) = Some\ (Some\ C)$
by *(metis invars' is-pos-def map-of-SomeD not-Some-eq decided-literals-all-neg-def weak-map-of-SomeI)*

thus $\exists s'. \text{ord-res-8 } N\ s\ s'$
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$
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**
 $\mathcal{R} = \text{ord-res-8}$ **and** $\mathcal{F} = \text{ord-res-8-final}$ **and** $\mathcal{I} = \text{ord-res-8-invars}$
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:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ clause-could-propagate } \Gamma C (Pos A)) \implies$
 $\Gamma' = (Neg A, None) \# \Gamma \implies$
 $\text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid$

propagate:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{ clause-could-propagate } \Gamma C (Pos A)\} C \implies$
 $\Gamma' = (Pos A, Some (\text{efac } C)) \# \Gamma \implies$
 $\mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies$
 $\text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid$

resolution:

$\text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma D\}$
 $D \implies$
 $\text{linorder-lit.is-maximal-in-mset } D (Neg A) \implies$
 $\text{map-of } \Gamma (Pos A) = Some (Some C) \implies$
 $U_{er}' = \text{finsert } (\text{eres } C D) U_{er} \implies$
 $\Gamma' = \text{dropWhile } (\lambda Ln. \forall K.$
 $\text{linorder-lit.is-maximal-in-mset } (\text{eres } C D) K \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst}$
 $Ln)) \Gamma \implies$
 $\text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

lemma right-unique-ord-res-9:

fixes $N :: 'f \text{ gclause fset}$

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 F U_{er} Γ A₁ Γ₁)*

show *?thesis*

using *step2 unfolding* $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases N (U_{er}, F, Γ) 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*
 $\text{linorder-trm.is-least-in-fset-iff}$)

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 Uer' Γ')
with hyps1 have False
  by (metis (no-types, lifting) linorder-cls.is-least-in-ffilter-iff)
thus ?thesis ..
qed
next
case hyps1: (propagate  $\mathcal{F}$  Uer Γ A1 C1 Γ1'  $\mathcal{F}$ 1')
show ?thesis
  using step2 unfolding ⟨x = (Uer,  $\mathcal{F}$ , Γ)⟩
proof (cases N (Uer,  $\mathcal{F}$ , 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 Uer' Γ')
with hyps1 have False
  by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
thus ?thesis ..
qed
next
case hyps1: (resolution Γ  $\mathcal{F}$  Uer D1 A1 C1 Uer1' Γ1')
show ?thesis
  using step2 unfolding ⟨x = (Uer,  $\mathcal{F}$ , Γ)⟩
proof (cases N (Uer,  $\mathcal{F}$ , 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 Uer' Γ')
have D1 = 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  $\langle D_1 = D \rangle$ 
    by (metis (no-types) Uniq-D linorder-lit.Uniq-is-maximal-in-mset option.inject
    uminus-Neg)
  show ?thesis
    using hyps1 hyps2
    unfolding  $\langle D_1 = D \rangle \langle C_1 = C \rangle$ 
    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  $N s s' \vee (\text{ord-res-8 } N \text{ OO } \text{ord-res-8 } N) s s'$ 
  using step
proof (cases  $N s s'$  rule: ord-res-9.cases)
  case step-hyps: (decide-neg  $\mathcal{F} U_{er} \Gamma A \Gamma'$ )
  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  $\mathcal{F} U_{er} \Gamma A C \Gamma' \mathcal{F}'$ )

```

have

```

  C-in:  $C \in | \text{iefac } \mathcal{F} \mid \mid (N \mid \cup \mid U_{er})$  and
  C-could-prop: clause-could-propagate  $\Gamma C (Pos A)$  and
  C-lt:  $\forall D \in | \text{iefac } \mathcal{F} \mid \mid (N \mid \cup \mid U_{er}).$ 
     $D \neq C \longrightarrow \text{clause-could-propagate } \Gamma D (Pos A) \longrightarrow C \prec_c D$ 

```

using step-hyps unfolding *atomize-conj linorder-cls.is-least-in-filter-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  $\mathcal{F}' = \text{finsert } C \ \mathcal{F}$ 
  using False step-hyps by simp

have  $\text{efac } C \neq C$ 
  using False C-max-lit by (metis greatest-literal-in-efacI literal.disc(1))

hence  $C \text{-in}' : C \in N \cup U_{er}$ 
  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  $\langle \text{efac } C \neq C \rangle$  factorizable-if-neq-efac
      fimage-iff
      ex1-efac-eq-factoring-chain iefac-def)

have fimage-iefac- $\mathcal{F}'$ -eq:
  iefac  $\mathcal{F}' \upharpoonright (N \cup U_{er}) = \text{finsert } (\text{efac } C) ((\text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})) - \{C\})$ 
proof (intro fsubset-antisym fsubsetI)
  fix  $x :: 'f \text{ gclause}$ 
  assume  $x \in \text{iefac } \mathcal{F}' \upharpoonright (N \cup U_{er})$ 
  then obtain  $x'$  where  $x' \in N \cup U_{er}$  and  $x = \text{iefac } \mathcal{F}' \ x'$ 
    by blast
  thus  $x \in \text{finsert } (\text{efac } C) ((\text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})) - \{C\})$ 
  proof (cases  $x' = C$ )
    case True
      hence  $x = \text{efac } C$ 
        using  $\langle x = \text{iefac } \mathcal{F}' \ x' \rangle$  by (simp add:  $\langle \mathcal{F}' = \text{finsert } C \ \mathcal{F} \rangle$  iefac-def)
      thus ?thesis
        using  $\langle \text{efac } C \neq C \rangle$  by simp
    case False
      hence  $x = \text{iefac } \mathcal{F}' \ x'$ 
        using  $\langle x = \text{iefac } \mathcal{F}' \ x' \rangle$  by (auto simp add:  $\langle \mathcal{F}' = \text{finsert } C \ \mathcal{F} \rangle$  iefac-def)
      then show ?thesis
        by (metis (no-types, lifting) False  $\langle x' \in N \cup U_{er} \rangle$  ex1-efac-eq-factoring-chain
            factorizable-if-neq-efac fimage-eqI finsertCI fminus-iff fsingletonE
            iefac-def)
      qed
  next
  case False
    hence  $x = \text{iefac } \mathcal{F}' \ x'$ 
      using  $\langle x = \text{iefac } \mathcal{F}' \ x' \rangle$  by (auto simp add:  $\langle \mathcal{F}' = \text{finsert } C \ \mathcal{F} \rangle$  iefac-def)
    then show ?thesis
      by (metis (no-types, lifting) False  $\langle x' \in N \cup U_{er} \rangle$  ex1-efac-eq-factoring-chain
          factorizable-if-neq-efac fimage-eqI finsertCI fminus-iff fsingletonE
          iefac-def)
    qed
  next
  case True
    fix  $x :: 'f \text{ gclause}$ 
    assume  $x \text{-in} : x \in \text{finsert } (\text{efac } C) (\text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})) - \{C\}$ 
    hence  $x = \text{efac } C \vee x \in \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er}) - \{C\}$ 
      by blast
    thus  $x \in \text{iefac } \mathcal{F}' \upharpoonright (N \cup U_{er})$ 

```

proof (*elim disjE*)
assume $x = \text{efac } C$
hence $x = \text{iefac } \mathcal{F}' C$
by (*simp add: $\langle \mathcal{F}' = \text{finsert } C \mathcal{F} \rangle \text{iefac-def}$*)
thus $x \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
using $\langle C \in \mid N \mid \cup \mid U_{er} \rangle$ **by** *blast*
next
assume $x \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \mid - \mid \{C\}$
hence $x \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $x \neq C$
by *simp-all*
then obtain x' **where** $x' \in \mid N \mid \cup \mid U_{er}$ **and** $x = \text{iefac } \mathcal{F} x'$
by *auto*
moreover have $x' \neq C$
using $\langle x \neq C \rangle \langle x = \text{iefac } \mathcal{F} x' \rangle$
using $\langle C \notin \mid \mathcal{F} \rangle \text{iefac-def}$ **by** *force*
ultimately show $x \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
by (*simp add: $\langle \mathcal{F}' = \text{finsert } C \mathcal{F} \rangle \text{iefac-def}$*)
qed
qed

have *first-step8*: *ord-res-8* $N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma)$
using *False step-hyps ord-res-8.factorize* **by** *simp*

moreover have *ord-res-8* $N (U_{er}, \mathcal{F}', \Gamma) (U_{er}, \mathcal{F}', \Gamma')$
proof (*rule ord-res-8.propagate*)
have $\neg \text{trail-false-cls } \Gamma C$
using *step-hyps using C-in* **by** *metis*

hence $\neg \text{trail-false-cls } \Gamma (\text{efac } C)$
by (*simp add: trail-false-cls-def*)

thus $\neg (\exists C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C)$
using *step-hyps unfolding fimage-iefac- \mathcal{F}' -eq* **by** *blast*

next
show *linorder-trm.is-least-in-fset* $\{A_2 \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \forall A_1 \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2 \mid\} A$
using *step-hyps* **by** *argo*

next
show *linorder-cls.is-least-in-fset* $\{C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C (\text{Pos } A) \mid\} (\text{efac } C)$
unfolding *linorder-cls.is-least-in-ffilter-iff*
proof (*intro conjI ballI impI*)
show $\text{efac } C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$
unfolding *fimage-iefac- \mathcal{F}' -eq* **by** *simp*

next
show *clause-could-propagate* $\Gamma (\text{efac } C) (\text{Pos } A)$
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  $\mathcal{F}'$  |' (N |∪| Uer) and
    D ≠ efac C and
    clause-could-propagate  $\Gamma$  D (Pos A)
  thus efac C  $\prec_c$  D
  using C-lt
  by (metis (no-types, opaque-lifting) C-in-efac-properties-if-not-ident(1)
      fimage-iefac- $\mathcal{F}'$ -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  $\Gamma' = (\text{Pos } A, \text{Some } (\text{efac } C)) \# \Gamma$ 
  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  $\Gamma$   $\mathcal{F}$  Uer D A C Uer'  $\Gamma'$ )
  show ?thesis
  proof (rule disjI1)
    show ord-res-8 N s s'
      using step-hyps ord-res-8.resolution by (simp only:)
    qed
  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: \neg *ord-res-8-final* (N, s) **and**
invars: *ord-res-8-invars* $N s$
shows $\exists s'. \text{ord-res-9 } N s s'$

proof –

obtain $U_{er} \mathcal{F} \Gamma$ **where** $s = (U_{er}, \mathcal{F}, \Gamma)$
by (*metis surj-pair*)

note $\text{invars}' = \text{invars}[\text{unfolded } \langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle \text{ord-res-8-invars-def}]$

have

undef-atm-or-false-cls:
 $(\exists x \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). x \mid \notin \mid \text{trail-atms } \Gamma) \wedge$
 $\neg (\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x) \vee$
 $(\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x)$ **and**
 $\{\#\} \mid \notin \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
unfolding *atomize-conj*
using *not-final[unfolded ord-res-8-final.simps]* $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ **by** *metis*

show *?thesis*

using *undef-atm-or-false-cls*

proof (*elim disjE conjE*)

assume

ex-undef-atm: $\exists x \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). x \mid \notin \mid \text{trail-atms } \Gamma$ **and**
no-conflict: $\neg (\exists x \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x)$

moreover have $\{ \mid A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1$
 $\prec_t A_2 \mid \} \neq \{ \mid \}$

proof –

obtain $A_2 :: 'f \text{ gterm}$ **where**
 $A_2\text{-in}$: $A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and**
 $A_2\text{-undef}$: $A_2 \mid \notin \mid \text{trail-atms } \Gamma$
using *ex-undef-atm* **by** *metis*

have $A_1 \prec_t A_2$ **if** $A_1\text{-in}$: $A_1 \mid \in \mid \text{trail-atms } \Gamma$ **for** $A_1 :: 'f \text{ gterm}$

proof –

have $A_1 \neq A_2$
using $A_1\text{-in } A_2\text{-undef}$ **by** *metis*

moreover have *linorder-trm.is-lower-fset* $(\text{trail-atms } \Gamma)$ $(\text{atms-of-clss } (N$
 $\mid \cup \mid U_{er}))$

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* $\{|A_2| \in| \text{atms-of-cls } (N \cup| U_{er})$.

$\forall A_1 | \in| \text{trail-atms } \Gamma. A_1 \prec_t A_2| \}$ A

using *ex-undef-atm* *linorder-trm.ex-least-in-fset* by *presburger*

show $\exists s'. \text{ord-res-9 } N s s'$

proof (*cases* $\exists C | \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er})$. *clause-could-propagate* ΓC (*Pos* A))

case *True*

hence $\{|C| \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er})$. *clause-could-propagate* ΓC (*Pos* A)|}

$\neq \{|\}$

by *force*

then obtain C where

C -least: *linorder-cls.is-least-in-fset*

$\{|C| \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er})$. *clause-could-propagate* ΓC (*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 $\exists x | \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er})$. *trail-false-cls* Γx

then obtain $D :: 'f$ gclause where

D -least: *linorder-cls.is-least-in-fset*

(*ffilter* (*trail-false-cls* Γ) (*iefac* $\mathcal{F} | \uparrow (N \cup| U_{er})$)) D

by (*metis* *femptyE* *ffmember-filter* *linorder-cls.ex-least-in-fset*)

hence $D | \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er})$ and *trail-false-cls* ΓD

unfolding *atomize-conj* *linorder-cls.is-least-in-ffilter-iff* by *argo*

moreover have $D \neq \{\#\}$

using $\langle D | \in| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er}) \rangle \langle \{\#\} | \notin| \text{iefac } \mathcal{F} | \uparrow (N \cup| U_{er}) \rangle$ by

metis

ultimately obtain A **where** $Neg-A-max: linorder-lit.is-maximal-in-mset D$
($Neg A$)

using $invars'$ $false-clc-is-mempty-or-has-neg-max-lit-def$ **by** *metis*

hence $trail-false-lit \Gamma (Neg A)$

using $\langle trail-false-clc \Gamma D \rangle$

by (*metis linorder-lit.is-maximal-in-mset-iff trail-false-clc-def*)

hence $Pos A \in fst \text{ ' set } \Gamma$

unfolding $trail-false-lit-def$ **by** *simp*

hence $\exists C. (Pos A, Some C) \in set \Gamma$

using $invars'$ [*unfolded decided-literals-all-neg-def, simplified*]

by *fastforce*

then obtain $C :: 'f gclause$ **where**

$map-of \Gamma (Pos A) = Some (Some C)$

by (*metis invars' is-pos-def map-of-SomeD not-Some-eq decided-literals-all-neg-def weak-map-of-SomeI*)

thus $\exists s'. ord-res-9 N s s'$

unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$

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**

$\mathcal{R} = ord-res-9$ **and** $\mathcal{F} = ord-res-8-final$ **and** $\mathcal{I} = ord-res-8-invars$]

using $ord-res-9-preserves-invars ex-ord-res-9-if-not-final invars$ **by** *metis*

sublocale $ord-res-9-semantic: semantics$ **where**

$step = constant-context ord-res-9$ **and**

$final = ord-res-8-final$

proof *unfold-locales*

fix $S :: 'f gclause fset \times 'f gclause fset \times 'f gclause fset \times$

$('f gliteral \times 'f gclause option) list$

obtain

$N U_{er} \mathcal{F} :: 'f gterm clause fset$ **and**

$\Gamma :: ('f gterm literal \times 'f gterm literal multiset option) list$ **where**

$S-def: S = (N, U_{er}, \mathcal{F}, \Gamma)$

by (*metis prod.exhaust*)

assume *ord-res-8-final S*
hence $\nexists x. \text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) x$
unfolding *S-def*
proof (*cases* $(N, U_{er}, \mathcal{F}, \Gamma)$ *rule: ord-res-8-final.cases*)
case *model-found*

have $\nexists A. \text{linorder-trm.is-least-in-fset } \{|A_2| \in | \text{atms-of-cls } (N \cup U_{er}).$
 $\forall A_1 | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A$
using $\langle \neg (\exists A | \in | \text{atms-of-cls } (N \cup U_{er}). A | \notin | \text{trail-atms } \Gamma) \rangle$
using *linorder-trm.is-least-in-ffilter-iff* **by** *fastforce*

moreover have $\nexists C. \text{linorder-cls.is-least-in-fset}$
 $(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}))) C$
using $\langle \neg \text{fBex } (\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})) (\text{trail-false-cls } \Gamma) \rangle$
by (*metis linorder-cls.is-least-in-ffilter-iff*)

ultimately show *?thesis*
unfolding *ord-res-9.simps* **by** *blast*
next
case *contradiction-found*

hence $\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}). \text{trail-false-cls } \Gamma C$
using *trail-false-cls-empty* **by** *metis*

moreover have $\nexists D A. \text{linorder-cls.is-least-in-fset } (\text{ffilter } (\text{trail-false-cls } \Gamma)$
 $(\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}))) D \wedge \text{ord-res.is-maximal-lit } (\text{Neg } A) D$
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-empty)

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 × 'f gclause fset × 'f gclause fset × ('f gliteral × 'f gclause option)
list

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

inductive *ord-res-10 where*

decide-neg:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{ \mid A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} A \implies \\ & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C (Pos A)) \implies \\ & \Gamma' = (Neg A, None) \# \Gamma \implies \\ & \text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid \end{aligned}$$

decide-pos:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{ \mid A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{ \mid C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A) \} C \implies \\ & \Gamma' = (Pos A, None) \# \Gamma \implies \\ & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C) \implies \\ & \mathcal{F}' = (\text{if } \text{linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else } \text{finsert } C \mathcal{F}) \implies \\ & \text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid \end{aligned}$$

propagate:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{ \mid A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{ \mid C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A) \} C \implies \\ & \Gamma' = (Pos A, Some (efac C)) \# \Gamma \implies \\ & (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C) \implies \\ & \mathcal{F}' = (\text{if } \text{linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else } \text{finsert } C \mathcal{F}) \implies \\ & \text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid \end{aligned}$$

resolution:

$$\begin{aligned} & \text{linorder-cls.is-least-in-fset } \{ \mid D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma D \} \\ & D \implies \\ & \text{linorder-lit.is-maximal-in-mset } D (Neg A) \implies \\ & \text{map-of } \Gamma (Pos A) = Some (Some C) \implies \\ & U_{er}' = \text{finsert } (eres C D) U_{er} \implies \\ & \Gamma' = \text{dropWhile } (\lambda Ln. \forall K. \\ & \quad \text{linorder-lit.is-maximal-in-mset } (eres C D) K \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst} \\ & \quad Ln)) \Gamma \implies \end{aligned}$$

ord-res-10 $N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

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* $\mathcal{F} U_{er} \Gamma A1 \Gamma1'$)

show *?thesis*

using *step2* **unfolding** $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma) z$ *rule*: *ord-res-10.cases*)

case (*decide-neg* $A \Gamma'$)

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* $A C \Gamma' \mathcal{F}'$)

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* $A C \Gamma' \mathcal{F}'$)

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* $D A C U_{er}' \Gamma'$)

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* $\mathcal{F} U_{er} \Gamma A1 C1 \Gamma1' \mathcal{F}1'$)

show *?thesis*

using *step2* **unfolding** $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma) z$ *rule*: *ord-res-10.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 (*decide-pos* $A C \Gamma' \mathcal{F}'$)

```

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 hyps2: (propagate A C  $\Gamma'$   $\mathcal{F}'$ )
have A1 = A
  using hyps1 hyps2
  by (metis (no-types, lifting) linorder-trm.dual-order.asym
      linorder-trm.is-least-in-filter-iff)
hence trail-false-cls  $\Gamma 1' = \text{trail-false-cls } \Gamma'$ 
  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  $U_{er}' \Gamma'$ )
with hyps1 have False
  by (meson linorder-cls.is-least-in-filter-iff)
thus ?thesis ..
qed
next
case hyps1: (propagate  $\mathcal{F}$   $U_{er}$   $\Gamma$  A1 C1  $\Gamma 1' \mathcal{F} 1'$ )
show ?thesis
  using step2 unfolding  $\langle x = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
proof (cases  $N (U_{er}, \mathcal{F}, \Gamma) z$  rule: ord-res-10.cases)
case (decide-neg A  $\Gamma'$ )
with hyps1 have False
  by (metis (no-types, lifting) linorder-cls.is-least-in-filter-iff
      linorder-trm.dual-order.asym linorder-trm.is-least-in-filter-iff)
thus ?thesis ..
next
case hyps2: (decide-pos A C  $\Gamma'$   $\mathcal{F}'$ )
have A1 = A
  using hyps1 hyps2
  by (metis (no-types, lifting) linorder-trm.dual-order.asym
      linorder-trm.is-least-in-filter-iff)
hence trail-false-cls  $\Gamma 1' = \text{trail-false-cls } \Gamma'$ 
  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  $\Gamma'$   $\mathcal{F}'$ )
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 (resolution D A C Uer' Γ')
  with hyps1 have False
  by (meson linorder-cls.is-least-in-filter-iff)
  thus ?thesis ..
qed
next
  case hyps1: (resolution Γ  $\mathcal{F}$  Uer D1 A1 C1 Uer1' Γ1')
  show ?thesis
  using step2 unfolding ⟨x = (Uer,  $\mathcal{F}$ , Γ)⟩
  proof (cases N (Uer,  $\mathcal{F}$ , Γ) z rule: ord-res-10.cases)
  case (decide-neg A Γ')
  with hyps1 have False
  by (meson linorder-cls.is-least-in-filter-iff)
  thus ?thesis ..
  next
  case (decide-pos A C Γ'  $\mathcal{F}'$ )
  with hyps1 have False
  by (meson linorder-cls.is-least-in-filter-iff)
  thus ?thesis ..
  next
  case (propagate A C Γ'  $\mathcal{F}'$ )
  with hyps1 have False
  by (meson linorder-cls.is-least-in-filter-iff)
  thus ?thesis ..
  next
  case hyps2: (resolution D A C Uer' 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 $S :: 'f \text{ ord-res-10-state}$

obtain
 $N \ U_{er} \ \mathcal{F} :: 'f \text{ gclause fset and}$
 $\Gamma :: ('f \text{ gliteral} \times 'f \text{ gclause option}) \text{ list where}$
 $S\text{-def: } S = (N, U_{er}, \mathcal{F}, \Gamma)$
by *(metis prod.exhaust)*

assume *ord-res-8-final S*

hence $\nexists x. \text{ ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) x$
unfolding *S-def*
proof *(cases (N, U_{er}, F, Γ) rule: ord-res-8-final.cases)*
case *model-found*

have $\nexists A. \text{ linorder-trm.is-least-in-fset } \{ |A_2 | \in | \text{atms-of-clss } (N \ | \cup| \ U_{er}).$
 $\forall A_1 | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} A$
using $\langle \neg (\exists A | \in | \text{atms-of-clss } (N \ | \cup| \ U_{er}). A | \notin | \text{trail-atms } \Gamma) \rangle$
using *linorder-trm.is-least-in-ffilter-iff* **by** *fastforce*

moreover have $\nexists C. \text{ linorder-cls.is-least-in-fset}$
 $(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup| \ U_{er}))) C$
using $\langle \neg fBex (\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup| \ U_{er})) (\text{trail-false-cls } \Gamma) \rangle$
by *(metis linorder-cls.is-least-in-ffilter-iff)*

ultimately show *?thesis*
unfolding *ord-res-10.simps* **by** *blast*
next
case *contradiction-found*

hence $\exists C | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup| \ U_{er}). \text{ trail-false-cls } \Gamma \ C$
using *trail-false-cls-empty* **by** *metis*

moreover have $\nexists D A. \text{ linorder-cls.is-least-in-fset } (\text{ffilter } (\text{trail-false-cls } \Gamma)$
 $(\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup| \ U_{er}))) D \wedge \text{ord-res.is-maximal-lit } (\text{Neg } A) \ D$
by *(metis empty-iff linorder-cls.is-least-in-ffilter-iff linorder-cls.leD*
 $\text{linorder-lit.is-maximal-in-mset-iff local.contradiction-found(1) mempty-lesseq-clsset-mset-empty trail-false-clss-empty})$

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 $N (U_{er}, \mathcal{F}, \Gamma)$ **if**
sorted-wrt $(\lambda x y. atm-of (fst y) \prec_t atm-of (fst x)) \Gamma$ **and**
 $\forall Ln \in set \Gamma. \forall C. snd Ln = Some C \longrightarrow linorder-lit.is-greatest-in-mset C (fst Ln)$ **and**
linorder-trm.is-lower-fset $(trail-atms \Gamma) (atms-of-clss (N \mid\cup\mid U_{er}))$ **and**
 $\forall Ln \Gamma'. \Gamma = Ln \# \Gamma' \longrightarrow$
 $(snd Ln \neq None \longleftrightarrow (\exists C \mid\in\mid iefac \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). trail-false-cls \Gamma C))$
 \wedge
 $(snd Ln \neq None \longrightarrow is-pos (fst Ln)) \wedge$
 $(\forall C. snd Ln = Some C \longrightarrow C \mid\in\mid iefac \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})) \wedge$
 $(\forall C. snd Ln = Some C \longrightarrow clause-could-propagate \Gamma' C (fst Ln)) \wedge$
 $(\forall x \in set \Gamma'. snd x = None)$ **and**
 $\forall \Gamma_1 Ln \Gamma_0. \Gamma = \Gamma_1 @ Ln \# \Gamma_0 \longrightarrow$
 $snd Ln = None \longrightarrow \neg(\exists C \mid\in\mid iefac \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er}). trail-false-cls (Ln \# \Gamma_0) C)$

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*: $(1 \Gamma U_{er} \mathcal{F})$

note Γ -*sorted* = $\langle sorted-wrt (\lambda x y. atm-of (fst y) \prec_t atm-of (fst x)) \Gamma \rangle$

note Γ -*lower* = $\langle linorder-trm.is-lower-fset (trail-atms \Gamma) (atms-of-clss (N \mid\cup\mid U_{er})) \rangle$

have *trail-consistent* Γ

using Γ -*sorted* *trail-consistent-if-sorted-wrt-atoms* **by** *metis*

show *?thesis*

using *step* **unfolding** $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma) s'$ *rule*: *ord-res-10.cases*)

case *step-hyps*: $(decide-neg A \Gamma')$

have

A-in: $A \mid\in\mid atms-of-clss (N \mid\cup\mid U_{er})$ **and**

A-gt: $\forall A_1 \mid\in\mid trail-atms \Gamma. A_1 \prec_t A$ **and**

A-lt: $\forall y \mid\in\mid atms-of-clss (N \mid\cup\mid U_{er}).$

$y \neq A \longrightarrow (\forall A_1 \mid\in\mid trail-atms \Gamma. A_1 \prec_t y) \longrightarrow A \prec_t y$

using $\langle linorder-trm.is-least-in-fset - A \rangle$

unfolding *atomize-conj* *linorder-trm.is-least-in-filter-iff*

by *argo*

have *trail-atms* $\Gamma' = finsert A (trail-atms \Gamma)$

```

unfolding ⟨ $\Gamma' = (\text{Neg } A, -) \# \Gamma$ ⟩ by simp

show ?thesis
  unfolding ⟨ $s' = (-, -, -)$ ⟩
proof (intro ord-res-10-invars.intros allI impI conjI)
  show sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma'$ 
    unfolding ⟨ $\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma$ ⟩ sorted-wrt.simps
proof (intro conjI ballI)
  fix  $Ln :: 'f \text{ gliteral} \times 'f \text{ gclause option}$ 
  assume  $Ln \in \text{set } \Gamma$ 

  hence  $\text{atm-of } (fst Ln) \in \text{trail-atms } \Gamma$ 
  by (simp add: fset-trail-atms)

  thus  $\text{atm-of } (fst Ln) \prec_t \text{atm-of } (fst (\text{Neg } A, \text{None}))$ 
  unfolding prod.sel literal.sel
  using A-gt by metis
next
  show sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma$ 
  using  $\Gamma\text{-sorted}$  .
qed

show  $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{ord-res.is-strictly-maximal-lit}$ 
(fst Ln)  $C$ 
  unfolding ⟨ $\Gamma' = (-, \text{None}) \# \Gamma$ ⟩
  using invars by simp

show linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-clss ( $N \cup U_{er}$ ))
  unfolding ⟨ $\text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$ ⟩ finsert.rep-eq
proof (intro linorder-trm.is-lower-set-insertI ballI impI)
  show  $A \in \text{atms-of-clss } (N \cup U_{er})$ 
  using A-in .
next
  fix  $w :: 'f \text{ gterm}$ 
  assume  $w \in \text{atms-of-clss } (N \cup U_{er})$  and  $w \prec_t A$ 
  thus  $w \in \text{trail-atms } \Gamma$ 
  by (metis A-lt  $\Gamma$ -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  $\Gamma$ ) (atms-of-clss ( $N \cup U_{er}$ ))
  using  $\Gamma\text{-lower}$  .
qed

{
  fix
   $Ln :: 'f \text{ gliteral} \times 'f \text{ gclause option}$  and
   $\Gamma'' :: ('f \text{ gliteral} \times 'f \text{ gclause option}) \text{ list}$ 

  assume  $\Gamma' = Ln \# \Gamma''$ 

```



```

have snd Ln = None
  using  $\langle \Gamma' = Ln \# \Gamma'' \rangle$  step-hyps by auto

moreover have  $\neg (\exists C | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } ((\text{Neg } A,$ 
None) \# \Gamma) C)
proof (rule notI , elim bexE)
  fix C :: 'f gclause'
  assume
    C-in:  $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$  and
    C-false:  $\text{trail-false-cls } ((\text{Neg } A, \text{None}) \# \Gamma) C$ 

have clause-could-propagate  $\Gamma C$  (Pos A)
  unfolding clause-could-propagate-def
proof (intro conjI)
  show  $\neg \text{trail-defined-lit } \Gamma$  (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  $\neg \text{trail-false-cls } \Gamma C$ 
    using step-hyps C-in by metis

thus Pos A  $\in \# C$ 
  using C-false by (metis subtrail-falseI uminus-Neg)
next

fix L :: 'f gliteral'
assume L-in:  $L \in \# C$  and L-neq:  $L \neq \text{Pos } A$ 

have trail-false-lit  $((\text{Neg } A, \text{None}) \# \Gamma) L$ 
  using C-false L-in unfolding trail-false-cls-def by metis

hence  $L \in \text{fst } \text{'set } \Gamma$ 
  unfolding trail-false-lit-def
  using L-neq
  by (cases L) simp-all

hence trail-defined-lit  $\Gamma L$ 
  unfolding trail-defined-lit-def by argo

hence atm-of L  $\mid \in \mid \text{trail-atms } \Gamma$ 
  unfolding trail-defined-lit-iff-trail-defined-atm .

moreover have  $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A$ 
  using step-hyps unfolding linorder-trm.is-least-in-ffilter-iff by argo

```

```

ultimately have atm-of L <_t A
  by metis

hence L <_i Pos A
  by (cases L) simp-all

thus ¬ Pos A <_i 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 F |↑| (N |∪| Uer)).
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 F |↑| (N |∪| Uer)) (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 F |↑| (N |∪| Uer)
  using ⟨Γ' = Ln # Γ''⟩ ⟨Γ' = (Neg A, None) # Γ⟩ by force
}

have ¬ (∃ C |∈| iefac F |↑| (N |∪| Uer)). 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 |∪| Uer) 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  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
  ¬ (∃ C |∈| iefac  $\mathcal{F}$  |' (N |∪| Uer). trail-false-cls (Ln # Γ0) C)

```

```

      unfolding  $\langle \Gamma' = (-, None) \# \Gamma \rangle$ 
      by (metis suffixI trail-false-cls-if-trail-false-suffix)
    qed
  next
  case step-hyps: (decide-pos A C  $\Gamma'$   $\mathcal{F}'$ )

  have
    A-in:  $A \in | \text{atms-of-clss } (N \cup U_{er})$  and
    A-gt:  $\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A$  and
    A-lt:  $\forall y \in | \text{atms-of-clss } (N \cup U_{er}).$ 
       $y \neq A \longrightarrow (\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t y) \longrightarrow A \prec_t y$ 
    using  $\langle \text{linorder-trm.is-least-in-fset} - A \rangle$ 
    unfolding atomize-conj linorder-trm.is-least-in-filter-iff
    by argo

  have trail-atms  $\Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$ 
    unfolding  $\langle \Gamma' = (\text{Pos } A, -) \# \Gamma \rangle$  by simp

  show ?thesis
    unfolding  $\langle s' = (-, -, -) \rangle$ 
  proof (intro ord-res-10-invars.intros allI impI conjI)
    show sorted-wrt  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma'$ 
      unfolding  $\langle \Gamma' = (\text{Pos } A, None) \# \Gamma \rangle$  sorted-wrt.simps
    proof (intro conjI ballI)
      fix Ln :: 'f gliteral  $\times$  'f gclause option
      assume Ln  $\in$  set  $\Gamma$ 

      hence atm-of (fst Ln)  $\in | \text{trail-atms } \Gamma$ 
        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  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$ 
      using  $\Gamma$ -sorted .
    qed

  show  $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{ord-res.is-strictly-maximal-lit}$ 
    (fst Ln) C
    unfolding  $\langle \Gamma' = (-, None) \# \Gamma \rangle$ 
    using invars by simp

  show linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-clss (N  $\cup$   $U_{er}$ ))
    unfolding  $\langle \text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma) \rangle$  finsert.rep-eq
  proof (intro linorder-trm.is-lower-set-insertI ballI impI)
    show  $A \in | \text{atms-of-clss } (N \cup U_{er})$ 
      using A-in .
    next

```

```

fix  $w :: 'f\ gterm$ 
assume  $w \in |atms-of-cls (N \cup U_{er})$  and  $w \prec_t A$ 
thus  $w \in |trail-atms \Gamma$ 
  by (metis A-lt  $\Gamma$ -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  $\Gamma$ ) (atms-of-cls (N  $\cup$   $U_{er}$ ))
    using  $\Gamma$ -lower .
qed

{
  fix  $Ln$  and  $\Gamma''$ 
  assume  $\Gamma' = Ln \# \Gamma''$ 

  have snd  $Ln = None$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle$  step-hyps by auto

  moreover have  $\neg (\exists C \in |iefac \mathcal{F}' | \uparrow (N \cup U_{er}). trail-false-cls ((Pos A,$ 
None)  $\# \Gamma$ ) C)
  proof (rule notI , elim bexE)
    fix  $D :: 'f\ gclause$ 
    assume D-in:  $D \in |iefac \mathcal{F}' | \uparrow (N \cup U_{er})$ 

    hence  $D = efac C \vee D \in |iefac \mathcal{F}' | \uparrow (N \cup U_{er})$ 
    unfolding  $\langle \mathcal{F}' = (if\ ord-res.is-strictly-maximal-lit (Pos\ A)\ C\ then\ \mathcal{F}\ else$ 
finsert C  $\mathcal{F}$ ) \rangle
    by (smt (z3) fimage-iff finsert-iff iefac-def)

    hence  $\neg trail-false-cls \Gamma' D$ 
    proof (elim disjE)
      assume  $D = efac C$ 

      hence trail-false-cls  $\Gamma' D \longleftrightarrow trail-false-cls \Gamma' C$ 
      by (simp add: trail-false-cls-def)

      moreover have  $\neg trail-false-cls \Gamma' C$ 
        using step-hyps unfolding linorder-cls.is-least-in-filter-iff by metis

      ultimately show  $\neg trail-false-cls \Gamma' D$ 
        by argo
    next
    assume  $D \in |iefac \mathcal{F}' | \uparrow (N \cup U_{er})$ 
    thus  $\neg trail-false-cls \Gamma' D$ 
      using step-hyps by metis
  qed

  moreover assume trail-false-cls ((Pos A, None)  $\# \Gamma$ ) D

  ultimately show False

```

unfolding $\langle \Gamma' = (Pos\ A, None) \# \Gamma \rangle$
by *contradiction*
qed

ultimately show $snd\ Ln \neq None \longleftrightarrow$
 $(\exists C | \in | iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er}).\ trail\text{-}false\text{-}cls\ \Gamma' C)$
unfolding $\langle \Gamma' = (Pos\ A, None) \# \Gamma \rangle$ **by** *argo*

show $snd\ Ln \neq None \implies is\text{-}pos\ (fst\ Ln)$
using $\langle snd\ Ln = None \rangle$ **by** *argo*

have $\neg fBex\ (iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er}))\ (trail\text{-}false\text{-}cls\ \Gamma)$
using *step-hyps* **by** *argo*

hence $\forall x \in set\ \Gamma.\ snd\ x = None$
using *invars* **by** (*metis list.set-cases*)

thus $\forall x \in set\ \Gamma''.\ snd\ x = None$
using $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (Pos\ A, None) \# \Gamma \rangle$ **by** *simp*

show $\bigwedge C.\ snd\ Ln = Some\ C \implies clause\text{-}could\text{-}propagate\ \Gamma''\ C\ (fst\ Ln)$
using $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (Pos\ A, None) \# \Gamma \rangle$ **by** *force*

show $\bigwedge D.\ snd\ Ln = Some\ D \implies D \ | \in | iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er})$
using $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (Pos\ A, None) \# \Gamma \rangle$ **by** *force*

}

show $\bigwedge \Gamma_1\ Ln\ \Gamma_0.\ \Gamma' = \Gamma_1 \ @\ Ln \ \# \ \Gamma_0 \implies snd\ Ln = None \implies$
 $\neg (\exists C | \in | iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er}).\ trail\text{-}false\text{-}cls\ (Ln \ \# \ \Gamma_0)\ C)$
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' \mathcal{F}'*)

have

A-in: $A \ | \in | atms\text{-}of\text{-}class\ (N \ | \cup \ | U_{er})$ **and**
A-gt: $\forall A_1 | \in | trail\text{-}atms\ \Gamma.\ A_1 \prec_t A$ **and**
A-lt: $\forall y | \in | atms\text{-}of\text{-}class\ (N \ | \cup \ | U_{er}).$
 $y \neq A \longrightarrow (\forall A_1 | \in | trail\text{-}atms\ \Gamma.\ A_1 \prec_t y) \longrightarrow A \prec_t y$
using $\langle linorder\text{-}trm.is\text{-}least\text{-}in\text{-}fset - A \rangle$
unfolding *atomize-conj linorder-trm.is-least-in-filter-iff*
by *argo*

have

C-in: $C \ | \in | iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er})$ **and**
C-prop: *clause-could-propagate $\Gamma\ C\ (Pos\ A)$* **and**
C-lt: $\forall D \ | \in | iefac\ \mathcal{F}' \ | \uparrow (N \ | \cup \ | U_{er}).$

```

     $D \neq C \longrightarrow \text{clause-could-propagate } \Gamma \ D \ (Pos \ A) \longrightarrow C \prec_c \ D$ 
using  $\langle \text{linorder-cls.is-least-in-fset} - C \rangle$ 
unfolding  $\text{atomize-conj linorder-cls.is-least-in-filter-iff}$  by argo

have  $\text{trail-atms } \Gamma' = \text{finsert } A \ (\text{trail-atms } \Gamma)$ 
unfolding  $\langle \Gamma' = (Pos \ A, -) \# \Gamma \rangle$  by simp

show ?thesis
unfolding  $\langle s' = (-, -, -) \rangle$ 
proof (intro ord-res-10-invars.intros allI impI conjI)
show  $\text{sorted-wrt } (\lambda x \ y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma'$ 
unfolding  $\langle \Gamma' = (Pos \ A, -) \# \Gamma \rangle$  sorted-wrt.simps
proof (intro conjI ballI)
fix  $L_n :: 'f \ \text{gliteral} \times 'f \ \text{gclause} \ \text{option}$ 
assume  $L_n \in \text{set } \Gamma$ 

hence  $\text{atm-of } (fst \ L_n) \in \text{trail-atms } \Gamma$ 
by (simp add: fset-trail-atms)

thus  $\text{atm-of } (fst \ L_n) \prec_t \text{atm-of } (fst \ (Pos \ A, \text{Some } (efac \ C)))$ 
unfolding prod.sel literal.sel
using A-gt by metis

next
show  $\text{sorted-wrt } (\lambda x \ y. \text{atm-of } (fst \ y) \prec_t \text{atm-of } (fst \ x)) \ \Gamma$ 
using  $\Gamma\text{-sorted}$  .

qed

show  $\forall L_n \in \text{set } \Gamma'. \forall C. \text{snd } L_n = \text{Some } C \longrightarrow \text{ord-res.is-strictly-maximal-lit}$ 
 $(fst \ L_n) \ C$ 
unfolding  $\langle \Gamma' = (Pos \ A, \text{Some } (efac \ C)) \# \Gamma \rangle$ 
using invars step-hyps
by (simp add: clause-could-propagate-def greatest-literal-in-efacI
linorder-cls.is-least-in-filter-iff)

show  $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma') \ (\text{atms-of-clss } (N \ \cup \ U_{er}))$ 
unfolding  $\langle \text{trail-atms } \Gamma' = \text{finsert } A \ (\text{trail-atms } \Gamma) \rangle$  finsert.rep-eq
proof (intro linorder-trm.is-lower-set-insertI ballI impI)
show  $A \in \text{atms-of-clss } (N \ \cup \ U_{er})$ 
using A-in .

next
fix  $w :: 'f \ \text{gterm}$ 
assume  $w \in \text{atms-of-clss } (N \ \cup \ U_{er})$  and  $w \prec_t A$ 
thus  $w \in \text{trail-atms } \Gamma$ 
by (metis A-lt \Gamma-lower linorder-trm.dual-order.asym linorder-trm.neq-iff
linorder-trm.not-in-lower-setI)

next
show  $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) \ (\text{atms-of-clss } (N \ \cup \ U_{er}))$ 
using  $\Gamma\text{-lower}$  .

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 ℱ |↑| (N |∪| Uer) and D-false:
trail-false-cls Γ' D
  using step-hyps by metis

  have (∃ C |∈| iefac ℱ' |↑| (N |∪| Uer). 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 |∪| Uer)
  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 |∪| Uer). 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  $\mathcal{F}'$  | $\uparrow$ | (N | $\cup$ |  $U_{er}$ )
  proof (cases ord-res.is-strictly-maximal-lit (Pos A) C)
    case True
      thus ?thesis
        unfolding  $\langle \mathcal{F}' = - \rangle$ 
        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  $\langle \mathcal{F}' = - \rangle$ 
        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  $\bigwedge D. \text{snd } Ln = \text{Some } D \implies D |∈| iefac \mathcal{F}' | $\uparrow$ | (N | $\cup$ |  $U_{er}$ )$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (\text{Pos } A, \text{Some } (\text{efac } C)) \# \Gamma \rangle$  by force
}

show  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
 $\neg (\exists C |∈| iefac \mathcal{F}' | $\uparrow$ | (N | $\cup$ |  $U_{er}$ ). \text{trail-false-cls } (Ln \# \Gamma_0) C)$ 
  unfolding  $\langle \Gamma' = (-, \text{Some } -) \# \Gamma \rangle$ 
  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  $U_{er}' \Gamma'$ )

note D-max-lit =  $\langle \text{ord-res.is-maximal-lit } (\text{Neg } A) D \rangle$ 
have C-max-lit:  $\langle \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C \rangle$ 
  using invars by (metis map-of-SomeD split-pairs step-hyps(4))

have
  D-in:  $D |∈| iefac \mathcal{F} | $\uparrow$ | (N | $\cup$ |  $U_{er})$  and
  D-false: trail-false-cls  $\Gamma D$  and
  D-lt:  $\forall E |∈| iefac \mathcal{F} | $\uparrow$ | (N | $\cup$ |  $U_{er}). E \neq D \implies \text{trail-false-cls } \Gamma E \implies D \prec_c$ 
E
  using  $\langle \text{linorder-cls.is-least-in-fset } - D \rangle$ 
  unfolding atomize-conj linorder-cls.is-least-in-filter-iff by argo

have A |∈| atms-of-clss (N | $\cup$ |  $U_{er}$ )
  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 Γb where Γ = (Pos A, Some C) # Γb
  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) # Γb⟩ 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 (λLn. True) Γ
    using step-hyps(6) by simp

also have ... = []
  by simp

```

finally show *?thesis*
using $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_b \rangle$ **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* \prec_t *atm-of (Neg A)*
proof (*rule lit-in-eres-lt-greatest-lit-in-greatest-resolvent [of C D]*)
show *eres C D* \neq *D*
using \langle *eres C D* \neq *D* \rangle .
next
show *ord-res.is-maximal-lit (Neg A) D*
using *D-max-lit* .
next
show \neg *Neg A* \notin *# D*
using *D-false* \langle *trail-consistent* Γ \rangle
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* \in *# eres C D*
using *eres-max-lit unfolding linorder-lit.is-maximal-in-mset-iff* **by** *argo*
qed

hence *atm-of K* \prec_t *A*
unfolding *literal.sel* .

hence *atm-of K* \preceq_t *A*
by *order*

have $\Gamma' = dropWhile\ (\lambda Ln. atm-of\ K\ \preceq_t\ atm-of\ (fst\ Ln))\ \Gamma$
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 $\dots = dropWhile\ (\lambda Ln. atm-of\ K\ \preceq_t\ atm-of\ (fst\ Ln))\ \Gamma_b$
unfolding $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_b \rangle$
unfolding *dropWhile.simps prod.sel literal.sel*
using \langle *atm-of K* \preceq_t *A* \rangle **by** *simp*

finally have $\Gamma' = dropWhile\ (\lambda Ln. atm-of\ K\ \preceq_t\ atm-of\ (fst\ Ln))\ \Gamma_b$.

hence *trail-atms* Γ' $|\subseteq|$ *trail-atms* Γ_b
by (*simp only: suffix-dropWhile trail-atms-subset-if-suffix*)

moreover have *A* \notin *trail-atms* Γ_b
proof (*rule notI*)
assume *A* \in *trail-atms* Γ_b
then obtain *Ln* **where** *Ln* \in *set* Γ_b **and** *atm-of (fst Ln) = A*

unfolding *fset-trail-atms* **by** *blast*
moreover have $\forall y \in \text{set } \Gamma_b. \text{ atm-of } (fst\ y) \prec_t A$
using $\Gamma\text{-sorted}[\text{unfolded } \langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_b \rangle]$ **by** *simp*
ultimately have $A \prec_t A$
by *metis*
thus *False*
by *order*
qed

moreover have $\text{trail-atms } \Gamma_b \mid \subseteq \mid \text{trail-atms } \Gamma$
proof (*rule trail-atms-subset-if-suffix*)
show *suffix* $\Gamma_b\ \Gamma$
by (*simp only:* $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_b \rangle$ *suffix-ConsI*)
qed

ultimately show *?thesis*
by *fast*
qed

ultimately have $\Gamma' \neq \Gamma$
by *metis*

have *C-in:* $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
using *invars*
by (*meson* $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_b \rangle$ *snd-conv*)

define $P :: 'f\ \text{gliteral} \times 'f\ \text{gclause option} \Rightarrow \text{bool}$ **where**
 $P \equiv \lambda Ln. \forall K. \text{ord-res.is-maximal-lit } K\ (eres\ C\ D) \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (fst\ Ln)$

have $\Gamma = \text{takeWhile } P\ \Gamma\ @\ \Gamma'$
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* $\langle \Gamma' = \text{dropWhile } -\ \Gamma \rangle$ **by** *simp*

hence *suffix* $\Gamma'\ \Gamma$
unfolding *suffix-def* **by** *metis*

show *?thesis*
unfolding $\langle s' = (-, -, -) \rangle$
proof (*intro ord-res-10-invars.intros allI impI conjI*)
show $\Gamma'\text{-sorted: sorted-wrt } (\lambda x\ y. \text{atm-of } (fst\ y) \prec_t \text{atm-of } (fst\ x))\ \Gamma'$
by (*metis (no-types, lifting)* $\Gamma\text{-sorted } \langle \text{suffix } \Gamma'\ \Gamma \rangle$ *sorted-wrt-append suffix-def*)

show $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = Some\ C \longrightarrow \text{ord-res.is-strictly-maximal-lit } (fst\ Ln)\ C$
using $\langle \text{suffix } \Gamma'\ \Gamma \rangle$ *invars(3) set-mono-suffix* **by** *blast*

have $\bigwedge xs. \text{fset } (\text{trail-atms } xs) = \text{atm-of } 'fst\ 'set\ xs$

unfolding *fset-trail-atms* ..
also have $\bigwedge xs. \text{atm-of } \langle \text{fst } \rangle \text{ set } xs = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \text{ } xs)$
by (*simp add: image-comp*)
finally have $\bigwedge xs. \text{fset } (\text{trail-atms } xs) = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \text{ } xs)$.

have $\text{atms-of-clss } (N \mid\cup\mid U_{er}') = \text{atms-of-clss } (\text{eres } C \ D) \mid\cup\mid \text{atms-of-clss } (N \mid\cup\mid U_{er})$
unfolding $\langle U_{er}' = \text{finsert } (\text{eres } C \ D) \ U_{er} \rangle$ **by** *simp*

also have $\dots = \text{atms-of-clss } (N \mid\cup\mid U_{er})$
proof –
have $\text{atms-of-clss } (\text{eres } C \ D) \mid\subseteq\mid \text{atms-of-clss } C \ \mid\cup\mid \text{atms-of-clss } D$
by (*smt (verit, best) atm-of-clss-def fimage-iff fset-fset-mset fsubsetI*
unionCI
lit-in-one-of-resolvents-if-in-eres)

moreover have $\text{atms-of-clss } C \mid\subseteq\mid \text{atms-of-clss } (N \mid\cup\mid U_{er})$
using *C-in*
by (*metis atm-of-clss-fimage-iefac atm-of-clss-finsert finsert-absorb fsubset-funion-eq*)

moreover have $\text{atms-of-clss } D \mid\subseteq\mid \text{atms-of-clss } (N \mid\cup\mid U_{er})$
using *D-in*
by (*metis atm-of-clss-fimage-iefac atm-of-clss-finsert finsert-absorb fsubset-funion-eq*)

ultimately show *?thesis*
by *blast*
qed

finally have $\text{atms-of-clss } (N \mid\cup\mid U_{er}') = \text{atms-of-clss } (N \mid\cup\mid U_{er})$.

show Γ' -lower: *linorder-trm.is-lower-fset* (*trail-atms* Γ') ($\text{atms-of-clss } (N \mid\cup\mid U_{er}')$)
unfolding $\langle \text{fset } (\text{trail-atms } \Gamma') = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \Gamma') \rangle$
proof (*rule linorder-trm.sorted-and-lower-set-appendD-right(2)*)
have *sorted-wrt* $(\lambda x \ y. y \prec_t x)$ ($\text{map } (\text{atm-of } \circ \text{fst}) \Gamma$)
using Γ -sorted **by** (*simp add: sorted-wrt-map*)

thus *sorted-wrt* $(\lambda x \ y. y \prec_t x)$ ($\text{map } (\text{atm-of } \circ \text{fst})$ (*takeWhile* $P \ \Gamma$) @ $\text{map } (\text{atm-of } \circ \text{fst}) \Gamma'$)
using $\langle \Gamma = \text{takeWhile } P \ \Gamma \ @ \ \Gamma' \rangle$
by (*metis map-append*)

next
show *linorder-trm.is-lower-set*
 $(\text{set } (\text{map } (\text{atm-of } \circ \text{fst})$ (*takeWhile* $P \ \Gamma$) @ $\text{map } (\text{atm-of } \circ \text{fst}) \Gamma'))$
 $(\text{fset } (\text{atms-of-clss } (N \mid\cup\mid U_{er}')))$
unfolding *map-append[symmetric]*
unfolding $\langle \Gamma = \text{takeWhile } P \ \Gamma \ @ \ \Gamma' \rangle$ [*symmetric*]

```

using  $\Gamma$ -lower
unfolding  $\langle \text{fset } (\text{trail-atms } \Gamma) = \text{set } (\text{map } (\text{atm-of } o \text{ fst}) \Gamma) \rangle$ 
unfolding  $\langle \text{atms-of-clss } (N \mid \cup \mid U_{er}') = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$ 
.
qed

have  $\text{no-false-clss-in-}\Gamma'$ :  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')). \text{trail-false-clss } \Gamma' C$ 
if  $\text{eres-max-lit}$ :  $\text{ord-res.is-maximal-lit } K \text{ (eres } C D)$ 
for  $K$ 
proof –
  have  $\text{FOO}$ :  $\bigwedge Ln.$ 
     $(\forall K. \text{ord-res.is-maximal-lit } K \text{ (eres } C D) \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \longleftrightarrow$ 
     $\text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)$ 
  using  $\text{eres-max-lit}$ 
  by  $(\text{metis linorder-lit.Uniq-is-maximal-in-mset the1-equality})$ 

  have  $\forall x \in \text{set } \Gamma'. \text{atm-of } (\text{fst } x) \prec_t \text{atm-of } K$ 
  using  $\langle \Gamma' = \text{dropWhile } - \Gamma \rangle [\text{unfolded } \text{FOO}] \Gamma\text{-sorted}$ 
by  $(\text{metis linorder-trm.not-le mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone mono-atms-lt})$ 

hence  $\forall x \in \text{set } \Gamma'. \text{atm-of } (\text{fst } x) \neq \text{atm-of } K$ 
by  $\text{fastforce}$ 

hence  $\text{atm-of } K \notin \text{trail-atms } \Gamma'$ 
unfolding  $\text{fset-trail-atms}$  by  $\text{auto}$ 

hence  $\neg \text{trail-defined-lit } \Gamma' K$ 
unfolding  $\text{trail-defined-lit-iff-trail-defined-atm}$  .

hence  $\neg \text{trail-false-lit } \Gamma' K$ 
by  $(\text{metis trail-defined-lit-iff-true-or-false})$ 

hence  $\neg \text{trail-false-clss } \Gamma' \text{ (eres } C D)$ 
using  $\text{eres-max-lit}$ 
unfolding  $\text{linorder-lit.is-maximal-in-mset-iff trail-false-clss-def}$ 
by  $\text{metis}$ 

show  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')). \text{trail-false-clss } \Gamma' C$ 
unfolding  $\langle U_{er}' = \text{finsert } (\text{eres } C D) U_{er} \rangle$ 
proof  $(\text{rule notI } , \text{elim bexE})$ 
fix  $E :: 'f \text{ gclause}$ 
assume
   $E\text{-in}$ :  $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid \text{finsert } (\text{eres } C D) U_{er})$  and
   $E\text{-false}$ :  $\text{trail-false-clss } \Gamma' E$ 

have  $E = \text{iefac } \mathcal{F} \text{ (eres } C D) \vee E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 
using  $E\text{-in}$  by  $\text{simp}$ 

```

thus *False*
proof (*elim disjE*)
 assume $E = \text{iefac } \mathcal{F} \text{ (eres } C \ D)$

 hence *trail-false-cls* $\Gamma' \text{ (eres } C \ D)$
 using *E-false*
 by (*simp add: iefac-def trail-false-cls-def*)

thus *False*
 using $\langle \neg \text{ trail-false-cls } \Gamma' \text{ (eres } C \ D) \rangle$ **by** *contradiction*
next
 assume $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$

 moreover have *trail-false-cls* $\Gamma \ E$
 using *E-false* $\langle \text{suffix } \Gamma' \ \Gamma \rangle$ **by** (*metis trail-false-cls-if-trail-false-suffix*)

 ultimately have $D \preceq_c E$
 using *D-lt E-false* **by** *fast*

 then obtain L **where** $L \in \# E$ **and** $\text{Neg } A \preceq_l L$
 using *D-max-lit*
 by (*metis empty-iff linorder-cls.leD linorder-lit.is-maximal-in-mset-iff linorder-lit.leI ord-res.mulp-if-all-left-smaller set-mset-empty*)

 hence $A \preceq_t \text{atm-of } L$
 by (*cases L*) *simp-all*

 moreover have $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er} \prime)$
 using $\langle A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er} \prime) \rangle$
 using $\langle \text{atms-of-clss } (N \mid \cup \mid U_{er} \prime) = \text{atms-of-clss } (N \mid \cup \mid U_{er} \prime) \rangle$
 by (*simp only:*)

 ultimately have *atm-of* $L \mid \notin \mid \text{trail-atms } \Gamma'$
 using *linorder-trm.not-in-lower-setI* [*OF* Γ' -*lower*] $\langle A \mid \notin \mid \text{trail-atms } \Gamma' \rangle$
by *blast*

 hence $\neg \text{trail-defined-lit } \Gamma' \ L$
 unfolding *trail-defined-lit-iff-trail-defined-atm* .

 hence $\neg \text{trail-false-lit } \Gamma' \ L$
 by (*metis trail-defined-lit-iff-true-or-false*)

 moreover have *trail-false-lit* $\Gamma' \ L$
 using *E-false* $\langle L \in \# E \rangle$ **unfolding** *trail-false-cls-def* **by** *metis*

 ultimately show *False*
 by *contradiction*
qed

qed
qed

have *ex-max-lit-in-eres-if*: $\exists K. \text{ord-res.is-maximal-lit } K \text{ (eres } C D)$
if $\Gamma' = \Gamma_1 @ Ln \# \Gamma_0$ **for** Ln **and** $\Gamma_1 \Gamma_0$
proof –
have $\exists x xs. \Gamma' = x \# xs$
using *that neq-Nil-conv* **by** *blast*

hence $\exists x. \neg (\forall K. \text{ord-res.is-maximal-lit } K \text{ (eres } C D) \longrightarrow \text{atm-of } K \preceq_t \text{atm-of (fst } x))$
unfolding *step-hyps(6) dropWhile-eq-Cons-conv* **by** *auto*

thus *?thesis*
by *metis*
qed

{
fix Ln **and** Γ''
assume $\Gamma' = Ln \# \Gamma''$

then obtain K **where**
eres-max-lit: $\text{ord-res.is-maximal-lit } K \text{ (eres } C D)$
using *ex-max-lit-in-eres-if*[*of [], simplified*] **by** *metis*

have $\neg (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}'). \text{trail-false-cls } \Gamma' C)$
using *no-false-cls-in- Γ' eres-max-lit* **by** *metis*

moreover have $\text{snd } Ln = \text{None}$
using *invars* $\langle \Gamma' \neq \Gamma \rangle \langle \text{suffix } \Gamma' \Gamma \rangle$
by (*metis* $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma_b \rangle \langle \Gamma' = Ln \# \Gamma'' \rangle$ *in-set-conv-decomp*
suffix-Cons
suffix-def)

ultimately show $\text{snd } Ln \neq \text{None} \longleftrightarrow (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}'). \text{trail-false-cls } \Gamma' C)$
by *argo*

show $\text{snd } Ln \neq \text{None} \implies \text{is-pos (fst } Ln)$
using $\langle \text{snd } Ln = \text{None} \rangle$ **by** *argo*

show $\forall x \in \text{set } \Gamma''. \text{snd } x = \text{None}$
using *invars* $\langle \Gamma' \neq \Gamma \rangle \langle \text{suffix } \Gamma' \Gamma \rangle$
by (*metis* $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma_b \rangle \langle \Gamma' = Ln \# \Gamma'' \rangle$ *set-mono-suffix*
subsetD suffix-ConsD2)

show $\bigwedge C. \text{snd } Ln = \text{Some } C \implies \text{clause-could-propagate } \Gamma'' C \text{ (fst } Ln)$


```

    by (simp add: ⟨snd Ln = None⟩)

  show  $\bigwedge E. \text{snd } Ln = \text{Some } E \implies E \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})'$ 
    by (simp add: ⟨snd Ln = None⟩)
  }

  show  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
 $\neg (\exists C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})'. \text{trail-false-cls } (Ln \# \Gamma_0) C)$ 
  using ex-max-lit-in-eres-if no-false-cls-in- $\Gamma'$ 
  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:  $\neg \text{ord-res-8-final } (N, s)$  and
    invars: ord-res-10-invars N s
  shows  $\exists s'. \text{ord-res-10 } N s s'$ 
  using invars
  proof (cases N s rule: ord-res-10-invars.cases)
  case invars': (1  $\Gamma U_{er} \mathcal{F}$ )

```

have

```

  undef-atm-or-false-cls:
  ( $\exists x \in | \text{atms-of-cls } (N \mid \cup \mid U_{er}). x \notin | \text{trail-atms } \Gamma \mid \wedge$ 
 $\neg (\exists x \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x) \vee$ 
 $(\exists x \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x)$ ) and
  {#}  $\notin | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
  unfolding atomize-conj
  using not-final[unfolded ord-res-8-final.simps]  $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$  by metis

```

show *?thesis*

using *undef-atm-or-false-cls*

proof (*elim disjE conjE*)

assume

```

  ex-undef-atm:  $\exists x \in | \text{atms-of-cls } (N \mid \cup \mid U_{er}). x \notin | \text{trail-atms } \Gamma$  and
  no-conflict:  $\neg (\exists x \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma x)$ 

```

moreover have $\{ | A_2 \in | \text{atms-of-cls } (N \mid \cup \mid U_{er}). \forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2 \mid \} \neq \{ \mid \}$

proof –
obtain $A_2 :: 'f\ gterm\ where$
 $A_2\text{-in}: A_2 \mid\in\mid\ atms\text{-of}\text{-}cls (N \mid\cup\mid U_{er})$ **and**
 $A_2\text{-undef}: A_2 \mid\notin\mid\ trail\text{-}atms\ \Gamma$
using $ex\text{-}undef\text{-}atm$ **by** $metis$

have $A_1 \prec_t A_2$ **if** $A_1\text{-in}: A_1 \mid\in\mid\ trail\text{-}atms\ \Gamma$ **for** $A_1 :: 'f\ gterm$
proof –
have $A_1 \neq A_2$
using $A_1\text{-in}\ A_2\text{-undef}$ **by** $metis$

moreover have $linorder\text{-}trm.is\text{-}lower\text{-}fset (trail\text{-}atms\ \Gamma) (atms\text{-of}\text{-}cls (N \mid\cup\mid U_{er}))$
using $invars'$ [$unfolded\ trail\text{-}is\text{-}lower\text{-}set\text{-}def, simplified$] **by** $argo$

ultimately show $?thesis$
by ($meson\ A_2\text{-in}\ A_2\text{-undef}\ linorder\text{-}trm.is\text{-}lower\text{-}set\text{-}iff\ linorder\text{-}trm.linorder\text{-}cases$
 $that$)
qed

thus $?thesis$
using $A_2\text{-in}$
by ($smt (verit, ccfv\text{-}threshold) femptyE\ fmember\text{-}filter$)
qed

ultimately obtain $A :: 'f\ gterm\ where$
 $A\text{-least}: linorder\text{-}trm.is\text{-}least\text{-}in\text{-}fset \{ \mid A_2 \mid\in\mid\ atms\text{-of}\text{-}cls (N \mid\cup\mid U_{er}).$
 $\forall A_1 \mid\in\mid\ trail\text{-}atms\ \Gamma. A_1 \prec_t A_2 \}$ A
using $ex\text{-}undef\text{-}atm\ linorder\text{-}trm.ex\text{-}least\text{-}in\text{-}fset$ **by** $presburger$

show $\exists s'. ord\text{-}res\text{-}10\ N\ s\ s'$
proof ($cases\ \exists C \mid\in\mid\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\mid U_{er}).\ clause\text{-}could\text{-}propagate\ \Gamma\ C\ (Pos\ A)$)
case $True$
hence $\{ \mid C \mid\in\mid\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\mid U_{er}).\ clause\text{-}could\text{-}propagate\ \Gamma\ C\ (Pos\ A) \}$
 $\neq \{ \mid \}$
by $force$

then obtain C **where**
 $C\text{-least}: linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset$
 $\{ \mid C \mid\in\mid\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\mid U_{er}).\ clause\text{-}could\text{-}propagate\ \Gamma\ C\ (Pos\ A) \}$ C
using $linorder\text{-}cls.ex1\text{-}least\text{-}in\text{-}fset$ **by** $meson$

show $?thesis$
proof ($cases\ \exists D \mid\in\mid\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\mid U_{er}).\ trail\text{-}false\text{-}cls ((Pos\ A, Some (efac\ C)) \# \Gamma)\ D$)
case $True$
then show $?thesis$
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$

```

    using ord-res-10.propagate[OF no-conflict A-least C-least]
    by metis
next
case False
hence  $\neg (\exists C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). \text{trail-false-cls } ((\text{Pos } A, \text{None}) \# \Gamma))$ 
C)
    by (simp add: trail-false-cls-def trail-false-lit-def)
    then show ?thesis
    unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
    using ord-res-10.decide-pos[OF no-conflict A-least C-least]
    by metis
qed
next
case False
thus ?thesis
    unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
    using ord-res-10.decide-neg[OF no-conflict A-least]
    by metis
qed
next
assume  $\exists x \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). \text{trail-false-cls } \Gamma \ x$ 

then obtain  $D :: 'f \text{ gclause where}$ 
 $D\text{-least: linorder-cls.is-least-in-fset}$ 
 $(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}))) \ D$ 
by (metis emptyE ffilter-filter linorder-cls.ex-least-in-fset)

hence  $D\text{-in: } D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$  and  $D\text{-false: trail-false-cls } \Gamma \ D$ 
unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

have  $D \neq \{\#\}$ 
using  $D\text{-in } \langle \{\#\} \mid \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$  by metis

hence  $\Gamma \neq []$ 
using  $D\text{-false}$ 
by (auto simp add: trail-false-cls-def trail-false-lit-def)

then obtain  $Ln \ \Gamma'$  where  $\Gamma = Ln \ \# \ \Gamma'$ 
using neq-Nil-conv by metis

hence  $\text{snd } Ln \neq \text{None}$ 
using  $\text{invars}' \langle \exists x \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). \text{trail-false-cls } \Gamma \ x \rangle$  by presburger

have  $\forall x \in \text{set } \Gamma'. \text{snd } x = \text{None}$ 
using  $\text{invars}' \langle \Gamma = Ln \ \# \ \Gamma' \rangle$  by metis

have  $\neg (\exists x \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). \text{trail-false-cls } \Gamma' \ x)$ 
proof (cases  $\Gamma'$ )
case Nil

```

```

thus ?thesis
  using ⟨{#} |≠| iefac  $\mathcal{F}$  |† (N |∪| Uer)⟩
  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 ⟨∀ Γ1 Ln Γ0. Γ = Γ1 @ Ln # Γ0 → snd Ln = None →
    ¬ (∃ C |∈| -. trail-false-cls (Ln # Γ0) C)⟩[rule-format, of [Ln] x xs,
simplified]
  using Cons ⟨Γ = Ln # Γ'⟩ by blast
qed

hence ¬ trail-false-cls Γ' D
  using D-in by metis

hence – fst Ln ∈# 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 ∀ L ∈# D. atm-of L |∈| 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 ∀ x |∈| trail-atms Γ'. x <t atm-of (fst Ln)
  using ⟨sorted-wrt (λx y. atm-of (fst y) <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  $\langle \Gamma = Ln \# \Gamma' \rangle$ 
using  $\langle fst Ln = Pos A \rangle \langle snd Ln \neq None \rangle$  by force

thus  $\exists s'. ord-res-10 N s s'$ 
unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma) \rangle$ 
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-preserved-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  $\times$  'f gclause fset  $\times$  'f gclause fset  $\times$  ('f gliteral  $\times$  'f gclause option)
list  $\times$ 
'f gclause option

```

```

context simulation-SCLFOL-ground-ordered-resolution begin

```

```

lemma
fixes  $N U_{er} \mathcal{F} \Gamma A$ 
assumes
 $no-false-cls: \neg (\exists C \in |iefac \mathcal{F} | \uparrow (N \cup U_{er}). trail-false-cls \Gamma C)$  and
 $A-least: linorder-trm.is-least-in-fset \{A_2 \in |atms-of-cls (N \cup U_{er}).$ 
 $\forall A_1 \in |trail-atms \Gamma. A_1 \prec_t A_2\} A$  and
 $C-least: linorder-cls.is-least-in-fset \{C \in |iefac \mathcal{F} | \uparrow (N \cup U_{er}).$ 
 $clause-could-propagate \Gamma C (Pos A)\} C$ 
defines
 $\Gamma' \equiv (Pos A, None) \# \Gamma$  and
 $\mathcal{F}' \equiv (if linorder-lit.is-greatest-in-mset C (Pos A) then \mathcal{F} else finsert C \mathcal{F})$ 
shows
 $(\exists C \in |iefac \mathcal{F} | \uparrow (N \cup U_{er}). trail-false-cls \Gamma' C) \longleftrightarrow$ 
 $(\exists C \in |iefac \mathcal{F}' | \uparrow (N \cup U_{er}). trail-false-cls \Gamma' C)$ 
by (meson bex-trail-false-cls-simp)

```

inductive ord-res-11 where

decide-neg:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ clause-could-propagate } \Gamma C (Pos A)) \implies \\ & \Gamma' = (Neg A, None) \# \Gamma \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}, \Gamma', None) \mid \end{aligned}$$

decide-pos:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A)\} C \implies \\ & \Gamma' = (Pos A, None) \# \Gamma \implies \\ & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma' C) \implies \\ & \mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}', \Gamma', None) \mid \end{aligned}$$

propagate:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A)\} C \implies \\ & \Gamma' = (Pos A, Some (efac C)) \# \Gamma \implies \\ & (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma' C) \implies \\ & \mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}', \Gamma', None) \mid \end{aligned}$$

conflict:

$$\begin{aligned} & \text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma D\} \\ & D \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}, \Gamma, Some D) \mid \end{aligned}$$

skip: $- L \notin \# C \implies$

$$\text{ord-res-11 } N (U_{er}, \mathcal{F}, (L, n) \# \Gamma, Some C) (U_{er}, \mathcal{F}, \Gamma, Some C) \mid$$

resolution:

$$\begin{aligned} & \Gamma = (L, Some D) \# \Gamma' \implies - L \in \# C \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, Some C) (U_{er}, \mathcal{F}, \Gamma, Some ((C - \{\#- L\#}) + (D \\ & - \{\#L\#}))) \mid \end{aligned}$$

backtrack:

$$\begin{aligned} & \Gamma = (L, None) \# \Gamma' \implies - L \in \# C \implies \\ & \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, Some C) (\text{finsert } C U_{er}, \mathcal{F}, \Gamma', None) \end{aligned}$$

```

lemma right-unique-ord-res-11:
  fixes  $N :: 'f\ gclause\ fset$ 
  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  $\mathcal{F}$   $U_{er}$   $\Gamma$   $A1$   $\Gamma1'$ )
      show ?thesis
        using step2 unfolding  $\langle x = \rightarrow$ 
  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 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  $\Gamma'$   $\mathcal{F}'$ )
      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  $\Gamma'$   $\mathcal{F}'$ )
      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  $\mathcal{F}$   $U_{er}$   $\Gamma$   $A1$   $C1$   $\Gamma1'$   $\mathcal{F}1'$ )
  show ?thesis
    using step2 unfolding  $\langle x = \rightarrow$ 
  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 (decide-pos 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 hyps2: (propagate 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  $\Gamma 1' = \text{trail-false-cls } \Gamma'$ 
  unfolding < $\Gamma 1' = - \# \Gamma$ > < $\Gamma' = - \# \Gamma$ >
  unfolding trail-false-cls-def trail-false-lit-def
  by simp
hence False
  using hyps1 hyps2 by argo
thus ?thesis ..
next
case (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  $\Gamma 1'$   $\mathcal{F} 1'$ )
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  $\Gamma 1' = \text{trail-false-cls } \Gamma'$ 
  unfolding < $\Gamma 1' = - \# \Gamma$ > < $\Gamma' = - \# \Gamma$ >
  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  $\langle x = - \rangle$ 
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) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
  case (decide-pos  $A\ C\ \Gamma'\ \mathcal{F}'$ )
    with hyps1 have False
      by (metis (no-types) linorder-cls.is-least-in-ffilter-iff)
    thus ?thesis ..
  next
  case (propagate  $A\ C\ \Gamma'\ \mathcal{F}'$ )
    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  $L\ C\ U_{er}\ \mathcal{F}\ n\ \Gamma$ )
show ?thesis
  using step2 unfolding  $\langle x = - \rangle$ 
proof (cases  $N (U_{er}, \mathcal{F}, (L, n) \# \Gamma, Some\ C)$   $z\ rule: ord-res-11.cases$ )
  case skip
    with hyps1 show ?thesis
      by argo
  next
  case (resolution  $L'\ D'\ \Gamma'$ )
    with hyps1 have False
      by simp
    thus ?thesis ..
  next
  case (backtrack  $K\ \Gamma'$ )
    with hyps1 have False
      by simp
    thus ?thesis ..
  qed
next
case hyps1: (resolution  $\Gamma\ L1\ D1\ \Gamma1'\ C\ U_{er}\ \mathcal{F}$ )

```

```

show ?thesis
  using step2 unfolding ⟨x = -⟩
proof (cases N (Uer, 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 Uer F)
show ?thesis
  using step2 unfolding ⟨x = -⟩
proof (cases N (Uer, 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:

$$\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \mid \notin \mid \text{trail-atms } \Gamma) \implies$$

$$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies$$

$$\text{ord-res-11-final } (N, U_{er}, \mathcal{F}, \Gamma, \text{None}) \mid$$

contradiction-found:

$$\text{ord-res-11-final } (N, U_{er}, \mathcal{F}, [], \text{Some } \{\#\})$$

sublocale *ord-res-11-semantic: 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* $N U_{er} \Gamma \mathcal{F}$)

have $\nexists A. \text{linorder-trm.is-least-in-fset } \{|A_2 | \in | \text{atms-of-clss } (N \cup | U_{er}).$
 $\forall A_1 | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A$
using $\langle \neg (\exists A | \in | \text{atms-of-clss } (N \cup | U_{er}). A | \notin | \text{trail-atms } \Gamma) \rangle$
unfolding *linorder-trm.is-least-in-ffilter-iff*
by *blast*

moreover have $\nexists D. \text{linorder-cls.is-least-in-fset}$
 $(\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er}))) D$
using $\langle \neg (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er}). \text{trail-false-cls } \Gamma C) \rangle$
unfolding *linorder-cls.is-least-in-ffilter-iff*
by *metis*

ultimately have $\nexists x. \text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, \text{None}) x$
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 $\nexists x. \text{ord-res-11 } N (U_{er}, \mathcal{F}, [], \text{Some } \{\#\}) x$
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 $N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ **if**
ord-res-10-invars $N (U_{er}, \mathcal{F}, \Gamma)$ **and**
 $\{\#\} | \in | N \cup | U_{er} \longrightarrow \Gamma = []$ **and**
 $\forall C. \mathcal{C} = \text{Some } C \longrightarrow \text{atms-of-cl } C | \subseteq | \text{atms-of-clss } (N \cup | U_{er})$ **and**
 $\forall C. \mathcal{C} = \text{Some } C \longrightarrow \text{trail-false-cl } \Gamma C$ **and**
 $\text{atms-of-clss } U_{er} | \subseteq | \text{atms-of-clss } N$ **and**
 $\forall C | \in | \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{linorder-lit.is-maximal-in-mset } C L$

lemma *ord-res-11-invars-initial-state: ord-res-11-invars* $N (\{\|\}, \{\|\}, [], \text{None})$
by (*intro ord-res-11-invars.intros ord-res-10-invars.intros simp-all*)

lemma *mempty-in-fimage-iefac*[simp]: $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \cdot \mid N \longleftrightarrow \{\#\} \mid \in \mid N$
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 $\langle \text{ord-res-10-invars } N (U_{er}, \mathcal{F}, \Gamma) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma)$ *rule*: *ord-res-10-invars.cases*)

case *invars*: 1

note Γ -*sorted* = $\langle \text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma \rangle$

note Γ -*lower* = $\langle \text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er})) \rangle$

have *trail-consistent* Γ

using Γ -*sorted* *trail-consistent-if-sorted-wrt-atoms* **by** *metis*

show *?thesis*

using *step unfolding* $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$

proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ s' *rule*: *ord-res-11.cases*)

case *step-hyps*: $(\text{decide-neg } A \Gamma')$

have

A-in: $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and**

A-gt: $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A$ **and**

A-lt: $\forall y \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$

$y \neq A \longrightarrow (\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t y) \longrightarrow A \prec_t y$

using $\langle \text{linorder-trm.is-least-in-fset } - A \rangle$

unfolding *atomize-conj linorder-trm.is-least-in-filter-iff*

by *argo*

have *trail-atms* $\Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$

unfolding $\langle \Gamma' = (\text{Neg } A, -) \# \Gamma \rangle$ **by** *simp*

show *?thesis*

unfolding $\langle s' = (-, -, -) \rangle$

proof (*intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI*)

show *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$

unfolding $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ *sorted-wrt.simps*

proof (*intro conjI ballI*)

fix $Ln :: 'f \text{ gliteral} \times 'f \text{ gclause option}$

assume $Ln \in \text{set } \Gamma$

```

hence  $atm\text{-of } (fst Ln) \in trail\text{-atms } \Gamma$ 
  by (simp add: fset-trail-atms)

thus  $atm\text{-of } (fst Ln) \prec_t atm\text{-of } (fst (Neg A, None))$ 
  unfolding prod.sel literal.sel
  using A-gt by metis
next
  show sorted-wrt  $(\lambda x y. atm\text{-of } (fst y) \prec_t atm\text{-of } (fst x)) \Gamma$ 
  using  $\Gamma\text{-sorted}$  .
qed

show  $\forall Ln \in set \Gamma'. \forall C. snd Ln = Some C \longrightarrow ord\text{-res.is-strictly-maximal-lit}$ 
(fst Ln) C
  unfolding  $\langle \Gamma' = (-, None) \# \Gamma \rangle$ 
  using invars by simp

show linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-cls  $(N \cup U_{er})$ )
  unfolding  $\langle trail\text{-atms } \Gamma' = fin\text{sert } A (trail\text{-atms } \Gamma) \rangle fin\text{sert.rep-eq}$ 
proof (intro linorder-trm.is-lower-set-insertI ballI impI)
  show  $A \in atms\text{-of-cls } (N \cup U_{er})$ 
  using A-in .
next
  fix  $w :: 'f gterm$ 
  assume  $w \in atms\text{-of-cls } (N \cup U_{er})$  and  $w \prec_t A$ 
  thus  $w \in trail\text{-atms } \Gamma$ 
  by (metis A-lt  $\Gamma\text{-lower linorder-trm.dual-order.asym linorder-trm.neq-iff}$ 
linorder-trm.not-in-lower-setI)
next
  show linorder-trm.is-lower-fset (trail-atms  $\Gamma$ ) (atms-of-cls  $(N \cup U_{er})$ )
  using  $\Gamma\text{-lower}$  .
qed

{
  fix
     $Ln :: 'f gliteral \times 'f gclause\ option$  and
     $\Gamma'' :: ('f gliteral \times 'f gclause\ option) list$ 

  assume  $\Gamma' = Ln \# \Gamma''$ 

  have  $snd Ln = None$ 
  using  $\langle \Gamma' = Ln \# \Gamma'' \rangle step\text{-hyps}$  by auto

  moreover have  $\neg (\exists C \in iefac \mathcal{F} \mid (N \cup U_{er}). trail\text{-false-cls } ((Neg A,$ 
None)  $\# \Gamma) C)$ 
  proof (rule notI , elim bexE)
  fix  $C :: 'f gclause$ 
  assume
     $C\text{-in}: C \in iefac \mathcal{F} \mid (N \cup U_{er})$  and
     $C\text{-false}: trail\text{-false-cls } ((Neg A, None) \# \Gamma) C$ 

```

```

have clause-could-propagate  $\Gamma$   $C$  (Pos A)
  unfolding clause-could-propagate-def
proof (intro conjI)
  show  $\neg$  trail-defined-lit  $\Gamma$  (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  $\neg$  trail-false-cls  $\Gamma$   $C$ 
    using step-hyps C-in by metis

  thus Pos A  $\in\#$   $C$ 
    using C-false by (metis subtrail-falseI uminus-Neg)
next

  fix  $L :: 'f$  gliteral
  assume L-in:  $L \in\#$   $C$  and L-neq:  $L \neq$  Pos A

  have trail-false-lit ((Neg A, None)  $\#$   $\Gamma$ )  $L$ 
    using C-false L-in unfolding trail-false-cls-def by metis

  hence  $L \in$  fst ' set  $\Gamma$ 
    unfolding trail-false-lit-def
    using L-neq
    by (cases L) simp-all

  hence trail-defined-lit  $\Gamma$   $L$ 
    unfolding trail-defined-lit-def by argo

  hence atm-of L  $\in$  trail-atms  $\Gamma$ 
    unfolding trail-defined-lit-iff-trail-defined-atm .

  moreover have  $\forall A_1 \in$  trail-atms  $\Gamma$ .  $A_1 \prec_t$   $A$ 
    using step-hyps unfolding linorder-trm.is-least-in-filter-iff by argo

  ultimately have atm-of L  $\prec_t$   $A$ 
    by metis

  hence  $L \preceq_i$  Pos A
    by (cases L) simp-all

  thus  $\neg$  Pos A  $\prec_i$   $L$ 
    by order
qed
next
show trail-false-cls  $\Gamma$   $\{\#K \in\#$   $C$ .  $K \neq$  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 F |↑| (N |∪| Uer)).
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 F |↑| (N |∪| Uer)) (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 F |↑| (N |∪| Uer)
  using ⟨Γ' = Ln # Γ''⟩ ⟨Γ' = (Neg A, None) # Γ⟩ by force
}

C) have ¬ (∃ C |∈| iefac F |↑| (N |∪| Uer)). trail-false-cls ((Neg A, None) # Γ)
proof (intro notI , elim bexE)
  fix C :: 'f gclause
  assume
    C-in: C |∈| iefac F |↑| (N |∪| Uer) 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  $\neg$  trail-false-cls  $\Gamma$  C
  using C-in step-hyps by metis

thus trail-false-cls  $\Gamma$  {#K  $\in$  # C. K  $\neq$  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  $\in$  # C
    by (metis C-false  $\langle \neg$  trail-false-cls  $\Gamma$  C  $\rangle$  subtrail-falseI uminus-Neg)
next
fix L
  assume L  $\in$  # C and L  $\neq$  Pos A
  hence atm-of L | $\in$ | trail-atms  $\Gamma$ 
    using  $\langle$  trail-false-cls  $\Gamma$  {#K  $\in$  # C. K  $\neq$  Pos A# $\rangle$ 
  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  $\prec_t$  A
    using A-gt by metis
  hence L  $\prec_l$  Pos A
    by (cases L) simp-all
  thus  $\neg$  Pos A  $\prec_l$  L
    by order
qed
qed

moreover have  $\neg$  clause-could-propagate  $\Gamma$  C (Pos A)
  using C-in step-hyps by metis

ultimately show False
  by contradiction
qed

thus  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
 $\neg (\exists C | \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } (Ln \# \Gamma_0) C)$ 
  unfolding  $\langle \Gamma' = (-, \text{None}) \# \Gamma \rangle$ 
  by (metis suffixI trail-false-cls-if-trail-false-suffix)

show {#} | $\in$ | N | $\cup$ | Uer  $\implies \Gamma' = []$ 
using bex-trail-false-cls-simp more-invars(3) step-hyps(3) trail-false-cls-mempty
by blast
next
show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{atms-of-cls } C \mid \subseteq \mid \text{atms-of-cls } (N \mid \cup \mid U_{er})$ 
  by simp
next

```



```

show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{trail-false-cls } \Gamma' C$ 
  by simp
next
show  $\text{atms-of-cls } U_{er} \sqsubseteq \text{atms-of-cls } N$ 
  using more-invars by argo
next
show  $\forall C \in \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L C$ 
  using more-invars by argo
qed
next
case step-hyps: (decide-pos  $A C \Gamma' \mathcal{F}'$ )

have
  A-in:  $A \in \text{atms-of-cls } (N \cup U_{er})$  and
  A-gt:  $\forall A_1 \in \text{trail-atms } \Gamma. A_1 \prec_t A$  and
  A-lt:  $\forall y \in \text{atms-of-cls } (N \cup U_{er}).$ 
   $y \neq A \implies (\forall A_1 \in \text{trail-atms } \Gamma. A_1 \prec_t y) \implies A \prec_t y$ 
  using  $\langle \text{linorder-trm.is-least-in-fset} - A \rangle$ 
  unfolding atomize-conj linorder-trm.is-least-in-filter-iff
  by argo

have  $\text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma)$ 
  unfolding  $\langle \Gamma' = (\text{Pos } A, -) \# \Gamma \rangle$  by simp

show ?thesis
  unfolding  $\langle s' = (-, -, -) \rangle$ 
proof (intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI)
  show  $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma'$ 
  unfolding  $\langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$  sorted-wrt.simps
proof (intro conjI ballI)
  fix  $L_n :: 'f \text{ gliteral} \times 'f \text{ gclause option}$ 
  assume  $L_n \in \text{set } \Gamma$ 

  hence  $\text{atm-of } (fst L_n) \in \text{trail-atms } \Gamma$ 
  by (simp add: fset-trail-atms)

  thus  $\text{atm-of } (fst L_n) \prec_t \text{atm-of } (fst (\text{Pos } A, \text{None}))$ 
  unfolding prod.sel literal.sel
  using A-gt by metis
next
  show  $\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma$ 
  using  $\Gamma\text{-sorted}$  .
qed

show  $\forall L_n \in \text{set } \Gamma'. \forall C. \text{snd } L_n = \text{Some } C \implies \text{ord-res.is-strictly-maximal-lit}$ 
(fst  $L_n$ )  $C$ 
  unfolding  $\langle \Gamma' = (-, \text{None}) \# \Gamma \rangle$ 
  using invars by simp

```

```

show linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-cls ( $N \mid \cup \mid U_{er}$ ))
  unfolding  $\langle \text{trail-atms } \Gamma' = \text{finsert } A \text{ (trail-atms } \Gamma) \rangle \text{finsert.rep-eq}$ 
proof (intro linorder-trm.is-lower-set-insertI ballI impI)
  show  $A \mid \in \mid \text{atms-of-cls} (N \mid \cup \mid U_{er})$ 
    using A-in .
next
  fix  $w :: 'f \text{ gterm}$ 
  assume  $w \mid \in \mid \text{atms-of-cls} (N \mid \cup \mid U_{er})$  and  $w \prec_t A$ 
  thus  $w \mid \in \mid \text{trail-atms } \Gamma$ 
    by (metis A-lt  $\Gamma$ -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  $\Gamma$ ) (atms-of-cls ( $N \mid \cup \mid U_{er}$ ))
    using  $\Gamma$ -lower .
qed

{
  fix  $Ln$  and  $\Gamma''$ 
  assume  $\Gamma' = Ln \# \Gamma''$ 

  have snd  $Ln = None$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle$  step-hyps by auto

  moreover have  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } ((\text{Pos } A,$ 
None) \# \Gamma) C)
  proof (rule notI , elim bexE)
    fix  $D :: 'f \text{ gclause}$ 
    assume D-in:  $D \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 

    hence  $D = \text{efac } C \vee D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 
      unfolding  $\langle \mathcal{F}' = (\text{if } \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C \text{ then } \mathcal{F}$ 
else finsert } C \mathcal{F}) \rangle
      by (smt (z3) fimage-iff finsert-iff iefac-def)

    hence  $\neg \text{trail-false-cls } \Gamma' D$ 
    proof (elim disjE)
      assume  $D = \text{efac } C$ 

      hence  $\text{trail-false-cls } \Gamma' D \longleftrightarrow \text{trail-false-cls } \Gamma' C$ 
        by (simp add: trail-false-cls-def)

      moreover have  $\neg \text{trail-false-cls } \Gamma' C$ 
        using step-hyps unfolding linorder-cls.is-least-in-ffilter-iff by metis

      ultimately show  $\neg \text{trail-false-cls } \Gamma' D$ 
        by argo
    next
    assume  $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$ 
    thus  $\neg \text{trail-false-cls } \Gamma' D$ 

```

```

    using step-hyps by metis
  qed

  moreover assume trail-false-cls ((Pos A, None) #  $\Gamma$ ) D

  ultimately show False
    unfolding  $\langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$ 
    by contradiction
  qed

  ultimately show snd Ln  $\neq$  None  $\longleftrightarrow$ 
    ( $\exists C | \in | \text{iefac } \mathcal{F}' | \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C$ )
    unfolding  $\langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$  by argo

  show snd Ln  $\neq$  None  $\implies$  is-pos (fst Ln)
    using  $\langle \text{snd } Ln = \text{None} \rangle$  by argo

  have  $\neg$  fBex (iefac  $\mathcal{F}' | \uparrow (N \mid \cup \mid U_{er})$ ) (trail-false-cls  $\Gamma$ )
    using step-hyps by argo

  hence  $\forall x \in \text{set } \Gamma. \text{snd } x = \text{None}$ 
    using invars by (metis list.set-cases)

  thus  $\forall x \in \text{set } \Gamma''. \text{snd } x = \text{None}$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$  by simp

  show  $\bigwedge C. \text{snd } Ln = \text{Some } C \implies \text{clause-could-propagate } \Gamma'' C$  (fst Ln)
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$  by force

  show  $\bigwedge D. \text{snd } Ln = \text{Some } D \implies D | \in | \text{iefac } \mathcal{F}' | \uparrow (N \mid \cup \mid U_{er})$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \rangle$  by force
}

show  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
 $\neg (\exists C | \in | \text{iefac } \mathcal{F}' | \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } (Ln \# \Gamma_0) C)$ 
  using step-hyps
  unfolding bex-trail-false-cls-simp
  by (meson suffixI trail-false-cls-if-trail-false-suffix)

show  $\{\#\} | \in | N \mid \cup \mid U_{er} \implies \Gamma' = []$ 
  by (meson bex-trail-false-cls-simp step-hyps(3) trail-false-cls-mempty)
next
show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{atms-of-cls } C | \subseteq | \text{atms-of-cls } (N \mid \cup \mid U_{er})$ 
  by simp
next
show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{trail-false-cls } \Gamma' C$ 
  by simp
next
show atms-of-cls  $U_{er} | \subseteq | \text{atms-of-cls } N$ 

```

```

    using more-invars by argo
next
  have  $\exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
    using step-hyps
    by (metis (no-types, lifting) clause-could-propagate-def
        linorder-cls.is-least-in-ffilter-iff literal.discI(1))

  thus  $\forall C | \in | \mathcal{F}' . \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
    using more-invars  $\langle \mathcal{F}' = (\text{if - then } \mathcal{F} \text{ else fininsert } C \ \mathcal{F}) \rangle$  by simp
qed
next
case step-hyps: (propagate A C  $\Gamma'$   $\mathcal{F}'$ )

have
  A-in:  $A | \in | \text{atms-of-clss } (N \cup | U_{er})$  and
  A-gt:  $\forall A_1 | \in | \text{trail-atms } \Gamma . A_1 \prec_t A$  and
  A-lt:  $\forall y | \in | \text{atms-of-clss } (N \cup | U_{er}) .$ 
 $y \neq A \longrightarrow (\forall A_1 | \in | \text{trail-atms } \Gamma . A_1 \prec_t y) \longrightarrow A \prec_t y$ 
  using  $\langle \text{linorder-trm.is-least-in-fset - } A \rangle$ 
  unfolding atomize-conj linorder-trm.is-least-in-ffilter-iff
  by argo

have
  C-in:  $C | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup | U_{er})$  and
  C-prop: clause-could-propagate  $\Gamma \ C \ (\text{Pos } A)$  and
  C-lt:  $\forall D | \in | \text{iefac } \mathcal{F} \ | \uparrow (N \cup | U_{er}) .$ 
 $D \neq C \longrightarrow \text{clause-could-propagate } \Gamma \ D \ (\text{Pos } A) \longrightarrow C \prec_c D$ 
  using  $\langle \text{linorder-cls.is-least-in-fset - } C \rangle$ 
  unfolding atomize-conj linorder-cls.is-least-in-ffilter-iff by argo

have trail-atms  $\Gamma' = \text{fininsert } A \ (\text{trail-atms } \Gamma)$ 
  unfolding  $\langle \Gamma' = (\text{Pos } A, -) \# \Gamma \rangle$  by simp

show ?thesis
  unfolding  $\langle s' = (-, -, -) \rangle$ 
proof (intro ord-res-11-invars.intros ord-res-10-invars.intros allI impI conjI)
  show sorted-wrt  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma'$ 
    unfolding  $\langle \Gamma' = (\text{Pos } A, -) \# \Gamma \rangle$  sorted-wrt.simps
  proof (intro conjI ballI)
    fix Ln :: 'f gliteral  $\times$  'f gclause option
    assume Ln  $\in$  set  $\Gamma$ 

    hence atm-of (fst Ln)  $| \in | \text{trail-atms } \Gamma$ 
      by (simp add: fset-trail-atms)

    thus atm-of (fst Ln)  $\prec_t \text{atm-of } (\text{fst } (\text{Pos } A, \text{Some } (\text{efac } C)))$ 
      unfolding prod.sel literal.sel
      using A-gt by metis
  next

```

```

show sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma$ 
  using  $\Gamma$ -sorted .
qed

show  $\forall Ln \in \text{set } \Gamma'. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{ord-res.is-strictly-maximal-lit}$ 
( $\text{fst } Ln$ )  $C$ 
  unfolding  $\langle \Gamma' = (\text{Pos } A, \text{Some } (efac C)) \# \Gamma \rangle$ 
  using invars step-hyps
  by (simp add: clause-could-propagate-def greatest-literal-in-efacI
    linorder-cls.is-least-in-filter-iff)

show linorder-trm.is-lower-fset (trail-atms  $\Gamma'$ ) (atms-of-clss ( $N \mid \cup \mid U_{er}$ ))
  unfolding  $\langle \text{trail-atms } \Gamma' = \text{finsert } A (\text{trail-atms } \Gamma) \rangle$  finsert.rep-eq
proof (intro linorder-trm.is-lower-set-insertI ballI impI)
  show  $A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ 
    using A-in .
next
  fix  $w :: 'f \text{ gterm}$ 
  assume  $w \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$  and  $w \prec_t A$ 
  thus  $w \mid \in \mid \text{trail-atms } \Gamma$ 
    by (metis A-lt  $\Gamma$ -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  $\Gamma$ ) (atms-of-clss ( $N \mid \cup \mid U_{er}$ ))
    using  $\Gamma$ -lower .
qed

{
  fix  $Ln$  and  $\Gamma''$ 
  assume  $\Gamma' = Ln \# \Gamma''$ 
  hence  $Ln = (\text{Pos } A, \text{Some } (efac C))$  and  $\Gamma'' = \Gamma$ 
    using  $\langle \Gamma' = (\text{Pos } A, \text{Some } (efac C)) \# \Gamma \rangle$  by simp-all

  obtain  $D$  where D-in:  $D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$  and D-false:
trail-false-cls  $\Gamma' D$ 
    using step-hyps by metis

  have  $(\exists C \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C)$ 
  proof (rule bexI)
    show trail-false-cls  $\Gamma' D$ 
      using D-false .
  next
    have  $\neg \text{trail-false-lit } \Gamma' (\text{Pos } A)$ 
  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  $\text{Pos } A \in \# C$ 
    using C-prop
    unfolding clause-could-propagate-def linorder-lit.is-maximal-in-mset-iff

```

```

    by argo

    ultimately have  $\neg$  trail-false-cls  $\Gamma' C$ 
      unfolding trail-false-cls-def by metis

    hence  $D \neq C$ 
      using D-false by metis

    thus  $D \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ 
      unfolding  $\langle \mathcal{F}' = (\text{if ord-res.is-strictly-maximal-lit } (Pos A) C \text{ then } \mathcal{F}$ 
    else fininsert  $C \mathcal{F}) \rangle$ 
      using D-in iefac-def by auto
    qed

    moreover have  $\text{snd } Ln \neq None$ 
      using  $\langle \Gamma' = Ln \# \Gamma'' \rangle$  step-hyps by auto

    ultimately show  $\text{snd } Ln \neq None \longleftrightarrow$ 
       $(\exists C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C)$ 
      by argo

    show  $\text{snd } Ln \neq None \implies \text{is-pos } (\text{fst } Ln)$ 
      using  $\langle Ln = (Pos A, \text{Some } (efac C)) \rangle$  by auto

    show  $\forall x \in \text{set } \Gamma''. \text{snd } x = None$ 
      unfolding  $\langle \Gamma'' = \Gamma \rangle$ 
      using invars by (meson list.set-cases step-hyps)

    have clause-could-propagate  $\Gamma (efac C) (Pos A)$ 
      using C-prop clause-could-propagate-efac by metis

    thus  $\bigwedge C. \text{snd } Ln = \text{Some } C \implies \text{clause-could-propagate } \Gamma'' C (\text{fst } Ln)$ 
      using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (Pos A, \text{Some } (efac C)) \# \Gamma \rangle$ 
      by force

    have  $efac C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ 
    proof (cases ord-res.is-strictly-maximal-lit  $(Pos A) C$ )
      case True
        thus ?thesis
          unfolding  $\langle \mathcal{F}' = - \rangle$ 
          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  $\langle \mathcal{F}' = - \rangle$ 
            using C-in
    by (smt (z3) fimage-iff fininsert-iff iefac-def nex-strictly-maximal-pos-lit-if-neq-efac

```

```

      obtains-positive-greatest-lit-if-efac-not-ident)
    qed

  thus  $\bigwedge D. \text{snd } Ln = \text{Some } D \implies D \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er})$ 
    using  $\langle \Gamma' = Ln \# \Gamma'' \rangle \langle \Gamma' = (\text{Pos } A, \text{Some } (efac \ C)) \# \Gamma \rangle$  by force
  }

  show  $\bigwedge \Gamma_1 Ln \Gamma_0. \Gamma' = \Gamma_1 @ Ln \# \Gamma_0 \implies \text{snd } Ln = \text{None} \implies$ 
 $\neg (\exists C \in |iefac \mathcal{F}'|^{\uparrow} (N \cup U_{er}). \text{trail-false-cls } (Ln \# \Gamma_0) \ C)$ 
    unfolding  $\langle \Gamma' = (-, \text{Some } -) \# \Gamma \rangle$ 
    using invars
    unfolding be-trail-false-cls-simp
    by (metis list.inject not-None-eq split-pairs suffix-Cons suffix-def)

  show  $\{\#\} \in |N \cup U_{er} \implies \Gamma' = []$ 
    by (meson be-trail-false-cls-simp step-hyps(3) trail-false-cls-mempty)
  next
  show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{atms-of-cls } C \subseteq | \text{atms-of-clss } (N \cup U_{er})$ 
    by simp
  next
  show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{trail-false-cls } \Gamma' \ C$ 
    by simp
  next
  show  $\text{atms-of-clss } U_{er} \subseteq | \text{atms-of-clss } N$ 
    using more-invars by argo
  next
  have  $\exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
    using step-hyps
    by (metis (no-types, lifting) clause-could-propagate-def
      linorder-cls.is-least-in-ffilter-iff literal.discI(1))

  thus  $\forall C \in |\mathcal{F}'|. \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
    using more-invars  $\langle \mathcal{F}' = (\text{if - then } \mathcal{F} \text{ else } \text{finsert } C \ \mathcal{F}) \rangle$  by simp
  qed
  next
  case step-hyps: (conflict D)
  show ?thesis
    unfolding  $\langle s' = - \rangle$ 
  proof (intro ord-res-11-invars.intros allI impI)
  show ord-res-10-invars  $N (U_{er}, \mathcal{F}, \Gamma)$ 
    using more-invars by argo
  next
  show  $\{\#\} \in |N \cup U_{er} \implies \Gamma = []$ 
    using more-invars by argo
  next
  show  $\bigwedge C. \text{Some } D = \text{Some } C \implies \text{atms-of-cls } C \subseteq | \text{atms-of-clss } (N \cup U_{er})$ 
    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  $\wedge C. \text{Some } D = \text{Some } C \implies \text{trail-false-cls } \Gamma \ C$ 
    using  $\langle \text{linorder-cls.is-least-in-fset} - D \rangle$ 
    unfolding  $\text{linorder-cls.is-least-in-filter-iff}$ 
    by simp
next
  show  $\text{atms-of-clss } U_{er} \mid\subseteq\mid \text{atms-of-clss } N$ 
    using more-invars by argo
next
  show  $\forall C \in \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
    using more-invars by argo
qed
next
case step-hyps: (skip  $L \ C \ n \ \Gamma'$ )

have  $\wedge xs. \text{fset } (\text{trail-atms } xs) = \text{atm-of } \langle \text{fst } \langle \text{set } xs \rangle$ 
  unfolding fset-trail-atms ..
also have  $\wedge xs. \text{atm-of } \langle \text{fst } \langle \text{set } xs \rangle = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \ xs)$ 
  by (simp add: image-comp)
finally have  $\wedge xs. \text{fset } (\text{trail-atms } xs) = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \ xs) .$ 

show ?thesis
  unfolding  $\langle s' = (U_{er}, \mathcal{F}, \Gamma', \text{Some } C) \rangle$ 
  proof (intro ord-res-11-invars.intros ord-res-10-invars.intros ballI allI impI
conjI)
    show sorted-wrt  $(\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma'$ 
      using invars unfolding  $\langle \Gamma = (L, n) \# \Gamma' \rangle$  by simp
    next
      fix  $Ln :: 'f \text{ gliteral} \times 'f \text{ gclause option}$  and  $C :: 'f \text{ gclause}$ 
      assume  $Ln \in \text{set } \Gamma'$  and  $\text{snd } Ln = \text{Some } C$ 
      then show ord-res.is-strictly-maximal-lit  $(\text{fst } Ln) \ C$ 
        using invars unfolding  $\langle \Gamma = (L, n) \# \Gamma' \rangle$  by simp
    next
      show linorder-trm.is-lower-fset  $(\text{trail-atms } \Gamma') \ (\text{atms-of-clss } (N \mid\cup\mid U_{er}))$ 
        unfolding  $\langle \text{fset } (\text{trail-atms } \Gamma') = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma') \rangle$ 
      proof (rule linorder-trm.sorted-and-lower-set-appendD-right(2))
        have sorted-wrt  $(\lambda x \ y. y \prec_t x) \ (\text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma)$ 
          using  $\Gamma$ -sorted by (simp add: sorted-wrt-map)

        thus sorted-wrt  $(\lambda x \ y. y \prec_t x) \ ([\text{atm-of } L] \ @ \ \text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma')$ 
          unfolding  $\langle \Gamma = - \# \Gamma' \rangle$  by simp
      next
        have  $\text{set } ([\text{atm-of } L] \ @ \ \text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma') = \text{fset } (\text{trail-atms } \Gamma)$ 
          unfolding append-Cons append-Nil list.set
        unfolding  $\langle \text{fset } (\text{trail-atms } \Gamma') = \text{set } (\text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma') \rangle$  [symmetric]
        unfolding  $\langle \Gamma = - \# \Gamma' \rangle$  trail-atms.simps prod.sel
        by simp

      thus linorder-trm.is-lower-set  $(\text{set } ([\text{atm-of } L] \ @ \ \text{map } (\text{atm-of } \circ \text{fst}) \ \Gamma'))$ 

```



```

      (fset (atms-of-clss (N |∪| Uer)))
      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 |∪| Uer). 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 |∪| Uer)
    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
  Γ1 Γ0 :: ('f gliteral × 'f gclause option) list
  assume Γ' = Γ1 @ Ln # Γ0 and snd Ln = None
  thus ¬ (∃ C |∈| iefac  $\mathcal{F}$  |' (N |∪| Uer). trail-false-cls (Ln # Γ0) C)
    by (metis append-Cons invars(5) step-hyps(1))
next
show {#} |∈| N |∪| Uer ⇒ Γ' = []
  using step-hyps(1) more-invars(3) by fastforce
next
show ∧D. Some C = Some D ⇒ atms-of-clss D |⊆| atms-of-clss (N |∪|
Uer)
  using more-invars(4) step-hyps(2) by presburger
next
show ∧D. Some C = Some D ⇒ trail-false-cls Γ' D
  using more-invars ⟨C = Some C⟩ ⟨Γ = (L, n) # Γ'⟩ ⟨← L # C⟩
  using subtrail-falseI by auto
next

```

```

show atms-of-clss  $U_{er} \sqsubseteq$  atms-of-clss  $N$ 
  using more-invars by argo
next
show  $\bigwedge C. C \in \mathcal{F} \implies \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C$ 
  using more-invars by metis
qed
next
case step-hyps: (resolution  $L \ D \ \Gamma' \ C$ )

have D-max-lit: ord-res.is-strictly-maximal-lit  $L \ D$ 
  using invars  $\langle \Gamma = (L, \text{Some } D) \# \Gamma' \rangle$  by simp

show ?thesis
  unfolding  $\langle s' = - \rangle$ 
proof (intro ord-res-11-invars.intros allI impI)
  show ord-res-10-invars  $N \ (U_{er}, \mathcal{F}, \Gamma)$ 
    using more-invars by argo
next
show  $\{\#\} \in N \ \cup U_{er} \implies \Gamma = []$ 
  using more-invars by argo
next
have  $D \in \text{iefac } \mathcal{F} \ \uparrow \ (N \ \cup \ U_{er})$ 
  using invars  $\langle \Gamma = (L, \text{Some } D) \# \Gamma' \rangle$ 
  by (metis snd-conv)

hence atms-of-cls  $D \sqsubseteq$  atms-of-clss  $(N \ \cup \ U_{er})$ 
  by (metis atms-of-clss-fimage-iefac atms-of-clss-finsert finsert-absorb fun-union-upper1)

moreover have atms-of-cls  $C \sqsubseteq$  atms-of-clss  $(N \ \cup \ U_{er})$ 
  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  $\bigwedge E. \text{Some } (\text{remove1-mset } (- \ L) \ C + \text{remove1-mset } L \ D) = \text{Some } E \implies$ 
  atms-of-cls  $E \sqsubseteq$  atms-of-clss  $(N \ \cup \ U_{er})$ 
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  $E :: 'f \text{ gclause}$ 
assume  $\text{Some } (\text{remove1-mset } (- \ L) \ C + \text{remove1-mset } L \ D) = \text{Some } E$ 

hence  $E = \text{remove1-mset } (- \ L) \ C + \text{remove1-mset } L \ D$ 
  by simp

show trail-false-cls  $\Gamma \ E$ 
  unfolding trail-false-cls-def

```

```

proof (intro ballI)
  fix  $K :: 'f \text{ gliteral}$ 
  assume  $K \in\# E$ 

  hence  $K \in\# \text{remove1-mset } (- L) C \vee K \in\# \text{remove1-mset } L D$ 
  unfolding  $\langle E = \rightarrow \rangle$  by simp

  thus trail-false-lit  $\Gamma K$ 
  proof (elim disjE)
    assume  $K \in\# \text{remove1-mset } (- L) C$ 

    moreover have trail-false-cls  $\Gamma C$ 
      using more-invars  $\langle C = \text{Some } C \rangle$  by metis

    ultimately show trail-false-lit  $\Gamma K$ 
      unfolding trail-false-cls-def
      by (metis in-diffD)
  next
    assume  $K\text{-in}: K \in\# \text{remove1-mset } L D$ 

    have clause-could-propagate  $\Gamma' D L$ 
      using invars  $\langle \Gamma = (L, \text{Some } D) \# \Gamma' \rangle$  by simp

    hence trail-false-cls  $\Gamma' \{\#K \in\# D. K \neq L\# \}$ 
      by (simp only: clause-could-propagate-def)

    hence trail-false-cls  $\Gamma \{\#K \in\# D. K \neq L\# \}$ 
      unfolding  $\langle \Gamma = (L, \text{Some } D) \# \Gamma' \rangle$ 
      by (simp add: trail-false-cls-def trail-false-lit-def)

    moreover have  $K \in\# \{\#K \in\# D. K \neq L\# \}$ 
      using D-max-lit K-in in-diffD linorder-lit.is-greatest-in-mset-iff by
fastforce

    ultimately show trail-false-lit  $\Gamma K$ 
      unfolding trail-false-cls-def by metis
  qed
qed
next
  show atms-of-cls  $U_{er} \sqsubseteq \text{atms-of-cls } N$ 
    using more-invars by argo
next
  show  $\forall C \in \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L C$ 
    using more-invars by metis
qed
next
  case step-hyps: (backtrack L  $\Gamma'$  C)

  have  $\{\#\} \not\subseteq N \cup U_{er}$ 

```

using *more-invars step-hyps(3)* **by** *fastforce*

hence $\{\#\} \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
unfolding *mempty-in-fimage-iefac* .

hence $\neg(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma \ C)$
using *invars $\langle \Gamma = - \# \Gamma' \rangle$*
by (*metis (no-types, opaque-lifting) split-pairs*)

hence $\neg(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' \ C)$
by (*metis append.simps(1) step-hyps(3) suffix-ConsD suffix-def trail-false-cls-if-trail-false-suffix*)

moreover have $\neg \text{trail-false-cls } \Gamma' \ C$
by (*metis $\langle \text{trail-consistent } \Gamma \rangle \text{fst-conv list.distinct(1) list.inject step-hyps(3,4) trail-consistent.simps trail-defined-lit-def trail-false-cls-def trail-false-lit-def uminus-lit-swap}$)*

ultimately have $\neg(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid \text{finsert } C \ U_{er}). \text{trail-false-cls } \Gamma'$
C) **by** (*simp add: iefac-def trail-false-cls-def*)

have $\bigwedge xs. \text{fset } (\text{trail-atms } xs) = \text{atm-of } \text{'fst ' set } xs$
unfolding *fset-trail-atms ..*
also have $\bigwedge xs. \text{atm-of } \text{'fst ' set } xs = \text{set } (\text{map } (\text{atm-of } o \text{fst}) \ xs)$
by (*simp add: image-comp*)
finally have $\bigwedge xs. \text{fset } (\text{trail-atms } xs) = \text{set } (\text{map } (\text{atm-of } o \text{fst}) \ xs)$.

have $\text{atms-of-cls } C \mid \subseteq \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$
using *more-invars $\langle C = \text{Some } C \rangle$* **by** *metis*

hence $\text{atms-of-clss } (N \mid \cup \mid \text{finsert } C \ U_{er}) = \text{atms-of-clss } (N \mid \cup \mid U_{er})$
by *auto*

show *?thesis*
unfolding $\langle s' = (\text{finsert } C \ U_{er}, \mathcal{F}, \Gamma', \text{None}) \rangle$
proof (*intro ord-res-11-invars.intros ord-res-10-invars.intros ballI allI impI conjI*)

show *sorted-wrt* $(\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma'$
using *invars unfolding $\langle \Gamma = - \# \Gamma' \rangle$* **by** *simp*

next
fix *Ln* :: 'f gliteral \times 'f gclause option **and** *C* :: 'f gclause
assume *Ln* \in *set* Γ' **and** *snd* *Ln* = *Some C*
then show *ord-res.is-strictly-maximal-lit (fst Ln) C*
using *invars unfolding $\langle \Gamma = - \# \Gamma' \rangle$* **by** *simp*

next
show *linorder-trm.is-lower-fset (trail-atms Γ') (atms-of-clss (N $\mid \cup \mid$ finsert C U_{er}))*
unfolding $\langle \text{fset } (\text{trail-atms } \Gamma') = \text{set } (\text{map } (\text{atm-of } o \text{fst}) \ \Gamma') \rangle$

```

unfolding ⟨atms-of-clss (N |∪| finsert C Uer) = atms-of-clss (N |∪| Uer)⟩
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 ∘ 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 |∪| Uer))))
    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 ℱ |' (N |∪| finsert C Uer).
trail-false-clc Γ' C)
    using ⟨¬(∃ C|∈|iefac ℱ |' (N |∪| finsert C Uer). trail-false-clc 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 ℱ |' (N |∪| finsert C Uer)
    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
  Γ1 Γ0 :: ('f gliteral × 'f gclause option) list
assume Γ' = Γ1 @ Ln # Γ0 and snd Ln = None

```

```

thus  $\neg (\exists C | \in | \text{iefac } \mathcal{F} |^q (N | \cup | \text{finsert } C U_{er}). \text{trail-false-cls } (Ln \# \Gamma_0) C)$ 
  using  $\langle \neg (\exists C | \in | \text{iefac } \mathcal{F} |^q (N | \cup | \text{finsert } C U_{er}). \text{trail-false-cls } \Gamma' C) \rangle$ 
  by (metis suffixI trail-false-cls-if-trail-false-suffix)
next
  have  $C \neq \{\#\}$ 
  using  $\langle - L \in \# C \rangle$  by force

  hence  $\{\#\} | \notin | N | \cup | \text{finsert } C U_{er}$ 
  using  $\langle \{\#\} | \notin | N | \cup | U_{er} \rangle$  by simp

  thus  $\{\#\} | \in | N | \cup | \text{finsert } C U_{er} \implies \Gamma' = []$ 
  by contradiction
next
  show  $\bigwedge D. \text{None} = \text{Some } D \implies \text{atms-of-cls } D | \subseteq | \text{atms-of-clss } (N | \cup | \text{finsert } C U_{er})$ 
  by simp
next
  show  $\bigwedge C. \text{None} = \text{Some } C \implies \text{trail-false-cls } \Gamma' C$ 
  by simp
next
  have  $\text{atms-of-cls } C | \subseteq | \text{atms-of-clss } N$ 
  by (smt (verit, ccfv-threshold)  $\langle \text{atms-of-cls } C | \subseteq | \text{atms-of-clss } (N | \cup | U_{er}) \rangle$   

atms-of-clss-def fin-mono fmember-ffUnion-iff fsubsetI funion-iff  

more-invars(6))

  thus  $\text{atms-of-clss } (\text{finsert } C U_{er}) | \subseteq | \text{atms-of-clss } N$ 
  using  $\langle \text{atms-of-clss } U_{er} | \subseteq | \text{atms-of-clss } N \rangle$  by simp
next
  show  $\bigwedge C. C | \in | \mathcal{F} \implies \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L C$ 
  using more-invars by metis
qed
qed
qed
qed

```

lemma *rtranclp-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 *step invars ord-res-11-preserves-invars*

by (*smt (verit, del-insts) rtranclp-induct*)

lemma *tranclp-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 *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 *ord-res-11-final* (N , s) **and**
invars: *ord-res-11-invars* N s

shows $\exists s'. \text{ord-res-11 } N \ s \ s'$

using *invars*

proof (cases N s rule: *ord-res-11-invars.cases*)
case *more-invars*: ($1 \ U_{er} \ \mathcal{F} \ \Gamma \ C$)
show *?thesis*
using $\langle \text{ord-res-10-invars } N \ (U_{er}, \mathcal{F}, \Gamma) \rangle$
proof (cases $N \ (U_{er}, \mathcal{F}, \Gamma)$ rule: *ord-res-10-invars.cases*)
case *invars*: 1

show *?thesis*
proof (cases C)
case *None*
show *?thesis*
proof (cases $\exists C \ | \in | \text{iefac } \mathcal{F} \ | \uparrow \ (N \ | \cup \ U_{er}). \text{trail-false-cls } \Gamma \ C$)
case *True*

then obtain C **where**
 $\text{linorder-cls.is-least-in-fset } (\text{ffilter } (\text{trail-false-cls } \Gamma) (\text{iefac } \mathcal{F} \ | \uparrow \ (N \ | \cup \ U_{er}))) \ C$
using *linorder-cls.ex-is-least-in-ffilter-iff* **by** *metis*

thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle C = \text{None} \rangle$
using *ord-res-11.conflict* **by** *metis*

next
case *no-false-cls*: *False*

hence $\exists A_2 \ | \in | \text{atms-of-cls } (N \ | \cup \ U_{er}). \forall A_1 \ | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2$
using *not-final*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle C = \text{None} \rangle$
by (smt (verit) *invars*(β) *linorder-trm.antisym-conv3* *linorder-trm.not-in-lower-setI* *ord-res-11-final.model-found*)

then obtain A **where**
 $A\text{-least: linorder-trm.is-least-in-fset } \{ | A_2 \ | \in | \text{atms-of-cls } (N \ | \cup \ U_{er}). \forall A_1 \ | \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2 \} \ A$
using *linorder-trm.ex-is-least-in-ffilter-iff* **by** *presburger*

show *?thesis*
proof (cases $\exists C \ | \in | \text{iefac } \mathcal{F} \ | \uparrow \ (N \ | \cup \ U_{er}). \text{clause-could-propagate } \Gamma \ C \ (\text{Pos } A)$)
case *True*

then obtain C where
C-least: linorder-cls.is-least-in-fset
 $\{C \mid \exists \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ clause-could-propagate } \Gamma \ C \ (Pos \ A)\}$
C
using *linorder-cls.ex-is-least-in-filter-iff* by *presburger*
show *?thesis*
proof (*cases* $\exists C \mid \exists \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } ((Pos \ A, \ None)$
 $\# \Gamma) \ C$)
case *True*
moreover have *trail-false-cls* $((Pos \ A, \ None) \# \Gamma) =$
trail-false-cls $((Pos \ A, \ Some \ (efac \ C)) \# \Gamma)$
unfolding *trail-false-cls-def trail-false-lit-def* by *simp*
ultimately show *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle \mathcal{C} = None \rangle$
using *ord-res-11.propagate[OF no-false-cls A-least C-least]*
by *metis*
next
case *False*
thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle \mathcal{C} = None \rangle$
using *ord-res-11.decide-pos[OF no-false-cls A-least C-least]*
by *metis*
qed
next
case *False*
thus *?thesis*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle \mathcal{C} = None \rangle$
using *ord-res-11.decide-neg[OF no-false-cls A-least]* by *metis*
qed
next
case $(Some \ D)$
hence *D-false: trail-false-cls* $\Gamma \ D$
using *more-invars* by *metis*
show *?thesis*
proof (*cases* $D = \{\#\}$)
case *True*
hence $\Gamma \neq []$
using *not-final*
unfolding $\langle s = (U_{er}, \mathcal{F}, \Gamma, C) \rangle \langle \mathcal{C} = Some \ D \rangle$
unfolding *ord-res-11-final.simps* by *metis*
then obtain $L \ n \ \Gamma'$ where $\Gamma = (L, \ n) \# \ \Gamma'$


```

    by (metis list.exhaust prod.exhaust)

  moreover have  $- L \notin \# D$ 
    unfolding  $\langle D = \{\#\} \rangle$  by simp

  ultimately show ?thesis
    unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \mathcal{C} = \text{Some } D \rangle$ 
    using ord-res-11.skip by metis
next
  case False

  then obtain  $L n \Gamma'$  where  $\Gamma = (L, n) \# \Gamma'$ 
    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  $- L \in \# D$ )
    case True
      show ?thesis
      proof (cases  $n$ )
        case None
          show ?thesis
          proof (cases  $- L \in \# D$ )
            case True
              thus ?thesis
              unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \Gamma = (L, n) \# \Gamma' \rangle \langle n = \text{None} \rangle \langle \mathcal{C} =$ 
                Some D  $\rangle$ 
                using ord-res-11.backtrack by metis
            next
              case False
                thus ?thesis
                using ord-res-11.skip
                unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \Gamma = (L, n) \# \Gamma' \rangle \langle n = \text{None} \rangle \langle \mathcal{C} =$ 
                  Some D  $\rangle$  by metis
          qed
        next
          case (Some C)
            thus ?thesis
            unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \Gamma = (L, n) \# \Gamma' \rangle \langle n = \text{Some } C \rangle \langle \mathcal{C} =$ 
              = Some D  $\rangle$ 
              using ord-res-11.resolution[OF  $- \langle - L \in \# D \rangle$ ] by metis
            qed
        next
          case False
            thus ?thesis
            using ord-res-11.skip
            unfolding  $\langle s = (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle \langle \Gamma = (L, n) \# \Gamma' \rangle \langle \mathcal{C} = \text{Some } D \rangle$  by
              metis
            qed
  qed

```

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**
 $\mathcal{R} = \text{ord-res-11}$ **and** $\mathcal{F} = \text{ord-res-11-final}$ **and** $\mathcal{I} = \text{ord-res-11-invars}$]
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 $(\text{ord-res-11 } N)^{**} (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } D) (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } (eres\ C\ D))$
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 $(\text{ord-res-11 } N)^{**} (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } D) (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } CD)$
proof (*rule full-run-preserves-invariant[OF run]*)
show $(\text{ord-res-11 } N)^{**} (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } D) (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } D)$
by *simp*
next
fix $x y$
assume *ground-resolution* $C x y$

hence *ord-res-11* $N (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } x) (U, \mathcal{F}, (K, \text{Some } C) \# \Gamma, \text{Some } y)$

unfolding *ground-resolution-def*

proof (*cases* $x\ C\ y$ *rule*: *ord-res.ground-resolution.cases*)

case *res-hyps*: (*ground-resolutionI* $A\ P_1'\ P_2'$)

have $K = -\ \text{Neg } A$

using *C-max-lit*

by (*metis ord-res.Uniq-strictly-maximal-lit-in-ground-cls res-hyps(6) the1-equality' uminus-Neg*)

hence $- K \in\# x$

by (*simp add*: $\langle x = \text{add-mset } (\text{Neg } A)\ P_1' \rangle$)

moreover **have** $y = \text{remove1-mset } (- K)\ x + \text{remove1-mset } K\ C$

using *res-hyps* $\langle K = -\ \text{Neg } A \rangle$ **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  $\bigwedge N1 N2. N1 = N2 \implies \text{ord-res-1-final } N2 \implies \text{ord-res-final } N1$ 

```

```

    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) ffilter 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  $N2' = \text{finsert } C' N2$ 
    using hyps by argo
qed
next
case hyps: (resolution C L D CD)
show ?thesis
proof (rule ord-res.resolution)
  show  $\{\#\} \notin N2$ 
    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  $C \in N2$ 
    using hyps is-least-false-clause-def linorder-cls.is-least-in-fset-ffilterD(1)
by blast
next
  show  $D \in N2$ 
    using hyps by argo
next
  show ord-res.ground-resolution C D CD
    using hyps by argo
next
  show  $N2' = \text{finsert } CD N2$ 
    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.

$S1 = N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f \wedge$
 $(\forall C_f \mid\in\mid U_f. \exists C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}. \text{ord-res.ground-factorizing}^{++} C C_f \wedge$
 $C_f \neq \text{efac } C_f \wedge$
 $(\text{efac } C_f \mid\in\mid U_{ef} \vee \text{is-least-false-clause } (N \mid\cup\mid U_r \mid\cup\mid U_{ef}) C)))$

lemma *ord-res-1-matches-ord-res-2-simps'*:

$\text{ord-res-1-matches-ord-res-2 } S1 (N, (U_r, U_{ef})) \longleftrightarrow$
 $(\exists U_f. S1 = N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f \wedge$
 $(\forall C_f \mid\in\mid U_f. C_f \neq \text{efac } C_f \wedge (\exists C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}. \text{ord-res.ground-factorizing}^{++}$
 $C C_f \wedge$
 $(\text{efac } C_f \mid\in\mid U_{ef} \vee \text{is-least-false-clause } (N \mid\cup\mid U_r \mid\cup\mid U_{ef}) C))))$
unfolding *ord-res-1-matches-ord-res-2.simps* **by** *metis*

lemma *ord-res-1-matches-ord-res-2-simps''*:

$\text{ord-res-1-matches-ord-res-2 } S1 (N, (U_r, U_{ef})) \longleftrightarrow$
 $(\exists U_f. S1 = N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f \wedge$
 $(\forall C_f \mid\in\mid U_f. C_f \neq \text{efac } C_f \wedge (\exists C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}. \text{ord-res.ground-factorizing}^{++}$
 $C C_f \wedge$
 $(\text{efac } C \mid\in\mid U_{ef} \vee \text{is-least-false-clause } (N \mid\cup\mid U_r \mid\cup\mid U_{ef}) C))))$
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*: $\text{ord-res-1-matches-ord-res-2 } S_1 S_2$
shows $\text{ord-res-1-final } S_1 \longleftrightarrow \text{ord-res-2-final } S_2$

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**
 $S_1\text{-def}$: $S_1 = N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f$ **and**
 $U_f\text{-spec}$: $\forall C_f \mid\in\mid U_f. \exists C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}. \text{ord-res.ground-factorizing}^{++}$
 $C C_f \wedge C_f \neq \text{efac } C_f \wedge$
 $(\text{efac } C_f \mid\in\mid U_{ef} \vee \text{is-least-false-clause } (N \mid\cup\mid U_r \mid\cup\mid U_{ef}) C)$
by *auto*

have $U_f\text{-unproductive}$: $\forall C_f \mid\in\mid U_f. \text{ord-res.production } (fset (N \mid\cup\mid U_r \mid\cup\mid U_{ef}$
 $\mid\cup\mid U_f)) C_f = \{\}$

proof (*intro ballI*)

fix C_f

assume $C_f \mid\in\mid U_f$

hence $C_f \neq \text{efac } C_f$

using $U_f\text{-spec}$ **by** *metis*

hence $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L C_f$

using *nex-strictly-maximal-pos-lit-if-neq-efac* **by** *metis*

thus $\text{ord-res.production } (fset (N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)) C_f = \{\}$

using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*

qed

have *Interp-eq*: $\bigwedge C. \text{ord-res-Interp } (fset (N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)) C =$

$ord\text{-}res\text{-}Interp (fset (N \cup U_r \cup U_{ef})) C$
using $Interp\text{-}union\text{-}unproductive[of fset (N \cup U_r \cup U_{ef}) fset U_f, folded$
 $union\text{-}fset,$
 $OF finite\text{-}fset finite\text{-}fset U_f\text{-}unproductive]$.

have $\{\#\} \in N \cup U_r \cup U_{ef} \cup U_f \longleftrightarrow \{\#\} \in N \cup U_r \cup U_{ef}$

proof (*rule iffI*)

assume $\{\#\} \in N \cup U_r \cup U_{ef} \cup U_f$

hence $\{\#\} \in N \cup U_r \cup U_{ef} \vee \{\#\} \in U_f$

by *simp*

thus $\{\#\} \in N \cup U_r \cup U_{ef}$

proof (*elim disjE*)

assume $\{\#\} \in N \cup U_r \cup U_{ef}$

thus $\{\#\} \in N \cup U_r \cup U_{ef}$

by *assumption*

next

assume $\{\#\} \in U_f$

hence $\{\#\} \neq efac \{\#\}$

using $U_f\text{-}spec[rule\text{-}format, of \{\#\}]$ **by** *metis*

hence *False*

by *simp*

thus $\{\#\} \in N \cup U_r \cup U_{ef} ..$

qed

next

assume $\{\#\} \in N \cup U_r \cup U_{ef}$

thus $\{\#\} \in N \cup U_r \cup U_{ef} \cup U_f$

by *simp*

qed

moreover have $\neg ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef} \cup U_f)) \longleftrightarrow$

$\neg ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef}))$

proof (*rule iffI; erule contrapos-nn*)

assume $ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef}))$

thus $ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef} \cup U_f))$

unfolding $ex\text{-}false\text{-}clause\text{-}def Interp\text{-}eq$ **by** *auto*

next

assume $ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef} \cup U_f))$

then obtain C **where**

$C \in N \cup U_r \cup U_{ef} \cup U_f$ **and**

$C\text{-}false: \neg ord\text{-}res\text{-}Interp (fset (N \cup U_r \cup U_{ef})) C \models C$

unfolding $ex\text{-}false\text{-}clause\text{-}def Interp\text{-}eq$ **by** *metis*

hence $C \in N \cup U_r \cup U_{ef} \vee C \in U_f$

by *simp*

thus $ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef}))$

proof (*elim disjE*)

assume $C \in N \cup U_r \cup U_{ef}$

thus $ex\text{-}false\text{-}clause (fset (N \cup U_r \cup U_{ef}))$

unfolding $ex\text{-}false\text{-}clause\text{-}def$ **using** $C\text{-}false$ **by** *metis*

next

assume $C \in U_f$
then obtain C' **where** $C' \in N \cup U_r \cup U_{ef}$ **and**
ord-res.ground-factoring⁺⁺ $C' C$ **and**
 $C \neq \text{efac } C$ **and**
 $\text{efac } C \in U_{ef} \vee \text{is-least-false-clause } (N \cup U_r \cup U_{ef}) C'$
using $U_f\text{-spec}[\text{rule-format, of } C]$ **by** *metis*
thus *ex-false-clause* ($\text{fset } (N \cup U_r \cup U_{ef})$)
proof (*elim disjE exE conjE*)
assume $\text{efac } C \in U_{ef}$

show *ex-false-clause* ($\text{fset } (N \cup U_r \cup U_{ef})$)
proof (*cases ord-res-Interp* ($\text{fset } (N \cup U_r \cup U_{ef})$)) ($\text{efac } C$) $\models \text{efac } C$
case *efac-C-true*: *True*
have $\text{efac } C \subseteq\# C$
using *efac-subset*[of C].
hence $\text{efac } C \preceq_c C$
using *subset-implies-reflclp-multp* **by** *metis*
hence *ord-res-Interp* ($\text{fset } (N \cup U_r \cup U_{ef})$) $C \models \text{efac } C$
using *efac-C-true ord-res.entailed-clause-stays-entailed* **by** *fastforce*
hence *ord-res-Interp* ($\text{fset } (N \cup U_r \cup U_{ef})$) $C \models C$
using *efac-C-true* **by** (*simp add: true-cls-def*)
with *C-false* **have** *False*
by *contradiction*
thus *?thesis ..*
next
case *False*

moreover have $\text{efac } C \in N \cup U_r \cup U_{ef}$
using $\langle \text{efac } C \in U_{ef} \rangle$ **by** *simp*

ultimately show *ex-false-clause* ($\text{fset } (N \cup U_r \cup U_{ef})$)
unfolding *ex-false-clause-def* **by** *metis*
qed
next
assume *is-least-false-clause* ($N \cup U_r \cup U_{ef}$) C'
hence $C' \in N \cup U_r \cup U_{ef}$ **and** $\neg \text{ord-res-Interp } (\text{fset } (N \cup U_r \cup U_{ef})) C' \models C'$
using *linorder-cls.is-least-in-ffilter-iff is-least-false-clause-def* **by** *simp-all*
thus *ex-false-clause* ($\text{fset } (N \cup U_r \cup U_{ef})$)
unfolding *ex-false-clause-def* **by** *metis*
qed
qed
qed

ultimately show *?thesis*
by (*simp add: S₁-def* $\langle S_2 = (N, U_r, U_{ef}) \rangle$ *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* $S_1 \wedge$ *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 $\exists C. \text{is-least-false-clause } s1 C$ *then*
The $(\text{is-least-false-clause } s1)$
else
 $\{\#\}$ *)*

lemma *forward-simulation*:
assumes *match: ord-res-1-matches-ord-res-2* $s1 s2$ **and**
step1: ord-res-1 $s1 s1'$
shows $(\exists s2'. \text{ord-res-2-step}^{++} s2 s2' \wedge \text{ord-res-1-matches-ord-res-2 } s1' s2') \vee$
 $\text{ord-res-1-matches-ord-res-2 } s1' s2 \wedge \text{ord-res-1-measure } s1' \subset\# \text{ord-res-1-measure } s1$
proof –
let
 $?match = \text{ord-res-1-matches-ord-res-2}$ **and**
 $?measure = \text{ord-res-1-measure}$ **and**
 $?order = (\subset\#)$

obtain $N U_r U_{ef} :: 'f \text{ gterm clause fset}$ **where**
 $s2\text{-def: } s2 = (N, (U_r, U_{ef}))$
by *(metis prod.exhaust)*

from *match* **obtain** U_f **where**
 $s1\text{-def: } s1 = N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f$ **and**
 $U_f\text{-spec: } \forall C_f \mid\in\mid U_f. \exists C \mid\in\mid N \mid\cup\mid U_r \mid\cup\mid U_{ef}. \text{ord-res.ground-factoring}^{++}$
 $C C_f \wedge C_f \neq \text{efac } C_f \wedge$
 $(\text{efac } C_f \mid\in\mid U_{ef} \vee \text{is-least-false-clause } (N \mid\cup\mid U_r \mid\cup\mid U_{ef}) C)$
unfolding $s2\text{-def}$ *ord-res-1-matches-ord-res-2.simps* **by** *metis*

have $U_f\text{-unproductive: } \forall C_f \mid\in\mid U_f. \text{ord-res.production } (\text{fset } (N \mid\cup\mid U_r \mid\cup\mid U_{ef} \mid\cup\mid U_f)) C_f = \{\}$
proof *(intro ballI)*
fix C_f

assume $C_f \mid \in \mid U_f$
hence $C_f \neq \text{efac } C_f$
using $U_f\text{-spec}$ **by** *metis*
hence $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C_f$
using *nex-strictly-maximal-pos-lit-if-neq-efac* **by** *metis*
thus $\text{ord-res.production } (\text{fset } (N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f)) \ C_f = \{\}$
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

have $\text{Interp-eq: } \bigwedge C. \text{ord-res-Interp } (\text{fset } (N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f)) \ C =$
 $\text{ord-res-Interp } (\text{fset } (N \mid \cup \mid U_r \mid \cup \mid U_{ef})) \ C$
using *Interp-union-unproductive*[*of fset (N |cup| U_r |cup| U_{ef}) fset U_f, folded*
union-fset,
OF finite-fset finite-fset U_f-unproductive] .

show $(\exists s2'. \text{ord-res-2-step}^{++} \ s2 \ s2' \wedge ?\text{match } s1' \ s2') \vee$
 $?\text{match } s1' \ s2 \wedge ?\text{order } (?measure \ s1') \ (?measure \ s1)$
using *step1*
proof (*cases s1 s1' rule: ord-res-1.cases*)
case (*factoring C L C'*)

have $C\text{-least-false: is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f) \ C$
using *factoring*
unfolding *is-least-false-clause-def s1-def* **by** *argo*

hence $C\text{-in: } C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f$
unfolding *is-least-false-clause-def linorder.cls.is-least-in-filter-iff s1-def* **by**
argo

hence $C\text{-in-disj: } C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef} \vee C \mid \in \mid U_f$
by *simp*

show *?thesis*
proof (*cases C' = efac C'*)
case *True*
let $?s2' = (N, (U_r, \text{finsert } C' \ U_{ef}))$

have $\text{ord-res-2-step}^{++} \ s2 \ ?s2'$
proof (*rule tranclp.r-into-trancl*)
show $\text{ord-res-2-step } s2 \ (N, U_r, \text{finsert } C' \ U_{ef})$
using *C-in-disj*
proof (*elim disjE*)
assume $C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef}$
show *?thesis*
unfolding *s2-def*
proof (*intro ord-res-2-step.intros ord-res-2.factoring*)
show $\text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef}) \ C$
using *is-least-false-clause-if-is-least-false-clause-in-union-unproductive*[
OF U_f-unproductive <C |in| N |cup| U_r |cup| U_{ef}> C-least-false]
unfolding *is-least-false-clause-def* .

```

next
  show ord-res.is-maximal-lit L C
    using  $\langle \text{ord-res.is-maximal-lit } L \ C \rangle$  .
next
  show is-pos L
    using  $\langle \text{is-pos } L \rangle$  .
next
  show  $\text{finsert } C' \ U_{ef} = \text{finsert } (\text{efac } C) \ U_{ef}$ 
    using True factoring ground-factoring-preserves-efac by metis
qed
next
assume  $C \in U_f$ 
then obtain  $x$  where
   $x \in N \cup U_r \cup U_{ef}$  and
  ord-res.ground-factoring++ x C and
   $C \neq \text{efac } C$  and
   $\text{efac } C \in U_{ef} \vee \text{is-least-false-clause } (N \cup U_r \cup U_{ef}) \ x$ 
  using Uf-spec by metis

show ?thesis
  unfolding s2-def
proof (intro ord-res-2-step.intros ord-res-2.factoring)
  have  $\langle \text{efac } C \notin U_{ef} \rangle$ 
  proof (rule notI)
    have  $\text{efac } C \preceq_c C$ 
      using efac-subset[of C] subset-implies-reflclp-multp by metis
    hence  $\text{efac } C \prec_c C$ 
      using  $\langle C \neq \text{efac } C \rangle$  by order

  moreover assume  $\text{efac } C \in U_{ef}$ 

  ultimately show False
    using C-least-false[unfolded is-least-false-clause-def
      linorder-cls.is-least-in-filter-iff]
    by (metis  $\langle C \neq \text{efac } C \rangle$  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  $(N \cup U_r \cup U_{ef}) \ x$ 
    using  $\langle \text{efac } C \in U_{ef} \vee \text{is-least-false-clause } (N \cup U_r \cup U_{ef}) \ x \rangle$ 
by argo
next
  show ord-res.is-maximal-lit L x
    using  $\langle \text{ord-res.ground-factoring}^{++} \ x \ C \rangle \langle \text{ord-res.is-maximal-lit } L \ C \rangle$ 
    using ord-res.ground-factorings-preserves-maximal-literal
    by (metis tranclp-into-rtranclp)
next
  show is-pos L
    using  $\langle \text{is-pos } L \rangle$  .
next

```

```

    show  $finsert\ C'\ U_{ef} = finsert\ (efac\ x)\ U_{ef}$ 
    using  $\langle ord-res.ground-factoring^{++}\ x\ C \rangle \langle ord-res.ground-factoring\ C\ C' \rangle$ 
    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  $s1' = N\ |\cup|\ U_r\ |\cup|\ finsert\ C'\ U_{ef}\ |\cup|\ U_f$ 
    unfolding  $\langle s1' = finsert\ C'\ s1 \rangle\ s1-def$  by simp

    moreover have  $\exists C\ |\in|\ N\ |\cup|\ U_r\ |\cup|\ finsert\ C'\ U_{ef}.$ 
     $ord-res.ground-factoring^{++}\ C\ C_f \wedge C_f \neq efac\ C_f \wedge$ 
     $(efac\ C_f\ |\in|\ finsert\ C'\ U_{ef} \vee is-least-false-clause\ (N\ |\cup|\ U_r\ |\cup|\ finsert\ C'$ 
     $U_{ef})\ C)$ 
    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  $\langle C_f\ |\in|\ U_f \rangle\ U_f-spec$  by metis

      show ?thesis
      proof  $(intro\ bexI\ conjI)$ 
        show  $x\ |\in|\ N\ |\cup|\ U_r\ |\cup|\ finsert\ C'\ U_{ef}$ 
        using  $\langle x\ |\in|\ N\ |\cup|\ U_r\ |\cup|\ U_{ef} \rangle$  by simp
      next
        show  $ord-res.ground-factoring^{++}\ x\ C_f$ 
        using  $\langle ord-res.ground-factoring^{++}\ x\ C_f \rangle$  .
      next
        show  $C_f \neq efac\ C_f$ 
        using  $\langle C_f \neq efac\ C_f \rangle$  .
      next
        show  $efac\ C_f\ |\in|\ finsert\ C'\ U_{ef} \vee is-least-false-clause\ (N\ |\cup|\ U_r\ |\cup|\$ 
         $finsert\ C'\ U_{ef})\ x$ 
        using  $\langle efac\ C_f\ |\in|\ U_{ef} \vee is-least-false-clause\ (N\ |\cup|\ U_r\ |\cup|\ U_{ef})\ x \rangle$ 
        proof  $(elim\ disjE)$ 
          assume  $efac\ C_f\ |\in|\ U_{ef}$ 
          thus ?thesis
          by simp
        next
          assume  $is-least-false-clause\ (N\ |\cup|\ U_r\ |\cup|\ U_{ef})\ x$ 
          show ?thesis
          proof  $(cases\ C' = efac\ x)$ 
            case True

```

```

moreover have efac  $x = \text{efac } C_f$ 
using  $\langle \text{ord-res.ground-factorizing}^{++} \ x \ C_f \rangle$  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  $C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef}$ 
    then show ?thesis
      by (smt (verit) C-least-false True Uf-unproductive
         $\langle \text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef}) \ x \rangle$   $\langle \text{ord-res.ground-factorizing}^{++}$ 
x Cf⟩
        finsert-iff ground-factorizing-preserves-efac ground-factorings-preserves-efac
        linorder-cls.Uniq-is-least-in-fset local.factorizing(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  $C \mid \in \mid U_f$ 
      then show ?thesis
        using C-least-false
      using is-least-false-clause-if-is-least-false-clause-in-union-unproductive[
        OF Uf-unproductive]
      by (smt (z3) True Uf-spec  $\langle \text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup$ 
Uef) \ x \rangle
         $\langle \text{ord-res.ground-factorizing}^{++} \ x \ C_f \rangle$  finsert-absorb finsert-iff
        ground-factorizing-preserves-efac ground-factorings-preserves-efac
        linorder-cls.Uniq-is-least-in-fset local.factorizing(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  $?U_f' = \text{finsert } C' \ U_f$ 

  have ?match s1' s2

```

proof –
have $finsert\ C'\ s1 = N \mid\cup\ U_r \mid\cup\ U_{ef} \mid\cup\ ?U_f'$
unfolding $s1-def$ **by** $simp$

moreover have $\exists C \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef}$.
 $ord-res.ground-factoring^{++}\ C\ C_f \wedge C_f \neq efac\ C_f \wedge$
 $(efac\ C_f \mid\in\ U_{ef} \vee is-least-false-clause\ (N \mid\cup\ U_r \mid\cup\ U_{ef})\ C)$
if $C_f \mid\in\ ?U_f'$ **for** C_f

proof –
from $\langle C_f \mid\in\ ?U_f' \rangle$ **have** $C_f = C' \vee C_f \mid\in\ U_f$
by $simp$
thus $?thesis$

proof ($elim\ disjE$)
assume $C_f = C'$
thus $?thesis$
using $C-in-disj$

proof ($elim\ disjE$)
assume $C \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef}$
show $?thesis$
proof ($intro\ beqI\ conjI$)
show $C \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef}$
using $\langle C \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef} \rangle$.

next
show $ord-res.ground-factoring^{++}\ C\ C_f$
using $\langle ord-res.ground-factoring\ C\ C' \rangle\ \langle C_f = C' \rangle$ **by** $simp$

next
show $C_f \neq efac\ C_f$
using $False\ \langle C_f = C' \rangle$ **by** $argo$

next
have $is-least-false-clause\ (N \mid\cup\ U_r \mid\cup\ U_{ef})\ C$
using $factoring$

using $Interp-eq\ \langle C \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef} \rangle\ linorder-cls.is-least-in-filter-iff$
by ($simp\ add: s1-def\ is-least-false-clause-def$)
thus $efac\ C_f \mid\in\ U_{ef} \vee is-least-false-clause\ (N \mid\cup\ U_r \mid\cup\ U_{ef})\ C$..

qed

next
assume $C \mid\in\ U_f$
then obtain x **where**
 $x \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef}$ **and**
 $ord-res.ground-factoring^{++}\ x\ C$ **and**
 $C \neq efac\ C$ **and**
 $efac\ C \mid\in\ U_{ef} \vee is-least-false-clause\ (N \mid\cup\ U_r \mid\cup\ U_{ef})\ x$
using U_f-spec **by** $metis$

show $?thesis$
proof ($intro\ beqI\ conjI$)
show $x \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef}$
using $\langle x \mid\in\ N \mid\cup\ U_r \mid\cup\ U_{ef} \rangle$.

next

```

      show ord-res.ground-factorizing++ x Cf
      using ⟨ord-res.ground-factorizing++ x C⟩ ⟨ord-res.ground-factorizing C
C'⟩ ⟨Cf = C'⟩
      by simp
    next
      show Cf ≠ efac Cf
      using False ⟨Cf = C'⟩ by argo
    next
      show efac Cf |∈| Uef ∨ is-least-false-clause (N |∪| Ur |∪| Uef) x
      using ⟨efac C |∈| Uef ∨ is-least-false-clause (N |∪| Ur |∪| Uef) x⟩
      proof (elim disjE)
        assume efac C |∈| Uef

        moreover have efac C = efac Cf
          unfolding ⟨Cf = C'⟩
        using ⟨ord-res.ground-factorizing C C'⟩ ground-factorizing-preserves-efac
      by metis

      ultimately show ?thesis
        by argo
    next
      assume is-least-false-clause (N |∪| Ur |∪| Uef) x
      thus ?thesis
        by argo
      qed
    qed
  next
    assume Cf |∈| Uf
    thus ?thesis
      using Uf-spec by metis
    qed
  qed

ultimately have ord-res-1-matches-ord-res-2 (finsert C' s1) (N, (Ur, Uef))
  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  $C' \prec_c C$ 
    using factoring ord-res.ground-factoring-smaller-conclusion by metis

  have unproductive:  $\forall x \in \{C'\}. \text{ord-res.production (fset } s1 \cup \{C'\}) x = \{\}$ 
    using  $\langle C' \neq \text{efac } C' \rangle$ 
    by (simp add: nex-strictly-maximal-pos-lit-if-neq-efac
      unproductive-if-nex-strictly-maximal-pos-lit)

  have Interp-eq:  $\bigwedge D. \text{ord-res-Interp (fset } s1) D = \text{ord-res-Interp (fset (finsert } C' s1)) D$ 
    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  $\neg \text{ord-res-Interp (fset } s1) C \models C$ 
      using C-least-false s1-def is-least-false-clause-def
      linorder-cls.is-least-in-ffilter-iff by simp
    thus  $\neg \text{ord-res-Interp (fset (finsert } C' s1)) C' \models C'$ 
      by (metis Interp-eq \langle C' \prec_c C \rangle 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  $y \in \text{finsert } C' s1$  and  $y \neq C'$  and
       $y\text{-false}: \neg \text{ord-res-Interp (fset (finsert } C' s1)) y \models y$ 
    hence  $y \in s1$ 
      by simp

    moreover have  $\neg \text{ord-res-Interp (fset } s1) y \models y$ 
      using y-false
      unfolding Interp-eq .

    ultimately have  $C \preceq_c y$ 
      using C-least-false[folded s1-def, unfolded is-least-false-clause-def]
      unfolding linorder-cls.is-least-in-ffilter-iff
      by force
    thus  $C' \prec_c y$ 
      using  $\langle C' \prec_c C \rangle$  by order

  qed simp
  thus ?thesis
    unfolding ord-res-1-measure-def \langle s1' = finsert } C' s1 \rangle
    by (metis (mono-tags, lifting) linorder-cls.Uniq-is-least-in-fset
      is-least-false-clause-def the1-equality' the-equality)
  qed

```

moreover have $C' \subset\# C$
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
 $C \in s1$ **and**
 $\neg \text{ord-res-Interp} (fset\ s1)\ C \models C$ **and**
 $\forall x \in s1. \neg \text{ord-res-Interp} (fset\ s1)\ x \models x \longrightarrow x \neq C \longrightarrow C \prec_c x$
unfolding *is-least-false-clause-def linorder-cls.is-least-in-filter-iff* **by** *simp-all*

have $C \notin U_f$
proof (*rule notI*)
assume $C \in U_f$
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 $C \in N \cup U_r \cup U_{ef}$
using $\langle C \in s1 \rangle$ **by** (*simp add: s1-def*)

have *C-least-false: is-least-false-clause (N | \cup | U_r | \cup | U_{ef} | \cup | U_f) C*
using *resolution s1-def* **by** *metis*
hence *C-least-false': is-least-false-clause (N | \cup | U_r | \cup | U_{ef}) C*
using *is-least-false-clause-if-is-least-false-clause-in-union-unproductive[*
OF U_f-unproductive $\langle C \in N \cup U_r \cup U_{ef} \rangle$] **by** *argo*

define *s2'* **where**
 $s2' = (N, (finsert\ CD\ U_r,\ U_{ef}))$

have *ord-res-2-step⁺⁺ s2 s2'*
proof –
have $D \notin U_f$
proof (*rule notI*)
assume $D \in U_f$
thus *False*
using $\langle \text{ord-res.production} (fset\ s1)\ D = \{atm-of\ L\} \rangle$
using *U_f-unproductive s1-def* **by** *simp*

```

qed
hence D-in:  $D \in N \cup U_r \cup U_{ef}$ 
  using  $\langle D \in s1 \rangle$ [unfolded s1-def] by simp

have ord-res-2  $N \cup U_r \cup U_{ef}$  (insert CD  $U_r \cup U_{ef}$ )
proof (rule ord-res-2.resolution)
  show is-least-false-clause  $(N \cup U_r \cup U_{ef}) C$ 
    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]
Uf-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 insert CD  $(N \cup U_r \cup U_{ef} \cup U_f) = N \cup \text{insert } CD \ U_r \cup U_{ef} \cup U_f$ 
    by simp

  moreover have  $\exists C \in N \cup \text{insert } CD \ U_r \cup U_{ef}$ .
    ord-res.ground-factoring++  $C C_f \wedge C_f \neq \text{efac } C_f \wedge$ 
    (efac  $C_f \in U_{ef} \vee \text{is-least-false-clause } (N \cup \text{insert } CD \ U_r \cup U_{ef}) C$ )
  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

```

```

    Cf ≠ efac Cf and
    efac Cf |∈| Uef ∨ is-least-false-clause (N |∪| Ur |∪| Uef) x
    using ⟨Cf |∈| Uf⟩ Uf-spec by metis
  show ?thesis
  proof (intro beqI conjI)
    show x |∈| N |∪| fininsert CD Ur |∪| Uef
      using ⟨x |∈| N |∪| Ur |∪| Uef⟩ by simp
    next
      show ord-res.ground-factorizing++ x Cf
        using ⟨ord-res.ground-factorizing++ x Cf⟩ .
    next
      show Cf ≠ efac Cf
        using ⟨Cf ≠ efac Cf⟩ .
    next
      show ⟨efac Cf |∈| Uef ∨ is-least-false-clause (N |∪| fininsert CD Ur |∪|
Uef) x⟩
        using ⟨efac Cf |∈| Uef ∨ is-least-false-clause (N |∪| Ur |∪| Uef) x⟩ ⟨x
|∈| N |∪| Ur |∪| Uef⟩
        by (metis (no-types, lifting) C-least-false' Uniq-D ⟨ord-res.ground-factorizing++
x Cf⟩
          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-factorizing.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  $\forall s1. \text{ord-res-1-final } s1 \longrightarrow (\nexists s1'. \text{ord-res-1 } s1 \ s1')$ 
  using ord-res-1-semantic.final-finished
  by (simp add: finished-def)
next
show  $\forall s2. \text{ord-res-2-final } s2 \longrightarrow (\nexists s2'. \text{ord-res-2-step } s2 \ s2')$ 
  using ord-res-2-semantic.final-finished
  by (simp add: finished-def)
next
show  $\forall i \ s1 \ s2. \text{match } i \ s1 \ s2 \longrightarrow \text{ord-res-1-final } s1 = \text{ord-res-2-final } s2$ 
  using ord-res-1-final-iff-ord-res-2-final
  by (simp add: match-def)
next
show  $\forall i \ s1 \ s2. \text{match } i \ s1 \ s2 \longrightarrow$ 
  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
     $S2\text{-def}: S2 = (N, s2)$  and
     $i = \text{ord-res-1-measure } s1$  and
     $\text{match}: \text{ord-res-1-matches-ord-res-2 } s1 \ S2$ 
  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  $\forall i \ s1 \ s2 \ s1'. \text{match } i \ s1 \ s2 \longrightarrow \text{ord-res-1 } s1 \ s1' \longrightarrow$ 
   $(\exists i' \ s2'. \text{ord-res-2-step}^{++} \ s2 \ s2' \wedge \text{match } i' \ s1' \ s2') \vee (\exists i'. \text{match } i' \ s1' \ s2 \wedge$ 
   $i' \subset\# \ i)$ 
proof (intro allI impI)
  fix  $i \ s1 \ S2 \ s1'$ 
  assume match i s1 S2
  then obtain  $N \ s2$  where
     $S2\text{-def}: S2 = (N, s2)$  and  $i = \text{ord-res-1-measure } s1$  and  $\text{ord-res-1-matches-ord-res-2}$ 
   $s1 \ S2$ 
  unfolding match-def
  by (metis prod.exhaust)

  moreover assume ord-res-1 s1 s1'

  ultimately have  $(\exists S2'. \text{ord-res-2-step}^{++} \ S2 \ S2' \wedge \text{ord-res-1-matches-ord-res-2}$ 

```

$s1' S2' \vee$
 $ord-res-1-matches-ord-res-2 s1' S2 \wedge ord-res-1-measure s1' \subset\# ord-res-1-measure s1$
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 $(metis (mono-tags, lifting) \langle i = ord-res-1-measure s1 \rangle match-def)$
qed
qed

end

29 ORD-RES-3 (full resolve)

type-synonym $'f ord-res-3-state = 'f gclause fset \times 'f gclause fset \times 'f gclause fset$

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

inductive *ord-res-2-matches-ord-res-3* :: $- \Rightarrow 'f ord-res-3-state \Rightarrow bool$ **where**
 $(\forall C | \in | U_{pr}. \exists D1 | \in | N | \cup | U_{er} | \cup | U_{ef}. \exists D2 | \in | N | \cup | U_{er} | \cup | U_{ef}.$
 $(ground-resolution D1)^{++} D2 C \wedge C \neq eres D1 D2 \wedge eres D1 D2 | \in | U_{er})$
 \implies
 $ord-res-2-matches-ord-res-3 (N, (U_{pr} | \cup | U_{er}, U_{ef})) (N, (U_{er}, U_{ef}))$

lemma *ord-res-2-final-iff-ord-res-3-final*:
assumes *match: ord-res-2-matches-ord-res-3 S2 S3*
shows $ord-res-2-final S2 \longleftrightarrow ord-res-3-final S3$
using *match*

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 $U_{pr}\text{-spec}: \forall C | \in | U_{pr}. \exists D1 | \in | N | \cup | U_{er} | \cup | U_{ef}. \exists D2 | \in | N | \cup | U_{er} | \cup | U_{ef}.$
 $(ground-resolution D1)^{++} D2 C \wedge C \neq eres D1 D2 \wedge eres D1 D2 | \in | U_{er}$
using *invars by argo*

have *least-false-spec: 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 $U_{pr}\text{-unproductive}: \forall C | \in | U_{pr}. ord-res.production (fset (N | \cup | U_{er} | \cup | U_{ef} | \cup | U_{pr})) C = \{\}$
proof $(intro ballI)$

fix C
assume $C \mid \in \mid U_{pr}$
hence $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$
using $U_{pr}\text{-spec}$
by (*metis eres-eq-after-tranclp-ground-resolution nex-strictly-maximal-pos-lit-if-neq-eres*)
thus $\text{ord-res.production (fset (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef} \mid \cup \mid U_{pr})) } C = \{\}$
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
qed

hence $\text{Interp-N-U}_r\text{-U}_{ef}\text{-eq-Interp-N-U}_{er}\text{-U}_{ef}: \bigwedge C.$
 $\text{ord-res-Interp (fset (N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef})) } C =$
 $\text{ord-res-Interp (fset (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef})) } C$
using $\text{Interp-union-unproductive}[OF \text{ finite-fset finite-fset, folded union-fset,}$
 $\text{of } U_{pr} \ N \ \mid \cup \mid U_{er} \ \mid \cup \mid U_{ef}]$
by (*simp add: funion-left-commute sup-commute*)

have $\text{ex-false-clause (fset (N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))} \longleftrightarrow$
 $\text{ex-false-clause (fset (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))}$

proof (*rule iffI*)

assume $\text{ex-false-clause (fset (N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))}$
then obtain C **where** $\text{is-least-false-clause (N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) } C$
using *obtains-least-false-clause-if-ex-false-clause* **by** *metis*
thus $\text{ex-false-clause (fset (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))}$
using *least-false-spec ex-false-clause-iff* **by** *metis*

next

assume $\text{ex-false-clause (fset (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))}$
thus $\text{ex-false-clause (fset (N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}))}$
unfolding *ex-false-clause-def*
unfolding $\text{Interp-N-U}_r\text{-U}_{ef}\text{-eq-Interp-N-U}_{er}\text{-U}_{ef}$
by *auto*

qed

moreover have $\{\#\} \mid \in \mid N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef} \longleftrightarrow \{\#\} \mid \in \mid N \mid \cup \mid U_{er} \mid \cup$
 U_{ef}

proof (*rule iffI*)

assume $\{\#\} \mid \in \mid N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}$
hence $\{\#\} \mid \in \mid N \mid \cup \mid U_{ef} \vee \{\#\} \mid \in \mid U_{pr} \mid \cup \mid U_{er}$
by *auto*

thus $\{\#\} \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}$

proof (*elim disjE*)

assume $\{\#\} \mid \in \mid N \mid \cup \mid U_{ef}$

thus *?thesis*

by *auto*

next

have $\{\#\} \mid \notin \mid U_{pr}$

using $U_{pr}\text{-spec}[rule-format, \text{ of } \{\#\}]$

by (*metis eres-eq-after-tranclp-ground-resolution eres-mempty-right*)

moreover assume $\{\#\} \mid \in \mid U_{pr} \mid \cup \mid U_{er}$

ultimately show *?thesis*

by *simp*
 qed
 next
 assume $\{\#\} \in N \cup U_{er} \cup U_{ef}$
 then show $\{\#\} \in N \cup U_{pr} \cup U_{er} \cup U_{ef}$
 by *auto*
 qed

 ultimately have $\text{ord-res-final } (N \cup U_{pr} \cup U_{er} \cup U_{ef}) = \text{ord-res-final } (N \cup U_{er} \cup U_{ef})$
 unfolding *ord-res-final-def* by *argo*

 thus $\text{ord-res-2-final } S_2 \longleftrightarrow \text{ord-res-3-final } S_3$
 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 $\exists C.$ *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 $C D i = (\text{THE } CD. (\text{ground-resolution } C \rightsquigarrow i) D CD)$

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*: $(\text{ground-resolution } C \rightsquigarrow n) D CD$
 assumes $i < j$ and $j \leq n$
 shows *resolvent-at* $C D j \prec_c$ *resolvent-at* $C D i$
proof –
 obtain j' where
 $j = i + \text{Suc } j'$
 using $\langle i < j \rangle$ by (*metis less-iff-Suc-add nat-arith.suc1*)

 obtain n' where
 $n = j + n'$
 using $\langle j \leq n \rangle$ by (*metis le-add-diff-inverse*)

 obtain $CD_i CD_j CD_n$ where
 (*ground-resolution* $C \rightsquigarrow i$) $D CD_i$ and
 (*ground-resolution* $C \rightsquigarrow \text{Suc } j'$) $CD_i CD_j$
 (*ground-resolution* $C \rightsquigarrow n'$) $CD_j CD_n$
 using *reso-at* $\langle n = j + n' \rangle \langle j = i + \text{Suc } j' \rangle$ by (*metis relpowp-plusD*)

have *: *resolvent-at* $C \ D \ i = CD_i$
unfolding *resolvent-at-def*
using $\langle (\text{ground-resolution } C \ \sim i) \ D \ CD_i \rangle$
by (*simp add: Uniq-ground-resolution Uniq-relpowp the1-equality'*)

have (*ground-resolution* $C \ \sim j$) $D \ CD_j$
unfolding $\langle j = i + \text{Suc } j' \rangle$
using $\langle (\text{ground-resolution } C \ \sim i) \ D \ CD_i \rangle \langle (\text{ground-resolution } C \ \sim \text{Suc } j') \ CD_i \ CD_j \rangle$
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 $\langle (\text{ground-resolution } C \ \sim \text{Suc } j') \ CD_i \ CD_j \rangle$
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* $C \ \sim n$) $D \ CD$ **and** $i < n$
shows
left-premise-lt-resolvent-at: $C \prec_c$ *resolvent-at* $C \ D \ i$ **and**
max-lit-resolvent-at:
ord-res.is-maximal-lit $L \ D \implies$ *ord-res.is-maximal-lit* L (*resolvent-at* $C \ D \ i$)
and
nex-pos-strictly-max-lit-in-resolvent-at:
 $\nexists L.$ *is-pos* $L \wedge$ *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* $C \ \sim i$) D (*resolvent-at* $C \ D \ i$)
proof –
obtain j **where** *n-def*: $n = i + \text{Suc } j$
using $\langle i < n \rangle$ *less-natE* **by** *auto*

obtain CD' **where** (*ground-resolution* $C \ \sim i$) $D \ CD'$ **and** (*ground-resolution* $C \ \sim \text{Suc } j$) $CD' \ CD$
using *reso-at n-def* **by** (*metis relpowp-plusD*)

have *resolvent-at* $C \ D \ i = CD'$
unfolding *resolvent-at-def*
using $\langle (\text{ground-resolution } C \ \sim i) \ D \ CD' \rangle$
by (*simp add: Uniq-ground-resolution Uniq-relpowp the1-equality'*)

have $C \prec_c CD'$
proof (rule left-premise-lt-right-premise-if-tranclp-ground-resolution)
show $(\text{ground-resolution } C)^{++} CD' CD$
using $\langle (\text{ground-resolution } C \rightsquigarrow \text{Suc } j) CD' CD \rangle$
by (metis Zero-not-Suc tranclp-if-relpowp)
qed
thus $C \prec_c \text{resolvent-at } C D i$
unfolding $\langle \text{resolvent-at } C D i = CD' \rangle$ **by** argo

show *ord-res.is-maximal-lit* $L (\text{resolvent-at } C D i)$ **if** *ord-res.is-maximal-lit* $L D$
unfolding $\langle \text{resolvent-at } C D i = CD' \rangle$
using that
using $\langle (\text{ground-resolution } C \rightsquigarrow i) D CD' \rangle$
by (smt (verit, cfv-SIG) Uniq-ground-resolution Uniq-relpowp Zero-not-Suc
 $\langle \wedge \text{thesis. } (\wedge CD'. \llbracket (\text{ground-resolution } C \rightsquigarrow i) D CD'; (\text{ground-resolution } C \rightsquigarrow \text{Suc } j) CD' CD \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$
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 $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L (\text{resolvent-at } C D i)$
unfolding $\langle \text{resolvent-at } C D i = CD' \rangle$
by (metis Zero-not-Suc $\langle (\text{ground-resolution } C \rightsquigarrow \text{Suc } j) CD' CD \rangle$
nex-strictly-maximal-pos-lit-if-resolvable tranclpD tranclp-if-relpowp)

show *ground-resolution* $C (\text{resolvent-at } C D i) (\text{resolvent-at } C D (\text{Suc } i))$
proof –
obtain CD'' **where** *ground-resolution* $C CD' CD''$ **and** $(\text{ground-resolution } C \rightsquigarrow j) CD'' CD$
using $\langle (\text{ground-resolution } C \rightsquigarrow \text{Suc } j) CD' CD \rangle$ **by** (metis relpowp-Suc-D2)
hence $(\text{ground-resolution } C \rightsquigarrow \text{Suc } i) D CD''$
using $\langle (\text{ground-resolution } C \rightsquigarrow i) D CD' \rangle$ **by** auto
hence $\text{resolvent-at } C D (\text{Suc } i) = CD''$
unfolding *resolvent-at-def*
by (meson Uniq-ground-resolution Uniq-relpowp the1-equality')

show ?thesis
unfolding $\langle \text{resolvent-at } C D i = CD' \rangle \langle \text{resolvent-at } C D (\text{Suc } i) = CD'' \rangle$
using $\langle \text{ground-resolution } C CD' CD'' \rangle$.
qed

show $(\text{ground-resolution } C \rightsquigarrow i) D (\text{resolvent-at } C D i)$
using $\langle (\text{ground-resolution } C \rightsquigarrow i) D CD' \rangle \langle \text{resolvent-at } C D i = CD' \rangle$ **by** argo
qed

definition *resolvents-upto* **where**
resolvents-upto $C D n = \text{resolvent-at } C D \mid \uparrow \text{fset-upto } (\text{Suc } 0) n$

lemma *resolvents-upto-0[simp]*:

resolvents-upto $C D 0 = \{\{\}\}$
by (*simp add: resolvents-upto-def*)

lemma *resolvents-upto-Suc*[*simp*]:
resolvents-upto $C D (Suc\ n) = \text{finsert } (\text{resolvent-at } C D (Suc\ n)) (\text{resolvents-upto } C D\ n)$
by (*simp add: resolvents-upto-def*)

lemma *resolvent-at-fmember-resolvents-upto*:
assumes $k \neq 0$
shows *resolvent-at* $C D k \in | \text{resolvents-upto } C D\ k$
unfolding *resolvents-upto-def*
proof (*rule fimageI*)
show $k \in | \text{fset-upto } (Suc\ 0)\ k$
using *assms* **by** *simp*
qed

lemma *backward-simulation-2-to-3*:
fixes *match measure less*
defines $\text{match} \equiv \text{ord-res-2-matches-ord-res-3}$
assumes
match: match $S2\ S3$ **and**
step2: ord-res-3-step $S3\ S3'$
shows $(\exists S2'. \text{ord-res-2-step}^{++}\ S2\ S2' \wedge \text{match } S2'\ S3')$
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 $U_{pr}\text{-spec}: \forall C \in | U_{pr}. \exists D1 \in | N \cup | U_{er} \cup | U_{ef}. \exists D2 \in | N \cup | U_{er} \cup | U_{ef}. (\text{ground-resolution } D1)^{++}\ D2\ C \wedge C \neq \text{eres } D1\ D2 \wedge \text{eres } D1\ D2 \in | U_{er}$
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**
 $(\text{ground-resolution } D1)^{++}\ D2\ C$ **and**
 $C \neq \text{eres } D1\ D2$ **and**
 $\text{eres } D1\ D2 \in | U_{er}$
using $U_{pr}\text{-spec } C\text{-in}$ **by** *metis*

have $\text{eres } D1\ C = \text{eres } D1\ D2$
using $\langle (\text{ground-resolution } D1)^{++}\ D2\ C \rangle \text{eres-eq-after-tranclp-ground-resolution}$
by *metis*

hence $\text{eres } D1 \ C \prec_c \ C$
using $\text{eres-le}[of \ D1 \ C] \langle C \neq \text{eres } D1 \ D2 \rangle$ **by** *order*

show *?thesis*
proof (*cases ord-res-Interp (fset (N | \cup | U_{pr} | \cup | U_{er} | \cup | U_{ef})) (eres D1 D2)*
 $\models \text{eres } D1 \ D2$)
case *True*
then show *?thesis*
by (*metis (no-types, lifting) $\langle \text{ground-resolution } D1 \rangle^{++} \ D2 \ C \rangle \langle \text{eres } D1 \ C$*
 $= \text{eres } D1 \ D2 \rangle$
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 $\langle \text{eres } D1 \ C = \text{eres } D1 \ D2 \rangle \langle \text{eres } D1$*
 $C \prec_c \ C \rangle$
 $\langle \text{eres } D1 \ D2 \in | \ U_{er} \rangle$ 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: $\bigwedge N. \forall C \in | \ U_{pr}. \text{ord-res.production } N \ C = \{\}$*
proof (*intro ballI*)
fix *C*
assume $C \in | \ U_{pr}$
hence $\exists D \in | \ N \ | \cup \ U_{er} \ | \cup \ U_{ef}. (\exists C'. \text{ground-resolution } D \ C \ C')$
using *U_{pr}-spec* **by** (*metis eres-eq-after-tranclp-ground-resolution resolvable-if-neq-eres*)
hence $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ C$
using *nex-strictly-maximal-pos-lit-if-resolvable* **by** *metis*
thus $\bigwedge N. \text{ord-res.production } N \ C = \{\}$
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}:*
 $\bigwedge C. \text{ord-res-Interp (fset (N | \cup | U_{pr} | \cup | U_{er} | \cup | U_{ef})) } C = \text{ord-res-Interp (fset$
 $(N \ | \cup \ U_{er} \ | \cup \ U_{ef})) } C$
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  $Ca \mid \in \mid U_{pr}$ 
then obtain  $D1 D2$  where
   $D1 \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}$  and
   $D2 \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}$  and
   $(\text{ground-resolution } D1)^{++} D2 Ca$  and
   $Ca \neq \text{eres } D1 D2$  and
   $\text{eres } D1 D2 \mid \in \mid U_{er}$ 
  using  $U_{pr}\text{-spec}$  by metis

have  $\text{eres } D1 D2 = \text{eres } D1 Ca$ 
using  $\langle (\text{ground-resolution } D1)^{++} D2 Ca \rangle$  eres-eq-after-tranclp-ground-resolution
by metis

show  $\exists D \mid \in \mid U_{er}. D \prec_c Ca \wedge \{D\} \Vdash_e \{Ca\}$ 
proof (intro best conjI)
  have  $\text{eres } D1 Ca \preceq_c Ca$ 
  using eres-le .
  thus  $\text{eres } D1 D2 \prec_c Ca$ 
  using  $\langle Ca \neq \text{eres } D1 D2 \rangle$   $\langle \text{eres } D1 D2 = \text{eres } D1 Ca \rangle$  by order
next
  show  $\{\text{eres } D1 D2\} \Vdash_e \{Ca\}$ 
  using  $\langle (\text{ground-resolution } D1)^{++} D2 Ca \rangle$  eres-entails-resolvent by metis
next
  show  $\text{eres } D1 D2 \mid \in \mid U_{er}$ 
  using  $\langle \text{eres } D1 D2 \mid \in \mid U_{er} \rangle$  by simp
qed
qed

from step2 obtain  $s3'$  where  $S3'\text{-def}: S3' = (N, s3')$  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  $\langle \text{ord-res-3 } N (U_{er}, U_{ef}) s3' \rangle$ 
proof (cases  $N (U_{er}, U_{ef}) s3'$  rule: ord-res-3.cases)
  case (factoring  $C L U_{ef}'$ )

define  $S2'$  where
   $S2' = (N, (U_{pr} \mid \cup \mid U_{er}, \text{finsert } (\text{efac } C) U_{ef}))$ 

have ord-res-2-step++  $S2 S2'$ 
unfolding match-hyps(1,2)  $S2'\text{-def}$ 
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  $\langle \text{is-least-false-clause } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) C \rangle$ 
  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 $\text{finsert } (efac\ C)\ U_{ef} = \text{finsert } (efac\ C)\ U_{ef}$
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 $\forall Ca \in U_{pr}.$
 $\exists D1 \in N \mid U \mid U_{er} \mid \text{finsert } (efac\ C)\ U_{ef}. \exists D2 \in N \mid U \mid U_{er} \mid \text{finsert}$
 $(efac\ C)\ U_{ef}.$
 $(\text{ground-resolution } D1)^{++}\ D2\ Ca \wedge Ca \neq \text{eres } D1\ D2 \wedge \text{eres } D1\ D2 \in U_{er}$
using *U_{pr}-spec by auto*
qed

ultimately show *?thesis*
by *metis*
next
case (*resolution C L D U_{rr}'*)

have $(\text{ground-resolution } D)^{**}\ C\ (\text{eres } D\ C) \nexists x. \text{ground-resolution } D\ (\text{eres } D$
 $C)\ x$
unfolding *atomize-conj*
by (*metis ex1-eres-eq-full-run-ground-resolution full-run-def*)

moreover have $\exists x. \text{ground-resolution } D\ C\ x$
unfolding *ground-resolution-def*
using *resolution*
by (*metis Neg-atm-of-iff ex-ground-resolutionI ord-res.mem-productionE singletonI*)

ultimately have $(\text{ground-resolution } D)^{++}\ C\ (\text{eres } D\ C)$
by (*metis rtranclpD*)

then obtain *n where* $(\text{ground-resolution } D \rightsquigarrow \text{Suc } n)\ C\ (\text{eres } D\ C)$
by (*metis not0-implies-Suc not-gr-zero tranclp-power*)

hence *resolvent-at D C (Suc n) = eres D C*
by (*metis Uniq-ground-resolution Uniq-relpowp resolvent-at-def the1-equality'*)

have *steps: k ≤ Suc n ⇒ (ord-res-2-step ~ k)*
 $(N, U_{pr} \mid U \mid U_{er}, U_{ef})\ (N, U_{pr} \mid U \mid U_{er} \mid \text{resolvents-upto } D\ C\ k, U_{ef})$ **for**
k

```

proof (induction k)
  case 0
  show ?case
  by simp
next
  case (Suc k)
  have k < Suc n
  using ⟨Suc k ≤ Suc n⟩ by presburger
  hence k ≤ Suc n
  by presburger
  hence (ord-res-2-step  $\sim$  k) (N, Upr |∪| Uer, Uef)
  (N, Upr |∪| Uer |∪| resolvents-upto D C k, Uef)
  using Suc.IH by metis

moreover have ord-res-2-step
  (N, Upr |∪| Uer |∪| resolvents-upto D C k, Uef)
  (N, Upr |∪| Uer |∪| resolvents-upto D C (Suc k), Uef)
  unfolding resolvents-upto-Suc
  proof (intro ord-res-2-step.intros ord-res-2.resolution)
  show is-least-false-clause (N |∪| (Upr |∪| Uer |∪| resolvents-upto D C k)
|∪| Uef)
  (resolvent-at D C k)
  using ⟨k < Suc n⟩
  proof (induction k)
  case 0
  have is-least-false-clause (N |∪| Upr |∪| Uer |∪| Uef) C
  using ⟨is-least-false-clause (N |∪| Uer |∪| Uef) C⟩
  unfolding least-false-conv .
  thus ?case
  unfolding funion-fempty-right funion-assoc[symmetric]
  by simp
  next
  case (Suc k')

  have  $\bigwedge x$ . ord-res-Interp (fset (N |∪| (Upr |∪| Uer |∪| resolvents-upto D
C (Suc k')) |∪| Uef)) x =
  ord-res-Interp (fset (N |∪| Uer |∪| Uef) ∪ fset (Upr |∪| resolvents-upto
D C (Suc k'))) x
  by (simp add: funion-left-commute sup-assoc sup-commute)
  also have  $\bigwedge x$ . ord-res-Interp (fset (N |∪| Uer |∪| Uef) ∪ fset (Upr |∪|
resolvents-upto D C (Suc k'))) x =
  ord-res-Interp (fset (N |∪| Uer |∪| Uef)) x
  proof (intro Interp-union-unproductive ballI)
  fix x y assume y |∈| Upr |∪| resolvents-upto D C (Suc k')
  hence y |∈| Upr ∨ y |∈| resolvents-upto D C (Suc k')
  by blast
  thus ord-res.production (fset (N |∪| Uer |∪| Uef) ∪ fset (Upr |∪|
resolvents-upto D C (Suc k'))) y = {}
  proof (elim disjE)

```

```

    assume  $y \in U_{pr}$ 
    thus ?thesis
      using  $U_{pr}$ -unproductive by metis
    next
      assume  $y \in \text{resolvents-upto } D \ C \ (Suc \ k')$ 
      then obtain  $i$  where  $i \in \text{fset-upto } (Suc \ 0) \ (Suc \ k')$  and  $y =$ 
resolvent-at  $D \ C \ i$ 
      unfolding resolvents-upto-def by blast

    have  $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ (\text{resolvent-at } D \ C$ 
i)

    proof (rule nex-pos-strictly-max-lit-in-resolvent-at)
      show  $(\text{ground-resolution } D \ \sim\! \sim \text{Suc } n) \ C \ (\text{eres } D \ C)$ 
        using  $\langle \text{ground-resolution } D \ \sim\! \sim \text{Suc } n \rangle \ C \ (\text{eres } D \ C)$  .
      next
        have  $i \leq \text{Suc } k'$ 
          using  $\langle i \in \text{fset-upto } (Suc \ 0) \ (Suc \ k') \rangle$  by auto
        thus  $i < \text{Suc } n$ 
          using  $\langle \text{Suc } k' < \text{Suc } n \rangle$  by presburger
        qed

      then show ?thesis
        using  $\langle y = \text{resolvent-at } D \ C \ i \rangle$  unproductive-if-nex-strictly-maximal-pos-lit
          by metis
        qed
      qed simp-all
      finally have Interp-simp:  $\bigwedge x.$ 
         $\text{ord-res-Interp } (\text{fset } (N \ \cup \ (U_{pr} \ \cup \ U_{er} \ \cup \ \text{resolvents-upto } D \ C \ (Suc$ 
k'))  $\cup \ U_{ef})) \ x =$ 
         $\text{ord-res-Interp } (\text{fset } (N \ \cup \ U_{er} \ \cup \ U_{ef})) \ x .$ 

    show ?case
      unfolding is-least-false-clause-def linorder-cl.is-least-in-ffilter-iff
    proof (intro conjI ballI impI)
      have  $\text{resolvent-at } D \ C \ (Suc \ k') \in \text{resolvents-upto } D \ C \ (Suc \ k')$ 
        using resolvent-at-fmember-resolvents-upto by simp
      thus  $\text{resolvent-at } D \ C \ (Suc \ k') \in N \ \cup \ (U_{pr} \ \cup \ U_{er} \ \cup \ \text{resolvents-upto}$ 
D C (Suc k'))  $\cup \ U_{ef}$ 
        by simp
      next

        show  $\neg \text{ord-res-Interp } (\text{fset } (N \ \cup \ (U_{pr} \ \cup \ U_{er} \ \cup \ \text{resolvents-upto } D$ 
C (Suc k'))  $\cup \ U_{ef}))$ 
           $(\text{resolvent-at } D \ C \ (Suc \ k')) \models \text{resolvent-at } D \ C \ (Suc \ k')$ 
        unfolding Interp-simp
        by (metis (no-types, lifting) Suc.premis Zero-not-Suc
           $\langle \text{ground-resolution } D \ \sim\! \sim \text{Suc } n \rangle \ C \ (\text{eres } D \ C)$  clause-true-if-resolved-true
          insert-not-empty is-least-false-clause-def
          linorder-cl.is-least-in-fset-ffilterD(2) local.resolution(2) lo-

```


cal.resolution(7)
relpoup-to-resolvent-at tranclp-if-relpoup)
next
fix y
assume $y \neq \text{resolvent-at } D \ C \ (\text{Suc } k')$
assume $\neg \text{ord-res-Interp } (fset \ (N \ |\cup| \ (U_{pr} \ |\cup| \ U_{er} \ |\cup| \ \text{resolvents-upto } D \ C \ (\text{Suc } k')) \ |\cup| \ U_{ef})) \ y \models y$
hence $\neg \text{ord-res-Interp } (fset \ (N \ |\cup| \ U_{er} \ |\cup| \ U_{ef})) \ y \models y$
unfolding *Interp-simp* .
hence $\neg \text{ord-res-Interp } (fset \ (N \ |\cup| \ U_{pr} \ |\cup| \ U_{er} \ |\cup| \ U_{ef})) \ y \models y$
using *Interp-N-U_r-U_{ef}-eq-Interp-N-U_{er}-U_{ef}* **by** *metis*

assume $y \in N \ |\cup| \ (U_{pr} \ |\cup| \ U_{er} \ |\cup| \ \text{resolvents-upto } D \ C \ (\text{Suc } k')) \ |\cup| \ U_{ef}$
hence $y \in N \ |\cup| \ U_{pr} \ |\cup| \ U_{er} \ |\cup| \ U_{ef} \vee y \in \text{resolvents-upto } D \ C \ (\text{Suc } k')$
by *auto*
thus $\text{resolvent-at } D \ C \ (\text{Suc } k') \prec_c y$
proof (*elim disjE*)
assume $y \in N \ |\cup| \ U_{pr} \ |\cup| \ U_{er} \ |\cup| \ U_{ef}$
have $C \preceq_c y$
proof (*cases y = C*)
case *True*
thus *?thesis*
by *order*
next
case *False*
thus *?thesis*
using $\langle y \in N \ |\cup| \ U_{pr} \ |\cup| \ U_{er} \ |\cup| \ U_{ef} \rangle$
using $\langle \neg \text{ord-res-Interp } (fset \ (N \ |\cup| \ U_{pr} \ |\cup| \ U_{er} \ |\cup| \ U_{ef})) \ y \models y \rangle$
using $\langle \text{is-least-false-clause } (N \ |\cup| \ U_{er} \ |\cup| \ U_{ef}) \ C \rangle$
unfolding *least-false-conv[symmetric]*
unfolding *is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*
by *simp*
qed

moreover **have** $\text{resolvent-at } D \ C \ (\text{Suc } k') \prec_c C$
by (*metis Suc.premis* $\langle \text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n \rangle \ C \ (\text{eres } D \ C) \rangle$
less-or-eq-imp-le
resolvent-at-less-cls-resolvent-at resolvent-at-0 zero-less-Suc)

ultimately **show** $\text{resolvent-at } D \ C \ (\text{Suc } k') \prec_c y$
by *order*
next
assume $y \in \text{resolvents-upto } D \ C \ (\text{Suc } k')$
then **obtain** i **where**
i-in: $i \in fset\text{-upto } (\text{Suc } 0) \ (\text{Suc } k')$ **and** $y\text{-def}$: $y = \text{resolvent-at } D \ C \ i$
unfolding *resolvents-upto-def* **by** *blast*

```

hence  $i < \text{Suc } k'$ 
  using  $\langle y \neq \text{resolvent-at } D \ C \ (\text{Suc } k') \rangle$ 
  by auto

show  $\text{resolvent-at } D \ C \ (\text{Suc } k') \prec_c y$ 
  unfolding  $y\text{-def}$ 
proof (rule  $\text{resolvent-at-less-cls-resolvent-at}$ )
  show  $(\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C)$ 
    using  $\langle (\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C) \rangle$  .
next
  show  $i < \text{Suc } k'$ 
    using  $\langle y \neq \text{resolvent-at } D \ C \ (\text{Suc } k') \rangle$   $i\text{-in } y\text{-def}$  by auto
next
  show  $\text{Suc } k' \leq \text{Suc } n$ 
    using  $\langle \text{Suc } k' < \text{Suc } n \rangle$  by presburger
qed
qed
qed
next
show  $\text{ord-res.is-maximal-lit } L \ (\text{resolvent-at } D \ C \ k)$ 
proof (rule  $\text{max-lit-resolvent-at}$ )
  show  $(\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C)$ 
    using  $\langle (\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C) \rangle$  .
next
  show  $k < \text{Suc } n$ 
    using  $\langle k < \text{Suc } n \rangle$  .
next
  show  $\text{ord-res.is-maximal-lit } L \ C$ 
    using  $\langle \text{ord-res.is-maximal-lit } L \ C \rangle$  .
qed
next
show  $\text{is-neg } L$ 
  using  $\langle \text{is-neg } L \rangle$  .
next
show  $D \ |\in| \ N \ |\cup| \ (U_{pr} \ |\cup| \ U_{er} \ |\cup| \ \text{resolvents-upto } D \ C \ k) \ |\cup| \ U_{ef}$ 
  using  $\langle D \ |\in| \ N \ |\cup| \ U_{er} \ |\cup| \ U_{ef} \rangle$  by auto
next
show  $D \ \prec_c \text{resolvent-at } D \ C \ k$ 
proof (rule  $\text{left-premise-lt-resolvent-at}$ )
  show  $(\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C)$ 
    using  $\langle (\text{ground-resolution } D \ \overset{\sim}{\sim} \text{Suc } n) \ C \ (\text{eres } D \ C) \rangle$  .
next
  show  $k < \text{Suc } n$ 
    using  $\langle k < \text{Suc } n \rangle$  .
qed
next
have  $\text{ord-res.production } (\text{fset } (N \ |\cup| \ (U_{pr} \ |\cup| \ U_{er} \ |\cup| \ \text{resolvents-upto } D \ C \ k) \ |\cup| \ U_{ef})) \ D =$ 

```

$ord-res.production (fset (N \mid \cup U_{er} \mid \cup U_{ef}) \cup fset (U_{pr} \mid \cup resolvents-upto$
 $D C k)) D$
by (*simp add: funion-left-commute sup-assoc sup-commute*)
also have $\dots = ord-res.production (fset (N \mid \cup U_{er} \mid \cup U_{ef})) D$
proof (*intro production-union-unproductive ballI*)
fix x
assume $x \mid \in U_{pr} \mid \cup resolvents-upto D C k$
hence $\nexists L. is-pos L \wedge ord-res.is-strictly-maximal-lit L x$
unfolding *funion-iff*
proof (*elim disjE*)
assume $x \mid \in U_{pr}$
thus *?thesis*
using *U_{pr}-spec*
by (*metis eres-eq-after-tranclp-ground-resolution nex-strictly-maximal-pos-lit-if-neq-eres*)
next
assume $x \mid \in resolvents-upto D C k$
then obtain i **where** $i \mid \in fset-upto (Suc 0) k$ **and** $x-def: x = resolvent-at$
 $D C i$
unfolding *resolvents-upto-def* **by** *auto*

have $0 < i$ **and** $i \leq k$
using $\langle i \mid \in fset-upto (Suc 0) k \rangle$ **by** *simp-all*

show *?thesis*
unfolding *x-def*
proof (*rule nex-pos-strictly-max-lit-in-resolvent-at*)
show (*ground-resolution D \rightsquigarrow Suc n C (eres D C)*)
using $\langle (ground-resolution D \rightsquigarrow Suc n) C (eres D C) \rangle$.
next
show $i < Suc n$
using $\langle i \leq k \rangle \langle k < Suc n \rangle$ **by** *presburger*
qed
qed
thus $ord-res.production (fset (N \mid \cup U_{er} \mid \cup U_{ef}) \cup$
 $fset (U_{pr} \mid \cup resolvents-upto D C k)) x = \{\}$
using *unproductive-if-nex-strictly-maximal-pos-lit* **by** *metis*
next
show $D \mid \in N \mid \cup U_{er} \mid \cup U_{ef}$
using $\langle D \mid \in N \mid \cup U_{er} \mid \cup U_{ef} \rangle$.
qed *simp-all*
finally show $ord-res.production (fset (N \mid \cup (U_{pr} \mid \cup U_{er} \mid \cup resolvents-upto$
 $D C k) \mid \cup U_{ef})) D =$
 $\{atm-of L\}$
using $\langle ord-res.production (fset (N \mid \cup U_{er} \mid \cup U_{ef})) D = \{atm-of L\} \rangle$ **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* $D \rightsquigarrow \text{Suc } n$) C (*eres* D C)
using $\langle \text{ground-resolution } D \rightsquigarrow \text{Suc } n \rangle C$ (*eres* D C) .
next
show $k < \text{Suc } n$
using $\langle k < \text{Suc } n \rangle$.
qed
next
show $U_{pr} \mid \cup \mid U_{er} \mid \cup \mid \text{finsert} (\text{resolvent-at } D \ C \ (\text{Suc } k)) (\text{resolvents-upto } D \ C \ k) =$
 $\text{finsert} (\text{resolvent-at } D \ C \ (\text{Suc } k)) (U_{pr} \mid \cup \mid U_{er} \mid \cup \mid \text{resolvents-upto } D \ C \ k)$
by *simp*
qed

ultimately show *?case*
by (*meson relpowp-Suc-I*)
qed

hence (*ord-res-2-step* $\rightsquigarrow \text{Suc } n$) $S2$ ($N, U_{pr} \mid \cup \mid U_{er} \mid \cup \mid \text{resolvents-upto } D \ C$
 $(\text{Suc } n), U_{ef}$)
unfolding *match-hyps(1,2)* **by** *blast*

moreover have *match* ($N, U_{pr} \mid \cup \mid U_{er} \mid \cup \mid \text{resolvents-upto } D \ C \ (\text{Suc } n), U_{ef}$)
 $S3'$
proof –
have 1: $S3' = (N, \text{finsert} (\text{eres } D \ C) \ U_{er}, U_{ef})$
unfolding *S3'-def* $\langle s3' = (U_{rr}', U_{ef}) \rangle \langle U_{rr}' = \text{finsert} (\text{eres } D \ C) \ U_{er} \rangle ..$

have 2: $U_{pr} \mid \cup \mid U_{er} \mid \cup \mid \text{resolvents-upto } D \ C \ (\text{Suc } n) =$
 $U_{pr} \mid \cup \mid \text{resolvents-upto } D \ C \ n \mid \cup \mid \text{finsert} (\text{eres } D \ C) \ U_{er}$
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 $\forall E \mid \in \mid U_{pr} \mid \cup \mid \text{resolvents-upto } D \ C \ n.$
 $\exists D1 \mid \in \mid N \mid \cup \mid \text{finsert} (\text{eres } D \ C) \ U_{er} \mid \cup \mid U_{ef}. \exists D2 \mid \in \mid N \mid \cup \mid \text{finsert} (\text{eres}$
 $D \ C) \ U_{er} \mid \cup \mid U_{ef}.$
 $(\text{ground-resolution } D1)^{++} \ D2 \ E \wedge E \neq \text{eres } D1 \ D2 \wedge \text{eres } D1 \ D2 \mid \in \mid$
 $\text{finsert} (\text{eres } D \ C) \ U_{er}$
proof (*intro ballI*)
fix Ca
assume $Ca \mid \in \mid U_{pr} \mid \cup \mid \text{resolvents-upto } D \ C \ n$
hence $Ca \mid \in \mid U_{pr} \vee Ca \mid \in \mid \text{resolvents-upto } D \ C \ n$
by *simp*
thus $\exists D1 \mid \in \mid N \mid \cup \mid \text{finsert} (\text{eres } D \ C) \ U_{er} \mid \cup \mid U_{ef}. \exists D2 \mid \in \mid N \mid \cup \mid \text{finsert}$
 $(\text{eres } D \ C) \ U_{er} \mid \cup \mid U_{ef}.$
 $(\text{ground-resolution } D1)^{++} \ D2 \ Ca \wedge Ca \neq \text{eres } D1 \ D2 \wedge \text{eres } D1 \ D2 \mid \in \mid$
 $\text{finsert} (\text{eres } D \ C) \ U_{er}$

```

proof (elim disjE)
  show  $Ca \in U_{pr} \implies ?thesis$ 
    using  $U_{pr}\text{-spec}$  by auto
next
  assume  $Ca \in \text{resolvents-upto } D \ C \ n$ 
  then obtain  $i$  where  $i\text{-in}: i \in \text{fset-upto } (Suc \ 0) \ n$  and  $Ca\text{-def}: Ca =$ 
  resolvent-at } D \ C \ i
    unfolding resolvents-upto-def by auto

from  $i\text{-in}$  have  $0 < i \ i \leq n$ 
  by simp-all

show  $?thesis$ 
proof (intro beXI conjI)
  have  $(\text{ground-resolution } D \ \sim i) \ C \ Ca$ 
    unfolding  $\langle Ca = \text{resolvent-at } D \ C \ i \rangle$ 
proof (rule relpowp-to-resolvent-at)
  show  $(\text{ground-resolution } D \ \sim Suc \ n) \ C \ (\text{eres } D \ C)$ 
    using  $\langle (\text{ground-resolution } D \ \sim Suc \ n) \ C \ (\text{eres } D \ C) \rangle$  .
next
  show  $i < Suc \ n$ 
    using  $\langle i \leq n \rangle$  by presburger
qed
thus  $(\text{ground-resolution } D)^{++} \ C \ Ca$ 
  using  $\langle 0 < i \rangle$  by (simp add: tranclp-if-relpowp)
next
show  $Ca \neq \text{eres } D \ C$ 
  by (metis  $Ca\text{-def}$   $\langle (\text{ground-resolution } D \ \sim Suc \ n) \ C \ (\text{eres } D \ C) \rangle$ 
   $\langle \nexists x. \text{ground-resolution } D \ (\text{eres } D \ C) \ x \rangle \langle i \leq n \rangle$ 
  ground-resolution-resolvent-at-resolvent-at-Suc less-Suc-eq-le)
next
show  $\text{eres } D \ C \in \text{finsert } (\text{eres } D \ C) \ U_{er}$ 
  by simp
next
show  $D \in N \ \cup \ \text{finsert } (\text{eres } D \ C) \ U_{er} \ \cup \ U_{ef}$ 
  using resolution by simp
next
have  $C \in N \ \cup \ U_{er} \ \cup \ U_{ef}$ 
  using resolution
  by (simp add: is-least-false-clause-def linorder-cls.is-least-in-filter-iff)
thus  $C \in N \ \cup \ \text{finsert } (\text{eres } D \ C) \ U_{er} \ \cup \ U_{ef}$ 
  by simp
qed
qed
qed
qed

```

ultimately have $\exists S2'. (\text{ord-res-2-step } \sim Suc \ n) \ S2 \ S2' \wedge \text{match } S2' \ S3'$

```

    by metis

  thus  $\exists S2'. \text{ord-res-2-step}^{++} S2 S2' \wedge \text{match } S2' S3'$ 
    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  $\equiv \lambda S2 S3. \text{ord-res-2-matches-ord-res-3 } S2 S3$ 
  shows  $\exists (MATCH :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-2-state} \Rightarrow 'f \text{ord-res-3-state} \Rightarrow \text{bool})$ 
 $\mathcal{R}_f \mathcal{R}_b.$ 
    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  $\forall s1. \text{ord-res-2-final } s1 \longrightarrow (\nexists s1'. \text{ord-res-2-step } s1 s1')$ 
        by (metis finished-def ord-res-2-semantics.final-finished)
    next
      show  $\forall s2. \text{ord-res-3-final } s2 \longrightarrow (\nexists s2'. \text{ord-res-3-step } s2 s2')$ 
        by (metis finished-def ord-res-3-semantics.final-finished)
    next
      show  $\forall i s1 s2. \text{match } i s1 s2 \longrightarrow \text{ord-res-2-final } s1 = \text{ord-res-3-final } s2$ 
        unfolding match-def
        using ord-res-2-final-iff-ord-res-3-final by metis
    next
      show  $\forall i s1 s2. \text{match } i s1 s2 \longrightarrow$ 
        safe-state ord-res-2-step ord-res-2-final  $s1 \wedge \text{safe-state ord-res-3-step ord-res-3-final } s2$ 
        unfolding match-def
        using safe-states-if-ord-res-2-matches-ord-res-3 by metis
  next

```

```

show wfP ( $\lambda$ -. False)
  by simp
next
  show  $\forall i s1 s2 s2'$ .
     $match\ i\ s1\ s2 \longrightarrow$ 
     $ord-res-3-step\ s2\ s2' \longrightarrow$ 
     $(\exists i' s1'. ord-res-2-step^{++}\ s1\ s1' \wedge match\ i'\ s1'\ s2') \vee (\exists i'. match\ i'\ s1\ s2'$ 
 $\wedge False)$ 
    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 \times 'f gclause fset \times 'f gclause fset

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

inductive ord-res-3-matches-ord-res-4 :: 'f ord-res-3-state \Rightarrow 'f ord-res-4-state \Rightarrow bool **where**

$\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er} \Longrightarrow U_{ef} = iefac\ \mathcal{F} \mid \uparrow \{ \mid C \mid \in \mid N \mid \cup \mid U_{er}. iefac\ \mathcal{F}\ C \neq C \mid \} \Longrightarrow$
 $ord-res-3-matches-ord-res-4\ (N, (U_{er}, U_{ef}))\ (N, U_{er}, \mathcal{F})$

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 \longleftrightarrow 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 $\{ \# \} \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef} \longleftrightarrow \{ \# \} \mid \in \mid iefac\ \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using invars **by** (auto simp: iefac-def)

moreover have ex-false-clause (fset (N $\mid \cup \mid U_{er} \mid \cup \mid U_{ef}$)) \longleftrightarrow
ex-false-clause (fset (iefac $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$))

unfolding ex-false-clause-iff

unfolding union-union-eq-union-union-fimage-iefac-if[OF invars(2)]

unfolding is-least-false-clause-with-iefac-conv ..

ultimately have ord-res-final (N $\mid \cup \mid U_{er} \mid \cup \mid U_{ef}$) \longleftrightarrow ord-res-final (iefac $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$)

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 $(\exists S4'. \text{ord-res-4-step}^{++} S4 S4' \wedge \text{ord-res-3-matches-ord-res-4 } S3' S4')$

using *match*

proof (*cases S3 S4 rule: ord-res-3-matches-ord-res-4.cases*)

case *match-hyps: (1 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 $\neg \text{ord-res.is-strictly-maximal-lit } L \ C$

using $\langle \text{is-least-false-clause } (N \mid \cup \ U_{er} \mid \cup \ U_{ef}) \ C \rangle \langle \text{ord-res.is-maximal-lit } L \ C \rangle \langle \text{is-pos } L \rangle$

by (*metis (no-types, lifting) is-least-false-clause-def is-pos-def pos-lit-not-greatest-if-maximal-in-least-false-clause*)

have $C \mid \in \mid N \mid \cup \mid U_{er}$

proof –

have $C \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}$

using $\langle \text{is-least-false-clause } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) \ C \rangle$

by (*simp add: is-least-false-clause-def linorder-cls.is-least-in-ffilter-iff*)

moreover **have** $C \mid \notin \mid U_{ef}$

proof (*rule notI*)

assume $C \mid \in \mid U_{ef}$

then **obtain** C_0 **where** $C = \text{iefac } \mathcal{F} \ C_0$ **and** $C_0 \mid \in \mid N \mid \cup \mid U_{er}$ **and** $\text{iefac } \mathcal{F} \ C_0 \neq C_0$

using *match-invars(2)* **by** *force*

then **show** *False*

by (*metis Uniq-D $\langle \neg \text{ord-res.is-strictly-maximal-lit } L \ C \rangle \text{iefac-def}$*

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, Uer, finsert C 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 |∪| Uer |∪| Uef) C
      using factoring by argo
    hence is-least-false-clause (N |∪| Uer |∪| iefac F |∧| (N |∪| Uer)) C
    unfolding funion-funion-eq-funion-funion-fimage-iefac-if[OF match-invars(2)]

    thus is-least-false-clause (iefac F |∧| (N |∪| Uer)) 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, Uer, finsert C F)
  unfolding ⟨S3' = (N, s3')⟩ ⟨s3' = (Uer, Uef^')⟩ ⟨Uef' = finsert (efac C)
  Uef⟩
  proof (rule ord-res-3-matches-ord-res-4.intros)
    show finsert C F |⊆| N |∪| Uer
      using match-invars ⟨C |∈| N |∪| Uer⟩ 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 F) C ≠ C
    by (simp add: iefac-def)

  have { |Ca |∈| N |∪| Uer. iefac (finsert C F) Ca ≠ Ca |} =
    finsert C { |Ca |∈| N |∪| Uer. iefac F Ca ≠ Ca |}
  proof (intro fsubset-antisym fsubsetI)
    fix x
    assume x |∈| { |Ca |∈| N |∪| Uer. iefac (finsert C F) Ca ≠ Ca |}
    hence x |∈| N |∪| Uer and iefac (finsert C F) x ≠ x
      by simp-all
    then show x |∈| finsert C { |Ca |∈| N |∪| Uer. iefac F Ca ≠ Ca |}
      by (smt (verit, best) fmember-filter finsert-iff iefac-def)
  next
  fix x
  assume x |∈| finsert C { |Ca |∈| N |∪| Uer. iefac F Ca ≠ Ca |}
  hence x = C ∨ x |∈| N |∪| Uer ∧ iefac F x ≠ x

```

```

    by auto
  thus  $x \in \{Ca \in N \cup U_{er}. \text{iefac} (\text{finsert } C \mathcal{F}) Ca \neq Ca\}$ 
  proof (elim disjE conjE)
    assume  $x = C$ 
    thus ?thesis
      using  $\langle C \in N \cup U_{er} \rangle \langle \text{iefac} (\text{finsert } C \mathcal{F}) C \neq C \rangle$  by auto
  next
    assume  $x \in N \cup U_{er}$  and  $\text{iefac } \mathcal{F} x \neq x$ 
    thus ?thesis
      by (smt (verit, best) fmember-filter finsertCI iefac-def)
  qed
  qed
  thus  $\text{finsert} (\text{efac } C) U_{ef} =$ 
     $\text{iefac} (\text{finsert } C \mathcal{F}) \uparrow \{Ca \in N \cup U_{er}. \text{iefac} (\text{finsert } C \mathcal{F}) Ca \neq Ca\}$ 
    using iefac-def match-invars(2) by auto
  qed
  qed
next
  case (resolution C L D Urr)

  have  $D \in \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
  proof -
    have  $D \in N \cup U_{er} \cup U_{ef}$ 
      using resolution by argo
    hence  $D \in N \cup U_{er} \cup \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
    unfolding union-union-eq-union-union-fimage-iefac-if[OF match-invars(2)]

    moreover have  $D \notin N \cup U_{er} - \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
    by (metis clauses-for-iefac-are-unproductive insert-not-empty local.resolution( $\gamma$ ))
    ultimately show ?thesis
      by blast
  qed

  show ?thesis
  proof (intro exI conjI)
    show ord-res-4-step++  $S_4 (N, \text{finsert} (\text{eres } D C) U_{er}, \mathcal{F})$ 
      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  $(N \cup U_{er} \cup U_{ef}) C$ 
        using resolution by argo
      hence is-least-false-clause  $(N \cup U_{er} \cup \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) C$ 
      unfolding union-union-eq-union-union-fimage-iefac-if[OF match-invars(2)]

      thus is-least-false-clause  $(\text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) C$ 
        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  $D \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
    using  $\langle D \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}) \rangle$  .
next
  show  $D \prec_c C$ 
    using resolution by argo
next
  have ord-res.production ( $\text{fset } (N \cup U_{er} \cup U_{ef})$ )  $D =$ 
    ord-res.production ( $\text{fset } (N \cup U_{er} \cup \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}))$ )  $D$ 
  unfolding funion-funion-eq-funion-funion-fimage-iefac-if[OF match-invars(2)]
..
  also have  $\dots = \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) \cup \text{fset } (N \cup U_{er})) D$ 
    by (simp add: sup commute)
  also have  $\dots = \text{ord-res.production } (\text{fset } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er}))) D$ 
  proof (rule production-union-unproductive-strong)
    show  $\forall x \in \text{fset } (N \cup U_{er}) - \text{fset } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er}))$ .
      ord-res.production ( $\text{fset } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er})) \cup \text{fset } (N \cup U_{er})$ )  $x$ 
= {}
    using clauses-for-iefac-are-unproductive[of  $N \cup U_{er} \mathcal{F}$ ] by simp
  next
    show  $D \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ 
      using  $\langle D \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}) \rangle$  .
    qed (rule finite-fset)+

  finally show ord-res.production ( $\text{fset } (\text{iefac } \mathcal{F} \uparrow (N \cup U_{er}))$ )  $D =$ 
{atm-of L}
    using resolution by argo
    qed (rule refl)+
next
  show ord-res-3-matches-ord-res-4  $S3' (N, \text{finsert } (\text{eres } D C) U_{er}, \mathcal{F})$ 
    unfolding  $\langle S3' = (N, s3') \rangle \langle s3' = (U_{rr}', U_{ef}) \rangle \langle U_{rr}' = \text{finsert } (\text{eres } D C) U_{er} \rangle$ 
  proof (rule ord-res-3-matches-ord-res-4.intros)
    show  $\mathcal{F} \subseteq | N \cup \text{finsert } (\text{eres } D C) U_{er}$ 
      using match-invars by auto
  next
    show  $U_{ef} = \text{iefac } \mathcal{F} \uparrow \{ | C \in | N \cup \text{finsert } (\text{eres } D C) U_{er}. \text{iefac } \mathcal{F} C \neq C | \}$ 
  proof (cases eres D C  $\in | \mathcal{F}$ )
    case True
      then show ?thesis
        using  $\langle \mathcal{F} \subseteq | N \cup U_{er} \rangle$ 
        using match-invars by force
  next
    case False
      hence iefac  $\mathcal{F} (\text{eres } D C) = \text{eres } D C$ 

```

```

      by (simp add: iefac-def)
    hence  $\{|C| \in |N| \cup |finsert (eres D C) U_{er}. iefac \mathcal{F} C \neq C|\} = \{|C| \in |N| \cup |U_{er}. iefac \mathcal{F} C \neq C|\}$ 
      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  $\equiv \lambda. S3 S4. ord-res-3-matches-ord-res-4 S3 S4$ 
  shows  $\exists (MATCH :: nat \times nat \Rightarrow 'f ord-res-3-state \Rightarrow 'f ord-res-4-state \Rightarrow bool)$ 
 $\mathcal{R}_f \mathcal{R}_b.$ 
  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  $\forall s1. ord-res-3-final s1 \longrightarrow (\nexists s1'. ord-res-3-step s1 s1')$ 
    by (metis finished-def ord-res-3-semantics.final-finished)
  next
  show  $\forall s2. ord-res-4-final s2 \longrightarrow (\nexists s2'. ord-res-4-step s2 s2')$ 
    by (metis finished-def ord-res-4-semantics.final-finished)
  next
  show  $\forall i s1 s2. match i s1 s2 \longrightarrow ord-res-3-final s1 \longleftrightarrow ord-res-4-final s2$ 
    unfolding match-def
    using ord-res-3-final-iff-ord-res-4-final by metis
  next
  show  $\forall i s1 s2. match i s1 s2 \longrightarrow$ 
 $safe-state ord-res-3-step ord-res-3-final s1 \wedge safe-state ord-res-4-step ord-res-4-final$ 
 $s2$ 
    using ord-res-3-step-safe ord-res-4-step-safe
    by (simp add: safe-state-if-all-states-safe)
  next
  show wfp  $(\lambda i' i. False)$ 
    by simp
  next
  show  $\forall i s1 s2 s1'. match i s1 s2 \longrightarrow ord-res-3-step s1 s1' \longrightarrow$ 
 $(\exists i' s2'. ord-res-3-step^{++} s2 s2' \wedge match i' s1' s2') \vee (\exists i'. match i' s1' s2 \wedge$ 
 $False)$ 
    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 $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \implies$
 $(\forall C. C = \text{Some } C \longleftrightarrow \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) C) \implies$
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 $S_4 S_5$

shows ord-res-4-final $S_4 \longleftrightarrow$ ord-res-5-final S_5

using match

proof (cases $S_4 S_5$ 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 $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \vee \neg \text{ex-false-clause } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$

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 $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$

hence is-least-false-clause $(\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})) \{\#\}$

using is-least-false-clause-empty **by** metis

hence $C = \text{Some } \{\#\}$

by (smt (verit) all-smaller-clauses-true-wrt-respective-Interp-def is-least-false-clause-def
linorder-cls.is-least-in-filter-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 $\neg \text{ex-false-clause } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er})))$

hence $C = \text{None}$

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, Uer, F, M, C)
  thus ord-res-4-final (N, Uer, F)
  proof (cases (N, Uer, F, M, C) rule: ord-res-5-final.cases)
    case model-found
      have all-smaller-clauses-true-wrt-respective-Interp N (Uer, F, M, C)
        using ⟨ord-res-5-invars N (Uer, F, M, C)⟩
      unfolding ord-res-5-invars-def by metis
      hence ∀ C |∈| iefac F |↑| (N |∪| Uer). ord-res-Interp (iefac F ‘ (fset N ∪ fset
Uer)) C ⊨ C
        by (simp add: model-found all-smaller-clauses-true-wrt-respective-Interp-def)
      hence ¬ ex-false-clause (fset (iefac F |↑| (N |∪| Uer)))
        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 |∪| Uer)
        using ⟨ord-res-5-invars N (Uer, 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 Uer F M C)
  hence
    S4-def: S4 = (N, Uer, F) and
    S5-def: S5 = (N, Uer, 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 |∪| Uer))) 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 (Uer, F) s4'
  using step unfolding S4-def by (auto simp: ord-res-4-step.simps)

```

```

show ?thesis
using step'
proof (cases N (Uer, 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 C' :: 'f gclause option where
C' = The-optional (linorder-cls.is-least-in-fset (iefac F' |↑ (N |∪| Uer)))

have ord-res-5-step: ord-res-5 N (Uer, F, M, Some C) (Uer, F', M', C')
proof (rule ord-res-5.factoring)
have is-least-false-clause (iefac F' |↑ (N |∪| Uer)) 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-filter-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 ¬ 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 F' = finsert C F
using step-hyps by metis
qed (simp-all add: M'-def C'-def)

moreover have ∃ M'' C''.
(ord-res-5 N)** (Uer, F', M', C') (Uer, F', M'', C'') ∧
(∀ C. (C'' = Some C) ⟷ is-least-false-clause (iefac F' |↑ (N |∪| Uer)) C)
proof (rule ord-res-5-construct-model-upto-least-false-clause)
show ord-res-5-invars N (Uer, F', M', C')
using ord-res-5-step ⟨ord-res-5-invars N (Uer, F, M, C)⟩ ⟨C = Some C⟩
by (metis ord-res-5-preserves-invars)
qed

ultimately obtain M'' C'' where
s5-steps: (ord-res-5 N)++ (Uer, F, M, Some C) (Uer, F', M'', C'') and

```

```

next-clause-least-false:
  (∀ C. (C'' = Some C) ↔ is-least-false-clause (iefac F' |' (N |∪| Uer)) C)
by (meson rtranclp-into-tranclp2)

have ord-res-5-step++ S5 (N, Uer, F', M'', C'')
  unfolding S5-def ⟨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, Uer, F', M'', C'')
  unfolding S4'-def ⟨s4' = (Uer, F')⟩
proof (intro ord-res-4-matches-ord-res-5.intros)
  show ord-res-5-invars N (Uer, F', M'', C'')
    using s5-steps ⟨C = Some C⟩ ⟨ord-res-5-invars N (Uer, 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 |∪| Uer)) C
    using next-clause-least-false .
qed

ultimately show ?thesis
  by metis
next
case step-hyps: (resolution NN C L D Uer')
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 |∪| Uer'))))

have ord-res-5-step: ord-res-5 N (Uer, F, M, Some C) (Uer', F, M', C')
proof (rule ord-res-5.resolution)
  have is-least-false-clause (iefac F |' (N |∪| Uer)) 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 insertII
match-hyps(β)
    ord-res-5-invars-def)
next
show  $U_{er}' = \text{finsert } (\text{eres } D \ C) \ U_{er}$ 
    using step-hyps by metis
qed (simp-all add:  $\mathcal{M}'$ -def  $\mathcal{C}'$ -def)

moreover have  $\exists \mathcal{M}'' \ \mathcal{C}''$ .
  (ord-res-5 N)** ( $U_{er}', \mathcal{F}, \mathcal{M}', \mathcal{C}'$ ) ( $U_{er}', \mathcal{F}, \mathcal{M}'', \mathcal{C}''$ )  $\wedge$ 
  ( $\forall C. (\mathcal{C}'' = \text{Some } C) \longleftrightarrow \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) \ C$ )
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 ( $U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}$ )⟩ ⟨C = Some C⟩
    by (metis ord-res-5-preserves-invars)
qed

ultimately obtain  $\mathcal{M}'' \ \mathcal{C}''$  where
  s5-steps: (ord-res-5 N)** ( $U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C$ ) ( $U_{er}', \mathcal{F}, \mathcal{M}'', \mathcal{C}''$ ) and
  next-clause-least-false:
  ( $\forall C. (\mathcal{C}'' = \text{Some } C) \longleftrightarrow \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) \ C$ )
  by (meson rtranclp-into-tranclp2)

have ord-res-5-step** S5 (N,  $U_{er}', \mathcal{F}, \mathcal{M}'', \mathcal{C}''$ )
  unfolding S5-def ⟨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  $S_4'$  (N,  $U_{er}', \mathcal{F}, \mathcal{M}'', \mathcal{C}''$ )
  unfolding  $S_4'$ -def ⟨ $s_4' = (U_{er}', \mathcal{F})$ ⟩
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 ⟨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  $\forall C. (\mathcal{C}'' = \text{Some } C) = \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}')) \ C$ 
  using next-clause-least-false .
qed

ultimately show ?thesis
  by metis
qed
qed

theorem bisimulation-ord-res-4-ord-res-5:
  defines match  $\equiv \lambda-. \text{ord-res-4-matches-ord-res-5}$ 
  shows  $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-4-state} \Rightarrow 'f \text{ord-res-5-state} \Rightarrow \text{bool})$ 
 $\mathcal{R}_f \ \mathcal{R}_b$ .
  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  $\forall s. \text{ord-res-4-final } s \longrightarrow (\nexists s'. \text{ord-res-4-step } s s')$ 
    by (metis finished-def ord-res-4-semantics.final-finished)
next
  show  $\forall s. \text{ord-res-5-final } s \longrightarrow (\nexists s'. \text{ord-res-5-step } s s')$ 
    by (metis finished-def ord-res-5-semantics.final-finished)
next
  show  $\forall i s_4 s_5. \text{match } i s_4 s_5 \longrightarrow \text{ord-res-4-final } s_4 \longleftrightarrow \text{ord-res-5-final } s_5$ 
    unfolding match-def
    using ord-res-4-final-iff-ord-res-5-final by metis
next
  show  $\forall i S_4 S_5. \text{match } i S_4 S_5 \longrightarrow$ 
    safe-state ord-res-4-step ord-res-4-final S_4  $\wedge$  safe-state ord-res-5-step ord-res-5-final S_5
proof (intro allI impI conjI)
  fix i S_4 S_5
  show safe-state ord-res-4-step ord-res-4-final S_4
    using ord-res-4-step-safe safe-state-if-all-states-safe by metis

  assume match i S_4 S_5
  thus safe-state ord-res-5-step ord-res-5-final S_5
    using  $\langle \text{match } i S_4 S_5 \rangle$ 
    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 ( $\lambda - . \text{False}$ )
    by simp
next
  show  $\forall i s_1 s_2 s_1'. \text{match } i s_1 s_2 \longrightarrow$ 
    ord-res-4-step s_1 s_1' \longrightarrow
     $(\exists i' s_2'. \text{ord-res-5-step}^{++} s_2 s_2' \wedge \text{match } i' s_1' s_2') \vee (\exists i'. \text{match } i' s_1' s_2$ 
 $\wedge \text{False})$ 
    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 × '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-matches-ord-res-6 :: 'f ord-res-5-state ⇒ 'f ord-res-6-state ⇒
 bool **where**

ord-res-5-invars $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \implies$
 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 $\exists S5'. \text{ord-res-5-step}^{++} S5 S5' \wedge \text{ord-res-5-matches-ord-res-6 } S5' S6'$

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 $S5\text{-def}$: $S5 = (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$ **and** $S6\text{-def}$: $S6 = (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$

by metis+

obtain $s6'$ **where** $S6'\text{-def}$: $S6' = (N, s6')$ **and** $step'$: ord-res-6 $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) s6'$

using step unfolding $S6\text{-def}$

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 $C C'$)

define $S5'$ **where**

$S5' = (N, U_{er}, \mathcal{F}, \mathcal{M}, C')$

```

show ?thesis
proof (intro exI conjI)
  have step5: ord-res-5 N (Uer, F, M, Some C) (Uer, F, M, 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 (Uer, 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' = (Uer, 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, Uer, F, M', C')

show ?thesis
proof (intro exI conjI)
  have step5: ord-res-5 N (Uer, F, M, Some C) (Uer, 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 (Uer, 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' = (Uer, 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, Uer, F', M, Some (efac D))

have D ∈| iefac F |↑ (N |∪| Uer)
  by (metis match-hyps(3) next-clause-in-factorized-clause-def ord-res-5-invars-def
step-hyps(1))
hence iefac F' |↑ (N |∪| Uer) ≠ {||}
  by blast

```

then obtain C **where** C -least: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$) C
by (*metis linorder-cls.ex1-least-in-fset*)

have *efac* $D \neq D$
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* $\mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$)) =
Some C
proof (*rule The-optional-eq-SomeI*)
show $\exists_{\leq 1} x. \text{linorder-cls.is-least-in-fset} (\text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er})) x$
by *blast*
next
show *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F}' \mid \uparrow (N \mid \cup U_{er})$) C
using C -least .
qed
hence *step5: ord-res-5* $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) (U_{er}, \mathcal{F}', \text{Map.empty}, \text{Some } C)$
using *ord-res-5.factoring step-hyps* **by** *metis*
moreover have (*ord-res-5* N)** ... ($U_{er}, \mathcal{F}', \mathcal{M}, \text{Some} (\text{efac } D)$)
proof (*rule full-rtranclp-ord-res-5-run-upto*)
show *ord-res-6* $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some} (\text{efac } D))$
using *step' S6-def S6'-def* $\langle s6' = (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some} (\text{efac } D)) \rangle \langle C = \text{Some } D \rangle$ **by** *argo*
next
show *ord-res-5-invars* $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some} (\text{efac } D))$
using *match-hyps(3) ord-res-6-preserved-invars step' step-hyps(2)* **by** *blast*
next
have *iefac* $\mathcal{F} D = D$ **and** $D \in N \mid \cup U_{er}$
unfolding *atomize-conj*
using $\langle \text{efac } D \neq D \rangle \langle D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er}) \rangle$ [*unfolded fimage-iff*]
unfolding *iefac-def*
by (*metis ex1-efac-eq-factoring-chain factorizable-if-neq-efac*)

have *iefac- \mathcal{F}' -eq*: *iefac* $\mathcal{F}' = (\text{iefac } \mathcal{F})(D := \text{efac } D)$
unfolding $\langle \mathcal{F}' = \text{finsert } D \mathcal{F} \rangle$ *iefac-def* **by** *auto*

have *fimage-iefac- \mathcal{F}' -eq*:
iefac $\mathcal{F}' \mid \uparrow (N \mid \cup U_{er}) = \text{finsert} (\text{efac } D) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er} - \{D\}))$
unfolding *iefac- \mathcal{F}' -eq*
unfolding *fun-upd-fimage*[*of iefac* $\mathcal{F} D \text{efac } D$] $\langle D \in N \mid \cup U_{er} \rangle$
using $\langle D \in N \mid \cup U_{er} \rangle$ **by** *argo*

have $\{C \in \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup U_{er}). C \prec_c \text{efac } D\} =$
 $\{C \in \text{finsert} (\text{efac } D) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup U_{er} - \{D\})). C \prec_c \text{efac } D\}$
unfolding *fimage-iefac- \mathcal{F}' -eq* ..

also have ... = $\{|C \in| \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er} - \{|D\}) . C \prec_c \text{efac } D\}$
by *auto*

also have ... = $\{|C \in| \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) . C \prec_c \text{efac } D\}$
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 $\{|C \in| \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}) . C \prec_c \text{efac } D\} =$
 $\{|C \in| \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) . C \prec_c \text{efac } D\} .$

next

have *dom- \mathcal{M} -eq: dom $\mathcal{M} = \text{ord-res.interp (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))$*

D

using *<ord-res-5-invars $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$ > < $\mathcal{C} = \text{Some } D$ >*
unfolding *ord-res-5-invars-def model-eq-interp-upto-next-clause-def*
by *metis*

have *atm-of $K \notin \text{dom } \mathcal{M}$*
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 $A \prec_t \text{atm-of } K$ **if** $A \in \text{dom } \mathcal{M}$ **for** A
proof –

obtain C **where**
 $C \in| \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $C \prec_c D$ **and**
 $A \in \text{ord-res.production (fset (iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C$
using *< $A \in \text{dom } \mathcal{M}$ > unfolding dom- \mathcal{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 $\text{Pos } A \preceq_l K$
using *<ord-res.is-maximal-lit $K D$ > < $C \prec_c D$ >*

by (*metis ord-res.asymp-less-lit ord-res.transp-less-lit linorder-cls.less-asymp linorder-lit.is-maximal-in-mset-if-is-greatest-in-mset linorder-lit.leI linorder-lit.mulp_{HO}-if-maximal-less-that-maximal mulp-eq-mulp_{HO}*)

hence $A \preceq_t \text{atm-of } K$
by (*metis literal.collapse(1) literal.sel(1) ord-res.less-lit-simps(1) reflclp-iff step-hyps(5)*)

moreover have $A \neq \text{atm-of } K$
using *<atm-of $K \notin \text{dom } \mathcal{M}$ > < $A \in \text{dom } \mathcal{M}$ >* **by** *metis*

ultimately show *?thesis*

```

      by order
    qed
  hence  $\text{dom } \mathcal{M} \subseteq \{A. \exists K. \text{ord-res.is-maximal-lit } K \text{ (efac } D) \wedge A \prec_t \text{ atm-of } K\}$ 
    using linorder-lit.is-maximal-in-mset-iff step-hyps(4) by auto
  thus  $\mathcal{M} = \text{restrict-map } \mathcal{M} \{A. \exists K. \text{ord-res.is-maximal-lit } K \text{ (efac } D) \wedge A \prec_t \text{ atm-of } K\}$ 
    using restrict-map-ident-if-dom-subset by fastforce
  next
  show linorder-cls.is-least-in-fset (iefac  $\mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er})$ ) C
    using C-least .
  qed
  ultimately have steps5: (ord-res-5 N)++ (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some D) (Uer,  $\mathcal{F}'$ ,  $\mathcal{M}$ , Some (efac D))
    by simp
  thus 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 ord-res-5-invars N (Uer,  $\mathcal{F}'$ ,  $\mathcal{M}$ , Some (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  $\langle s6' = (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some (efac D)}) \rangle$ 
    using ord-res-5-matches-ord-res-6.intros by metis
  qed
  next
  case step-hyps: (resolution-bot C L D Uer'  $\mathcal{M}'$ )

  define S5' :: - × - × - × (f gterm ⇒ f gclause option) × f gclause option
  where
     $S5' = (N, U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } \{\#\})$ 

  show ?thesis
  proof (intro exI conjI)
  have  $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')$ 
    using  $\langle U_{er}' = \text{finsert (eres D C) } U_{er} \rangle \langle \text{eres D C} = \{\#\} \rangle$ 
    using iefac-def by simp

  hence linorder-cls.is-least-in-fset (iefac  $\mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')$ )  $\{\#\}$ 
    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 The-optional (linorder-cls.is-least-in-fset (iefac  $\mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}')$ )) = Some  $\{\#\}$ 
    by (metis linorder-cls.Uniq-is-least-in-fset The-optional-eq-SomeI)

  hence step5: ord-res-5 N (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some C) (Uer',  $\mathcal{F}$ ,  $\mathcal{M}'$ , Some  $\{\#\})$ 
    using ord-res-5.resolution step-hyps by metis

```

```

thus ord-res-5-step++ S5 S5'
  unfolding S5-def ⟨C = 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 Uer' M' K)

  define S5' :: - × - × - × ('f gterm ⇒ 'f gclause option) × 'f gclause option
where
  S5' = (N, Uer', F, M', Some (eres D E))

  hence iefac F |↑ (N |∪| Uer') ≠ {||}
  using ⟨Uer' = finsert (eres D E) Uer⟩ by simp
  then obtain C where C-least: linorder-cls.is-least-in-fset (iefac F |↑ (N |∪|
    Uer')) 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 F |↑ (N |∪| Uer'))) =
    Some C
    proof (rule The-optional-eq-SomeI)
      show ∃≤1 x. linorder-cls.is-least-in-fset (iefac F |↑ (N |∪| Uer')) x
      by blast
    next
      show linorder-cls.is-least-in-fset (iefac F |↑ (N |∪| Uer')) C
      using C-least .
    qed

  hence step5: ord-res-5 N (Uer, F, M, Some E) (Uer', F, Map.empty, Some
    C)
    using ord-res-5.resolution step-hyps by metis

  moreover have (ord-res-5 N)** ... (Uer', F, M', Some (eres D E))
  proof (rule full-rtranclp-ord-res-5-run-upto)
    show ord-res-6 N (Uer, F, M, Some E) (Uer', F, M', Some (eres D E))
    using step' ⟨C = Some E⟩ ⟨s6' = (Uer', F, M', Some (eres D E))⟩ by
    argo
  next
    show ord-res-5-invars N (Uer', F, 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 $\text{eres } D \ E \preceq_c \ E$
using *eres-le* .

ultimately have $\text{eres } D \ E \prec_c \ E$
by *order*

have $\forall F. \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \ F \longrightarrow E \preceq_c \ F$
using $\langle \text{ord-res-5-invars } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \rangle$
unfolding *ord-res-5-invars-def* $\langle \mathcal{C} = \text{Some } E \rangle$
using *next-clause-lt-least-false-clause*[*of* $N \ (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } E)$]
by *simp*

have *E-least-false: is-least-false-clause* $(\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \ E$
unfolding *is-least-false-clause-def linorder-cls.is-least-in-filter-iff*
proof (*intro conjI ballI impI*)
show $E \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
using $\langle \text{ord-res-5-invars } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \rangle$
unfolding *ord-res-5-invars-def* $\langle \mathcal{C} = \text{Some } E \rangle$
by (*metis next-clause-in-factorized-clause-def*)

next
have $\neg \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ E \models E$
using $\langle \text{ord-res-5-invars } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \rangle$
unfolding *ord-res-5-invars-def* $\langle \mathcal{C} = \text{Some } E \rangle$
using $\langle \neg \text{dom } \mathcal{M} \models E \rangle$ **by** (*metis model-eq-interp-upto-next-clause-def*)

moreover have *ord-res.production* $(\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ E = \{\}$
proof –
have $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L \ E$
using $\langle \text{ord-res.is-maximal-lit } L \ E \rangle \langle \text{is-neg } L \rangle$
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 $\neg \text{ord-res.Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ E \models E$
by *simp*

next
fix F
assume *F-in*: $F \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and** $F \neq E$ **and**
F-false: $\neg \text{ord-res.Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) \ F \models F$
have $\neg F \prec_c \ E$
using $\langle \text{ord-res-5-invars } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \rangle$
unfolding *ord-res-5-invars-def* $\langle \mathcal{C} = \text{Some } E \rangle$
unfolding *all-smaller-clauses-true-wrt-respective-Interp-def*
using *F-in F-false*
by (*metis option.inject*)

thus $E \prec_c \ F$
using $\langle F \neq E \rangle$ **by** *order*

qed

have D -prod: *ord-res.production* (*fset* (*iefac* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}$))) $D \neq \{\}$
using *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$
unfolding *ord-res-5-invars-def* $\langle \mathcal{C} = \text{Some } E \rangle$
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*)
 $\langle \forall F. \text{is-least-false-clause} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) F \longrightarrow (\prec_c)^{==} E F \rangle$
 $\langle \text{eres } D E \prec_c E \rangle$ *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 $\{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \cdot C \prec_c \text{eres } D E|\} =$
 $\{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}) \cdot C \prec_c \text{eres } D E|\}$
unfolding $\langle U_{er}' = \text{finsert} (\text{eres } D E) U_{er} \rangle$ **by** *auto*

next
show $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. \exists K. \text{ord-res.is-maximal-lit } K (\text{eres } D E)$
 $\wedge A \prec_t \text{atm-of } K\}$
using $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \rangle$
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* \mathcal{F} | \uparrow ($N \mid \cup \mid U_{er}'$)) C
using *C-least* .

qed

ultimately have *steps5*: (*ord-res-5* N)⁺⁺ ($U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } E$) ($U_{er}', \mathcal{F},$
 $\mathcal{M}', \text{Some} (\text{eres } D E)$)
by *simp*

thus *ord-res-5-step*⁺⁺ $S5 S5'$
unfolding *S5-def* $\langle \mathcal{C} = \text{Some } E \rangle$ *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* $\langle s6' = (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some} (\text{eres } D E)) \rangle$
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 F |↑ (N |∪| U_{er}')) = Some B*
proof (*rule The-optional-eq-SomeI*)
show *∃_{≤1} x. linorder-cls.is-least-in-fset (iefac F |↑ (N |∪| U_{er}') x*
by *blast*
next
show *linorder-cls.is-least-in-fset (iefac F |↑ (N |∪| U_{er}') B*
using *B-least .*
qed

hence *step5: ord-res-5 N (U_{er}, F, M, Some E) (U_{er}', F, Map.empty, Some B)*
using *ord-res-5.resolution step-hyps by metis*

moreover have (*ord-res-5 N*)** ... (*U_{er}'*, *F, M'*, *Some C*)
proof (*rule full-rtranclp-ord-res-5-run-upto*)
show *ord-res-6 N (U_{er}, F, M, Some E) (U_{er}', F, M', Some C)*
using *step' ⟨C = Some E⟩ ⟨s6' = (U_{er}', F, M', Some C)⟩ by argo*
next
show *ord-res-5-invars N (U_{er}', F, 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 *⟨M (atm-of K) = Some C⟩*
⟨ord-res-5-invars N (U_{er}, F, M, 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) <_i K*
using *⟨is-neg K⟩ by (cases K) simp-all*

ultimately have *C <_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.mulp_{HO}-if-maximal-less-that-maximal mulp-eq-mulp_{HO})

hence $C \prec_c E$
using *eres-le[of D E]* **by** *order*

have $C \prec_c \text{efac } (eres D E)$
by (*metis Uniq-D* $\langle C \prec_c eres D E \rangle$ *efac-spec is-pos-def linorder-lit.Uniq-is-maximal-in-mset*
step-hyps(10) step-hyps(11))

moreover have $\text{efac } (eres D E) \preceq_c eres D E$
by (*metis efac-subset subset-implies-reflclp-mulp*)

ultimately have $C \prec_c \text{iefac } \mathcal{F} (eres D E)$
unfolding *iefac-def* **by** *auto*

hence $\{ |Ca | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}') . Ca \prec_c C | \} =$
 $\{ |Ca | \in | \text{iefac } \mathcal{F} | \uparrow (N | \cup | U_{er}) . Ca \prec_c C | \}$
unfolding $\langle U_{er}' = \text{finsert } (eres D E) U_{er} \rangle$ **by** *auto*

have $(\exists K . \text{ord-res.is-maximal-lit } K C \wedge A \prec_t \text{atm-of } K) \longleftrightarrow A \prec_t \text{atm-of}$
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 $\{ A . \exists K . \text{ord-res.is-maximal-lit } K C \wedge A \prec_t \text{atm-of } K \} = \{ A . A \prec_t$
atm-of K
by *metis*

thus $\mathcal{M}' = \text{restrict-map } \mathcal{M} \{ A . \exists K . \text{ord-res.is-maximal-lit } K C \wedge A \prec_t$
atm-of K
using $\langle \mathcal{M}' = \text{restrict-map } \mathcal{M} \{ A . A \prec_t \text{atm-of } K \} \rangle$ **by** *argo*

next
show *linorder-cls.is-least-in-fset (iefac F | ↑ (N | ∪ | U_{er}')) B*
using *B-least .*

qed

ultimately have *steps5: (ord-res-5 N)⁺⁺ (U_{er}, F, M, Some E) (U_{er}', F,*
M', Some C)
by *simp*

thus *ord-res-5-step⁺⁺ S5 S5'*
unfolding *S5-def* $\langle C = \text{Some } E \rangle$ *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* $\langle s6' = (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } C) \rangle$
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  $\equiv \lambda-. \text{ord-res-5-matches-ord-res-6}$ 
  shows  $\exists (MATCH :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-5-state} \Rightarrow 'f \text{ord-res-6-state} \Rightarrow \text{bool})$ 
 $\mathcal{R}_f \mathcal{R}_b.$ 
  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  $\forall s. \text{ord-res-5-final } s \longrightarrow (\nexists s'. \text{ord-res-5-step } s s')$ 
    by (metis finished-def ord-res-5-semantics.final-finished)
next
  show  $\forall s. \text{ord-res-6-final } s \longrightarrow (\nexists s'. \text{ord-res-6-step } s s')$ 
    by (metis finished-def ord-res-6-semantics.final-finished)
next
  show  $\forall i S5 S6. \text{match } i S5 S6 \longrightarrow \text{ord-res-5-final } S5 \longleftrightarrow \text{ord-res-6-final } S6$ 
    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  $\langle \text{match } i S5 S6 \rangle$ 
      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  $\langle \text{match } i S5 S6 \rangle$ 
      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  $(\lambda-. \text{False})$ 
    by simp
next

```

```

show  $\forall i S5 S6 S6'$ .
  match  $i S5 S6 \longrightarrow$ 
  ord-res-6-step  $S6 S6' \longrightarrow$ 
   $(\exists i' S5'. \text{ord-res-5-step}^{++} S5 S5' \wedge \text{match } i' S5' S6') \vee (\exists i'. \text{match } i' S5$ 
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  $\times$  'f gclause fset  $\times$  'f gclause fset  $\times$  ('f gliteral  $\times$  'f gclause option)
list  $\times$ 
  'f gclause option

```

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

```

inductive ord-res-6-matches-ord-res-7 ::
  'f gterm fset  $\Rightarrow$  'f ord-res-6-state  $\Rightarrow$  'f ord-res-7-state  $\Rightarrow$  bool where
  ord-res-5-invars  $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \Longrightarrow$ 
  ord-res-7-invars  $N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \Longrightarrow$ 
   $(\forall A C. \mathcal{M} A = \text{Some } C \longleftrightarrow \text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C)) \Longrightarrow$ 
   $(\forall A. \mathcal{M} A = \text{None} \longleftrightarrow \text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma) \Longrightarrow$ 
   $i = \text{atms-of-clss } (N \mid \cup \mid U_{er}) - \text{trail-atms } \Gamma \Longrightarrow$ 
  ord-res-6-matches-ord-res-7  $i (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ 

```

```

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  $S6 \longleftrightarrow$  ord-res-7-final  $S7$ 
  using match
proof (cases  $i S6 S7$  rule: ord-res-6-matches-ord-res-7.cases)
  case match-hyps:  $(1 N U_{er} \mathcal{F} \mathcal{M} \mathcal{C} \Gamma)$ 

```

```

show ord-res-6-final  $S6 \longleftrightarrow$  ord-res-7-final  $S7$ 
proof (rule iffI)
  assume ord-res-6-final  $S6$ 
  thus ord-res-7-final  $S7$ 
  unfolding  $\langle S6 = (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \rangle$ 
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  $\langle S7 = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$ 
  using ord-res-7-final.model-found

```

```

    by metis
  next
  case contradiction-found
  thus ord-res-7-final S7
    unfolding ⟨S7 = (N, Uer, F, Γ, 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, Uer, F, Γ, C)⟩
proof (cases (N, Uer, F, Γ, C) rule: ord-res-7-final.cases)
  case model-found
  thus ord-res-6-final S6
    unfolding ⟨S6 = (N, Uer, 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, Uer, F, M, C)⟩
    using ord-res-6-final.contradiction-found
    by metis
  qed
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' |C| i)
  using match
proof (cases i S6 S7 rule: ord-res-6-matches-ord-res-7.cases)
  case match-hyps: (1 N Uer F M C Γ)

  note S6-def = ⟨S6 = (N, Uer, F, M, C)⟩
  note invars-6 = ⟨ord-res-5-invars N (Uer, F, M, C)⟩
  note invars-7 = ⟨ord-res-7-invars N (Uer, 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* $\mathcal{M} \models D$
if *D-true*: *trail-true-cls* Γ *D* **for** *D*

proof –

obtain *L* **where** $L \in \# D$ **and** *L-true*: *trail-true-lit* Γ *L*
using *D-true* **unfolding** *trail-true-cls-def* **by** *auto*

have $\exists C. (L, C) \in \text{set } \Gamma$
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*) $\in \text{set } \Gamma$
using *invars-7* $\langle \exists C. (L, C) \in \text{set } \Gamma \rangle$
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 $\mathcal{M} A = \text{Some } C$
using $\langle \forall A C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C)) \rangle$

by *metis*

hence $A \in \text{dom } \mathcal{M}$
by *blast*

then show *?thesis*
using $\langle L \in \# D \rangle$ $\langle L = \text{Pos } A \rangle$ **by** *blast*

next

case (*Neg A*)

hence (*Neg A*, *None*) $\in \text{set } \Gamma$
using *invars-7* $\langle \exists C. (L, C) \in \text{set } \Gamma \rangle$
by (*metis fst-conv literal.disc(2) snd-conv*)

hence *map-of* Γ (*Neg A*) $\neq \text{None}$
by (*simp add: weak-map-of-SomeI*)

hence $\mathcal{M} A = \text{None}$
using $\langle \forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma) \rangle$

by *metis*

hence $A \notin \text{dom } \mathcal{M}$
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

  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⟩ 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⟩ by blast

```

hence $\text{map-of } \Gamma \text{ (Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma$
using $\langle \forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma \text{ (Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma) \rangle$ by *blast*

hence $\text{trail-true-lit } \Gamma \text{ (Neg } A) \vee \neg \text{trail-defined-lit } \Gamma \text{ (Neg } A)$
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 $\langle \neg \text{trail-true-lit } \Gamma L \rangle \langle \text{trail-defined-lit } \Gamma L \rangle \langle L = \text{Neg } A \rangle$ by *argo*
qed
qed

obtain $s\gamma'$ where

$S\gamma' = (N, s\gamma')$ and
 $\text{step}' : \text{ord-res-}\gamma \ N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \ s\gamma'$
using *step unfolding* $\langle S\gamma' = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$
by *(auto elim: constant-context.cases)*

have $\text{invars-}s\gamma' : \text{ord-res-}\gamma\text{-invars } N \ s\gamma'$
using *ord-res-}\gamma\text{-preserves-invars[OF step' ord-res-}\gamma\text{-invars } N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})]*

show *?thesis*

using *step'*

proof *(cases N (U_{er}, F, Γ, C) sγ' rule: ord-res-γ.cases)*

case *step-hyps: (decide-neg C L A Γ')*

define i' where

$i' = \text{atms-of-clss } (N \cup U_{er}) \ \text{atms } \Gamma'$

have $A \in \text{atms-of-clss } (N \cup U_{er})$ and $A \prec_t \text{atm-of } L$ and $A \notin \text{trail-atms } \Gamma$

using *step-hyps unfolding atomize-conj linorder-trm.is-least-in-filter-iff* by *argo*

have $\text{ord-res-6-matches-ord-res-}\gamma \ i' \ S6 \ S\gamma'$

unfolding *S6-def* $\langle \mathcal{C} = \text{Some } C \rangle \langle S\gamma' = (N, s\gamma') \rangle \langle s\gamma' = (U_{er}, \mathcal{F}, \Gamma', \text{Some } C) \rangle$

proof *(rule ord-res-6-matches-ord-res-γ.intros)*

show $\text{ord-res-5-invars } N \ (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C)$

using *invars-6 unfolding* $\langle \mathcal{C} = \text{Some } C \rangle$.

next

show $\text{ord-res-}\gamma\text{-invars } N \ (U_{er}, \mathcal{F}, \Gamma', \text{Some } C)$

using *invars-}s\gamma' unfolding* $\langle s\gamma' = (U_{er}, \mathcal{F}, \Gamma', \text{Some } C) \rangle$.

next

show $\forall A \ C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma' \text{ (Pos } A) = \text{Some } (\text{Some } C))$

using *match-hyps unfolding* $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ by *simp*

```

next
  show  $\forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms}$ 
 $\Gamma')$ 
    unfolding  $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$ 
    using match-hyps  $\langle A \notin \text{trail-atms } \Gamma \rangle$  by force
next
  show  $i' = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \mid - \mid \text{trail-atms } \Gamma'$ 
    unfolding i'-def ..
qed

moreover have  $i' \mid \subset \mid i$ 
proof -
  have  $i = \text{finsert } A \ i'$ 
    unfolding match-hyps i'-def
    using  $\langle A \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle \langle A \notin \text{trail-atms } \Gamma \rangle$  step-hyps(6) by
force

    moreover have  $A \notin \mid i'$ 
      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  $C \ L \ C'$ )

  define  $S6'$  where
     $S6' = (N, U_{er}, \mathcal{F}, \mathcal{M}, C')$ 

  have C-almost-defined: trail-defined-cls  $\Gamma \ \{\#x \in \# \ C. \ x \neq L\# \}$ 
    using step-hyps by (metis clause-almost-definedI invars-7)

  hence C-defined: trail-defined-cls  $\Gamma \ C$ 
    using step-hyps unfolding trail-defined-cls-def by auto

  hence C-true: trail-true-cls  $\Gamma \ C$ 
    using step-hyps by (metis trail-true-or-false-cls-if-defined)

  have step6: ord-res-6  $N \ (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \ (U_{er}, \mathcal{F}, \mathcal{M}, C')$ 
proof (rule ord-res-6.skip)
  show  $\text{dom } \mathcal{M} \models C$ 
    using clause-true-wrt-model-if-true-wrt- $\Gamma$ [OF C-true] .
next
  show  $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset}$ 
 $(\text{filter } ((\prec_c) \ C) \ (\text{iefac } \mathcal{F} \ \mid \mid (N \ \mid \cup \mid U_{er}))))$ 

```

```

    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' = (Uer, F, Γ, C')⟩
  proof (rule ord-res-6-matches-ord-res-7.intros)
    show ord-res-5-invars N (Uer, F, M, 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 (Uer, F, Γ, C')
        using invars-s7' unfolding ⟨s7' = (Uer, 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-cls (N |∪| Uer) |−| 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, Uer, F, M, C')

  define i' where
    i' = atms-of-cls (N |∪| Uer) |−| 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 (Uer, F, M, Some C) (Uer, 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  $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset}
    (\text{ffilter } ((\prec_c) C) (\text{iefac } \mathcal{F} \mid \mid (N \mid \cup \mid U_{er}))))$ 
using step-hyps by argo
qed

hence  $\text{ord-res-6-step}^{++} S6 S6'$ 
using S6-def  $\langle C = \text{Some } C \rangle S6'\text{-def ord-res-6-step.intros}$  by blast

moreover have  $\text{ord-res-6-matches-ord-res-7 } i' S6' S7'$ 
unfolding S6'-def  $\langle S7' = (N, s7') \rangle \langle s7' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$ 
proof (rule ord-res-6-matches-ord-res-7.intros)
show  $\text{ord-res-5-invars } N (U_{er}, \mathcal{F}, \mathcal{M}, C')$ 
using invars-6 unfolding  $\langle C = \text{Some } C \rangle$ 
using ord-res-6-preserves-invars[OF step6] by argo
next
show  $\text{ord-res-7-invars } N (U_{er}, \mathcal{F}, \Gamma', C')$ 
using invars-s7' unfolding  $\langle s7' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$  .
next
show  $\forall A C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma' (\text{Pos } A) = \text{Some } (\text{Some } C))$ 
using match-hyps
unfolding  $\langle \Gamma' = (L, \text{None}) \# \Gamma \rangle$ 
by (metis literal.disc(1) map-of-Cons-code(2) step-hyps(7))
next
show  $\forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma')$ 
using match-hyps
unfolding  $\langle \Gamma' = (L, \text{None}) \# \Gamma \rangle$ 
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  $i' = \text{atms-of-cls } (N \mid \cup \mid U_{er}) \mid - \mid \text{trail-atms } \Gamma'$ 
using i'-def .
qed

ultimately show ?thesis
by metis
next
case step-hyps: (skip-undefined-pos C L D)

define S6' where
S6' = (N, Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some D)

have trail-defined-cls  $\Gamma \{ \#x \in \# C. x \neq L \wedge x \neq - L \# \}$ 

```

```

proof (rule clause-almost-almost-definedI)
  show  $C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
    using invars-7 step-hyps by metis
next
  show ord-res.is-maximal-lit L C
    using step-hyps by argo
next
  show  $\neg (\exists A \in \text{atms-of-cls} (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma)$ 
    using step-hyps by argo
qed

moreover have  $- L \notin \# C$ 
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  $\Gamma \{ \#x \in \# C. x \neq L \# \}$ 
  unfolding trail-defined-cls-def by auto

hence trail-true-cls  $\Gamma \{ \#x \in \# C. x \neq L \# \}$ 
using  $\langle \neg \text{trail-false-cls } \Gamma \{ \#K \in \# C. K \neq L \# \} \rangle$  by (metis trail-true-or-false-cls-if-defined)

hence C-true: trail-true-cls  $\Gamma C$ 
  by (auto simp: trail-true-cls-def)

have step6: ord-res-6 N (Uer, F, M, Some C) (Uer, F, M, Some D)
proof (rule ord-res-6.skip)
  show dom M  $\models C$ 
    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 F |↑ (N |∪| Uer)))))
    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  $\langle C = \text{Some } C \rangle$  S6'-def ord-res-6-step.intros by blast

moreover have ord-res-6-matches-ord-res-7 i S6' S7'
  unfolding S6'-def  $\langle S7' = (N, s7') \rangle$   $\langle s7' = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$ 
proof (rule ord-res-6-matches-ord-res-7.intros)
  show ord-res-5-invars N (Uer, F, M, Some D)
    using invars-6 unfolding  $\langle C = \text{Some } C \rangle$ 
    using ord-res-6-preserves-invars[OF step6] by argo
next
  show ord-res-7-invars N (Uer, F, Γ, Some D)
    using invars-s7' unfolding  $\langle s7' = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \rangle$  .
next

```

```

    show  $\forall A C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C))$ 
      using match-hyps by argo
  next
  show  $\forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms}$ 
 $\Gamma)$ 
    using match-hyps by argo
  next
  show  $i = \text{atms-of-cls} (N \cup U_{er}) \dashv \text{trail-atms } \Gamma$ 
    using match-hyps by argo
  qed

  ultimately show ?thesis
    by metis
  next
  case step-hyps: (skip-undefined-pos-ultimate  $C L \Gamma'$ )

  define  $S6'$  where
     $S6' = (N, U_{er}, \mathcal{F}, \mathcal{M}, \text{None} :: 'f \text{ gclause option})$ 

  define  $i'$  where
     $i' = \text{atms-of-cls} (N \cup U_{er}) \dashv \text{trail-atms } \Gamma'$ 

  have trail-defined-cls  $\Gamma \{ \#x \in \# C. x \neq L \wedge x \neq -L \# \}$ 
  proof (rule clause-almost-almost-definedI)
    show  $C \in \text{iefac } \mathcal{F} \upharpoonright (N \cup U_{er})$ 
      using invars-7 step-hyps by metis
  next
  show ord-res.is-maximal-lit  $L C$ 
    using step-hyps by argo
  next
  show  $\neg (\exists A \in \text{atms-of-cls} (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma)$ 
    using step-hyps by argo
  qed

  moreover have  $-L \notin \# C$ 
  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  $\Gamma \{ \#x \in \# C. x \neq L \# \}$ 
  unfolding trail-defined-cls-def by auto

  hence trail-true-cls  $\Gamma \{ \#x \in \# C. x \neq L \# \}$ 
  using  $\langle \neg \text{trail-false-cls } \Gamma \{ \#K \in \# C. K \neq L \# \} \rangle$  by (metis trail-true-or-false-cls-if-defined)

  hence C-true: trail-true-cls  $\Gamma C$ 
  by (auto simp: trail-true-cls-def)

  have step6: ord-res-6  $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, \text{None})$ 

```

```

proof (rule ord-res-6.skip)
  show dom  $\mathcal{M} \models C$ 
    using clause-true-wrt-model-if-true-wrt- $\Gamma$ [OF C-true] .
next
  have  $\neg (\exists D. \text{linorder-cls.is-least-in-fset} (\text{ffilter} ((\prec_c) C) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) D)$ 
    using  $\langle \neg \text{fBex} (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) ((\prec_c) C) \rangle$ 
    by (meson linorder-cls.is-least-in-ffilter-iff)

  thus None = The-optional (linorder-cls.is-least-in-fset
    (ffilter (( $\prec_c$ ) C) (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))))$ )
    unfolding The-optional-def by metis
qed

hence ord-res-6-step++ S6 S6'
  using S6-def  $\langle C = \text{Some } C \rangle$  S6'-def ord-res-6-step.intros by blast

moreover have ord-res-6-matches-ord-res-7 i' S6' S7'
  unfolding S6'-def  $\langle S7' = (N, s7') \rangle \langle s7' = (U_{er}, \mathcal{F}, \Gamma', \text{None}) \rangle$ 
proof (rule ord-res-6-matches-ord-res-7.intros)
  show ord-res-5-invars N (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , None)
    using invars-6 unfolding  $\langle C = \text{Some } C \rangle$ 
    using ord-res-6-preserves-invars[OF step6] by argo
next
  show ord-res-7-invars N (Uer,  $\mathcal{F}$ ,  $\Gamma'$ , None)
    using invars-s7' unfolding  $\langle s7' = (U_{er}, \mathcal{F}, \Gamma', \text{None}) \rangle$  .
next
  show  $\forall A C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma' (\text{Pos } A) = \text{Some } (\text{Some } C))$ 
    using match-hyps(3-)
    unfolding  $\langle \Gamma' = (- L, \text{None}) \# \Gamma \rangle$ 
  by (metis is-pos-neg-not-is-pos literal.disc(1) map-of-Cons-code(2) step-hyps(7))
next
  show  $\forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma')$ 
    using match-hyps(3-)
    unfolding  $\langle \Gamma' = (- L, \text{None}) \# \Gamma \rangle$ 
  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-cls (N  $\mid \cup \mid U_{er}$ )  $\mid - \mid \text{trail-atms } \Gamma'$ 
    using i'-def .
qed

ultimately show ?thesis
  by metis
next
  case step-hyps: (production C L  $\Gamma'$  C')

```


define $S6'$ **where**
 $S6' = (N, U_{er}, \mathcal{F}, \mathcal{M}(\text{atm-of } L \mapsto C), C')$

define i' **where**
 $i' = \text{atms-of-cls } (N \mid \cup \mid U_{er}) \mid - \mid \text{trail-atms } \Gamma'$

have $L \in \# C$
using *step-hyps* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

have *step6*: *ord-res-6* $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}(\text{atm-of } L \mapsto C), C')$

proof (*rule ord-res-6.production*)
have $\text{atm-of } L \notin \mid \text{trail-atms } \Gamma$
using $\langle \neg \text{trail-defined-lit } \Gamma L \rangle$
unfolding *trail-defined-lit-iff-trail-defined-atm* .

hence $\mathcal{M}(\text{atm-of } L) = \text{None}$
using *match-hyps(3-)* **by** *metis*

hence $\text{atm-of } L \notin \text{dom } \mathcal{M}$
unfolding *dom-def* **by** *simp*

hence $\neg \text{dom } \mathcal{M} \models L$
using $\langle \text{is-pos } L \rangle$ **unfolding** *true-lit-def* **by** *metis*

moreover have $\neg \text{dom } \mathcal{M} \models \{\#K \in \# C. K \neq L\# \}$
using *clause-false-wrt-model-if-false-wrt-Γ[OF ⟨trail-false-cls Γ {#K ∈ # C. K ≠ L#}⟩]* .

ultimately show $\neg \text{dom } \mathcal{M} \models C$
using $\langle L \in \# C \rangle$
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 $\mathcal{M}(\text{atm-of } L \mapsto C) = \mathcal{M}(\text{atm-of } L \mapsto C) ..$

next
show $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) C) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))))$
using *step-hyps* **by** *argo*

qed

hence *ord-res-6-step*⁺⁺ *S6 S6'*
using *S6-def* $\langle C = \text{Some } C \rangle$ *S6'-def* *ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7* *i' S6' S7'*
unfolding *S6'-def* $\langle S7' = (N, s7') \rangle \langle s7' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}(\text{atm-of } L \mapsto C), C')$
using *invars-6* **unfolding** $\langle C = \text{Some } C \rangle$
using *ord-res-6-preserves-invars*[*OF step6*] **by** *argo*

next
show *ord-res-7-invars* $N (U_{er}, \mathcal{F}, \Gamma', C')$
using *invars-s7'* **unfolding** $\langle s7' = (U_{er}, \mathcal{F}, \Gamma', C') \rangle$.

next
show $\forall A D. ((\mathcal{M}(\text{atm-of } L \mapsto C)) A = \text{Some } D) = (\text{map-of } \Gamma' (\text{Pos } A) = \text{Some } (\text{Some } D))$
using *match-hyps*(3-)

unfolding $\langle \Gamma' = (L, \text{Some } C) \# \Gamma \rangle$
using *step-hyps*(7) **by** *auto*

next
show $\forall A. ((\mathcal{M}(\text{atm-of } L \mapsto C)) A = \text{None}) = (\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma')$
using *match-hyps*(3-)

unfolding $\langle \Gamma' = (L, \text{Some } C) \# \Gamma \rangle$
by (*metis* (*no-types*, *opaque-lifting*) *domI* *domIff* *fininsert-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 $i' = \text{atms-of-clss } (N \cup U_{er}) \mid - \mid \text{trail-atms } \Gamma'$
using *i'-def* .

qed

ultimately show *?thesis*
by *metis*

next
case *step-hyps*: (*factoring* *C L F'*)

define *S6'* **where**
 $S6' = (N, U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (\text{efac } C))$

have $L \in \# C$
using *step-hyps* **unfolding** *linorder-lit.is-maximal-in-mset-iff* **by** *argo*

have *step6*: *ord-res-6* $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (\text{efac } C))$
proof (*rule ord-res-6.factoring*)
have *atm-of* $L \notin \text{trail-atms } \Gamma$
using $\langle \neg \text{trail-defined-lit } \Gamma L \rangle$

unfolding *trail-defined-lit-iff-trail-defined-atm* .

hence $\mathcal{M} \text{ (atm-of } L) = \text{None}$
using *match-hyps(3-)* **by** *metis*

hence $\text{atm-of } L \notin \text{dom } \mathcal{M}$
unfolding *dom-def* **by** *simp*

hence $\neg \text{dom } \mathcal{M} \models L$
using $\langle \text{is-pos } L \rangle$ **unfolding** *true-lit-def* **by** *metis*

moreover have $\neg \text{dom } \mathcal{M} \models \{\#K \in\# C. K \neq L\# \}$
using *clause-false-wrt-model-if-false-wrt- Γ [OF $\langle \text{trail-false-cls } \Gamma \{\#K \in\# C. K \neq L\# \} \rangle$]* .

ultimately show $\neg \text{dom } \mathcal{M} \models C$
using $\langle L \in\# C \rangle$
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 $\neg \text{ord-res.is-strictly-maximal-lit } L C$
using *step-hyps* **by** *argo*

next
show $\mathcal{F}' = \text{finsert } C \mathcal{F}$
using *step-hyps* **by** *argo*

qed

hence *ord-res-6-step⁺⁺* $S6 S6'$
using *S6-def* $\langle C = \text{Some } C \rangle$ *S6'-def* *ord-res-6-step.intros* **by** *blast*

moreover have *ord-res-6-matches-ord-res-7* $i S6' S7'$
unfolding *S6'-def* $\langle S7' = (N, s7') \rangle$ $\langle s7' = (U_{er}, \mathcal{F}', \Gamma, \text{Some } (\text{efac } C)) \rangle$
proof (*rule ord-res-6-matches-ord-res-7.intros*)
show *ord-res-5-invars* $N (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (\text{efac } C))$
using *invars-6* **unfolding** $\langle C = \text{Some } C \rangle$
using *ord-res-6-preserves-invars[OF step6]* **by** *argo*

next
show *ord-res-7-invars* $N (U_{er}, \mathcal{F}', \Gamma, \text{Some } (\text{efac } C))$
using *invars-s7'* **unfolding** $\langle s7' = (U_{er}, \mathcal{F}', \Gamma, \text{Some } (\text{efac } C)) \rangle$.

next
show $\forall A C. (\mathcal{M} A = \text{Some } C) = (\text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C))$
using *match-hyps(3-)* **by** *argo*

next
show $\forall A. (\mathcal{M} A = \text{None}) = (\text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms})$

```

Γ)
  using match-hyps(3-) by argo
next
  show  $i = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \mid - \mid \text{trail-atms } \Gamma$ 
  using match-hyps(3-) by argo
qed

ultimately show ?thesis
  by metis
next
  case step-hyps: (resolution-bot E L D Uer' Γ)

  define S6' where
    S6' = (N, Uer',  $\mathcal{F}$ , ( $\lambda$ -. None) :: 'f gterm  $\Rightarrow$  'f gclause option, Some ({#} ::
'f gclause))

  define i' where
    i' = atms-of-clss (N  $\mid \cup \mid$  Uer')  $\mid - \mid$  trail-atms Γ'

  have step6: ord-res-6 N (Uer,  $\mathcal{F}$ ,  $\mathcal{M}$ , Some E) (Uer',  $\mathcal{F}$ ,  $\lambda$ -. None, Some {#})
  proof (rule ord-res-6.resolution-bot)
    show  $\neg \text{dom } \mathcal{M} \Vdash E$ 
    using clause-false-wrt-model-if-false-wrt- $\Gamma$ [OF  $\langle$ trail-false-cl $\rangle$  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  $\mathcal{M} (\text{atm-of } L) = \text{Some } D$ 
    by (metis literal.collapse(2) match-hyps(5) step-hyps(5) step-hyps(6) umi-
nus-Neg)
  next
    show Uer' = finsert (eres D E) Uer
    using step-hyps by argo
  next
    show eres D E = {#}
    using step-hyps by argo
  next
    show (( $\lambda$ -. None)) = ( $\lambda$ -. None) ..
  qed

  hence ord-res-6-step++ S6 S6'
  using S6-def  $\langle C = \text{Some } E \rangle$  S6'-def ord-res-6-step.intros by blast

  moreover have ord-res-6-matches-ord-res-7 i' S6' S7'
  unfolding S6'-def  $\langle S7' = (N, s7') \rangle$   $\langle s7' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } \{ \# \}) \rangle$ 
  proof (rule ord-res-6-matches-ord-res-7.intros)

```

```

show ord-res-5-invars  $N (U_{er}', \mathcal{F}, \lambda-. None, Some \{\#\})$ 
  using invars-6 unfolding  $\langle C = Some E \rangle$ 
  using ord-res-6-preserves-invars[OF step6] by argo
next
show ord-res-7-invars  $N (U_{er}', \mathcal{F}, \Gamma', Some \{\#\})$ 
  using invars-s7' unfolding  $\langle s7' = (U_{er}', \mathcal{F}, \Gamma', Some \{\#\}) \rangle$  .
next
show  $\forall A C. (None = Some C) = (map-of \Gamma' (Pos A) = Some (Some C))$ 
  unfolding  $\langle \Gamma' = [] \rangle$  by simp
next
show  $\forall A. (None = None) = (map-of \Gamma' (Neg A) \neq None \vee A \notin trail-atms \Gamma')$ 
  unfolding  $\langle \Gamma' = [] \rangle$  by simp
next
show  $i' = atms-of-clss (N \cup U_{er}') \dashv\vdash trail-atms \Gamma'$ 
  using i'-def .
qed

ultimately show ?thesis
  by metis
next
case step-hyps: (resolution-pos  $E L D U_{er}' \Gamma' K$ )

define  $S6'$  where
   $S6' = (N, U_{er}', \mathcal{F}, restrict-map \mathcal{M} \{A. A \prec_t atm-of K\}, Some (eres D E))$ 

define  $i'$  where
   $i' = atms-of-clss (N \cup U_{er}') \dashv\vdash trail-atms \Gamma'$ 

have mem-set- $\Gamma'$ -iff:  $\bigwedge x. (x \in set \Gamma') = (atm-of (fst x) \prec_t atm-of K \wedge x \in set \Gamma)$ 
  unfolding  $\langle \Gamma' = dropWhile (\lambda Ln. atm-of K \preceq_t atm-of (fst Ln)) \Gamma \rangle$ 
  unfolding mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone[OF  $\Gamma$ -sorted mono-atms-lt]
  by auto

have step6: ord-res-6  $N (U_{er}, \mathcal{F}, \mathcal{M}, Some E) (U_{er}', \mathcal{F}, restrict-map \mathcal{M} \{A. A \prec_t atm-of K\}, Some (eres D E))$ 
  proof (rule ord-res-6.resolution-pos)
    show  $\neg dom \mathcal{M} \Vdash E$ 
    using clause-false-wrt-model-if-false-wrt- $\Gamma$ [OF  $\langle trail-false-cls \Gamma E \rangle$ ] .
  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  $\mathcal{M} (atm-of L) = Some D$ 

```

```

    by (metis literal.collapse(2) match-hyps(5) step-hyps(5) step-hyps(6) umi-
nus-Neg)
  next
    show  $U_{er}' = \text{finsert} (\text{eres } D \ E) \ U_{er}$ 
      using step-hyps by argo
  next
    show  $\text{eres } D \ E \neq \{\#\}$ 
      using step-hyps by argo
  next
    show  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\}$ 
    ..
  next
    show  $\text{ord-res.is-maximal-lit } K \ (\text{eres } D \ E)$ 
      using step-hyps by argo
  next
    show  $\text{is-pos } K$ 
      using step-hyps by argo
qed

hence  $\text{ord-res-6-step}^{++} \ S6 \ S6'$ 
  using  $S6\text{-def } \langle C = \text{Some } E \rangle \ S6'\text{-def } \text{ord-res-6-step.intros}$  by blast

moreover have  $\text{ord-res-6-matches-ord-res-7 } i' \ S6' \ S7'$ 
  unfolding  $S6'\text{-def } \langle S7' = (N, s7') \rangle \ \langle s7' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } (\text{eres } D \ E)) \rangle$ 
  proof (rule  $\text{ord-res-6-matches-ord-res-7.intros}$ )
    show  $\text{ord-res-5-invars } N \ (U_{er}', \mathcal{F}, \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\}, \text{Some } (\text{eres } D \ E))$ 
      using  $\text{invars-6 unfolding } \langle C = \text{Some } E \rangle$ 
      using  $\text{ord-res-6-preserves-invars}[OF \ \text{step6}]$  by argo
  next
    show  $\text{ord-res-7-invars } N \ (U_{er}', \mathcal{F}, \Gamma', \text{Some } (\text{eres } D \ E))$ 
      using  $\text{invars-s7' unfolding } \langle s7' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } (\text{eres } D \ E)) \rangle$  .
  next
    show  $\forall A \ C. (\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \ A = \text{Some } C) = (\text{map-of } \Gamma' \ (\text{Pos } A) = \text{Some } (\text{Some } C))$ 
    proof (intro allI)
      fix  $A :: 'f \ \text{gterm}$  and  $C :: 'f \ \text{gclause}$ 
      show  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \ A = \text{Some } C \iff \text{map-of } \Gamma' \ (\text{Pos } A) = \text{Some } (\text{Some } C)$ 
      proof (cases  $A \in \text{dom } \mathcal{M} \wedge A \prec_t \text{atm-of } K$ )
        case True
          have  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \ A = \text{Some } C \iff \mathcal{M} \ A = \text{Some } C$ 
            using True by simp
      also have  $\dots \iff \text{map-of } \Gamma' \ (\text{Pos } A) = \text{Some } (\text{Some } C)$ 
        using  $\text{match-hyps}(3-)$  by metis

```

also have $\dots \longleftrightarrow \text{map-of } \Gamma' (Pos A) = \text{Some } (Some C)$
proof –
have $Pos A \in \text{fst ' set } \Gamma$
using *True*
by (*metis domIff map-of-eq-None-iff match-hyps(5) not-None-eq*)

hence $\exists \mathcal{C}. (Pos A, \mathcal{C}) \in \text{set } \Gamma$
by *fastforce*

hence $\exists \mathcal{C}. (Pos A, \mathcal{C}) \in \text{set } \Gamma \wedge (Pos A, \mathcal{C}) \in \text{set } \Gamma'$
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 $\text{map-of } \Gamma (Pos A) = \text{map-of } \Gamma' (Pos A)$
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 $\mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}$*
using *False unfolding restrict-map-def* **by** *auto*

moreover have $\text{map-of } \Gamma' (Pos A) \neq \text{Some } (Some C)$
using *False unfolding de-Morgan-conj*
proof (*elim disjE*)
assume $A \notin \text{dom } \mathcal{M}$

hence $\bigwedge \mathcal{C}. (Pos A, \mathcal{C}) \notin \text{set } \Gamma$
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 $\bigwedge \mathcal{C}. (Pos A, \mathcal{C}) \notin \text{set } \Gamma'$
unfolding *mem-set- Γ' -iff* **by** *simp*

then show $\text{map-of } \Gamma' (Pos A) \neq \text{Some } (Some C)$
by (*meson map-of-SomeD*)
next

```

assume  $\neg A \prec_t \text{atm-of } K$ 

hence  $\bigwedge C. (\text{Pos } A, C) \notin \text{set } \Gamma'$ 
unfolding mem-set- $\Gamma'$ -iff by simp

then show  $\text{map-of } \Gamma' (\text{Pos } A) \neq \text{Some } (\text{Some } C)$ 
by (meson map-of-SomeD)
qed

ultimately show ?thesis
by simp
qed
qed
next
show  $\forall A. (\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}) =$ 
 $(\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma')$ 
proof (intro allI)
fix  $A :: 'f \text{ gterm}$ 
show  $(\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}) =$ 
 $(\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma')$ 
proof (cases A  $\prec_t$  atm-of K)
case True

have  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None} \longleftrightarrow \mathcal{M} A = \text{None}$ 
using True by simp

also have  $\dots \longleftrightarrow \text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma$ 
using match-hyps(6) by metis

also have  $\dots \longleftrightarrow \text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma$ 
using True mem-set- $\Gamma'$ -iff
by (metis eq-fst-iff literal.sel(2) map-of-SomeD not-None-eq weak-map-of-SomeI)

also have  $\dots \longleftrightarrow \text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma'$ 
using True mem-set- $\Gamma'$ -iff
by (smt (verit, best) fset-trail-atms image-iff)

finally show ?thesis .
next
case False

have  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}$ 
using False by simp

moreover have  $A \notin \text{trail-atms } \Gamma'$ 
using False mem-set- $\Gamma'$ -iff
by (smt (verit, ccfv-threshold) fset-trail-atms image-iff)

ultimately show ?thesis

```



```

      by metis
    qed
  qed
next
  show  $i' = \text{atms-of-clss } (N \mid \cup \mid U_{er}') \mid - \mid \text{trail-atms } \Gamma'$ 
    using  $i'\text{-def}$  .
  qed

  ultimately show ?thesis
    by metis
  next
  case step-hyps: (resolution-neg  $E L D U_{er}' \Gamma' K C$ )

  define  $S6'$  where
     $S6' = (N, U_{er}', \mathcal{F}, \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\}, \text{Some } C)$ 

  define  $i'$  where
     $i' = \text{atms-of-clss } (N \mid \cup \mid U_{er}') \mid - \mid \text{trail-atms } \Gamma'$ 

  have mem-set- $\Gamma'$ -iff:  $\bigwedge x. (x \in \text{set } \Gamma') = (\text{atm-of } (fst\ x) \prec_t \text{atm-of } K \wedge x \in \text{set } \Gamma)$ 
    unfolding  $\langle \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (fst\ Ln)) \Gamma \rangle$ 
    unfolding mem-set-dropWhile-conv-if-list-sorted-and-pred-monotone[OF  $\Gamma$ -sorted mono-atms-lt]
    by auto

  have step6: ord-res-6  $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } E) (U_{er}', \mathcal{F}, \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\}, \text{Some } C)$ 
    proof (rule ord-res-6.resolution-neg)
      show  $\neg \text{dom } \mathcal{M} \Vdash E$ 
        using clause-false-wrt-model-if-false-wrt- $\Gamma$ [OF  $\langle \text{trail-false-cls } \Gamma\ E \rangle$ ] .
    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  $\mathcal{M} (\text{atm-of } L) = \text{Some } D$ 
        by (metis literal.collapse(2) match-hyps(5) step-hyps(5) step-hyps(6) uminus-Neg)
    next
      show  $U_{er}' = \text{finsert } (eres\ D\ E)\ U_{er}$ 
        using step-hyps by argo
    next
      show eres  $D\ E \neq \{\#\}$ 
        using step-hyps by argo
    next
      show restrict-map  $\mathcal{M} \{A. A \prec_t \text{atm-of } K\} = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of}$ 

```

```

K}
..
next
  show ord-res.is-maximal-lit K (eres D E)
    using step-hyps by argo
next
  show is-neg K
    using step-hyps by argo
next
  show  $\mathcal{M}$  (atm-of K) = 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  $\langle C = \text{Some } E \rangle$  S6'-def ord-res-6-step.intros by blast

moreover have ord-res-6-matches-ord-res-7 i' S6' S7'
  unfolding S6'-def  $\langle S7' = (N, s7') \rangle$   $\langle s7' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } C) \rangle$ 
  proof (rule ord-res-6-matches-ord-res-7.intros)
    show ord-res-5-invars N (Uer',  $\mathcal{F}$ , restrict-map  $\mathcal{M}$  {A. A  $\prec_t$  atm-of K},
Some C)
      using invars-6 unfolding  $\langle C = \text{Some } E \rangle$ 
      using ord-res-6-preserves-invars[OF step6] by argo
  next
    show ord-res-7-invars N (Uer',  $\mathcal{F}$ ,  $\Gamma'$ , Some C)
      using invars-s7' unfolding  $\langle s7' = (U_{er}', \mathcal{F}, \Gamma', \text{Some } C) \rangle$  .
  next
    show  $\forall A C. (\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{Some } C) =$ 
      (map-of  $\Gamma'$  (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  $\longleftrightarrow$  map-of  $\Gamma'$ 
(Pos A) = Some (Some C)
      proof (cases A  $\in$  dom  $\mathcal{M} \wedge A \prec_t$  atm-of K)
        case True
          have restrict-map  $\mathcal{M}$  {A. A  $\prec_t$  atm-of K} A = Some C  $\longleftrightarrow$   $\mathcal{M}$  A =
Some C
          using True by simp

      also have ...  $\longleftrightarrow$  map-of  $\Gamma$  (Pos A) = Some (Some C)
      using match-hyps(3-) by metis

      also have ...  $\longleftrightarrow$  map-of  $\Gamma'$  (Pos A) = Some (Some C)
      proof -
        have Pos A  $\in$  fst ' set  $\Gamma$ 
        using True
        by (metis domIff map-of-eq-None-iff match-hyps(5) not-None-eq)

```

```

hence  $\exists C. (Pos\ A, C) \in set\ \Gamma$ 
  by fastforce

hence  $\exists C. (Pos\ A, C) \in set\ \Gamma \wedge (Pos\ A, C) \in set\ \Gamma'$ 
  using True unfolding mem-set- $\Gamma'$ -iff prod.sel literal.sel by metis

moreover have distinct (map fst  $\Gamma'$ )
  using  $\Gamma$ -distinct-atoms
proof (rule distinct-suffix)
  show suffix (map fst  $\Gamma'$ ) (map fst  $\Gamma$ )
    using map-mono-suffix step-hyps(9) suffix-dropWhile by blast
qed

ultimately have map-of  $\Gamma$  (Pos A) = map-of  $\Gamma'$  (Pos A)
  using  $\Gamma$ -distinct-atoms by (auto dest: map-of-is-SomeI)

thus ?thesis
  by argo
qed

finally show ?thesis .
next
case False
have restrict-map  $\mathcal{M}$  {A. A  $\prec_t$  atm-of K} A = None
  using False unfolding restrict-map-def by auto

moreover have map-of  $\Gamma'$  (Pos A)  $\neq$  Some (Some C)
  using False unfolding de-Morgan-conj
proof (elim disjE)
  assume A  $\notin$  dom  $\mathcal{M}$ 

  hence  $\bigwedge C. (Pos\ A, C) \notin set\ \Gamma$ 
    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  $\bigwedge C. (Pos\ A, C) \notin set\ \Gamma'$ 
    unfolding mem-set- $\Gamma'$ -iff by simp

  then show map-of  $\Gamma'$  (Pos A)  $\neq$  Some (Some C)
    by (meson map-of-SomeD)
next
assume  $\neg A \prec_t atm\ of\ K$ 

  hence  $\bigwedge C. (Pos\ A, C) \notin set\ \Gamma'$ 
    unfolding mem-set- $\Gamma'$ -iff by simp

  then show map-of  $\Gamma'$  (Pos A)  $\neq$  Some (Some C)

```

```

      by (meson map-of-SomeD)
    qed

    ultimately show ?thesis
      by simp
    qed
  qed
next
show  $\forall A. (\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}) =$ 
  ( $\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma'$ )
proof (intro allI)
  fix A :: 'f gterm
  show  $(\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}) =$ 
    ( $\text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma'$ )
  proof (cases A  $\prec_t$  atm-of K)
    case True
      have  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None} \longleftrightarrow \mathcal{M} A = \text{None}$ 
        using True by simp
      also have  $\dots \longleftrightarrow \text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma$ 
        using match-hyps(6) by metis
      also have  $\dots \longleftrightarrow \text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma$ 
        using True mem-set- $\Gamma'$ -iff
      by (metis eq-fst-iff literal.sel(2) map-of-SomeD not-None-eq weak-map-of-SomeI)
      also have  $\dots \longleftrightarrow \text{map-of } \Gamma' (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma'$ 
        using True mem-set- $\Gamma'$ -iff
        by (smt (verit, best) fset-trail-atms image-iff)
    finally show ?thesis .
  next
  case False
    have  $\text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} A = \text{None}$ 
      using False by simp
    moreover have  $A \notin \text{trail-atms } \Gamma'$ 
      using False mem-set- $\Gamma'$ -iff
      by (smt (verit, ccfv-threshold) fset-trail-atms image-iff)
    ultimately show ?thesis
      by metis
  qed
  qed
next
show  $i' = \text{atms-of-class } (N \cup U_{er}') \mid\text{-} \text{trail-atms } \Gamma'$ 
  using i'-def .

```

```

qed

ultimately show ?thesis
  by metis
qed
qed

theorem bisimulation-ord-res-6-ord-res-7:
  defines match  $\equiv$  ord-res-6-matches-ord-res-7
  shows  $\exists (MATCH :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ ord-res-6-state} \Rightarrow 'f \text{ ord-res-7-state} \Rightarrow \text{bool})$ 
 $\mathcal{R}_f \mathcal{R}_b.$ 
  bisimulation ord-res-6-step ord-res-6-final (constant-context ord-res-7) ord-res-7-final
  MATCH  $\mathcal{R}_f \mathcal{R}_b$ 
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  $\forall S. \text{ord-res-6-final } S \longrightarrow (\exists S'. \text{ord-res-6-step } S S')$ 
    by (metis finished-def ord-res-6-semantics.final-finished)
next
  show  $\forall S. \text{ord-res-7-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-7 } S S')$ 
    by (metis finished-def ord-res-7-semantics.final-finished)
next
  show  $\forall i S6 S7. \text{match } i S6 S7 \longrightarrow \text{ord-res-6-final } S6 \longleftrightarrow \text{ord-res-7-final } S7$ 
    unfolding match-def
    using ord-res-6-final-iff-ord-res-7-final by metis
next
  show  $\forall i S6 S7. \text{match } i S6 S7 \longrightarrow$ 
    safe-state ord-res-6-step ord-res-6-final  $S6 \wedge$ 
    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  $\langle \text{match } i S6 S7 \rangle$ [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  $\langle \text{match } i S6 S7 \rangle$ [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 ( $|\cdot|$ )
    using wfp-pfssubset .

```

next

show $\forall i S6 S7 S7'. \text{match } i S6 S7 \longrightarrow \text{constant-context ord-res-7 } S7 S7' \longrightarrow$
 $(\exists i' S6'. \text{ord-res-6-step}^{++} S6 S6' \wedge \text{match } i' S6' S7') \vee$
 $(\exists i'. \text{match } i' S6 S7' \wedge i' \sqsubset i)$
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**

$\neg \text{trail-false-cls } \Gamma C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C L \Longrightarrow$
linorder-trm.is-least-in-fset $\{A \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er})\} A \Longrightarrow$
 $A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma \} A \Longrightarrow$
ord-res-8-can-decide-neg $N U_{er} \mathcal{F} \Gamma C$

inductive *ord-res-8-can-skip-undefined-neg* **where**

$\neg \text{trail-false-cls } \Gamma C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C L \Longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma) \Longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L \Longrightarrow$
is-neg $L \Longrightarrow$
ord-res-8-can-skip-undefined-neg $N U_{er} \mathcal{F} \Gamma C$

inductive *ord-res-8-can-skip-undefined-pos-ultimate* **where**

$\neg \text{trail-false-cls } \Gamma C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C L \Longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma) \Longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L \Longrightarrow$
is-pos $L \Longrightarrow$
 $\neg \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq L\# \} \Longrightarrow$
 $\neg (\exists D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). C \prec_c D) \Longrightarrow$
ord-res-8-can-skip-undefined-pos-ultimate $N U_{er} \mathcal{F} \Gamma C$

inductive *ord-res-8-can-produce* **where**

$\neg \text{trail-false-cls } \Gamma C \Longrightarrow$
linorder-lit.is-maximal-in-mset $C L \Longrightarrow$
 $\neg (\exists A \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma) \Longrightarrow$
 $\neg \text{trail-defined-lit } \Gamma L \Longrightarrow$
is-pos $L \Longrightarrow$
trail-false-cls $\Gamma \{\#K \in \# C. K \neq L\# \} \Longrightarrow$
linorder-lit.is-greatest-in-mset $C L \Longrightarrow$

ord-res-8-can-produce $N U_{er} \mathcal{F} \Gamma C$

inductive *ord-res-8-can-factorize* **where**

\neg *trail-false-cls* $\Gamma C \implies$
linorder-lit.is-maximal-in-mset $C L \implies$
 $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma) \implies$
 \neg *trail-defined-lit* $\Gamma L \implies$
is-pos $L \implies$
trail-false-cls $\Gamma \{\#K \in \# C. K \neq L\# \} \implies$
 \neg *linorder-lit.is-greatest-in-mset* $C L \implies$
ord-res-8-can-factorize $N U_{er} \mathcal{F} \Gamma C$

definition *is-least-nonskipped-clause* **where**

is-least-nonskipped-clause $N U_{er} \mathcal{F} \Gamma C \longleftrightarrow$
linorder-cls.is-least-in-fset $\{|C| \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}).$
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-empty*:

assumes *bot-in*: $\{\#\} | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$

shows *is-least-nonskipped-clause* $N U_{er} \mathcal{F} \Gamma \{\#\}$

unfolding *is-least-nonskipped-clause-def* *linorder-cls.is-least-in-filter-iff*

proof (*intro conjI ballI impI*)

show $\{\#\} | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$

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 $C :: 'f \text{ gclause}$

assume $C \neq \{\#\}$

thus $\{\#\} \prec_c C$

using *empty-lesseq-cls* **by** *blast*

qed

lemma *nex-is-least-nonskipped-clause-if*:

assumes

no-undef-atom: $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \notin | \text{trail-atms } \Gamma)$ **and**

no-false-clause: $\neg fBex (\text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})) (i \text{trail-false-cls } \Gamma)$

shows $\# C. i \text{is-least-nonskipped-clause } N U_{er} \mathcal{F} \Gamma C$

unfolding *not-ex*
proof (*intro allI notI*)
fix $C :: 'f\ gclause$
assume *is-least-nonskipped-clause* $N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

hence $C\text{-in}$: $C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})$ **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* $C\ L\ \wedge\ \neg\ trail\text{-defined-lit}\ \Gamma\ L$
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* $L\ |\in|\ atms\text{-of-clss}\ (N\ |\cup|\ U_{er})\ \wedge\ atm\text{-of}\ L\ |\notin|\ trail\text{-atms}\ \Gamma$
using $C\text{-in}$
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: $\nexists C'.\ ord\text{-res-7}\ N\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C})\ (U_{er},\ \mathcal{F},\ \Gamma,\ \mathcal{C}')$
shows $(\forall C.\ \mathcal{C} = Some\ C \longleftrightarrow is\text{-least-nonskipped-clause}\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C)$
proof (*cases* \mathcal{C})
case *None*

have *trail-atms* $\Gamma = atms\text{-of-clss}\ (N\ |\cup|\ U_{er})$
using *None invars[unfolded ord-res-7-invars-def]* **by** *simp*

have $\forall C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}).\ trail\text{-true-cl}\ \Gamma\ C$
using *None invars[unfolded ord-res-7-invars-def]* **by** *simp*

hence *no-false*: $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \neg \text{trail-false-cls } \Gamma C$
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 $\# C. \text{is-least-nonskipped-clause } N U_{er} \mathcal{F} \Gamma C$
proof (*rule notI, elim exE*)
fix C
assume *is-least-nonskipped-clause* $N U_{er} \mathcal{F} \Gamma C$

hence

C-in: $C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **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 $\langle \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$
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 $\langle \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$
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 $\langle \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$
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 $\langle \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er}) \rangle$

```

    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 Uer F Γ C
thus False
    using ⟨trail-atms Γ = atms-of-clss (N |∪| Uer)⟩
    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 F |↑| (N |∪| Uer)
    using Some invars[unfolded ord-res-7-invars-def] by simp

have is-least-nonskipped-clause N Uer F Γ D
proof (cases D = {#})
case True
    thus ?thesis
        using D-in is-least-nonskipped-clause-empty by metis
next
case False

then obtain LD where D-max-lit: linorder-lit.is-maximal-in-mset D LD
    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 F |↑| (N |∪| Uer)
        using D-in .
next
show trail-false-cls Γ D ∨
    ord-res-8-can-decide-neg N Uer F Γ D ∨
    ord-res-8-can-skip-undefined-neg N Uer F Γ D ∨
    ord-res-8-can-skip-undefined-pos-ultimate N Uer F Γ D ∨
    ord-res-8-can-produce N Uer F Γ D ∨ ord-res-8-can-factorize N Uer 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* $\exists A | \in | \text{atms-of-cls} (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms}$
 Γ)
case *True*
hence *ord-res-8-can-decide-neg N U_{er} F Γ D*
using *ord-res-8-can-decide-neg.intros[OF D-not-false D-max-lit]*
by (*metis (no-types, lifting) equals_iffemptyD ffilter_member_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 $\exists C'. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, C) (U_{er}, \mathcal{F}, \Gamma, C')$
unfolding $\langle C = \text{Some } D \rangle$
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 \in # D. K \neq 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  $\exists E \mid \in \mid \text{iefac } \mathcal{F} \mid ^\dagger \mid (N \mid \cup \mid U_{er}) \cdot D \prec_c E$ )
    case True

    hence  $\exists C' \cdot \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, C) (U_{er}, \mathcal{F}, \Gamma, C')$ 
      unfolding  $\langle C = \text{Some } D \rangle$ 
      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 fffmember-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  $N U_{er} \mathcal{F} \Gamma D$ 
    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  $N U_{er} \mathcal{F} \Gamma D$ 
  unfolding  $\langle C = \text{Some } D \rangle$ 
  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

$E\text{-in}: E \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**

$E\text{-neg}: E \neq D$ **and**

$E\text{-spec}: \text{trail-false-cls } \Gamma E \vee$

$\text{ord-res-8-can-decide-neg } N U_{er} \mathcal{F} \Gamma E \vee$

$\text{ord-res-8-can-skip-undefined-neg } N U_{er} \mathcal{F} \Gamma E \vee$

$\text{ord-res-8-can-skip-undefined-pos-ultimate } N U_{er} \mathcal{F} \Gamma E \vee$

$\text{ord-res-8-can-produce } N U_{er} \mathcal{F} \Gamma E \vee$

$\text{ord-res-8-can-factorize } N U_{er} \mathcal{F} \Gamma E$

have true-cls-if-lt-D :

$\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma C$

using $\text{invars}[\text{unfolded ord-res-7-invars-def}]$ **Some** **by** simp

have $\Gamma\text{-lower-set}: \text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$

using $\text{invars}[\text{unfolded ord-res-7-invars-def}]$ **by** simp

have FOO :

$\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$

$(\forall K. \text{ord-res.is-maximal-lit } K C \longrightarrow$

$\neg (\exists A \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma))$

using $\text{invars}[\text{unfolded ord-res-7-invars-def}]$ **Some** $E\text{-in}$ **by** simp

hence BAR :

$\forall C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma K \longrightarrow (\text{trail-true-cls } \Gamma \{ \#x \in \# C. x \neq K \# \} \wedge \text{is-pos } K))$

using $\text{invars}[\text{unfolded ord-res-7-invars-def}]$ **Some** **by** simp

show $D \prec_c E$

using $E\text{-spec}$

proof (elim disjE)

assume $\text{trail-false-cls } \Gamma E$

hence $\neg \text{trail-true-cls } \Gamma E$

using $\text{invars not-trail-true-cls-and-trail-false-cls}$

$\text{ord-res-7-invars-implies-trail-consistent}$ **by** blast

thus $D \prec_c E$

using $E\text{-in } E\text{-neg true-cls-if-lt-D}$ **by** force

next

assume $\text{ord-res-8-can-decide-neg } N U_{er} \mathcal{F} \Gamma E$

thus $D \prec_c E$

proof ($\text{cases } N U_{er} \mathcal{F} \Gamma E$ $\text{rule: ord-res-8-can-decide-neg.cases}$)

case ($1 L_E A$)

hence $\exists A \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \prec_t \text{atm-of } L_E \wedge A \notin | \text{trail-atms}$

Γ

unfolding $\text{linorder-trm.is-least-in-ffilter-iff}$ **by** metis

thus $?thesis$

```

      using FOO[rule-format, OF E-in - ⟨ord-res.is-maximal-lit LE E⟩] E-in
E-neq
  by force
  qed
next
  assume ord-res-8-can-skip-undefined-neg N Uer F Γ E
  thus D <c E
  proof (cases N Uer F Γ E rule: ord-res-8-can-skip-undefined-neg.cases)
    case hyps: (1 LE)
    thus ?thesis
      using BAR[rule-format, OF E-in - ⟨ord-res.is-maximal-lit LE 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 Uer F Γ E
  thus D <c E
  proof (cases N Uer 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 Uer F Γ E
  hence ¬ trail-true-cls Γ E
  proof (cases N Uer 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, cfv-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 Uer F Γ E
  hence ¬ trail-true-cls Γ E
  proof (cases N Uer 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, cfv-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
qed

```

```

moreover have Uniq (is-least-nonskipped-clause N Uer 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-γ-invars N (Uer, F, Γ, C)
  shows  $\exists C'. (ord-res-γ N)^{**} (U_{er}, F, \Gamma, C) (U_{er}, F, \Gamma, C') \wedge$ 
    ( $\nexists C''. ord-res-γ N (U_{er}, F, \Gamma, C') (U_{er}, F, \Gamma, C'')$ )
proof –
  define R where
     $R = (\lambda C C'. ord-res-γ N (U_{er}, F, \Gamma, C) (U_{er}, F, \Gamma, C'))$ 

  define f :: 'f gclause option ⇒ 'f gclause fset where
     $f = (\lambda C. case C of None \Rightarrow \{\}\mid Some C \Rightarrow \{D \mid \in |iefac F \mid \} (N \mid \cup \mid U_{er}).$ 
     $C \preceq_c D \})$ 

  let ?less-f = ( $|\subset|$ )

  have Wellfounded.wfp-on  $\{x'. R^{**} C x'\} R^{-1-1}$ 
  proof (rule wfp-on-if-convertible-to-wfp-on)
    have wfp ( $|\subset|$ )
      by auto
    thus Wellfounded.wfp-on ( $f \text{ ' } \{x'. R^{**} C x'\}$ ) ?less-f
      using Wellfounded.wfp-on-subset subset-UNIV by metis
  next
    fix Cx Cy :: '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  $C_x \in \{x'. R^{**} C x'\}$  and  $C_y \in \{x'. R^{**} C x'\}$ 
  hence
     $(ord-res-γ N)^{**} (U_{er}, F, \Gamma, C) (U_{er}, F, \Gamma, C_x)$  and

```

(*ord-res-7 N*)** ($U_{er}, \mathcal{F}, \Gamma, C$) ($U_{er}, \mathcal{F}, \Gamma, C_y$)
unfolding *atomize-conj mem-Collect-eq R-def*
by (*auto intro: rtranclp-with-constsD*)

hence

x-invars: *ord-res-7-invars N* ($U_{er}, \mathcal{F}, \Gamma, C_x$) **and**
y-invars: *ord-res-7-invars N* ($U_{er}, \mathcal{F}, \Gamma, C_y$)
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* ($f C_y$) ($f C_x$)

if $C_x = \text{Some } C$ **and**

C_y-disj: $C_y = \text{None} \vee C_y = \text{Some } D \wedge C \prec_c D$ **and**

C-in: $C \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$

for $C D$

proof –

have *f-x*: $f C_x = \{D \mid D \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}). C \preceq_c D\}$
by (*auto simp add: <C_x = Some C> f-def*)

moreover have $C \in | f C_x$

using *C-in f-x by simp*

moreover have $C \notin | f C_y \wedge f C_y \subseteq | f C_x$

using *C_y-disj*

proof (*elim disjE conjE; intro conjI*)

assume $C_y = \text{None}$

thus $C \notin | f C_y$ **and** $f C_y \subseteq | f C_x$

unfolding *f-x*

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

next

assume $C_y = \text{Some } D$ **and** $C \prec_c D$

have $\bigwedge x. D \preceq_c x \implies C \preceq_c x$

using $\langle C \prec_c D \rangle$ **by** *auto*

moreover have *fst-f-y*: $f C_y = \{E \mid E \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}). D \preceq_c E\}$

by (*auto simp add: <C_y = Some D> f-def*)

ultimately show $f C_y \subseteq | f C_x$

using *f-x by auto*

show $C \notin | f C_y$

using $\langle C \prec_c D \rangle$ *fst-f-y by auto*

qed

ultimately have $f C_y \subseteq | f C_x$

by *blast*

thus *?thesis*
by (*simp add: lex-prodp-def*)
qed

have *eres-not-in-if: eres D E | \notin | U_{er}*
if $C_x = \text{Some } E$ **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:
 $\forall C \in |iefac \mathcal{F} |^| (N \cup U_{er}). C \prec_c E \longrightarrow \text{trail-true-cls } \Gamma C$ **and**
 Γ -prop-greatest:
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$
(*fst Ln*)
using *x-invars unfolding $\langle C_x = \text{Some } E \rangle$ ord-res- γ -invars-def* **by** *simp-all*

have $(- L, \text{Some } D) \in \text{set } \Gamma$
using $\langle \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \rangle$ **by** (*metis map-of-SomeD*)

hence *D-greatest-lit: linorder-lit.is-greatest-in-mset D (- L)*
using *Γ -prop-greatest* **by** *fastforce*

have $\text{eres } D E \prec_c E$
using *eres-lt-if*
using *E-max-lit L-neg D-greatest-lit*
by *metis*

hence $\text{eres } D E \neq E$
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 $\forall K \in\# \text{eres } D E. \text{atm-of } K \prec_t \text{atm-of } L$
using *lit-in-eres-lt-greatest-lit-in-greatest-resolvent[OF $\langle \text{eres } D E \neq E \rangle$*
E-max-lit]
by *metis*

hence $\forall K \in\# \text{eres } D E. K \neq L \wedge K \neq - L$
by *fastforce*

moreover have $\forall L \in\# \text{eres } D E. L \in\# D \vee L \in\# E$

```

using lit-in-one-of-resolvents-if-in-eres by metis

moreover have D-almost-false: trail-false-cls  $\Gamma \{\#K \in \# D. K \neq - L\# \}$ 
using ord-res-7-invars-implies-propagated-clause-almost-false
using  $\langle (- L, \text{Some } D) \in \text{set } \Gamma \rangle$  x-invars
by metis

ultimately have trail-false-cls  $\Gamma$  (eres  $D E$ )
using E-false unfolding trail-false-cls-def by fastforce

have eres  $D E \notin N \mid \cup \mid U_{er}$ 
using eres-not-in-known-clauses-if-trail-false-cls
using  $\Gamma$ -consistent clauses-lt-E-true  $\langle \text{eres } D E \prec_c E \rangle$   $\langle \text{trail-false-cls } \Gamma$  (eres
D E) $\rangle$ 
by metis

thus eres  $D E \notin U_{er}$ 
by blast
qed

assume  $R^{-1-1} C_y C_x$ 
hence ord-res-7 N ( $U_{er}, \mathcal{F}, \Gamma, C_x$ ) ( $U_{er}, \mathcal{F}, \Gamma, C_y$ )
unfolding conversep-iff R-def .
thus ?less-f ( $f C_y$ ) ( $f C_x$ )
proof (cases  $N$  ( $U_{er}, \mathcal{F}, \Gamma, C_x$ ) ( $U_{er}, \mathcal{F}, \Gamma, C_y$ ) rule: ord-res-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  $C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ 
using ord-res-7-invars-def step-hyps(1) x-invars by presburger

moreover have  $\exists D. C_y = \text{None} \vee C_y = \text{Some } D \wedge C \prec_c D$ 
proof (cases  $C_y$ )
case None
thus ?thesis
by simp
next
case (Some  $D$ )
thus ?thesis
using step-hyps
by (metis linorder-cls.is-least-in-filter-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 |∈| iefac F |↑| (N |∪| Uer)
  using ord-res-7-invars-def step-hyps(1) x-invars by presburger

moreover have ∃ D. Cy = None ∨ Cy = Some D ∧ C <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 F |↑| (N |∪| Uer)
  using ord-res-7-invars-def step-hyps(1) x-invars by presburger

moreover have efac C ≠ C
  using step-hyps by (metis greatest-literal-in-efacI)

ultimately have C |∉| F
by (smt (verit, ccfv-threshold) fimage-iff iefac-def ex1-efac-eq-factoring-chain
  factorizable-if-neq-efac)

hence False
  using ⟨F = finsert C F⟩ by blast

thus ?thesis ..
next
case step-hyps: (resolution-bot E L D)
hence eres D E |∉| Uer
  using eres-not-in-if by metis
hence False
  using ⟨Uer = finsert (eres D E) Uer⟩ by blast
thus ?thesis ..
next
case (resolution-pos E L D K)
hence eres D E |∉| Uer

```

```

    using eres-not-in-if by metis
  hence False
    using ⟨ $U_{er} = \text{finsert } (eres D E) U_{er}$ ⟩ by blast
  thus ?thesis ..
next
  case (resolution-neg E L D K C)
  hence eres D E  $\notin$   $U_{er}$ 
    using eres-not-in-if by metis
  hence False
    using ⟨ $U_{er} = \text{finsert } (eres D E) U_{er}$ ⟩ by blast
  thus ?thesis ..
qed
qed

then obtain  $C'$  where  $R^{**} C C'$  and  $\nexists z. R C' z$ 
  using ex-terminating-rtranclp-strong by metis

show ?thesis
proof (intro exI conjI)
  show (ord-res-7 N)** ( $U_{er}, \mathcal{F}, \Gamma, C$ ) ( $U_{er}, \mathcal{F}, \Gamma, C'$ )
    using ⟨ $R^{**} C C'$ ⟩
    by (induction C rule: converse-rtranclp-induct) (auto simp: R-def)
next
  show  $\nexists C''. \text{ord-res-7 N } (U_{er}, \mathcal{F}, \Gamma, C') (U_{er}, \mathcal{F}, \Gamma, C'')$ 
    using ⟨ $\nexists z. R C' z$ ⟩
    by (simp add: R-def)
qed
qed

inductive ord-res-7-matches-ord-res-8 :: 'f ord-res-7-state  $\Rightarrow$  'f ord-res-8-state  $\Rightarrow$ 
bool where
  ord-res-7-invars N ( $U_{er}, \mathcal{F}, \Gamma, C$ )  $\Longrightarrow$ 
  ord-res-8-invars N ( $U_{er}, \mathcal{F}, \Gamma$ )  $\Longrightarrow$ 
  ( $\forall C. C = \text{Some } C \longleftrightarrow \text{is-least-nonskipped-clause } N U_{er} \mathcal{F} \Gamma C$ )  $\Longrightarrow$ 
  ord-res-7-matches-ord-res-8 (N,  $U_{er}, \mathcal{F}, \Gamma, C$ ) (N,  $U_{er}, \mathcal{F}, \Gamma$ )

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  $\longleftrightarrow$  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  $U_{er} \mathcal{F} \Gamma C$ )

  note invars7 = ⟨ord-res-7-invars N ( $U_{er}, \mathcal{F}, \Gamma, C$ )⟩[unfolded ord-res-7-invars-def,
  rule-format, OF refl]

  have  $\Gamma$ -consistent: trail-consistent  $\Gamma$ 
    using invars7 by (metis trail-consistent-if-sorted-wrt-atoms)

```

show *ord-res-7-final S7* \longleftrightarrow *ord-res-8-final S8*
proof (*rule iffI*)
 assume *ord-res-7-final S7*
 thus *ord-res-8-final S8*
 unfolding $\langle S7 = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$
 proof (*cases (N, U_{er}, F, Γ, C) rule: ord-res-7-final.cases*)
 case *model-found*
 show *ord-res-8-final S8*
 unfolding $\langle S8 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$
 proof (*rule ord-res-8-final.model-found*)
 have $\mathcal{C} = \text{None} \longrightarrow \text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er})$
 using *invars7* **by** *argo*

 hence $\text{trail-atms } \Gamma = \text{atms-of-clss } (N \mid \cup \mid U_{er})$
 using *model-found* **by** *argo*

 thus $\neg (\exists A \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). A \mid \notin \mid \text{trail-atms } \Gamma)$
 by *metis*
 next
 have $\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-true-cl } \Gamma \ C$
 using *invars7 model-found* **by** *simp*

 moreover **have** $\neg (\text{trail-true-cl } \Gamma \ C \wedge \text{trail-false-cl } \Gamma \ C)$ **for** *C*
 using *not-trail-true-cl-and-trail-false-cl[OF Γ-consistent]* .

 ultimately show $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er}). \text{trail-false-cl } \Gamma \ C)$
 by *metis*
 qed
 next
 case *contradiction-found*
 show *ord-res-8-final S8*
 unfolding $\langle S8 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$
 proof (*rule ord-res-8-final.contradiction-found*)
 show $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er})$
 using *invars7* $\langle \mathcal{C} = \text{Some } \{\#\} \rangle$ **by** *metis*
 qed
 qed
 next
 assume *ord-res-8-final S8*
 thus *ord-res-7-final S7*
 unfolding $\langle S8 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$
 proof (*cases (N, U_{er}, F, Γ) rule: ord-res-8-final.cases*)
 case *model-found*

 hence $\# C. \text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ C$
 using *nex-is-least-nonskipped-clause-if* **by** *metis*

 hence $\mathcal{C} = \text{None}$

```

using match-hyps by simp

thus ord-res-7-final  $S7$ 
  unfolding  $\langle S7 = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$ 
  using ord-res-7-final.model-found by metis
next
case contradiction-found

hence is-least-nonskipped-clause  $N U_{er} \mathcal{F} \Gamma \{\#\}$ 
  using is-least-nonskipped-clause-empty by metis

hence  $\mathcal{C} = \text{Some } \{\#\}$ 
  using match-hyps by presburger

thus ord-res-7-final  $S7$ 
  unfolding  $\langle S7 = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$ 
  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  $\exists S7'. (\text{constant-context } \text{ord-res-7})^{++} S7 S7' \wedge \text{ord-res-7-matches-ord-res-8}$ 
 $S7' S8'$ 
  using match
proof (cases  $S7 S8$  rule: ord-res-7-matches-ord-res-8.cases)
  case match-hyps:  $(1 N U_{er} \mathcal{F} \Gamma \mathcal{C})$ 

  note  $S7\text{-def} = \langle S7 = (N, U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle$ 

  note  $\text{invars7} = \langle \text{ord-res-7-invars } N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \rangle [\text{unfolded } \text{ord-res-7-invars-def},$ 
rule-format, OF refl]

  have  $\Gamma\text{-sorted}$ : sorted-wrt  $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$ 
  using  $\text{invars7}$  by argo

  have  $\Gamma\text{-consistent}$ : trail-consistent  $\Gamma$ 
  using trail-consistent-if-sorted-wrt-atoms [OF  $\Gamma\text{-sorted}$ ].

  have  $\Gamma\text{-lower-set}$ : linorder-trm.is-lower-fset  $(\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid$ 
 $U_{er}))$ 
  using  $\text{invars7}$  by argo

  have  $\mathcal{C}\text{-eq-None-implies-all-atoms-defined}$ :  $\mathcal{C} = \text{None} \longrightarrow \text{trail-atms } \Gamma = \text{atms-of-clss}$ 
 $(N \mid \cup \mid U_{er})$ 
  using  $\text{invars7}$  by argo

```

obtain $s8'$ **where**
S8'-def: $S8' = (N, s8')$ **and**
step': $\text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) s8'$
using *step unfolding* $\langle S8 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$
by (*auto elim: constant-context.cases*)

have *invars-s8'*: $\text{ord-res-8-invars } N s8'$
using *ord-res-8-preserves-invars* [*OF step'* $\langle \text{ord-res-8-invars } N (U_{er}, \mathcal{F}, \Gamma) \rangle$].

show *?thesis*
using *step'*
proof (*cases* $N (U_{er}, \mathcal{F}, \Gamma) s8'$ *rule: ord-res-8.cases*)
case *step-hyps*: (*decide-neg* $A \Gamma'$)

have
A-in: $A \in | \text{atms-of-clss } (N \cup | U_{er})$ **and**
A-undef: $A \notin | \text{trail-atms } \Gamma$ **and**
A-least: $\forall y \in | \text{atms-of-clss } (N \cup | U_{er}). y \neq A \longrightarrow (\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t y) \longrightarrow A \prec_t y$
using *step-hyps(3) unfolding linorder-trm.is-least-in-fset-iff* **by** *auto*

have $C \neq \text{None}$
using *C-eq-None-implies-all-atoms-defined* *A-in* *A-undef* **by** *metis*

then obtain $D :: 'f \text{ gclause where } C = \text{Some } D$
by *blast*

hence *D-in*: $D \in | \text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er})$
by (*metis* $\langle C = \text{Some } D \rangle$ *invars7*)

have *is-least-nonskipped-clause* $N U_{er} \mathcal{F} \Gamma D$
using *match-hyps* $\langle C = \text{Some } D \rangle$ **by** *metis*

moreover have *D-not-false*: $\neg \text{trail-false-cl } \Gamma D$
using *D-in step-hyps* **by** *metis*

moreover have $\neg \text{ord-res-8-can-produce } N U_{er} \mathcal{F} \Gamma D$
proof (*rule notI*)
assume *ord-res-8-can-produce* $N U_{er} \mathcal{F} \Gamma D$
thus *False*
proof (*cases* $N U_{er} \mathcal{F} \Gamma D$ *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 \neg ord-res-8-can-factorize $N U_{er} \mathcal{F} \Gamma D$

proof (rule notI)

 assume ord-res-8-can-factorize $N U_{er} \mathcal{F} \Gamma D$

 thus False

 proof (cases $N U_{er} \mathcal{F} \Gamma D$ 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-clsI
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 $N U_{er} \mathcal{F} \Gamma D \vee$
ord-res-8-can-skip-undefined-neg $N U_{er} \mathcal{F} \Gamma D \vee$
ord-res-8-can-skip-undefined-pos-ultimate $N U_{er} \mathcal{F} \Gamma D$

 unfolding is-least-nonskipped-clause-def

 unfolding linorder-cls.is-least-in-ffilter-iff

 by argo

then obtain C' where first-step7: ord-res-7 $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) (U_{er}, \mathcal{F}, \Gamma', C')$

proof (elim disjE; atomize-elim)

 assume ord-res-8-can-decide-neg $N U_{er} \mathcal{F} \Gamma D$

 thus $\exists C'. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) (U_{er}, \mathcal{F}, \Gamma', C')$

 proof (cases $N U_{er} \mathcal{F} \Gamma D$ 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.neg-iff linorder-trm.not-in-lower-setI linorder-trm.order.strict-trans
step-hyps(3))

 thus ?thesis

 using hyps $\langle \Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \rangle$

 using ord-res-7.decide-neg[of $\Gamma D - N U_{er} A \Gamma' \mathcal{F}$] by blast

 qed

next

 assume ord-res-8-can-skip-undefined-neg $N U_{er} \mathcal{F} \Gamma D$

 thus $\exists C'. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) (U_{er}, \mathcal{F}, \Gamma', C')$

 proof (cases $N U_{er} \mathcal{F} \Gamma D$ rule: ord-res-8-can-skip-undefined-neg.cases)

 case hyps: (1 L)

 hence $L = \text{Neg } A$

 by (smt (verit) A-in A-least A-undef D-in Γ -lower-set atm-of-in-atms-of-clsI
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 ⟨ $\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma$ ⟩
    using ord-res-7.skip-undefined-neg by blast
qed
next
assume ord-res-8-can-skip-undefined-pos-ultimate  $N U_{er} \mathcal{F} \Gamma D$ 
thus  $\exists C'. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) (U_{er}, \mathcal{F}, \Gamma', C')$ 
proof (cases  $N U_{er} \mathcal{F} \Gamma D$  rule: ord-res-8-can-skip-undefined-pos-ultimate.cases)
  case hyps: (1 L)
  hence  $L = \text{Pos } A$ 
  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 ⟨ $\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma$ ⟩
    using ord-res-7.skip-undefined-pos-ultimate by fastforce
qed
qed

moreover have ord-res-7-invars  $N (U_{er}, \mathcal{F}, \Gamma', C')$ 
  using ⟨ $C = \text{Some } D$ ⟩ first-step7 match-hyps(3) ord-res-7-preserves-invars by
blast

ultimately obtain  $C''$  where
  following-steps7:  $(\text{ord-res-7 } N)^{**} (U_{er}, \mathcal{F}, \Gamma', C') (U_{er}, \mathcal{F}, \Gamma', C'')$  and
  no-more-step7:  $(\nexists C'''. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma', C'') (U_{er}, \mathcal{F}, \Gamma', C'''))$ 
  using MAGIC6 by metis

show ?thesis
proof (intro exI conjI)
  have  $(\text{ord-res-7 } N)^{++} (U_{er}, \mathcal{F}, \Gamma, C) (U_{er}, \mathcal{F}, \Gamma', C'')$ 
    unfolding ⟨ $C = \text{Some } D$ ⟩
    using first-step7 following-steps7 by simp

  thus  $(\text{constant-context ord-res-7})^{++} S7 (N, U_{er}, \mathcal{F}, \Gamma', C'')$ 
    unfolding S7-def by (simp add: tranclp-constant-context)

  show ord-res-7-matches-ord-res-8  $(N, U_{er}, \mathcal{F}, \Gamma', C'')$  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  $N (U_{er}, \mathcal{F}, \Gamma', C'')$ 
      using ⟨ord-res-7-invars  $N (U_{er}, \mathcal{F}, \Gamma', C')$ ⟩ 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 $C'' = \text{Some } C \iff \text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma' \ C$
using *MAGIC5* $\langle \text{ord-res-7-invars } N \ (U_{er}, \mathcal{F}, \Gamma', C'') \ \text{no-more-step7} \rangle$ **by** *metis*
qed
qed
next
case *step-hyps*: $(\text{propagate } A \ C \ \Gamma')$

have
A-in: $A \in \text{atms-of-clss } (N \ \cup \ U_{er})$ **and**
A-undef: $A \notin \text{trail-atms } \Gamma$ **and**
A-least: $\forall y \in \text{atms-of-clss } (N \ \cup \ U_{er}). y \neq A \longrightarrow (\forall A_1 \in \text{trail-atms } \Gamma. A_1 \prec_t y) \longrightarrow A \prec_t y$
using *step-hyps*(3) **unfolding** *linorder-trm.is-least-in-fset-iff* **by** *auto*

have
C-in: $C \in \text{iefac } \mathcal{F} \ |\uparrow \ (N \ \cup \ U_{er})$ **and**
C-can-prop: *clause-could-propagate* $\Gamma \ C \ (\text{Pos } A)$ **and**
C-least: $\forall D \in \text{iefac } \mathcal{F} \ |\uparrow \ (N \ \cup \ U_{er}). D \neq C \longrightarrow \text{clause-could-propagate } \Gamma \ D \ (\text{Pos } A) \longrightarrow C \prec_c D$
using *step-hyps* **unfolding** *atomize-conj linorder-cls.is-least-in-ffilter-iff* **by** *argo*

hence
Pos-A-undef: $\neg \text{trail-defined-lit } \Gamma \ (\text{Pos } A)$ **and**
C-max-lit: *linorder-lit.is-maximal-in-mset* $C \ (\text{Pos } A)$ **and**
C-almost-false: *trail-false-cls* $\Gamma \ \{\#K \in \# C. K \neq \text{Pos } A\# \}$
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 $C \in \text{iefac } \mathcal{F} \ |\uparrow \ (N \ \cup \ U_{er})$
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 $\neg \text{trail-false-cls } \Gamma \ C$
using *step-hyps C-in* **by** *metis*
next
show *ord-res.is-maximal-lit* $(\text{Pos } A) \ C$
using *step-hyps* **by** *blast*
next
show $\neg (\exists Aa \in \text{atms-of-clss } (N \ \cup \ U_{er}). Aa \prec_t \text{atm-of } (\text{Pos } A) \wedge Aa \notin \text{trail-atms } \Gamma)$
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  $\Gamma$  (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  $\Gamma$  {#K  $\in$  # C. K  $\neq$  Pos A#}
    using  $\langle$  clause-could-propagate  $\Gamma$  C (Pos A)  $\rangle$ 
    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  $\Gamma$  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
  by argo
next
fix D :: 'f gclause
assume
  D-in: D  $\in$  |efac  $\mathcal{F}$  | $\uparrow$  (N  $\cup$  |  $U_{er}$ ) and
  D-neq: D  $\neq$  C and
  D-spec-disj: trail-false-cls  $\Gamma$  D  $\vee$ 
    ord-res-8-can-decide-neg N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
    ord-res-8-can-skip-undefined-neg N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
    ord-res-8-can-skip-undefined-pos-ultimate N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
    ord-res-8-can-produce N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
    ord-res-8-can-factorize N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D

hence
  ord-res-8-can-decide-neg N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
  ord-res-8-can-skip-undefined-neg N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
  ord-res-8-can-skip-undefined-pos-ultimate N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
  ord-res-8-can-produce N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D  $\vee$ 
  ord-res-8-can-factorize N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D
  using step-hyps D-in by metis

thus C  $\prec_c$  D
proof (elim disjE)
  assume ord-res-8-can-decide-neg N  $U_{er}$   $\mathcal{F}$   $\Gamma$  D
  thus C  $\prec_c$  D
  proof (cases N  $U_{er}$   $\mathcal{F}$   $\Gamma$  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 $A \prec_t \text{atm-of } L$
using *hyps*
unfolding *linorder-trm.is-least-in-ffilter-iff*
by *argo*
hence $\text{Pos } A \prec_l L$
by (*cases L*) *simp-all*
thus *?thesis*
using *C-max-lit $\langle \text{linorder-lit.is-maximal-in-mset } D \ L \rangle$*
by (*metis linorder-lit.mulp-if-maximal-less-that-maximal*)

qed

next

assume *ord-res-8-can-skip-undefined-neg $N U_{er} \mathcal{F} \Gamma D$*
thus $C \prec_c D$
proof (*cases $N U_{er} \mathcal{F} \Gamma D$ rule: ord-res-8-can-skip-undefined-neg.cases*)
case *hyps: (1 L)*
hence $\text{atm-of } L = A$
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 $\text{Pos } A \prec_l L$
using *$\langle \text{is-neg } L \rangle$* **by** (*cases L*) *simp-all*
thus *?thesis*
using *C-max-lit $\langle \text{linorder-lit.is-maximal-in-mset } D \ L \rangle$*
by (*metis linorder-lit.mulp-if-maximal-less-that-maximal*)

qed

next

assume *ord-res-8-can-skip-undefined-pos-ultimate $N U_{er} \mathcal{F} \Gamma D$*
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
 $\neg (\exists A | \in | \text{atms-of-clss} (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma)$

and

L-undef: $\neg \text{trail-defined-lit } \Gamma L$ 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-clsI
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 \prec_c D*
using *D-in D-neq C-least by metis*

next
assume *ord-res-8-can-factorize N U_{er} \mathcal{F} Γ D*

then obtain L where
D-max-lit: ord-res.is-maximal-lit L D and
 $\neg (\exists A | \in | \text{atms-of-cls} (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma)$

and
L-undef: \neg trail-defined-lit Γ L and
D-almost-false: trail-false-cls Γ {#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-clsI
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 \prec_c D*
using *D-in D-neq C-least by metis*

qed
qed

hence $C = \text{Some } C$
using *match-hyps* **by** *metis*

define C' **where**
 $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) C) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))))$

have *first-step7*: $\text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', C')$
proof (*rule ord-res-7.production*)
show $\neg \text{trail-false-cls } \Gamma C$
using *C-in step-hyps(2)* **by** *blast*

next
show $\text{ord-res.is-maximal-lit } (\text{Pos } A) C$
using *C-max-lit* **by** *force*

next
show $\neg (\exists Aa \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). Aa \prec_t \text{atm-of } (\text{Pos } A) \wedge Aa \notin \mid \text{trail-atms } \Gamma)$
by (*metis A-least Γ -lower-set linorder-trm.dual-order.asym linorder-trm.neq-iff linorder-trm.not-in-lower-setI literal.sel(1)*)

next
show $\neg \text{trail-defined-lit } \Gamma (\text{Pos } A)$
using *Pos-A-undef* .

next
show $\text{is-pos } (\text{Pos } A)$
by *simp*

next
show $\text{trail-false-cls } \Gamma \{ \#K \in \# C. K \neq \text{Pos } A \# \}$
using *C-almost-false* .

next
show $\text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C$
using *step-hyps* **by** *argo*

next
show $\Gamma' = (\text{Pos } A, \text{Some } C) \# \Gamma$
using *step-hyps* **by** *argo*

next
show $C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } (\text{ffilter } ((\prec_c) C) (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))))$
using *C'-def* .

qed

moreover have $\text{ord-res-7-invars } N (U_{er}, \mathcal{F}, \Gamma', C')$
using $\langle C = \text{Some } C \rangle$ *first-step7 match-hyps(3) ord-res-7-preserves-invars* **by** *blast*

ultimately obtain C'' **where**
following-steps7: $(\text{ord-res-7 } N)^{**} (U_{er}, \mathcal{F}, \Gamma', C') (U_{er}, \mathcal{F}, \Gamma', C'')$ **and**
no-more-step7: $(\exists C'''. \text{ord-res-7 } N (U_{er}, \mathcal{F}, \Gamma', C'') (U_{er}, \mathcal{F}, \Gamma', C'''))$
using *MAGIC6* **by** *metis*

show *?thesis*
proof (*intro exI conjI*)
have (*ord-res-7 N*)⁺⁺ ($U_{er}, \mathcal{F}, \Gamma, C$) ($U_{er}, \mathcal{F}, \Gamma', C''$)
unfolding $\langle C = \text{Some } C \rangle$
using *first-step7 following-steps7* **by** *simp*

thus (*constant-context ord-res-7*)⁺⁺ $S7$ ($N, U_{er}, \mathcal{F}, \Gamma', C''$)
unfolding $S7\text{-def}$ **by** (*simp add: tranclp-constant-context*)

show *ord-res-7-matches-ord-res-8* ($N, U_{er}, \mathcal{F}, \Gamma', C''$) $S8'$
unfolding $S8'\text{-def}$ $\langle s8' = (U_{er}, \mathcal{F}, \Gamma') \rangle$
proof (*intro ord-res-7-matches-ord-res-8.intros allI*)
show *ord-res-7-invars* N ($U_{er}, \mathcal{F}, \Gamma', C''$)
using $\langle \text{ord-res-7-invars } N (U_{er}, \mathcal{F}, \Gamma', C') \rangle$ *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 $C :: 'f \text{ gclause}$
show $C'' = \text{Some } C \longleftrightarrow \text{is-least-nonskipped-clause } N U_{er} \mathcal{F} \Gamma' C$
using $\text{MAGIC5} \langle \text{ord-res-7-invars } N (U_{er}, \mathcal{F}, \Gamma', C') \rangle$ *no-more-step7* **by**
metis
qed
qed
next
case *step-hyps: (factorize A C F')*

have
 $A\text{-in}: A \in | \text{atms-of-clss } (N \cup | U_{er})$ **and**
 $A\text{-undef}: A \notin | \text{trail-atms } \Gamma$ **and**
 $A\text{-least}: \forall y \in | \text{atms-of-clss } (N \cup | U_{er}). y \neq A \longrightarrow (\forall A_1 \in | \text{trail-atms } \Gamma. A_1$
 $\prec_t y) \longrightarrow A \prec_t y$
using *step-hyps(3)* **unfolding** *linorder-trm.is-least-in-fset-iff* **by** *auto*

have
 $C\text{-in}: C \in | \text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er})$ **and**
 $C\text{-can-prop: clause-could-propagate } \Gamma C (\text{Pos } A)$ **and**
 $C\text{-least}: \forall D \in | \text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er}).$
 $D \neq C \longrightarrow \text{clause-could-propagate } \Gamma D (\text{Pos } A) \longrightarrow C \prec_c D$
using *step-hyps* **unfolding** *atomize-conj linorder-cls.is-least-in-filter-iff* **by**
argo

hence
 $\text{Pos-}A\text{-undef}: \neg \text{trail-defined-lit } \Gamma (\text{Pos } A)$ **and**
 $C\text{-max-lit}: \text{linorder-lit.is-maximal-in-mset } C (\text{Pos } A)$ **and**
 $C\text{-almost-false}: \text{trail-false-cls } \Gamma \{\#K \in \# C. K \neq \text{Pos } A\# \}$
unfolding *atomize-conj clause-could-propagate-def* **by** *argo*

```

have C-not-false:  $\neg$  trail-false-cls  $\Gamma$  C
  using C-in step-hyps by metis

have no-undef-atm-lt-A:
   $\neg (\exists Aa | \in | \text{atms-of-clss} (N \cup U_{er}). Aa \prec_t A \wedge Aa \notin | \text{trail-atms } \Gamma)$ 
by (metis A-least  $\Gamma$ -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-filter-iff
proof (intro conjI ballI impI)
  show  $C \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er})$ 
  using C-in .
next
have ord-res-8-can-factorize  $N U_{er} \mathcal{F} \Gamma C$ 
proof (intro ord-res-8-can-factorize.intros)
  show  $\neg$  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  $\neg (\exists Aa | \in | \text{atms-of-clss} (N \cup U_{er}). Aa \prec_t \text{atm-of} (Pos A) \wedge Aa \notin |$ 
trail-atms  $\Gamma)$ 
  using no-undef-atm-lt-A by simp
next
show  $\neg$  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  $\neg$  ord-res.is-strictly-maximal-lit (Pos A) C
  using step-hyps by argo
qed

thus 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  $N U_{er} \mathcal{F} \Gamma C$ 
  using  $\langle$ ord-res-8-can-factorize  $N U_{er} \mathcal{F} \Gamma C \rangle$  by argo
next

```


fix $D :: 'f \text{ gclause}$
assume
 $D\text{-in}: D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
 $D\text{-neg}: D \neq C$ **and**
 $D\text{-spec-disj}: \text{trail-false-cls } \Gamma D \vee$
 $\text{ord-res-8-can-decide-neg } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-skip-undefined-neg } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-skip-undefined-pos-ultimate } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-produce } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-factorize } N U_{er} \mathcal{F} \Gamma D$

hence
 $\text{ord-res-8-can-decide-neg } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-skip-undefined-neg } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-skip-undefined-pos-ultimate } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-produce } N U_{er} \mathcal{F} \Gamma D \vee$
 $\text{ord-res-8-can-factorize } N U_{er} \mathcal{F} \Gamma D$
using $\text{step-hyps } D\text{-in}$ **by** metis

thus $C \prec_c D$
proof (elim disjE)
assume $\text{ord-res-8-can-decide-neg } N U_{er} \mathcal{F} \Gamma D$
thus $C \prec_c D$
proof ($\text{cases } N U_{er} \mathcal{F} \Gamma D$ $\text{rule: ord-res-8-can-decide-neg.cases}$)
case $\text{hyps}: (1 L A')$
hence $A = A'$
using step-hyps
by ($\text{smt (verit, del-insts) } \Gamma\text{-lower-set linorder-trm.antisym-conv3}$
 $\text{linorder-trm.dual-order.strict-implies-not-eq linorder-trm.dual-order.strict-trans}$
 $\text{linorder-trm.is-least-in-ffilter-iff linorder-trm.not-in-lower-setI}$)
hence $A \prec_t \text{atm-of } L$
using hyps
unfolding $\text{linorder-trm.is-least-in-ffilter-iff}$
by argo
hence $\text{Pos } A \prec_l L$
by ($\text{cases } L$) simp-all
thus $?thesis$
using $C\text{-max-lit } \langle \text{linorder-lit.is-maximal-in-mset } D L \rangle$
by ($\text{metis linorder-lit.mulp-if-maximal-less-that-maximal}$)

qed
next
assume $\text{ord-res-8-can-skip-undefined-neg } N U_{er} \mathcal{F} \Gamma D$
thus $C \prec_c D$
proof ($\text{cases } N U_{er} \mathcal{F} \Gamma D$ $\text{rule: ord-res-8-can-skip-undefined-neg.cases}$)
case $\text{hyps}: (1 L)$
hence $\text{atm-of } L = A$
using step-hyps
by (smt (verit, best)
 $A\text{-in } A\text{-least } A\text{-undef } D\text{-in } \Gamma\text{-lower-set atm-of-in-atms-of-clsI}$)

$linorder-lit.is-maximal-in-mset-iff\ linorder-trm.antisym-conv3$
 $linorder-trm.not-in-lower-setI\ trail-defined-lit-iff-trail-defined-atm)$
hence $Pos\ A \prec_l L$
using $\langle is-neg\ L \rangle$ **by** $(cases\ L)\ simp-all$
thus $?thesis$
using $C-max-lit\ \langle linorder-lit.is-maximal-in-mset\ D\ L \rangle$
by $(metis\ linorder-lit.multip-if-maximal-less-that-maximal)$
qed
next
assume $ord-res-8-can-skip-undefined-pos-ultimate\ N\ U_{er}\ \mathcal{F}\ \Gamma\ D$
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 $hypos:\ (1\ L)$
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**
 $\neg (\exists A | \in |atms-of-clss\ (N\ \cup\ U_{er}).\ A \prec_t\ atm-of\ L \wedge A \notin | trail-atms\ \Gamma)$
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 $(smt\ (verit,\ ccfv-SIG)\ A-in\ A-least\ A-undef\ D-in\ \Gamma-lower-set$
 $linorder-lit.is-maximal-in-mset-iff\ linorder-trm.antisym-conv3$
 $linorder-trm.not-in-lower-setI\ atm-of-in-atms-of-clssI$
 $trail-defined-lit-iff-trail-defined-atm)$

hence $L = Pos\ A$
using $\langle is-pos\ L \rangle$ **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** $metis$
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**

$\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma)$
and
 $L\text{-undef}: \neg \text{trail-defined-lit } \Gamma L$ **and**
 $D\text{-almost-false}: \text{trail-false-clc } \Gamma \{\#K \in \# D. K \neq L\# \}$ **and**
 $\text{is-pos } L$
by (*auto elim: ord-res-8-can-factorize.cases ord-res-8-can-produce.cases*)

hence $\text{atm-of } L = A$
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 $L = \text{Pos } A$
using *is-pos L* **by** (*cases L*) *simp-all*

hence *clause-could-propagate* $\Gamma D (\text{Pos } A)$
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 $C = \text{Some } C$
using *match-hyps* **by** *metis*

define C' **where**
 $C' = \text{Some } (\text{efac } C)$

have *first-step7: ord-res-7* $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}', \Gamma, C')$
unfolding *C'-def*
proof (*rule ord-res-7.factoring*)
show $\neg \text{trail-false-clc } \Gamma C$
using *C-not-false* .

next
show *ord-res.is-maximal-lit* $(\text{Pos } A) C$
using *C-max-lit* .

next
show $\neg (\exists Aa | \in | \text{atms-of-clss } (N \cup U_{er}). Aa \prec_t \text{atm-of } (\text{Pos } A) \wedge Aa \notin | \text{trail-atms } \Gamma)$
using *no-undef-atm-lt-A* **by** *simp*

next
show $\neg \text{trail-defined-lit } \Gamma (\text{Pos } A)$
using *Pos-A-undef* .

next
show *is-pos* $(\text{Pos } A)$

```

    by simp
  next
  show trail-false-cls  $\Gamma$  { $\#K \in \# C. K \neq Pos A \#$ }
    using C-almost-false .
  next
  show  $\neg ord-res.is-strictly-maximal-lit (Pos A) C$ 
    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  $\langle C = Some C \rangle first-step7 match-hyps(3) ord-res-7-preserves-invars$  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  $\langle C = Some C \rangle$ 
    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  $\langle s8' = (U_{er}, \mathcal{F}', \Gamma) \rangle$ 
  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  $\langle ord-res-7-invars N (U_{er}, \mathcal{F}', \Gamma, \mathcal{C}') \rangle following-steps7$ 
      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  $C :: 'f g clause$ 
  show  $\mathcal{C}'' = Some C \iff is-least-nonskipped-clause N U_{er} \mathcal{F}' \Gamma C$ 
    using MAGIC5  $\langle ord-res-7-invars N (U_{er}, \mathcal{F}', \Gamma, \mathcal{C}'') \rangle no-more-step7$  by
metis
qed
qed
next
case step-hyps: (resolution E A D  $U_{er}' \Gamma'$ )

```

note $E\text{-max-lit} = \langle \text{ord-res.is-maximal-lit } (\text{Neg } A) \ E \rangle$

have

$E\text{-in}$: $E \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})$ **and**

$E\text{-false}$: $\text{trail-false-cls } \Gamma \ E$ **and**

$E\text{-least}$: $\forall F \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er}). F \neq E \longrightarrow \text{trail-false-cls } \Gamma \ F \longrightarrow E$

$\prec_c F$

using step-hyps

unfolding atomize-conj

unfolding $\text{linorder-cls.is-least-in-filter-iff}$

by argo

have $\text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ E$

unfolding $\text{is-least-nonskipped-clause-def}$

unfolding $\text{linorder-cls.is-least-in-filter-iff}$

proof ($\text{intro } \text{conjI } \text{ballI } \text{impI}$)

show $E \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})$

using $E\text{-in}$.

next

show $\text{trail-false-cls } \Gamma \ E \vee$

$\text{ord-res-8-can-decide-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ E \vee$

$\text{ord-res-8-can-skip-undefined-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ E \vee$

$\text{ord-res-8-can-skip-undefined-pos-ultimate } N \ U_{er} \ \mathcal{F} \ \Gamma \ E \vee$

$\text{ord-res-8-can-produce } N \ U_{er} \ \mathcal{F} \ \Gamma \ E \vee$

$\text{ord-res-8-can-factorize } N \ U_{er} \ \mathcal{F} \ \Gamma \ E$

using $E\text{-false}$ **by** argo

next

fix $F :: 'f \ \text{gclause}$

assume

$F\text{-in}$: $F \in | \text{iefac } \mathcal{F} \ |^{\uparrow} (N \ | \cup \ | \ U_{er})$ **and**

$F\text{-neg}$: $F \neq E$ **and**

$D\text{-spec-disj}$: $\text{trail-false-cls } \Gamma \ F \vee$

$\text{ord-res-8-can-decide-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ F \vee$

$\text{ord-res-8-can-skip-undefined-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ F \vee$

$\text{ord-res-8-can-skip-undefined-pos-ultimate } N \ U_{er} \ \mathcal{F} \ \Gamma \ F \vee$

$\text{ord-res-8-can-produce } N \ U_{er} \ \mathcal{F} \ \Gamma \ F \vee$

$\text{ord-res-8-can-factorize } N \ U_{er} \ \mathcal{F} \ \Gamma \ F$

show $E \prec_c F$

using $D\text{-spec-disj}$

proof ($\text{elim } \text{disjE}$)

assume $\text{trail-false-cls } \Gamma \ F$

thus $E \prec_c F$

using $E\text{-least } F\text{-in } F\text{-neg}$ **by** metis

next

assume $\text{ord-res-8-can-decide-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ F$

thus $E \prec_c F$

proof ($\text{cases } N \ U_{er} \ \mathcal{F} \ \Gamma \ F \ \text{rule: } \text{ord-res-8-can-decide-neg.cases}$)

case $\text{hyps: } (1 \ L' \ A')$

```

      thus ?thesis
      using no-undef-atom-le-max-lit-if-lt-false-clause[
        OF  $\Gamma$ -lower-set  $E$ -in  $E$ -false  $E$ -max-lit  $F$ -in  $\langle$ ord-res.is-maximal-lit  $L'$ 
F>]
      by (metis (no-types, lifting)  $F$ -neg linorder-cls.neq-iff
        linorder-trm.is-least-in-ffilter-iff reflclp-iff)
    qed
  next
  assume ord-res-8-can-skip-undefined-neg  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$ 
  thus  $E \prec_c F$ 
  proof (cases  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$  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  $\Gamma$ -lower-set  $E$ -in  $E$ -false  $E$ -max-lit  $F$ -in  $\langle$ ord-res.is-maximal-lit  $L'$ 
F>]
      by (metis  $F$ -in  $F$ -neg atm-of-in-atms-of-clsI 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  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$ 
  thus  $E \prec_c F$ 
  proof (cases  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$  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  $\Gamma$ -lower-set  $E$ -in  $E$ -false  $E$ -max-lit  $F$ -in  $\langle$ ord-res.is-maximal-lit  $L'$ 
F>]
      by (metis  $F$ -in  $F$ -neg atm-of-in-atms-of-clsI 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  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$ 
  thus  $E \prec_c F$ 
  proof (cases  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$  rule: ord-res-8-can-produce.cases)
    case (1  $L'$ )
      thus ?thesis
      using no-undef-atom-le-max-lit-if-lt-false-clause[
        OF  $\Gamma$ -lower-set  $E$ -in  $E$ -false  $E$ -max-lit  $F$ -in  $\langle$ ord-res.is-maximal-lit  $L'$ 
F>]
      by (metis  $F$ -in  $F$ -neg atm-of-in-atms-of-clsI 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  $N$   $U_{er}$   $\mathcal{F}$   $\Gamma$   $F$ 
  thus  $E \prec_c F$ 

```

```

proof (cases N Uer  $\mathcal{F}$   $\Gamma$  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  $\Gamma$ -lower-set E-in E-false E-max-lit F-in  $\langle$ ord-res.is-maximal-lit L'
F $\rangle$ ]
  by (metis F-in F-neq atm-of-in-atms-of-clsI 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 C = Some E
using match-hyps by metis

obtain C' where first-step $\gamma$ : ord-res-7 N (Uer,  $\mathcal{F}$ ,  $\Gamma$ , Some E) (Uer',  $\mathcal{F}$ ,  $\Gamma'$ ,
C')
proof atomize-elim
  have (Pos A, Some D)  $\in$  set  $\Gamma$ 
    using  $\langle$ map-of  $\Gamma$  (Pos A) = Some (Some D) $\rangle$  by (metis map-of-SomeD)

  hence D-almost-false: trail-false-cls  $\Gamma$  {#K  $\in$ # D. K  $\neq$  Pos A#}
    using ord-res-7-invars-implies-propagated-clause-almost-false
       $\langle$ ord-res-7-invars N (Uer,  $\mathcal{F}$ ,  $\Gamma$ , C) $\rangle$  by metis

  have eres-false: trail-false-cls  $\Gamma$  (eres D E)
    unfolding trail-false-cls-def
  proof (intro ballI)
    fix K :: 'f gliteral
    assume K  $\in$ # eres D E
    hence K  $\in$ # D  $\wedge$  K  $\neq$  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  $\Gamma$  K
    proof (elim disjE conjE)
      assume K  $\in$ # D and K  $\neq$  Pos A
      thus trail-false-lit  $\Gamma$  K
        using D-almost-false unfolding trail-false-cls-def by simp
    next
      assume K  $\in$ # E
      thus trail-false-lit  $\Gamma$  K
        using E-false unfolding trail-false-cls-def by simp
    qed
  qed

  show  $\exists$  C'. ord-res-7 N (Uer,  $\mathcal{F}$ ,  $\Gamma$ , Some E) (Uer',  $\mathcal{F}$ ,  $\Gamma'$ , C')
  proof (cases eres D E = {#})
    case True
      hence  $\bigwedge$ Ln.  $\forall$  K. ord-res.is-maximal-lit K (eres D E)  $\longrightarrow$  atm-of K  $\preceq_t$ 

```

```

atm-of (fst Ln)
  unfolding linorder-lit.is-maximal-in-mset-iff
  by simp
  hence  $\Gamma' = \text{dropWhile } (\lambda Ln. \text{True}) \Gamma$ 
  using step-hyps by meson
  hence  $\Gamma' = []$ 
  by simp
  show ?thesis
  proof (intro exI)
    show ord-res-7 N ( $U_{er}, \mathcal{F}, \Gamma, \text{Some } E$ ) ( $U_{er}', \mathcal{F}, \Gamma', \text{Some } \{\#\}$ )
    using ord-res-7.resolution-bot[OF E-false E-max-lit]
    <map-of  $\Gamma$  ( $\text{Pos } A$ ) =  $\text{Some } (\text{Some } D)$ > < $U_{er}' = \text{finsert } (\text{eres } D \ E) \ U_{er}$ >
True < $\Gamma' = []$ >
  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  $\bigwedge Ln. (\forall K. \text{ord-res.is-maximal-lit } K \ (\text{eres } D \ E) \longrightarrow \text{atm-of } K \preceq_t$ 
atm-of (fst Ln))  $\longleftrightarrow$ 
  atm-of K  $\preceq_t$  atm-of (fst Ln)
  by (metis linorder-lit.Uniq-is-maximal-in-mset the1-equality')
  hence  $\Gamma'$ -eq:  $\Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma$ 
  using step-hyps by meson
  show ?thesis
  proof (cases K)
    case (Pos AK)
    hence K-pos: is-pos K
    by simp

    then show ?thesis
    using ord-res-7.resolution-pos[OF E-false E-max-lit - - - False  $\Gamma'$ -eq
eres-max-lit K-pos]
    using step-hyps by fastforce
  next
  case (Neg AK)

  hence K-neg: is-neg K
  by simp

  have trail-false-lit  $\Gamma$  K
  using eres-false eres-max-lit
  unfolding linorder-lit.is-maximal-in-mset-iff trail-false-cls-def by metis

  hence  $\exists \text{opt}. (- K, \text{opt}) \in \text{set } \Gamma$ 
  unfolding trail-false-lit-def by auto

  moreover have  $\forall Ln \in \text{set } \Gamma. \text{is-neg } (\text{fst } Ln) = (\text{snd } Ln = \text{None})$ 

```



```

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 (Uer', 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)** (Uer', ℱ, Γ', C') (Uer', ℱ, Γ', C'') and
  no-more-step7: (∄ C'''. ord-res-7 N (Uer', ℱ, Γ', C'') (Uer', ℱ, Γ', C'''))
  using MAGIC6 by metis

show ?thesis
proof (intro exI conjI)
  have (ord-res-7 N)** (Uer, ℱ, Γ, C) (Uer', ℱ, Γ', C'')
    unfolding ⟨C = Some E⟩
    using first-step7 following-steps7 by simp

  thus (constant-context ord-res-7)** S7 (N, Uer', ℱ, Γ', C'')
    unfolding S7-def by (simp add: tranclp-constant-context)

  show ord-res-7-matches-ord-res-8 (N, Uer', ℱ, Γ', C'') S8'
    unfolding S8'-def ⟨s8' = (Uer', ℱ, Γ')⟩
  proof (intro ord-res-7-matches-ord-res-8.intros allI)
    show ord-res-7-invars N (Uer', ℱ, Γ', C'')
      using ⟨ord-res-7-invars N (Uer', ℱ, Γ', C')⟩ following-steps7
      rtranclp-ord-res-7-preserves-ord-res-7-invars by blast

    show ord-res-8-invars N (Uer', ℱ, Γ')
      using invars-s8' step-hyps(1) by blast

  fix C :: 'f gclause

```

```

    show  $C'' = \text{Some } C \iff \text{is-least-nonskipped-clause } N \ U_{er'} \ \mathcal{F} \ \Gamma' \ C$ 
    using MAGIC5 <ord-res-7-invars  $N \ (U_{er'}, \mathcal{F}, \Gamma', C'')$ > no-more-step7 by
metis
  qed
  qed
  qed
  qed

theorem bisimulation-ord-res-7-ord-res-8:
  defines match  $\equiv \lambda-. \text{ord-res-7-matches-ord-res-8}$ 
  shows  $\exists (MATCH :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-7-state} \Rightarrow 'f \text{ord-res-8-state} \Rightarrow \text{bool})$ 
 $\mathcal{R}_f \ \mathcal{R}_b.$ 
  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  $\forall S. \text{ord-res-7-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-7 } S \ S')$ 
  by (metis finished-def ord-res-7-semantics.final-finished)
next
  show  $\forall S. \text{ord-res-8-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-8 } S \ S')$ 
  by (metis finished-def ord-res-8-semantics.final-finished)
next
  show  $\forall i \ S7 \ S8. \text{match } i \ S7 \ S8 \longrightarrow \text{ord-res-7-final } S7 \iff \text{ord-res-8-final } S8$ 
  unfolding match-def
  using ord-res-7-final-iff-ord-res-8-final by metis
next
  show  $\forall i \ S7 \ S8. \text{match } i \ S7 \ S8 \longrightarrow$ 
  safe-state (constant-context ord-res-7) ord-res-7-final  $S7 \wedge$ 
  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 ( $\lambda - . \text{False}$ )
    by simp
next
  show  $\forall i S7 S8 S8'. \text{match } i S7 S8 \longrightarrow \text{constant-context ord-res-8 } S8 S8' \longrightarrow$ 
     $(\exists i' S7'. (\text{constant-context ord-res-7})^{++} S7 S7' \wedge \text{match } i' S7' S8') \vee$ 
     $(\exists i'. \text{match } i' S7 S8' \wedge \text{False})$ 
    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 \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-8-matches-ord-res-9* :: *'f ord-res-8-state* \Rightarrow *'f ord-res-9-state* \Rightarrow
bool **where**

ord-res-8-invars $N (U_{er}, \mathcal{F}, \Gamma) \Longrightarrow$
ord-res-8-matches-ord-res-9 $(N, U_{er}, \mathcal{F}, \Gamma) (N, U_{er}, \mathcal{F}, \Gamma)$

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* $S8 \longleftrightarrow$ *ord-res-8-final* $S9$
using *match*

proof (*cases* $S8 S9$ *rule*: *ord-res-8-matches-ord-res-9.cases*)

case $(1 N U_{er} \mathcal{F} \Gamma)$
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 $\exists S8'. (\text{constant-context ord-res-8})^{++} S8 S8' \wedge \text{ord-res-8-matches-ord-res-9}$
 $S8' S9'$

using *match*

proof (*cases* $S8 S9$ *rule*: *ord-res-8-matches-ord-res-9.cases*)

case *match-hyps*: $(1 N U_{er} \mathcal{F} \Gamma)$

note $S8\text{-def} = \langle S8 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$

note $S9\text{-def} = \langle S9 = (N, U_{er}, \mathcal{F}, \Gamma) \rangle$

```

note invars = ⟨ord-res-8-invars N (Uer, F, Γ)⟩

obtain s9' where S9'-def: S9' = (N, s9') and step': ord-res-9 N (Uer, F, Γ)
s9'
  using step unfolding S9-def
  using constant-context.cases by blast

have ord-res-8 N (Uer, F, Γ) s9' ∨ (ord-res-8 N OO ord-res-8 N) (Uer, F, Γ)
s9'
  using step' ord-res-9-is-one-or-two-ord-res-9-steps by metis

hence steps8: (ord-res-8 N)++ (Uer, 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)
Rf Rb.
    bisimulation
      (constant-context ord-res-8) ord-res-8-final
      (constant-context ord-res-9) ord-res-8-final
      MATCH Rf Rb
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  $\forall i S8 S9. \text{match } i S8 S9 \longrightarrow \text{ord-res-8-final } S8 \longleftrightarrow \text{ord-res-8-final } S9$ 
    unfolding match-def
    using ord-res-8-final-iff-ord-res-9-final by metis
next
  show  $\forall i S8 S9. \text{match } i S8 S9 \longrightarrow$ 
     $\text{safe-state } (\text{constant-context } \text{ord-res-8}) \text{ ord-res-8-final } S8 \wedge$ 
     $\text{safe-state } (\text{constant-context } \text{ord-res-9}) \text{ ord-res-8-final } S9$ 
  proof (intro allI impI conjI)
    fix i S8 S9
    assume match: match i S8 S9
    show  $\text{safe-state } (\text{constant-context } \text{ord-res-8}) \text{ ord-res-8-final } S8$ 
      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  $\text{safe-state } (\text{constant-context } \text{ord-res-9}) \text{ ord-res-8-final } S9$ 
      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 ( $\lambda - . \text{False}$ )
    by simp
next
  show  $\forall i S8 S9 S9'. \text{match } i S8 S9 \longrightarrow \text{constant-context } \text{ord-res-9 } S9 S9' \longrightarrow$ 
     $(\exists i' S8'. (\text{constant-context } \text{ord-res-8})^{++} S8 S8' \wedge \text{match } i' S8' S9') \vee$ 
     $(\exists i'. \text{match } i' S8 S9' \wedge \text{False})$ 
    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* \Rightarrow '*f* *ord-res-10-state*
 \Rightarrow *bool* **where**

```

  ord-res-8-invars  $N (U_{er}, \mathcal{F}, \Gamma_9) \Longrightarrow$ 
  ord-res-10-invars  $N (U_{er}, \mathcal{F}, \Gamma_{10}) \Longrightarrow$ 
  list-all2  $(\lambda x y. \text{fst } x = \text{fst } y) \Gamma_9 \Gamma_{10} \Longrightarrow$ 
  list-all2  $(\lambda x y. \text{snd } y \neq \text{None} \longrightarrow x = y) \Gamma_9 \Gamma_{10} \Longrightarrow$ 
  ord-res-9-matches-ord-res-10  $(N, U_{er}, \mathcal{F}, \Gamma_9) (N, U_{er}, \mathcal{F}, \Gamma_{10})$ 

```

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  $ord-res-8-final\ S9 \longleftrightarrow ord-res-8-final\ S10$ 
using  $match$ 
proof ( $cases\ S9\ S10\ rule: ord-res-9-matches-ord-res-10.cases$ )
  case  $match-hyps: (1\ N\ U_{er}\ \mathcal{F}\ \Gamma_9\ \Gamma_{10})$ 
  then show  $?thesis$ 
    using  $trail-atms-eq-trail-atms-if-same-lits[OF\ \langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle]$ 
    using  $trail-false-cls-eq-trail-false-cls-if-same-lits[OF\ \langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle]$ 
    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  $\exists S9'. (constant-context\ ord-res-9)^{++}\ S9\ S9' \wedge ord-res-9-matches-ord-res-10\ S9'\ S10'$ 
  using  $match$ 
proof ( $cases\ S9\ S10\ rule: ord-res-9-matches-ord-res-10.cases$ )
  case  $match-hyps: (1\ N\ U_{er}\ \mathcal{F}\ \Gamma_9\ \Gamma_{10})$ 

  note  $S9-def = \langle S9 = (N,\ U_{er},\ \mathcal{F},\ \Gamma_9) \rangle$ 
  note  $S10-def = \langle S10 = (N,\ U_{er},\ \mathcal{F},\ \Gamma_{10}) \rangle$ 
  note  $invars9 = \langle ord-res-8-invars\ N\ (U_{er},\ \mathcal{F},\ \Gamma_9) \rangle$ 
  note  $invars10 = \langle ord-res-10-invars\ N\ (U_{er},\ \mathcal{F},\ \Gamma_{10}) \rangle$ 

  have  $trail-atms\ \Gamma_9 = trail-atms\ \Gamma_{10}$ 
  using  $\langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle\ trail-atms-eq-trail-atms-if-same-lits$ 
  by  $metis$ 

  have  $trail-false-lit\ \Gamma_9 = trail-false-lit\ \Gamma_{10}$ 
  using  $\langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle\ trail-false-lit-eq-trail-false-lit-if-same-lits$ 
  by  $metis$ 

  have  $trail-false-cls\ \Gamma_9 = trail-false-cls\ \Gamma_{10}$ 
  using  $\langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle\ trail-false-cls-eq-trail-false-cls-if-same-lits$ 
  by  $metis$ 

  have  $trail-defined-lit\ \Gamma_9 = trail-defined-lit\ \Gamma_{10}$ 
  using  $\langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle$ 
   $trail-defined-lit-eq-trail-defined-lit-if-same-lits$  by  $metis$ 

  have  $trail-defined-cls\ \Gamma_9 = trail-defined-cls\ \Gamma_{10}$ 
  using  $\langle list-all2\ (\lambda x\ y.\ fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \rangle$ 

```

trail-defined-cls-eq-trail-defined-cls-if-same-lits **by** *metis*

have *clause-could-propagate* $\Gamma_9 = \text{clause-could-propagate } \Gamma_{10}$
unfolding *clause-could-propagate-def*
unfolding $\langle \text{trail-defined-lit } \Gamma_9 = \text{trail-defined-lit } \Gamma_{10} \rangle$
unfolding $\langle \text{trail-false-cls } \Gamma_9 = \text{trail-false-cls } \Gamma_{10} \rangle \dots$

have $\Gamma_9\text{-sorted}$: *sorted-wrt* $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma_9$
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* $N (U_{er}, \mathcal{F}, \Gamma_{10})$
 $s10'$
using *step unfolding S10-def* **by** *(auto elim: constant-context.cases)*

show *?thesis*
using *step10*
proof *(cases N (U_{er}, F, Γ₁₀) s10' rule: ord-res-10.cases)*
case *step-hyps: (decide-neg A Γ₁₀)*

define Γ_9' **where**
 $\Gamma_9' = (\text{Neg } A, \text{None}) \# \Gamma_9$

show *?thesis*
proof *(intro exI conjI)*
have *step9*: *ord-res-9* $N (U_{er}, \mathcal{F}, \Gamma_9) (U_{er}, \mathcal{F}, \Gamma_9')$
proof *(rule ord-res-9.decide-neg)*
show $\neg (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}). \text{trail-false-cls } \Gamma_9 C)$
using *step-hyps* $\langle \text{trail-false-cls } \Gamma_9 = \text{trail-false-cls } \Gamma_{10} \rangle$ **by** *argo*
next
show *linorder-trm.is-least-in-fset*
 $\{|A_2 | \in | \text{atms-of-cls } (N \cup U_{er}). \forall A_1 | \in | \text{trail-atms } \Gamma_9. A_1 \prec_t A_2|\} A$
using *step-hyps* $\langle \text{trail-atms } \Gamma_9 = \text{trail-atms } \Gamma_{10} \rangle$ **by** *metis*
next
show $\neg (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup U_{er}). \text{clause-could-propagate } \Gamma_9 C (\text{Pos } A))$
using *step-hyps* $\langle \text{clause-could-propagate } \Gamma_9 = \text{clause-could-propagate } \Gamma_{10} \rangle$
by *metis*
next
show $\Gamma_9' = (\text{Neg } A, \text{None}) \# \Gamma_9$
using $\Gamma_9'\text{-def}$.
qed

thus *(constant-context ord-res-9)⁺⁺ S9* $(N, U_{er}, \mathcal{F}, \Gamma_9')$
unfolding *S9-def* **by** *(auto intro: constant-context.intros)*

show *ord-res-9-matches-ord-res-10* $(N, U_{er}, \mathcal{F}, \Gamma_9') S10'$
unfolding $\langle S10' = (N, s10') \rangle \langle s10' = (U_{er}, \mathcal{F}, \Gamma_{10}') \rangle$
proof *(rule ord-res-9-matches-ord-res-10.intros)*

```

    show ord-res-8-invars N (Uer, F, Γ9')
      using invars9 step9 ord-res-9-preserves-invars by metis
  next
    show ord-res-10-invars N (Uer, F, Γ10')
      using invars10 step10 ord-res-10-preserves-invars ⟨s10' = (Uer, F, Γ10')⟩
by metis
  next
    show list-all2 (λx y. fst x = fst y) Γ9' Γ10'
      unfolding ⟨Γ9' = (Neg A, None) # Γ9⟩ ⟨Γ10' = (Neg 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' = (Neg A, None) # Γ9⟩ ⟨Γ10' = (Neg A, None) # Γ10⟩
      using ⟨list-all2 (λx y. snd y ≠ None → x = y) Γ9 Γ10⟩
      by simp
  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 (Uer, F, Γ9) (Uer, F', Γ9')
  proof (rule ord-res-9.propagate)
    show ¬ (∃ C |∈| iefac F |' (N |∪| Uer). 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-cls (N |∪| Uer). ∀ 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 |∪| Uer). clause-could-propagate Γ9 C (Pos A)} C
      using step-hyps ⟨clause-could-propagate Γ9 = clause-could-propagate Γ10⟩
by metis
  next
  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, Uer, F', Γ9')
  unfolding S9-def by (auto intro: constant-context.intros)

```



```

show ord-res-9-matches-ord-res-10 (N, Uer, F', Γ9') S10'
  unfolding ⟨S10' = (N, s10')⟩ ⟨s10' = (Uer, F', Γ10')⟩
proof (rule ord-res-9-matches-ord-res-10.intros)
  show ord-res-8-invars N (Uer, F', Γ9')
    using invars9 step9 ord-res-9-preserves-invars by metis
next
  show ord-res-10-invars N (Uer, F', Γ10')
    using invars10 step10 ord-res-10-preserves-invars ⟨s10' = (Uer, 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 (Uer, F, Γ9) (Uer, F', Γ9')
  proof (rule ord-res-9.propagate)
    show ¬ (∃ C |∈| iefac F |↑| (N |∪| Uer). 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-cls (N |∪| Uer). ∀ 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 |∪| Uer). 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, Uer, F', Γ9')
  unfolding S9-def by (auto intro: constant-context.intros)

show ord-res-9-matches-ord-res-10 (N, Uer, F', Γ9') S10'
  unfolding ⟨S10' = (N, s10')⟩ ⟨s10' = (Uer, F', Γ10')⟩
proof (rule ord-res-9-matches-ord-res-10.intros)
  show ord-res-8-invars N (Uer, F', Γ9')
    using invars9 step9 ord-res-9-preserves-invars by metis
  next
  show ord-res-10-invars N (Uer, F', Γ10')
    using invars10 step10 ord-res-10-preserves-invars ⟨s10' = (Uer, 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, Some
(efac C)) # 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, Some
(efac C)) # Γ10⟩
    using ⟨list-all2 (λx y. snd y ≠ None → x = y) Γ9 Γ10⟩
    by simp
  qed
  qed
next
case step-hyps: (resolution D A C Uer' Γ10')

have ∀ Ln Γ'. Γ10 = Ln # Γ' →
  (snd Ln ≠ None) = fBex (iefac F |·| (N |∪| Uer)) (trail-false-cls Γ10) ∧
  (∀ x ∈ set Γ'. snd x = None)
  using invars10 by (simp add: ord-res-10-invars.simps)

then obtain Γ10'' where Γ10 = (Pos A, Some C) # Γ10''
  using ⟨map-of Γ10 (Pos A) = Some (Some C)⟩
  by (metis list.set-cases map-of-SomeD not-Some-eq snd-conv)

then obtain Γ9'' where Γ9 = (Pos A, Some C) # Γ9''
  using ⟨list-all2 (λx y. snd y ≠ None → x = y) Γ9 Γ10⟩
  by (smt (verit, best) list-all2-Cons2 option.discI snd-conv)

define Γ9' where
  Γ9' = dropWhile (λLn. ∀ K. ord-res.is-maximal-lit K (eres C D) →
atm-of K ≼t atm-of (fst Ln)) Γ9

```

```

show ?thesis
proof (intro exI conjI)
  have step9: ord-res-9 N (Uer,  $\mathcal{F}$ ,  $\Gamma_9$ ) (Uer',  $\mathcal{F}$ ,  $\Gamma_9'$ )
  proof (rule ord-res-9.resolution)
    show linorder-cls.is-least-in-fset (ffilter (trail-false-cls  $\Gamma_9$ ) (iefac  $\mathcal{F}$  | $\uparrow$  (N
| $\cup$ | Uer))) D
    using step-hyps  $\langle$ trail-false-cls  $\Gamma_9 =$  trail-false-cls  $\Gamma_{10}$  $\rangle$  by argo
  next
    show ord-res.is-maximal-lit (Neg A) D
    using step-hyps by argo
  next
    show map-of  $\Gamma_9$  (Pos A) = Some (Some C)
    unfolding  $\langle$  $\Gamma_9 =$  (Pos A, Some C) #  $\Gamma_9''$  $\rangle$  by simp
  next
    show Uer' = finsert (eres C D) Uer
    using step-hyps by argo
  next
    show  $\Gamma_9' =$  dropWhile ( $\lambda Ln. \forall K. \text{ord-res.is-maximal-lit } K \text{ (eres } C \ D) \longrightarrow$ 
atm-of  $K \preceq_t$  atm-of (fst Ln))  $\Gamma_9$ 
    using  $\Gamma_9'$ -def .
qed

thus (constant-context ord-res-9)++ S9 (N, Uer',  $\mathcal{F}$ ,  $\Gamma_9'$ )
  unfolding S9-def by (auto intro: constant-context.intros)

show ord-res-9-matches-ord-res-10 (N, Uer',  $\mathcal{F}$ ,  $\Gamma_9'$ ) S10'
  unfolding  $\langle$ S10' = (N, s10') $\rangle$   $\langle$ s10' = (Uer',  $\mathcal{F}$ ,  $\Gamma_{10}'$ ) $\rangle$ 
proof (rule ord-res-9-matches-ord-res-10.intros)
  show ord-res-8-invars N (Uer',  $\mathcal{F}$ ,  $\Gamma_9'$ )
    using invars9 step9 ord-res-9-preserves-invars by metis
  next
    show ord-res-10-invars N (Uer',  $\mathcal{F}$ ,  $\Gamma_{10}'$ )
    using invars10 step10 ord-res-10-preserves-invars  $\langle$ s10' = (Uer',  $\mathcal{F}$ ,  $\Gamma_{10}'$ ) $\rangle$ 
by metis
  next
    define P :: 'f gterm literal  $\times$  'f gterm literal multiset option  $\Rightarrow$  bool where
      P  $\equiv$   $\lambda Ln. \forall K. \text{ord-res.is-maximal-lit } K \text{ (eres } C \ D) \longrightarrow$  atm-of  $K \preceq_t$  atm-of
      (fst Ln)

    have length (takeWhile P  $\Gamma_9$ ) = length (takeWhile P  $\Gamma_{10}$ )
      using  $\langle$ list-all2 ( $\lambda x y. \text{fst } x = \text{fst } y$ )  $\Gamma_9$   $\Gamma_{10}$  $\rangle$ 
    proof (induction  $\Gamma_9$   $\Gamma_{10}$  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 $\Gamma_9 = \text{takeWhile } P \Gamma_9 @ \Gamma_9'$
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* $\langle \Gamma_9' = \text{dropWhile } - \Gamma_9 \rangle$ by *simp*

moreover have $\Gamma_{10} = \text{takeWhile } P \Gamma_{10} @ \Gamma_{10}'$
unfolding *takeWhile-dropWhile-id*
unfolding *P-def* $\langle \Gamma_{10}' = \text{dropWhile } - \Gamma_{10} \rangle$ by *simp*

ultimately have $\bigwedge Q. \text{list-all2 } Q \Gamma_9 \Gamma_{10} \longleftrightarrow$
 $(\text{list-all2 } Q (\text{takeWhile } P \Gamma_9) (\text{takeWhile } P \Gamma_{10}) \wedge \text{list-all2 } Q \Gamma_9' \Gamma_{10}')$
using *list-all2-append* by *metis*

thus

list-all2 $(\lambda x y. \text{fst } x = \text{fst } y) \Gamma_9' \Gamma_{10}'$
list-all2 $(\lambda x y. \text{snd } y \neq \text{None} \longrightarrow x = y) \Gamma_9' \Gamma_{10}'$
unfolding *atomize-conj*
using *list-all2* $(\lambda x y. \text{fst } x = \text{fst } y) \Gamma_9 \Gamma_{10}$
using *list-all2* $(\lambda x y. \text{snd } y \neq \text{None} \longrightarrow x = y) \Gamma_9 \Gamma_{10}$
by (*simp only*.)

qed

qed

qed

qed

theorem *bisimulation-ord-res-9-ord-res-10*:

defines *match* $\equiv \lambda-. \text{ord-res-9-matches-ord-res-10}$

shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-8-state} \Rightarrow 'f \text{ord-res-9-state} \Rightarrow \text{bool})$
 $\mathcal{R}_f \mathcal{R}_b.$

bisimulation

(constant-context ord-res-9) ord-res-8-final

(constant-context ord-res-10) 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-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 $\forall S. \text{ord-res-8-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-9 } S S')$

by (*metis finished-def ord-res-9-semantics.final-finished*)

next

show $\forall S. \text{ord-res-8-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-10 } S S')$

by (*metis finished-def ord-res-10-semantics.final-finished*)

next

show $\forall i S9 S10. \text{match } i S9 S10 \longrightarrow \text{ord-res-8-final } S9 \longleftrightarrow \text{ord-res-8-final } S10$

unfolding *match-def*

```

    using ord-res-9-final-iff-ord-res-10-final by metis
next
show  $\forall i S9 S10. \text{match } i S9 S10 \longrightarrow$ 
    safe-state (constant-context ord-res-9) ord-res-8-final S9  $\wedge$ 
    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 ( $\lambda - . \text{False}$ )
  by simp
next
show  $\forall i S9 S10 S10'. \text{match } i S9 S10 \longrightarrow \text{constant-context ord-res-10 } S10 S10'$ 
 $\longrightarrow$ 
  ( $\exists i' S9'. (\text{constant-context ord-res-9})^{++} S9 S9' \wedge \text{match } i' S9' S10'$ )  $\vee$ 
  ( $\exists i'. \text{match } i' S9 S10' \wedge \text{False}$ )
  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 \Rightarrow '*f* ord-res-11-state
 \Rightarrow bool **where**
ord-res-10-invars $N (U_{er10}, \mathcal{F}, \Gamma) \Longrightarrow$
ord-res-11-invars $N (U_{er11}, \mathcal{F}, \Gamma, \mathcal{C}) \Longrightarrow$
 $U_{er11} = U_{er10} - \{\{\#\}\} \Longrightarrow$
 if $\{\#\} \in \text{iefac } \mathcal{F} \mid^{\dagger} (N \mid \cup \mid U_{er10})$ then $\Gamma = [] \wedge \mathcal{C} = \text{Some } \{\#\}$ else $\mathcal{C} =$
None \Longrightarrow
ord-res-10-matches-ord-res-11 $(N, U_{er10}, \mathcal{F}, \Gamma) (N, U_{er11}, \mathcal{F}, \Gamma, \mathcal{C})$

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* $S10 \longleftrightarrow$ *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 Uer10 F Γ Uer11 C)
  show ?thesis
  proof (rule iffI)
    assume ord-res-8-final S10
    thus ord-res-11-final S11
    unfolding  $\langle S10 = \rightarrow \rangle$ 
    proof (cases (N, Uer10, F, Γ) rule: ord-res-8-final.cases)
      case model-found
      hence  $\{ \# \} \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er10})$ 
        using trail-false-cls-empty by blast
      hence  $C = \text{None}$ 
        using match-hyps by argo
      moreover have  $U_{er11} = U_{er10}$ 
        using match-hyps
        by (metis  $\langle \{ \# \} \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er10}) \rangle$  fimage-eqI finsert-fminus1
finsert-iff
fminus-finsert-absorb funionI2 iefac-empty)
      ultimately show ?thesis
        unfolding  $\langle S11 = \rightarrow \rangle$ 
        using model-found
        using ord-res-11-final.model-found
        by metis
    next
      case contradiction-found
      hence  $\Gamma = [] \wedge C = \text{Some } \{ \# \}$ 
        using match-hyps by argo
      thus ?thesis
        unfolding  $\langle S11 = \rightarrow \rangle$ 
        using ord-res-11-final.contradiction-found by metis
    qed
  next
    assume ord-res-11-final S11
    thus ord-res-8-final S10
    unfolding  $\langle S11 = \rightarrow \rangle$ 
    proof (cases (N, Uer11, F, Γ, C) rule: ord-res-11-final.cases)
      case model-found
      show ?thesis
        unfolding  $\langle S10 = \rightarrow \rangle$ 
        proof (rule ord-res-8-final.model-found)
          show  $\neg (\exists A \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er10}). A \mid \notin \mid \text{trail-atms } \Gamma)$ 
            by (metis (no-types, lifting) fimage-iff fminus-finsert-absorb fminus-idemp
funionCI
iefac-empty local.model-found(1,2) match-hyps(5,6) option.simps(3))
          next
            show  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er10}). \text{trail-false-cls } \Gamma \ C)$ 
              by (metis finsertCI finsert-fminus1 fminus-finsert-absorb funionI2 iefac-empty
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  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )
      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  $\exists S11'. (\text{constant-context ord-res-11})^{++} S11 S11' \wedge \text{ord-res-10-matches-ord-res-11}$ 
    S10' S11'
    using match
  proof (cases S10 S11 rule: ord-res-10-matches-ord-res-11.cases)
    case match-hyps: (1 N  $U_{er10}$   $\mathcal{F}$   $\Gamma$   $U_{er11}$   $\mathcal{C}$ )

    note S10-def = ⟨S10 = (N,  $U_{er10}$ ,  $\mathcal{F}$ ,  $\Gamma$ )⟩
    note S11-def = ⟨S11 = (N,  $U_{er11}$ ,  $\mathcal{F}$ ,  $\Gamma$ ,  $\mathcal{C}$ )⟩
    note invars10 = ⟨ord-res-10-invars N ( $U_{er10}$ ,  $\mathcal{F}$ ,  $\Gamma$ )⟩
    note invars11 = ⟨ord-res-11-invars N ( $U_{er11}$ ,  $\mathcal{F}$ ,  $\Gamma$ ,  $\mathcal{C}$ )⟩

    have mempty-not-in-if-no-false-cls: {#} | $\notin$ | iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )
      if  $\neg fBex$  (iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )) (trail-false-cls  $\Gamma$ )
      using that by force

    have C-eq-None-if-no-false-cls:  $\mathcal{C} = \text{None}$ 
      if  $\neg fBex$  (iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )) (trail-false-cls  $\Gamma$ )
      using match-hyps mempty-not-in-if-no-false-cls[OF that] by argo

    have mempty-not-in-if: {#} | $\notin$ | N | $\cup$ |  $U_{er10}$ 
      if  $\neg fBex$  (iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )) (trail-false-cls  $\Gamma$ )
      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  $\neg fBex$  (iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ |  $U_{er10}$ )) (trail-false-cls  $\Gamma$ )
      using mempty-not-in-if[OF that] ⟨ $U_{er11} = U_{er10} \mid - \mid \{\{\#\}\}$ ⟩
    by (metis (no-types, opaque-lifting) finsertI1 finsert-ident fminusD2 funionCI
      funion-fempty-right funion-finsert-right funion-fminus-cancel2)
  end

```

```

obtain  $s10'$  where  $S10' = (N, s10')$  and  $step10: ord-res-10 N (U_{er10}, \mathcal{F}, \Gamma)$ 
 $s10'$ 
  using step unfolding S10-def by (auto elim: constant-context.cases)

show ?thesis
  using step10
proof (cases N (U_{er10}, \mathcal{F}, \Gamma) s10' rule: ord-res-10.cases)
  case step-hyps: (decide-neg A \Gamma)

  have  $C = None$ 
    using step-hyps C-eq-None-if-no-false-cls by argo

  have  $\{\#\} \not\subseteq N \mid \cup \mid U_{er10}$ 
    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  $step11: ord-res-11 N (U_{er11}, \mathcal{F}, \Gamma, None) (U_{er11}, \mathcal{F}, \Gamma', None)$ 
  proof (rule ord-res-11.decide-neg)
    show  $\neg (\exists C \in |iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er11}). trail-false-cls \Gamma C)$ 
      using step-hyps unfolding \langle U_{er11} = U_{er10} \rangle by argo
    next
      show linorder-trm.is-least-in-fset
         $\{A_2 \mid \in |atms-of-clss (N \mid \cup \mid U_{er11}). \forall A_1 \in |trail-atms \Gamma. A_1 \prec_t A_2\} A$ 
        using step-hyps unfolding \langle U_{er11} = U_{er10} \rangle by argo
      next
        show  $\neg (\exists C \in |iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er11}). clause-could-propagate \Gamma C (Pos$ 
         $A))$ 
          using step-hyps unfolding \langle U_{er11} = U_{er10} \rangle by argo
        next
          show  $\Gamma' = (Neg A, None) \# \Gamma$ 
            using step-hyps by argo
        qed

  thus (constant-context ord-res-11)^{++} S11 (N, U_{er11}, \mathcal{F}, \Gamma', None)
    unfolding S11-def \langle C = None \rangle by (auto intro: constant-context.intros)

  show ord-res-10-matches-ord-res-11 S10' (N, U_{er11}, \mathcal{F}, \Gamma', None)
    unfolding \langle S10' = (N, s10') \rangle \langle s10' = - \rangle
  proof (rule ord-res-10-matches-ord-res-11.intros)
    show ord-res-10-invars N (U_{er10}, \mathcal{F}, \Gamma')
      using step10 \langle s10' = - \rangle invars10 ord-res-10-preserves-invars by metis
    next
      show ord-res-11-invars N (U_{er11}, \mathcal{F}, \Gamma', None)
        using step11 invars11 \langle C = None \rangle ord-res-11-preserves-invars by metis
    next

```



```

    show  $U_{er11} = U_{er10} \mid - \mid \{\{\#\}\}$ 
      unfolding  $\langle U_{er11} = U_{er10} \rangle$ 
      using  $\langle \{\#\} \mid \notin \mid N \mid \cup \mid U_{er10} \rangle$  by simp
  next
    have  $\{\#\} \mid \notin \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er10})$ 
      using  $\langle \{\#\} \mid \notin \mid N \mid \cup \mid U_{er10} \rangle$  by (simp add: iefac-def)
    thus if  $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er10})$  then  $\Gamma' = [] \wedge \text{None} = \text{Some } \{\#\}$ 
else None = None
      by argo
    qed
  qed
next
case step-hyps: (decide-pos A C  $\Gamma'$   $\mathcal{F}'$ )

have C = None
  using step-hyps 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 ( $U_{er11}$ ,  $\mathcal{F}$ ,  $\Gamma$ , None) ( $U_{er11}$ ,  $\mathcal{F}'$ ,  $\Gamma'$ , None)
  proof (rule ord-res-11.decide-pos)
    show  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er11}). \text{trail-false-cls } \Gamma C)$ 
      using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  by argo
    next
      show linorder-trm.is-least-in-fset
         $\{|A_2 \mid \in \mid \text{atms-of-cls } (N \mid \cup \mid U_{er11}). \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2 \mid\} A$ 
        using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  by argo
    next
      show linorder-cls.is-least-in-fset
         $\{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er11}). \text{clause-could-propagate } \Gamma C (\text{Pos } A) \mid\} C$ 
        using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  by argo
    next
      show  $\Gamma' = (\text{Pos } A, \text{None}) \# \Gamma$ 
        using step-hyps by argo
    next
      show  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N \mid \cup \mid U_{er11}). \text{trail-false-cls } \Gamma' C)$ 
        using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  by argo
    next
      show  $\mathcal{F}' = (\text{if } \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C \text{ then } \mathcal{F} \text{ else } \text{finsert } C \mathcal{F})$ 
        using step-hyps by argo
    qed
  qed

```

```

thus (constant-context ord-res-11)++ S11 (N, Uer11, F', Γ', None)
  unfolding S11-def ⟨C = None⟩ by (auto intro: constant-context.intros)

show ord-res-10-matches-ord-res-11 S10' (N, Uer11, F', Γ', None)
  unfolding ⟨S10' = (N, s10')⟩ ⟨s10' = -⟩
proof (rule ord-res-10-matches-ord-res-11.intros)
  show ord-res-10-invars N (Uer10, F', Γ')
    using step10 ⟨s10' = -⟩ invars10 ord-res-10-preserves-invars by metis
next
  show ord-res-11-invars N (Uer11, F', Γ', None)
    using step11 invars11 ⟨C = None⟩ ord-res-11-preserves-invars by metis
next
  show Uer11 = Uer10 |-| {{#}}
    unfolding ⟨Uer11 = Uer10⟩
    using {#} |∉| N |∪| Uer10 by simp
next
  have {#} |∉| iefac F' |↑| (N |∪| Uer10)
    using {#} |∉| N |∪| Uer10 by (simp add: iefac-def)
    thus if {#} |∈| iefac F' |↑| (N |∪| Uer10) then Γ' = [] ∧ None = Some
    {#} else None = None
    by argo
  qed
qed
next
case step-hyps: (propagate A C Γ' F')

have C = None
  using step-hyps C-eq-None-if-no-false-cls by argo

have {#} |∉| N |∪| Uer10
  using step-hyps mempty-not-in-if by argo

have Uer11 = Uer10
  using step-hyps Uer11-eq-Uer10-if by argo

show ?thesis
proof (intro exI conjI)
  have step11: ord-res-11 N (Uer11, F, Γ, None) (Uer11, F', Γ', None)
  proof (rule ord-res-11.propagate)
    show ¬ (∃ C |∈| iefac F' |↑| (N |∪| Uer11). trail-false-cl Γ C)
    using step-hyps unfolding ⟨Uer11 = Uer10⟩ by argo
  next
  show linorder-trm.is-least-in-fset
    { |A2 |∈| atms-of-clss (N |∪| Uer11). ∀ A1 |∈| trail-atms Γ. A1 <t A2 |} A
    using step-hyps unfolding ⟨Uer11 = Uer10⟩ by argo
  next
  show linorder-cls.is-least-in-fset
    { |C |∈| iefac F' |↑| (N |∪| Uer11). clause-could-propagate Γ C (Pos A) |} C
    using step-hyps unfolding ⟨Uer11 = Uer10⟩ by argo

```

```

next
  show  $\Gamma' = (Pos\ A, Some\ (efac\ C)) \# \Gamma$ 
    using step-hyps by argo
next
  show  $\exists C \in |iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er11}).\ trail\ false\ cls\ \Gamma'\ C$ 
    using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  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  $(constant\ context\ ord\ res\ 11)^{++}\ S11\ (N,\ U_{er11},\ \mathcal{F}',\ \Gamma',\ None)$ 
  unfolding  $S11\ def\ \langle C = None \rangle$  by  $(auto\ intro:\ constant\ context.\ intros)$ 

show ord-res-10-matches-ord-res-11  $S10'\ (N,\ U_{er11},\ \mathcal{F}',\ \Gamma',\ None)$ 
  unfolding  $\langle S10' = (N,\ s10') \rangle\ \langle s10' = - \rangle$ 
proof  $(rule\ ord\ res\ 10\ matches\ ord\ res\ 11.\ intros)$ 
  show ord-res-10-invars  $N\ (U_{er10},\ \mathcal{F}',\ \Gamma')$ 
    using step10  $\langle s10' = - \rangle\ invars10\ ord\ res\ 10\ preserves\ invars$  by metis
next
  show ord-res-11-invars  $N\ (U_{er11},\ \mathcal{F}',\ \Gamma',\ None)$ 
    using step11 invars11  $\langle C = None \rangle\ ord\ res\ 11\ preserves\ invars$  by metis
next
  show  $U_{er11} = U_{er10} \mid - \mid \{\#\}$ 
    unfolding  $\langle U_{er11} = U_{er10} \rangle$ 
    using  $\langle \#\ \notin\ N\ |\cup\ U_{er10} \rangle$  by simp
next
  have  $\{\#\} \notin |iefac\ \mathcal{F}'\ |\uparrow\ (N\ |\cup\ U_{er10})$ 
    using  $\langle \#\ \notin\ N\ |\cup\ U_{er10} \rangle$  by  $(simp\ add:\ iefac\ def)$ 
  thus if  $\{\#\} \in |iefac\ \mathcal{F}'\ |\uparrow\ (N\ |\cup\ U_{er10})$  then  $\Gamma' = [] \wedge None = Some\ \{\#\}$ 
  else  $None = None$ 
    by argo
  qed
qed
next
  case step-hyps:  $(resolution\ D\ A\ C\ U_{er10}'\ \Gamma')$ 

  note  $D\ max\ lit = \langle ord\ res.\ is\ maximal\ lit\ (Neg\ A)\ D \rangle$ 

  have  $\{\#\} \notin |iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er10})$ 
    using  $\langle linorder\ cls.\ is\ least\ in\ fset\ -\ D \rangle\ \langle linorder\ lit.\ is\ maximal\ in\ mset\ D\ - \rangle$ 
    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  $C = None$ 
    using match-hyps  $\langle \#\ \notin\ |iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er10}) \rangle$  by argo

```

```

have  $U_{er11} = U_{er10}$ 
  using match-hyps  $\langle \{\#\} \mid \notin \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er10}) \rangle$  by force

have step11-conf: ord-res-11  $N (U_{er11}, \mathcal{F}, \Gamma, \text{None}) (U_{er11}, \mathcal{F}, \Gamma, \text{Some } D)$ 
proof (rule ord-res-11.conflict)
  show linorder-cls.is-least-in-fset
    (ffilter (trail-false-cls  $\Gamma$ ) (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er11})$ ))  $D$ 
  using step-hyps unfolding  $\langle U_{er11} = U_{er10} \rangle$  by argo
qed

have  $\Gamma$ -spec:  $\forall Ln \Gamma'. \Gamma = Ln \# \Gamma' \longrightarrow$ 
  (snd  $Ln \neq \text{None}$ ) = fBex (iefac  $\mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er10})$ ) (trail-false-cls  $\Gamma$ )  $\wedge$ 
  ( $\forall x \in \text{set } \Gamma'. \text{snd } x = \text{None}$ )
  using invars10 by (simp add: ord-res-10-invars.simps)

then obtain  $\Gamma'''$  where  $\Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma'''$ 
  using  $\langle \text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C) \rangle$ 
  by (metis list.set-cases map-of-SomeD not-Some-eq snd-conv)

have C-max-lit: linorder-lit.is-greatest-in-mset  $C (\text{Pos } A)$ 
  using invars10 by (simp add: ord-res-10-invars.simps  $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''' \rangle$ )

have ord-res.ground-resolution  $D C ((D - \{\#\text{Neg } A\}) + (C - \{\#\text{Pos } A\}))$ 
proof (rule ord-res.ground-resolutionI)
  show  $D = \text{add-mset } (\text{Neg } A) (D - \{\#\text{Neg } A\})$ 
    using D-max-lit unfolding linorder-lit.is-maximal-in-mset-iff by simp
next
  show  $C = \text{add-mset } (\text{Pos } A) (C - \{\#\text{Pos } A\})$ 
    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  $\{\#\} = \{\#\} \wedge \text{ord-res.is-maximal-lit } (\text{Neg } A) D \vee \text{Neg } A \in \{\#\}$ 
    using D-max-lit by argo
next
  show  $\{\#\} = \{\#\}$ 
    by argo
next
  show ord-res.is-strictly-maximal-lit  $(\text{Pos } A) C$ 
    using C-max-lit .
next
  show remove1-mset  $(\text{Neg } A) D + \text{remove1-mset } (\text{Pos } A) C = (D - \{\#\text{Neg } A\}) + (C - \{\#\text{Pos } A\})$ 
  ..

```

qed

hence $\text{eres } C D \neq D$

unfolding $\text{eres-ident-iff not-not ground-resolution-def}$ **by** metis

have $D\text{-false: trail-false-cls } \Gamma D$

using $\text{step-hyps unfolding linorder-cls.is-least-in-filter-iff}$ **by** argo

have $\text{clause-could-propagate } \Gamma''' C (\text{Pos } A)$

using $\text{invars10 } \langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''' \rangle$ **by** $(\text{simp add: ord-res-10-invars.simps})$

hence $\text{trail-false-cls } \Gamma''' \{ \#L \in \# C. L \neq \text{Pos } A \# \}$

unfolding $\text{clause-could-propagate-def}$ **by** argo

hence $C\text{-almost-false: trail-false-cls } \Gamma \{ \#L \in \# C. L \neq \text{Pos } A \# \}$

unfolding $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''' \rangle$

by $(\text{meson suffix-ConsD suffix-order.dual-order.refl trail-false-cls-if-trail-false-suffix})$

have $\text{eres-false: trail-false-cls } \Gamma (\text{eres } C D)$

unfolding $\text{trail-false-cls-def}$

proof (intro ballI)

fix $K :: 'f \text{ gliteral}$

assume $K \in \# \text{eres } C D$

hence $K \in \# C \wedge K \neq \text{Pos } A \vee K \in \# D$

using $\text{strong-lit-in-one-of-resolvents-if-in-eres}[OF D\text{-max-lit}]$ **by** simp

thus $\text{trail-false-lit } \Gamma K$

proof $(\text{elim disjE conjE})$

assume $K \in \# C$ **and** $K \neq \text{Pos } A$

thus $\text{trail-false-lit } \Gamma K$

using $C\text{-almost-false unfolding trail-false-cls-def}$ **by** simp

next

assume $K \in \# D$

thus $\text{trail-false-lit } \Gamma K$

using $D\text{-false unfolding trail-false-cls-def}$ **by** simp

qed

qed

have $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{ord-res.is-strictly-maximal-lit}$
 $(\text{fst } Ln) C$

using invars10 **by** $(\text{simp add: ord-res-10-invars.simps})$

hence $C\text{-max-lit: ord-res.is-strictly-maximal-lit } (\text{Pos } A) C$

unfolding $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''' \rangle$ **by** simp

have $\text{steps11-reso: } (\text{ord-res-11 } N)^{**} (U_{er11}, \mathcal{F}, \Gamma, \text{Some } D) (U_{er11}, \mathcal{F}, \Gamma, \text{Some}$
 $(\text{eres } C D))$

unfolding $\langle \Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''' \rangle$

using $C\text{-max-lit rtrancl-ord-res-11-all-resolution-steps}$ **by** metis

```

define strict-P :: 'f gterm literal × 'f gterm literal multiset option ⇒ bool where
  strict-P ≡ λLn. ∀ K. ord-res.is-maximal-lit K (eres C D) → atm-of K <t
atm-of (fst Ln)

have ∀ K ∈# eres C D. - K ∈ fst ' set Γ
using eres-false unfolding trail-false-cls-def trail-false-lit-def .

moreover have Γ-sorted: sorted-wrt (λx y. atm-of (fst y) <t atm-of (fst x)) Γ
using invars10 by (simp add: ord-res-10-invars.simps)

ultimately have dropWhile strict-P Γ = dropWhile (λLn. - fst Ln ∈# 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.prem True <- fst Ln ∈# eres C D
  using image-iff by fastforce

```

```

next
  show sorted-wrt ( $\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)$ )  $\Gamma$ 
    using Cons.premis by simp
qed

ultimately show ?thesis
  unfolding dropWhile.simps
  unfolding strict-P-Ln-iff
  by simp
next
case False

hence atm-of (fst Ln)  $\preceq_t$  atm-of L
  by order

hence atm-of (fst Ln) = atm-of L
  using Cons.premis
  using atm-of-eq-atm-of eres-max-lit linorder-lit.is-maximal-in-mset-iff
by fastforce

hence  $L = fst Ln \vee L = - fst Ln$ 
  by (metis atm-of-eq-atm-of)

moreover have  $- fst Ln \notin \text{fst } 'set \Gamma$ 
  using  $\langle \text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) (Ln \# \Gamma) \rangle$ 
  by fastforce

moreover have  $- L \in \text{fst } '(set (Ln \# \Gamma))$ 
  using Cons.premis(1) eres-max-lit linorder-lit.is-maximal-in-mset-iff by
blast

ultimately have  $- fst Ln \in \# \text{eres } C D$ 
  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)**
  ( $U_{er11}, \mathcal{F}, \Gamma, \text{Some } (\text{eres } C D)$ )
  ( $U_{er11}, \mathcal{F}, \text{dropWhile strict-P } \Gamma, \text{Some } (\text{eres } C D)$ )
  using rtrancl-ord-res-11-all-skip-steps by metis

have most-steps11: (ord-res-11 N)++ ( $U_{er11}, \mathcal{F}, \Gamma, \text{None}$ )

```

```

( $U_{er11}$ ,  $\mathcal{F}$ ,  $dropWhile\ strict-P\ \Gamma$ ,  $Some\ (eres\ C\ D)$ )
using step11-conf steps11-reso steps11-skip by simp

show ?thesis
proof (cases eres C D = {#})
  case True

  hence  $dropWhile\ strict-P\ \Gamma = []$ 
    unfolding strict-P-def  $\langle eres\ C\ D = \{ \# \} \rangle$ 
    unfolding linorder-lit.is-maximal-in-mset-iff
    by simp

  have  $\Gamma' = []$ 
    unfolding  $\langle \Gamma' = dropWhile\ -\ \Gamma \rangle \langle eres\ C\ D = \{ \# \} \rangle$ 
    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  $\langle C = None \rangle$ 
      using most-steps11[unfolded  $\langle dropWhile\ strict-P\ \Gamma = [] \rangle \langle eres\ C\ D =$ 
 $\{ \# \} \rangle]$ 
      using tranclp-constant-context by metis

    show ord-res-10-matches-ord-res-11 S10' ( $N$ ,  $U_{er11}$ ,  $\mathcal{F}$ ,  $[], Some\ \{ \# \}$ )
      unfolding  $\langle S10' = (N, s10') \rangle \langle s10' = - \rangle \langle \Gamma' = [] \rangle$ 
    proof (rule ord-res-10-matches-ord-res-11.intros)
      show ord-res-10-invars  $N$  ( $U_{er10}'$ ,  $\mathcal{F}$ ,  $[], []$ )
        using step10  $\langle s10' = - \rangle \langle \Gamma' = [] \rangle$  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  $\langle C = None \rangle \langle dropWhile\ strict-P\ \Gamma = [] \rangle \langle eres\ C\ D = \{ \# \} \rangle$ 
        by (metis tranclp-ord-res-11-preserves-invars)

    next
      show  $U_{er11} = U_{er10}' \mid - \mid \{ \{ \# \} \}$ 
      unfolding  $\langle U_{er11} = U_{er10} \rangle \langle U_{er10}' = finsert\ (eres\ C\ D)\ U_{er10} \rangle \langle eres\ C\ D = \{ \# \} \rangle$ 
      using  $\langle \{ \# \} \mid \notin\ iefac\ \mathcal{F} \mid \uparrow\ (N \mid \cup\ U_{er10}) \rangle$ 
      by force

    next
      show if  $\{ \# \} \mid \in \mid iefac\ \mathcal{F} \mid \uparrow\ (N \mid \cup\ U_{er10})$  then
         $[] = [] \wedge Some\ \{ \# \} = Some\ \{ \# \}$  else  $Some\ \{ \# \} = None$ 
      unfolding  $\langle U_{er10}' = finsert\ (eres\ C\ D)\ U_{er10} \rangle \langle eres\ C\ D = \{ \# \} \rangle$ 
      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 $L \in\#$ $C \wedge L \neq \text{Pos } A \vee L \in\#$ $D \wedge L \neq \text{Neg } A$
using *stronger-lit-in-one-of-resolvents-if-in-eres* \langle *eres* C $D \neq D$ \rangle *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* \langle $L \in\#$ *eres* C D \rangle *eres-lt-if ord-res.asymp-less-lit*
ord-res.less-lit-simps(3,4) *ord-res.transp-less-lit in-remove1-mset-neg*
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* $\Gamma = \text{dropWhile } (\lambda Ln. \text{atm-of } L \prec_t \text{atm-of } (\text{fst } Ln))$
 Γ

unfolding *strict-P-def*
using *eres-max-lit*
by (*metis (no-types) Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

also have $\dots = (- L, \text{None}) \# \text{dropWhile } (\lambda Ln. \text{atm-of } L \preceq_t \text{atm-of } (\text{fst } Ln))$ Γ

proof –
have $- L \neq \text{Pos } A$
using \langle $L \prec_l$ *Neg* A \rangle **by** (*cases* L) *simp-all*

hence $- L \in \text{fst 'set } \Gamma'''$
using *eres-false* \langle $L \in\#$ *eres* C D \rangle
unfolding *trail-false-cls-def trail-false-lit-def*
unfolding \langle $\Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma'''$ \rangle
by *auto*

hence $(- L, \text{None}) \in \text{set } \Gamma'''$
using Γ -*spec* **unfolding** \langle $\Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma'''$ \rangle **by** *auto*

then obtain Γ_0 Γ_1 **where** $\Gamma'''' = \Gamma_1 @ (- L, \text{None}) \# \Gamma_0$
by (*meson split-list*)

hence $\Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma_1 @ (- L, \text{None}) \# \Gamma_0$
unfolding \langle $\Gamma = (\text{Pos } A, \text{Some } C) \# \Gamma''''$ \rangle **by** (*simp only:*)

have *AAA*: $\forall Ln \in \text{set } ((\text{Pos } A, \text{Some } C) \# \Gamma_1). \text{atm-of } L \prec_t \text{atm-of } (\text{fst } Ln)$

Ln)
using Γ -sorted **unfolding** $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_1 @ (-\ L, None)$
 $\# \Gamma_0 \rangle$
by (*simp add: sorted-wrt-append*)

hence $BBB: \forall Ln \in set\ ((Pos\ A, Some\ C) \# \Gamma_1 @ [(-\ L, None)])$. *atm-of*
 $L \preceq_t atm-of\ (fst\ Ln)$
by *simp*

have $\forall Ln \in set\ \Gamma_0$. *atm-of* $(fst\ Ln) \prec_t atm-of\ L$
using Γ -sorted **unfolding** $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_1 @ (-\ L, None)$
 $\# \Gamma_0 \rangle$
by (*simp add: sorted-wrt-append*)

hence $CCC: \forall Ln \in set\ \Gamma_0$. $\neg atm-of\ L \preceq_t atm-of\ (fst\ Ln)$
using *linorder-trm.leD* **by** *blast*

have *dropWhile* $(\lambda Ln. atm-of\ L \prec_t atm-of\ (fst\ Ln))\ \Gamma = (-\ L, None) \# \Gamma_0$
unfolding $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_1 @ (-\ L, None) \# \Gamma_0 \rangle$
using *dropWhile-append2 AAA* **by** *simp*

also have $\dots = (-\ L, None) \# dropWhile\ (\lambda Ln. atm-of\ L \preceq_t atm-of\ (fst$
 $Ln))\ \Gamma_0$
using CCC **by** (*simp add: dropWhile-ident-if-pred-always-false*)

also have $\dots = (-\ L, None) \# dropWhile\ (\lambda Ln. atm-of\ L \preceq_t atm-of\ (fst$
 $Ln))\ \Gamma$
unfolding $\langle \Gamma = (Pos\ A, Some\ C) \# \Gamma_1 @ (-\ L, None) \# \Gamma_0 \rangle$
using *dropWhile-append2 BBB* **by** *simp*

finally show *?thesis* .
qed

also have $\dots = (-\ L, None) \# \Gamma'$
unfolding $\langle \Gamma' = dropWhile\ -\ \Gamma \rangle$
using *eres-max-lit*
by (*metis (no-types) Uniq-D linorder-lit.Uniq-is-maximal-in-mset*)

finally have *dropWhile strict-P* $\Gamma = (-\ L, None) \# \Gamma'$.

have *step10-back: ord-res-11 N*
 $(U_{er11}, \mathcal{F}, dropWhile\ strict-P\ \Gamma, Some\ (eres\ C\ D))$
 $(finsert\ (eres\ C\ D)\ U_{er11}, \mathcal{F}, \Gamma', None)$
unfolding $\langle dropWhile\ strict-P\ \Gamma = (-\ L, None) \# \Gamma' \rangle$

proof (*rule ord-res-11.backtrack*)
show $(-\ L, None) \# \Gamma' = (-\ L, None) \# \Gamma' ..$

next
show $(-\ L) \in \#\ eres\ C\ D$
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)++ (Uer11,  $\mathcal{F}$ ,  $\Gamma$ , None)
  (finsert (eres C D) Uer11,  $\mathcal{F}$ ,  $\Gamma'$ , None)
  using most-steps11 by simp

show ?thesis
proof (intro exI conjI)
  show (constant-context ord-res-11)++ S11 (N, finsert (eres C D) Uer11,
 $\mathcal{F}$ ,  $\Gamma'$ , None)
    unfolding S11-def  $\langle C = \text{None} \rangle$ 
    using all-steps11 tranclp-constant-context by metis

  have {#} | $\notin$ | iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ | Uer10) $\wedge$ 
    by (smt (verit, del-insts) False  $\langle \{ \# \} | \notin | \text{iefac } \mathcal{F} | \uparrow | (N | \cup | U_{er10}) \rangle$ )
  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) Uer11,  $\mathcal{F}$ ,
 $\Gamma'$ , None)
    unfolding  $\langle S10' = (N, s10') \rangle$   $\langle s10' = - \rangle$ 
    proof (rule ord-res-10-matches-ord-res-11.intros)
      show ord-res-10-invars N (Uer10',  $\mathcal{F}$ ,  $\Gamma'$ )
        using step10  $\langle s10' = - \rangle$  invars10 ord-res-10-preserves-invars by metis
      next
      show ord-res-11-invars N (finsert (eres C D) Uer11,  $\mathcal{F}$ ,  $\Gamma'$ , None)
        using all-steps11 invars11
        unfolding  $\langle C = \text{None} \rangle$ 
        by (metis tranclp-ord-res-11-preserves-invars)
      next
      show finsert (eres C D) Uer11 = Uer10' | $-$ | { |{#}| }
        unfolding  $\langle U_{er11} = U_{er10} \rangle$   $\langle U_{er10}' = \text{finsert } (eres C D) U_{er10} \rangle$ 
        using  $\langle \{ \# \} | \notin | \text{iefac } \mathcal{F} | \uparrow | (N | \cup | U_{er10}') \rangle$ 
        using False  $\langle U_{er11} = U_{er10} \rangle$   $\langle U_{er11} = U_{er10} | - | \{ | \{ \# \} | \} \rangle$  by auto
      next
      show if {#} | $\in$ | iefac  $\mathcal{F}$  | $\uparrow$ | (N | $\cup$ | Uer10') then
         $\Gamma' = \square \wedge \text{None} = \text{Some } \{ \# \}$  else None = None
        using  $\langle \{ \# \} | \notin | \text{iefac } \mathcal{F} | \uparrow | (N | \cup | U_{er10}') \rangle$  by simp
    qed
  qed
qed
qed
qed
qed

theorem bisimulation-ord-res-10-ord-res-11:
  defines match  $\equiv \lambda \cdot$ . ord-res-10-matches-ord-res-11

```

```

shows  $\exists (MATCH :: nat \times nat \Rightarrow 'f \text{ ord-res-10-state} \Rightarrow 'f \text{ ord-res-11-state} \Rightarrow$ 
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  $\forall S. \text{ord-res-8-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-10 } S S')$ 
    by (metis finished-def ord-res-10-antics.final-finished)
next
  show  $\forall S. \text{ord-res-11-final } S \longrightarrow (\exists S'. \text{constant-context ord-res-11 } S S')$ 
    by (metis finished-def ord-res-11-antics.final-finished)
next
  show  $\forall i S10 S11. \text{match } i S10 S11 \longrightarrow \text{ord-res-8-final } S10 \longleftrightarrow \text{ord-res-11-final}$ 
S11
    unfolding match-def
    using ord-res-10-final-iff-ord-res-11-final by metis
next
  show  $\forall i S10 S11. \text{match } i S10 S11 \longrightarrow$ 
    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 ( $\lambda - . \text{False}$ )
    by simp
next
  show  $\forall i S10 S11 S10'. \text{match } i S10 S11 \longrightarrow \text{constant-context ord-res-10 } S10$ 
S10'  $\longrightarrow$ 
    ( $\exists i' S11'. (\text{constant-context ord-res-11})^{++} S11 S11' \wedge \text{match } i' S10' S11'$ )  $\vee$ 
    ( $\exists i'. \text{match } i' S10' S11 \wedge \text{False}$ )
    unfolding match-def

```

```

    using forward-simulation-between-10-and-11 by metis
qed

end

type-synonym bisim-index-1-2 = nat × nat
type-synonym bisim-index-1-3 = bisim-index-1-2 × (nat × nat)
type-synonym bisim-index-1-4 = bisim-index-1-3 × (nat × nat)
type-synonym bisim-index-1-5 = bisim-index-1-4 × (nat × nat)
type-synonym bisim-index-1-6 = bisim-index-1-5 × (nat × nat)
type-synonym bisim-index-1-7 = bisim-index-1-6 × (nat × 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
   $\mathcal{R}_f \mathcal{R}_b :: \textit{bisim-index-1-11} \Rightarrow \textit{bisim-index-1-11} \Rightarrow \textit{bool}$ 
  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
   $\mathcal{R}_f \mathcal{R}_b :: \textit{bisim-index-1-11} \Rightarrow \textit{bisim-index-1-11} \Rightarrow \textit{bool}$ 
  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**
 $\bigwedge j S1 S11. \text{MATCH } j S1 S11 \implies \text{ord-res-1-final } S1 \longleftrightarrow \text{ord-res-11-final } S11$
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 $\equiv \lambda(L, \text{opt}).$
(*lit-of-glit* L , *map-option* ($\lambda C. (\text{cls-of-gcls } \{\#K \in\# C. K \neq L\# \}, \text{lit-of-glit } L,$
 $\text{Var})) \text{opt}$)

fun *state-of-gstate* $:: - \Rightarrow ('f, 'v) \text{SCL-FOL.state}$ **where**

state-of-gstate ($U_G, -, \Gamma_G, \mathcal{C}_G$) =
(*let*
 $\Gamma = \text{map } \text{gtrailelem-of-trailelem } \Gamma_G;$
 $U = \text{cls-of-gcls } |\uparrow U_G;$
 $\mathcal{C} = \text{map-option } (\lambda C_G. (\text{cls-of-gcls } C_G, \text{Var})) \mathcal{C}_G$
in (Γ, U, \mathcal{C})

lemma *fst-case-prod-simp*: $\text{fst } (\text{case } p \text{ of } (x, y) \Rightarrow (f x, g x y)) = f (\text{fst } p)$
by (*cases p*) *simp*

lemma *trail-false-cls-nonground-iff-trail-false-cls-ground*:

fixes Γ_G **and** $D_G :: 'f \text{gclause}$
fixes $\Gamma :: ('f, 'v) \text{SCL-FOL.trail}$ **and** $D :: ('f, 'v) \text{term clause}$
defines $\Gamma \equiv \text{map } \text{gtrailelem-of-trailelem } \Gamma_G$ **and** $D \equiv \text{cls-of-gcls } D_G$
shows *trail-false-cls* $\Gamma D \longleftrightarrow \text{trail-false-cls } \Gamma_G D_G$

proof –

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

unfolding *trail-false-cls-def* ..

also have $\dots \longleftrightarrow (\forall L_G \in\# D_G. \text{trail-false-lit } \Gamma (\text{lit-of-glit } L_G))$

unfolding *D-def cls-of-gcls-def* **by** *simp*

also have $\dots \longleftrightarrow (\forall L_G \in\# D_G. \text{trail-false-lit } \Gamma_G L_G)$

proof –

have *trail-false-lit* $\Gamma (\text{lit-of-glit } L_G) \longleftrightarrow \text{trail-false-lit } \Gamma_G L_G$

for $L_G :: 'f \text{gterm literal}$

proof –

have *trail-false-lit* $\Gamma (\text{lit-of-glit } L_G) \longleftrightarrow - \text{lit-of-glit } L_G \in \text{fst ' set } \Gamma$

unfolding *trail-false-lit-def* ..

also have $\dots \longleftrightarrow$

$- (\text{lit-of-glit } L_G :: ('f, 'v) \text{term literal}) \in \text{set } (\text{map } (\lambda x. \text{lit-of-glit } (\text{fst } x))$

$\Gamma_G)$

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 $\dots \longleftrightarrow (\text{lit-of-glit } (- L_G) :: ('f, 'v) \text{ term literal}) \in \text{lit-of-glit } \text{'fst}$
' set Γ_G
by (*cases* L_G) (*auto simp: lit-of-glit-def*)
also have $\dots \longleftrightarrow - L_G \in \text{fst } \text{'set } \Gamma_G$
using *inj-image-mem-iff inj-lit-of-glit* **by** *metis*
also have $\dots \longleftrightarrow \text{trail-false-lit } \Gamma_G L_G$
unfolding *trail-false-lit-def* ..
finally show $\text{trail-false-lit } \Gamma (\text{lit-of-glit } L_G) = \text{trail-false-lit } \Gamma_G L_G$.
qed
thus *?thesis* **by** *metis*
qed
also have $\dots \longleftrightarrow \text{trail-false-cls } \Gamma_G D_G$
unfolding *trail-false-cls-def* ..
finally show *?thesis* .
qed

theorem *ord-res-11-is-strategy-for-regular-scl:*

fixes
 $N_G :: 'f \text{ gclause fset}$ **and**
 $N :: ('f, 'v) \text{ term clause fset}$ **and**
 $\beta_G :: 'f \text{ gterm}$ **and**
 $\beta :: ('f, 'v) \text{ term}$ **and**
 $S_G S_G' :: 'f \text{ gclause fset} \times 'f \text{ gclause fset} \times ('f \text{ gliteral} \times 'f \text{ gclause option}) \text{ list}$
 $\times 'f \text{ gclause option}$ **and**
 $S S' :: ('f, 'v) \text{ SCL-FOL.state}$
defines
 $N \equiv \text{cls-of-gcls } |\cdot| N_G$ **and**
 $\beta \equiv \text{term-of-gterm } \beta_G$ **and**
 $S \equiv \text{state-of-gstate } S_G$ **and**
 $S' \equiv \text{state-of-gstate } S_G'$
assumes
 $\text{ball-le-}\beta_G: \forall A_G \in | \text{atms-of-cls } N_G. A_G \preceq_t \beta_G$ **and**
 $\text{run: } (\text{ord-res-11 } N_G)^{**} (\{\{\}\}, \{\{\}\}, [], \text{None}) S_G$ **and**
 $\text{step: } \text{ord-res-11 } N_G S_G S_G'$
shows
 $\text{scl-fol.regular-scl } N \beta S S'$
proof –
have $\text{ord-res-11-invars } N_G (\{\{\}\}, \{\{\}\}, [], \text{None})$
using *ord-res-11-invars-initial-state* .

hence $\text{ord-res-11-invars } N_G S_G$
using *run rtranclp-ord-res-11-preserves-invars* **by** *metis*

obtain $U_G \mathcal{F} \Gamma_G \mathcal{C}_G$ **where** $S_G\text{-def: } S_G = (U_G, \mathcal{F}, \Gamma_G, \mathcal{C}_G)$
by (*metis surj-pair*)

obtain $\Gamma U \mathcal{C}$ **where** $S\text{-def: } S = (\Gamma, U, \mathcal{C})$
by (*metis surj-pair*)

have Γ -def: $\Gamma = \text{map } \text{gtrailelem-of-trailelem } \Gamma_G$
using S -def S_G -def $\langle S \equiv \text{state-of-gstate } S_G \rangle$ **by** *simp*

have U -def: $U = \text{cls-of-gcls } | \uparrow U_G$
using S -def S_G -def $\langle S \equiv \text{state-of-gstate } S_G \rangle$ **by** *simp*

have C -def: $C = \text{map-option } (\lambda C_G. (\text{cls-of-gcls } C_G, \text{Var})) C_G$
using S -def S_G -def $\langle S \equiv \text{state-of-gstate } S_G \rangle$ **by** *simp*

obtain $\mathcal{F}' U_G' :: 'f \text{ gclause fset and } \Gamma_G' :: - \text{ list and } C_G' :: - \text{ option where}$
 S_G' -def: $S_G' = (U_G', \mathcal{F}', \Gamma_G', C_G')$
by (*metis surj-pair*)

obtain $\Gamma' :: - \text{ list and } U' :: - \text{ fset and } C' :: - \text{ option where}$
 S' -def: $S' = (\Gamma', U', C')$
by (*metis surj-pair*)

have Γ' -def: $\Gamma' = \text{map } \text{gtrailelem-of-trailelem } \Gamma_G'$
using S' -def S_G' -def $\langle S' \equiv \text{state-of-gstate } S_G' \rangle$ **by** *simp*

have U' -def: $U' = \text{cls-of-gcls } | \uparrow U_G'$
using S' -def S_G' -def $\langle S' \equiv \text{state-of-gstate } S_G' \rangle$ **by** *simp*

have C' -def: $C' = \text{map-option } (\lambda C_G. (\text{cls-of-gcls } C_G, \text{Var})) C_G'$
using S' -def S_G' -def $\langle S' \equiv \text{state-of-gstate } S_G' \rangle$ **by** *simp*

have *atms-of-clss* $U_G \subseteq |$ *atms-of-clss* N_G
using $\langle \text{ord-res-11-invars } N_G S_G \rangle$ [*unfolded* S_G -def]
unfolding *ord-res-11-invars.simps* **by** *simp*

have *atms-of-clss* $(N_G \cup | U_G) = \text{atms-of-clss } N_G \cup |$ *atms-of-clss* U_G
by (*simp add: atms-of-clss-def fimage-union*)

also have $\dots = \text{atms-of-clss } N_G$
using $\langle \text{atms-of-clss } U_G \subseteq |$ *atms-of-clss* $N_G \rangle$ **by** *auto*

finally have *atms-of-clss* $(N_G \cup | U_G) = \text{atms-of-clss } N_G$.

have *clauses-in-F-have-pos-max-lit*: $\forall C \in | \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit}$
 $L C$
using $\langle \text{ord-res-11-invars } N_G S_G \rangle$ [*unfolded* S_G -def *ord-res-11-invars.simps*]
by *simp*

have *nex-conflict-if-nbex-trail-false*:
 $\neg \text{fBex } (\text{iefac } \mathcal{F} | \uparrow (N_G \cup | U_G)) (\text{trail-false-cls } \Gamma_G) \implies \neg \text{Ex } (\text{scl-fol.conflict}$
 $N \beta S)$
proof (*elim contrapos-nn exE*)
fix $x :: ('f, 'v) \text{ state}$
assume *scl-fol.conflict* $N \beta S x$

hence $fBex (N_G \mid \cup \mid U_G) (trail\text{-}false\text{-}cls \Gamma_G)$
unfolding $S\text{-}def$
proof ($cases N \beta (\Gamma, U, \mathcal{C})$ x rule: $scl\text{-}fol.conflict.cases$)
case ($conflictI D \gamma$)

obtain D_G **where** $D_G \mid \in \mid N_G \mid \cup \mid U_G$ **and** $D\text{-}def: D = cls\text{-}of\text{-}gcls D_G$
using $\langle D \mid \in \mid N \mid \cup \mid U \rangle$
unfolding $N\text{-}def U\text{-}def$ **by** $blast$

moreover have $trail\text{-}false\text{-}cls \Gamma_G D_G$
proof –
have $is\text{-}ground\text{-}cls D$
using $\langle D = cls\text{-}of\text{-}gcls D_G \rangle$ **by** $simp$
hence $D \cdot \gamma = D$
by $simp$
hence $trail\text{-}false\text{-}cls \Gamma D$
using $conflictI$
unfolding $SCL\text{-}FOL.trail\text{-}false\text{-}cls\text{-}def$ $trail\text{-}false\text{-}cls\text{-}def$
unfolding $SCL\text{-}FOL.trail\text{-}false\text{-}lit\text{-}def$ $trail\text{-}false\text{-}lit\text{-}def$
by $argo$

thus $?thesis$
unfolding $\Gamma\text{-}def D\text{-}def$
unfolding $trail\text{-}false\text{-}cls\text{-}nonground\text{-}iff\text{-}trail\text{-}false\text{-}cls\text{-}ground$.
qed
ultimately show $?thesis$ **by** $metis$
qed

thus $fBex (iefac \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G)) (trail\text{-}false\text{-}cls \Gamma_G)$
unfolding $bex\text{-}trail\text{-}false\text{-}cls\text{-}simp$.
qed

have $nex\text{-}conflict\text{-}if\text{-}alread\text{-}in\text{-}conflict: \mathcal{C}_G = Some C_G \implies \neg Ex (scl\text{-}fol.conflict N \beta S)$ **for** C_G
unfolding $S\text{-}def \mathcal{C}\text{-}def$ **by** ($simp$ add: $scl\text{-}fol.conflict.simps$)

have $nex\text{-}conflict\text{-}if\text{-}no\text{-}clause\text{-}could\text{-}propagate\text{-}comp:$
 $\neg Ex (scl\text{-}fol.conflict N \beta ((lit\text{-}of\text{-}glit L_G, None) \# \Gamma, U, \mathcal{C}))$
if
 $nex\text{-}false\text{-}clause\text{-}wrt\text{-}\Gamma_G: \neg fBex (iefac \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G)) (trail\text{-}false\text{-}cls \Gamma_G)$

and
 $ball\text{-}lit\text{-}atm\text{-}L_G: \forall x \mid \in \mid trail\text{-}atms \Gamma_G. x \prec_t atm\text{-}of L_G$ **and**
 $nex\text{-}clause\text{-}that\text{-}propagate: \neg (\exists C \mid \in \mid iefac \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G)).$
 $clause\text{-}could\text{-}propagate \Gamma_G C (- L_G)$

for L_G
proof ($intro notI, elim exE$)
fix $S'' :: ('f, 'v) SCL\text{-}FOL.state$
assume $scl\text{-}fol.conflict N \beta ((lit\text{-}of\text{-}glit L_G, None) \# \Gamma, U, \mathcal{C}) S''$
thus $False$

proof (*cases* $N \beta ((\text{lit-of-glit } L_G, \text{None}) \# \Gamma, U, C) S''$ rule: *scl-fol.conflict.cases*)
case (*conflictI* $D \gamma$)

obtain D_G **where** $D_G \mid \in \mid N_G \mid \cup \mid U_G$ **and** $D\text{-def}$: $D = \text{cls-of-gcls } D_G$
using $\langle D \mid \in \mid N \mid \cup \mid U \rangle N\text{-def } U\text{-def}$ **by** *blast*

have (*lit-of-glit* $L_G :: ('f, 'v)$ term literal, *None*) $\# \Gamma =$
(map gtrailelem-of-trailelem ((L_G, None) $\# \Gamma_G) :: ('f, 'v)$ SCL-FOL.trail)
by (*simp add*: $\Gamma\text{-def gtrailelem-of-trailelem-def}$)

moreover have $D \cdot \gamma = \text{cls-of-gcls } D_G$
unfolding $D\text{-def}$ **by** *simp*

ultimately have *trail-false-cls*
(map gtrailelem-of-trailelem ((L_G, None) $\# \Gamma_G) :: ('f, 'v)$ SCL-FOL.trail)
(cls-of-gcls D_G)
using $\langle \text{SCL-FOL.trail-false-cls } ((\text{lit-of-glit } L_G, \text{None}) \# \Gamma) (D \cdot \gamma) \rangle$
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* $((L_G, \text{None}) \# \Gamma_G) D_G$
using *trail-false-cls-nonground-iff-trail-false-cls-ground* **by** *blast*

hence *trail-false-cls* $\Gamma_G \{ \#K_G \in \# D_G. K_G \neq - L_G \# \}$
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 $- L_G \in \# D_G$
using $\langle D_G \mid \in \mid N_G \mid \cup \mid U_G \rangle \langle \text{trail-false-cls } ((L_G, \text{None}) \# \Gamma_G) D_G \rangle$
subtrail-falseI
nex-false-clause-wrt- Γ_G
unfolding *bex-trail-false-cls-simp*
by *blast*

next
fix K_G **assume** $K_G \in \# D_G$ **and** $K_G \neq - L_G$
hence *trail-false-lit* $\Gamma_G K_G$
using $\langle \text{trail-false-cls } \Gamma_G \{ \#K_G \in \# D_G. K_G \neq - L_G \# \} \rangle$
unfolding *trail-false-cls-def* **by** *simp*
hence *trail-defined-lit* $\Gamma_G K_G$
by (*simp add*: *trail-defined-lit-iff-true-or-false*)
hence *atm-of* $K_G \mid \in \mid \text{trail-atms } \Gamma_G$
unfolding *trail-defined-lit-iff-trail-defined-atm* .
hence *atm-of* $K_G \prec_t \text{atm-of } L_G$
using *ball-lt-atm- L_G* **by** *metis*
hence $K_G \prec_l - L_G$

```

    by (cases LG; cases KG) simp-all
  thus ¬ - LG <l KG
    by order
qed

moreover have ¬ trail-defined-lit ΓG (- LG)
  by (metis atm-of-uminus linorder-trm.less-irrefl that(2)
      trail-defined-lit-iff-trail-defined-atm)

ultimately have clause-could-propagate ΓG DG (- LG)
  unfolding clause-could-propagate-def by argo

hence ∃ C |∈| NG |∪| UG. clause-could-propagate ΓG C (- LG)
  using ⟨DG |∈| NG |∪| UG⟩ 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 SG-def SG'-def
proof (cases NG (UG, ℱ, ΓG, CG) (UG', ℱ', ΓG', CG') rule: ord-res-11.cases)
  case step-hyps: (decide-neg AG)

  define A :: ('f, 'v) term where
    A = term-of-gterm AG

  let ?f = gtrailelem-of-trailelem
  have Γ' = map ?f ΓG'
    unfolding Γ'-def ..
  also have ... = map ?f ((Neg AG, None) # ΓG)
    unfolding ⟨ΓG' = (Neg AG, None) # ΓG⟩ ..
  also have ... = ?f (Neg AG, None) # map ?f ΓG
    unfolding list.map ..
  also have ... = ?f (Neg AG, None) # Γ
    unfolding Γ-def ..
  also have ... = (lit-of-glit (Neg AG), None) # Γ
    unfolding gtrailelem-of-trailelem-def prod.case option.map ..
  also have ... = (Neg (term-of-gterm AG), 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 ⟨UG' = UG⟩ U-def ..

have ¬ Ex (scl-fol.conflict N β S)
  using ⟨¬ fBex (iefac  $\mathcal{F}$  |q (NG | $\cup$ | UG)) (trail-false-cls  $\Gamma_G$ )⟩ 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 ⟨CG = None⟩ ⟨CG' = None⟩
    option.map
      proof (rule scl-fol.decideI')
        show is-ground-lit (Neg A ·l Var)
          by (simp add: A-def)
      next
        have  $\forall x \in | \text{trail-atms } \Gamma_G. x \prec_t A_G$ 
          using step-hyps linorder-trm.is-least-in-ffilter-iff by simp
        hence AG | $\notin$ | trail-atms  $\Gamma_G$ 
          by blast
        hence AG  $\notin$  atm-of 'fst' set  $\Gamma_G$ 
          unfolding fset-trail-atms .
        hence term-of-gterm AG  $\notin$  term-of-gterm 'atm-of' 'fst' set  $\Gamma_G$ 
          using inj-image-mem-iff inj-term-of-gterm by metis
        hence term-of-gterm AG  $\notin$  set (map (λx. term-of-gterm (atm-of (fst x)))
 $\Gamma_G$ )
          unfolding image-set list.map-comp comp-def .
        hence A  $\notin$  set (map (λx. atm-of (lit-of-glit (fst x)))  $\Gamma_G$ )
          unfolding A-def atm-of-lit-of-glit-conv .
        hence A  $\notin$  atm-of 'fst' set  $\Gamma$ 
        unfolding image-set list.map-comp comp-def  $\Gamma$ -def gtrailelem-of-trailelem-def
          fst-case-prod-simp .
        hence A | $\notin$ | trail-atms  $\Gamma$ 
          unfolding fset-trail-atms .
        hence ¬ trail-defined-lit  $\Gamma$  (Neg A ·l Var)
          by (simp add: trail-defined-lit-iff-trail-defined-atm)
        thus ¬ SCL-FOL.trail-defined-lit  $\Gamma$  (Neg A ·l Var)
          by (simp add: SCL-FOL.trail-defined-lit-def trail-defined-lit-def)
      next
        have AG | $\in$ | atms-of-cls (NG | $\cup$ | UG)
          using step-hyps linorder-trm.is-least-in-ffilter-iff by blast
        hence AG | $\in$ | atms-of-cls NG
          unfolding ⟨atms-of-cls (NG | $\cup$ | UG) = atms-of-cls NG⟩ .
        hence AG  $\preceq_t$  βG
          using ball-le-βG by metis
        moreover have gterm-of-term A = AG
          by (simp add: A-def)

```

```

moreover have gterm-of-term  $\beta = \beta_G$ 
  by (simp add:  $\beta$ -def)
ultimately have gterm-of-term  $A \preceq_t$  gterm-of-term  $\beta$ 
  by argo
thus less-B== (atm-of (Neg  $A$ )  $\cdot$  Var)  $\beta$ 
  using inj-term-of-gterm[THEN injD]
  by (auto simp: less-B-def A-def  $\beta$ -def)
next
  show  $\Gamma' = \text{trail-decide } \Gamma$  (Neg  $A$ )  $\cdot$  l Var
    using  $\langle \Gamma' = \text{decide-lit } (\text{Neg } A) \# \Gamma \rangle$ 
    unfolding subst-lit-id-subst .
qed

thus scl-fol.scl  $N \beta S'$ 
  unfolding scl-fol.scl-def by argo
next
fix  $S'' :: ('f, 'v)$  SCL-FOL.state
assume scl-fol.conflict  $N \beta S' S''$ 

moreover have  $\nexists S''. \text{scl-fol.conflict } N \beta S' S''$ 
proof –
  have  $\neg \text{Ex } (\text{scl-fol.conflict } N \beta ((\text{lit-of-glit } (\text{Neg } A_G), \text{None}) \# \Gamma, U, C))$ 
  proof (rule nex-conflict-if-no-clause-could-propagate-comp)
    show  $\neg \text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } \Gamma_G)$ 
    using step-hyps by argo
  next
    show  $\forall x \mid \in \mid \text{trail-atms } \Gamma_G. x \prec_t \text{atm-of } (\text{Neg } A_G)$ 
    unfolding literal.sel
    using step-hyps linorder-trm.is-least-in-fset-iff by simp
  next
    show  $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G). \text{clause-could-propagate } \Gamma_G C$ 
    ( $\neg \text{Neg } A_G$ ))
    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  $\langle \Gamma' = \text{decide-lit } (\text{Neg } A) \# \Gamma \rangle$  decide-lit-def
    using C'-def C-def  $\langle U' = U \rangle$  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 AG

let ?f = gtrailelem-of-trailelem
have Γ' = map ?f ΓG'
  unfolding Γ'-def ..
also have ... = map ?f ((Pos AG, None) # ΓG)
  unfolding ⟨ΓG' = (Pos AG, None) # ΓG⟩ ..
also have ... = ?f (Pos AG, None) # map ?f ΓG
  unfolding list.map ..
also have ... = ?f (Pos AG, None) # Γ
  unfolding Γ-def ..
also have ... = (lit-of-glit (Pos AG), None) # Γ
  unfolding prod.case option.map gtrailelem-of-trailelem-def ..
also have ... = (Pos (term-of-gterm AG), None) # Γ
  unfolding lit-of-glit-def literal.map ..
also have ... = (Pos A, None) # Γ
  unfolding A-def ..
finally have Γ' = decide-lit (Pos A) # Γ
  unfolding decide-lit-def .

have U' = U
  unfolding U'-def ⟨UG' = UG⟩ U-def ..

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.decide N β S S'
    unfolding S-def S'-def ⟨U' = U⟩ C-def C'-def ⟨CG = None⟩ ⟨CG' = None⟩
  option.map
  proof (rule scl-fol.decideI')
    show is-ground-lit (Pos A · l Var)
      by (simp add: A-def)
  next
  have ∀ x |∈| trail-atms ΓG. x ≺t AG
    using step-hyps linorder-trm.is-least-in-ffilter-iff by simp
  hence AG |∉| trail-atms ΓG
    by blast
  hence AG ∉ atm-of 'fst ' set ΓG
    unfolding fset-trail-atms .
  hence term-of-gterm AG ∉ term-of-gterm ' atm-of ' fst ' set ΓG
    using inj-image-mem-iff inj-term-of-gterm by metis
  hence term-of-gterm AG ∉ set (map (λx. term-of-gterm (atm-of (fst x)))
ΓG)
    unfolding image-set list.map-comp comp-def .

```

hence $A \notin \text{set } (\text{map } (\lambda x. \text{atm-of } (\text{lit-of-glit } (\text{fst } x))) \Gamma_G)$
unfolding $A\text{-def atm-of-lit-of-glit-conv}$.
hence $A \notin \text{atm-of 'fst ' set } \Gamma$
unfolding $\text{image-set list.map-comp comp-def } \Gamma\text{-def gtrailelem-of-trailelem-def}$
 $\text{fst-case-prod-simp}$.
hence $A \notin \text{trail-atms } \Gamma$
unfolding fset-trail-atms .
hence $\neg \text{trail-defined-lit } \Gamma (Pos A \cdot l Var)$
by $(\text{simp add: trail-defined-lit-iff-trail-defined-atm})$
thus $\neg \text{SCL-FOL.trail-defined-lit } \Gamma (Pos A \cdot l Var)$
by $(\text{simp add: SCL-FOL.trail-defined-lit-def trail-defined-lit-def})$
next
have $A_G \in \text{atms-of-clss } (N_G \mid \cup \mid U_G)$
using $\text{step-hyps linorder-trm.is-least-in-ffilter-iff}$ **by** simp
hence $A_G \in \text{atms-of-clss } N_G$
unfolding $\langle \text{atms-of-clss } (N_G \mid \cup \mid U_G) = \text{atms-of-clss } N_G \rangle$.
hence $A_G \preceq_t \beta_G$
using $\text{ball-le-}\beta_G$ **by** metis
moreover have $\text{gterm-of-term } A = A_G$
by $(\text{simp add: } A\text{-def})$
moreover have $\text{gterm-of-term } \beta = \beta_G$
by $(\text{simp add: } \beta\text{-def})$
ultimately have $\text{gterm-of-term } A \preceq_t \text{gterm-of-term } \beta$
by argo
thus $\text{less-B}^{\text{==}} (\text{atm-of } (Pos A) \cdot a Var) \beta$
using $\text{inj-term-of-gterm[THEN injD]}$
by $(\text{auto simp: less-B-def } A\text{-def } \beta\text{-def})$
next
show $\Gamma' = \text{trail-decide } \Gamma (Pos A \cdot l Var)$
using $\langle \Gamma' = \text{decide-lit } (Pos A) \# \Gamma \rangle$
unfolding $\text{subst-lit-id-subst}$.
qed

thus $\text{scl-fol.scl } N \beta S S'$
unfolding scl-fol.scl-def **by** argo
next
fix $S'' :: ('f, 'v) \text{SCL-FOL.state}$
assume $\text{scl-fol.conflict } N \beta S' S''$

moreover have $\nexists S''. \text{scl-fol.conflict } N \beta S' S''$
proof –
have $\neg \text{Ex } (\text{scl-fol.conflict } N \beta ((\text{lit-of-glit } (Pos A_G), \text{None}) \# \Gamma, U, C))$
proof $(\text{rule nex-conflict-if-no-clause-could-propagate-comp})$
show $\neg \text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } \Gamma_G)$
using step-hyps **by** argo
next
show $\forall x \in \text{trail-atms } \Gamma_G. x \prec_t \text{atm-of } (Pos A_G)$
unfolding literal.sel
using $\text{step-hyps linorder-trm.is-least-in-ffilter-iff}$ **by** simp

next
have *clause-could-propagate* $\Gamma_G C (Neg A_G) \implies \text{trail-false-cls } \Gamma_G' C$ **for**
 C

unfolding $\langle \Gamma_G' = (Pos A_G, None) \# \Gamma_G \rangle$
using *trail-false-if-could-have-propagated* **by** *fastforce*

thus $\neg (\exists C | \in | \text{iefac } \mathcal{F} | \uparrow (N_G | \cup | U_G). \text{clause-could-propagate } \Gamma_G C (-$
 $Pos A_G))$

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* $\langle \Gamma' = \text{decide-lit } (Pos A) \# \Gamma \rangle$ *decide-lit-def*
using *C'-def C-def* $\langle U' = U \rangle$ *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 $C_G | \in | \text{iefac } \mathcal{F} | \uparrow (N_G | \cup | U_G)$ **and** $C_G\text{-prop: clause-could-propagate } \Gamma_G$
 $C_G (Pos A_G)$
using *step-hyps linorder-cls.is-least-in-fset-iff* **by** *simp-all*

define $A :: (f, 'v)$ *term* **where**
 $A = \text{term-of-gterm } A_G$

define $C :: (f, 'v)$ *term clause* **where**
 $C = \text{cls-of-gcls } C_G$

have *ord-res.is-maximal-lit* $(Pos A_G) C_G$ **and** *trail-false-cls* $\Gamma_G \{ \#K \in \# C_G.$
 $K \neq Pos A_G \# \}$
using $\langle \text{clause-could-propagate } \Gamma_G C_G (Pos A_G) \rangle$
unfolding *clause-could-propagate-def* **by** *metis+*

then obtain C_G' **where** $C_G = \text{add-mset } (Pos A_G) C_G'$
by *(metis linorder-lit.is-maximal-in-mset-iff mset-add)*

define $C' :: (f, 'v)$ *term clause* **where**
 $C' = \text{cls-of-gcls } C_G'$

let $?f = \text{gtrailelem-of-trailelem}$
have $\Gamma' = \text{map } ?f \Gamma_G'$


```

unfolding  $\Gamma'$ -def ..
also have ... = map ?f ((Pos  $A_G$ , Some (efac  $C_G$ )) #  $\Gamma_G$ )
unfolding  $\langle \Gamma_G' = (Pos A_G, Some -) \# \Gamma_G \rangle$  ..
also have ... = ?f (Pos  $A_G$ , Some (efac  $C_G$ )) # map ?f  $\Gamma_G$ 
unfolding list.map ..
also have ... = ?f (Pos  $A_G$ , Some (efac  $C_G$ )) #  $\Gamma$ 
unfolding  $\Gamma$ -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)) #  $\Gamma$ 
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 ≠ Pos  $A_G$ #}.
  K ≠ Pos  $A_G$ #},
  lit-of-glit (Pos  $A_G$ ), Var)) #  $\Gamma$ 
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 ≠ Pos  $A_G$ #}, lit-of-glit (Pos  $A_G$ ), Var))
#  $\Gamma$ 
by (simp add: filter-filter-mset)
also have ... = (Pos  $A$ , Some (cls-of-gcls {#K ∈#  $C_G$ . K ≠ Pos  $A_G$ #}, Pos
 $A$ , Var)) #  $\Gamma$ 
by (simp add: A-def lit-of-glit-def)
also have ... = (Pos  $A$ , Some (cls-of-gcls {#L ∈#  $C_G$ . lit-of-glit L ≠ Pos
 $A$ #}, Pos  $A$ , Var)) #  $\Gamma$ 
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 ≠ Pos  $A$ #}, Pos  $A$ ,
  Var)) #  $\Gamma$ 
unfolding cls-of-gcls-def
unfolding image-mset-filter-mset-swap[of lit-of-glit  $\lambda L$ . L ≠ Pos  $A$   $C_G$ ]
unfolding cls-of-gcls-def[symmetric] ..
also have ... = (Pos  $A \cdot l$  Var, Some ({#L ∈# cls-of-gcls  $C_G$ . L ≠ Pos  $A$ #},
  Pos  $A$ , Var)) #  $\Gamma$ 
by simp
also have ... = (Pos  $A \cdot l$  Var, Some ({#L ∈#  $C$ . L ≠ Pos  $A$ #}, Pos  $A$ , Var))
#  $\Gamma$ 
unfolding C-def ..
finally have  $\Gamma' = propagate-lit$  (Pos  $A$ ) {#L ∈#  $C$ . L ≠ Pos  $A$ #} Var #  $\Gamma$ 
unfolding propagate-lit-def .

```

```

have  $U' = U$ 
  unfolding  $U'$ -def  $\langle U_G' = U_G \rangle$   $U$ -def ..

obtain  $C_{G0}$  where  $C_{G0} \mid \in \mid N_G \mid \cup \mid U_G$  and  $C_G = \text{iefac } \mathcal{F} \ C_{G0}$ 
  using  $\langle C_G \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G) \rangle$  by blast

define  $C_0 :: ('f, 'v)$  term clause where
   $C_0 = \text{cls-of-gcls } C_{G0}$ 

have  $\text{ord-res.is-maximal-lit } (Pos \ A_G) \ C_{G0}$ 
  using  $\langle \text{ord-res.is-maximal-lit } (Pos \ A_G) \ C_G \rangle \langle C_G = \text{iefac } \mathcal{F} \ C_{G0} \rangle$ 
  by ( $\text{metis iefac-def linorder-lit.is-maximal-in-mset-iff set-mset-efac}$ )

have  $\neg \text{Ex } (\text{scl-fol.conflict } N \ \beta \ S)$ 
  using  $\text{step-hyps nex-conflict-if-nbex-trail-false}$  by metis

moreover have  $\text{scl-fol.reasonable-scl } N \ \beta \ S \ S'$ 
  unfolding  $\text{scl-fol.reasonable-scl-def}$ 
proof ( $\text{intro conjI impI notI ; (elim exE) ?}$ )
  have  $\text{scl-fol.propagate } N \ \beta \ S \ S'$ 
    unfolding  $S$ -def  $S'$ -def  $\langle U' = U \rangle$   $C$ -def  $C'$ -def  $\langle C_G = \text{None} \rangle \langle C_G' = \text{None} \rangle$ 
  option.map
  proof ( $\text{rule scl-fol.propagateI}$ )
    show  $C_0 \mid \in \mid N \mid \cup \mid U$ 
      unfolding  $C_0$ -def  $N$ -def  $U$ -def
      using  $\langle C_{G0} \mid \in \mid N_G \mid \cup \mid U_G \rangle$ 
      by blast
    next
    show  $\text{is-ground-cls } (C_0 \cdot \text{Var})$ 
      by ( $\text{simp add: } C_0$ -def)

  have  $Pos \ A \in \# \ C_0$ 
    unfolding  $A$ -def  $C_0$ -def
    by ( $\text{metis (no-types, lifting) } \langle C_G = \text{iefac } \mathcal{F} \ C_{G0} \rangle \langle \text{ord-res.is-maximal-lit } (Pos \ A_G) \ C_G \rangle$ 
       $\text{cls-of-gcls-def iefac-def image-eqI linorder-lit.is-maximal-in-mset-iff}$ 
       $\text{lit-of-glit-def literal.map(1) multiset.set-map set-mset-efac}$ )

  then show  $C_0 = \text{add-mset } (Pos \ A) \ (C_0 - \{\#Pos \ A\# \})$ 
    by simp

  have  $A_G \mid \in \mid \text{atms-of-clss } (N_G \mid \cup \mid U_G)$ 
    using  $\text{step-hyps linorder-trm.is-least-in-ffilter-iff}$  by simp
  hence  $A_G \mid \in \mid \text{atms-of-clss } N_G$ 
    unfolding  $\langle \text{atms-of-clss } (N_G \mid \cup \mid U_G) = \text{atms-of-clss } N_G \rangle$  .
  hence  $A_G \preceq_t \beta_G$ 
    using  $\text{ball-le-}\beta_G$  by metis
  moreover have  $\text{gterm-of-term } A = A_G$ 
    by ( $\text{simp add: } A$ -def)

```

moreover have $gterm\text{-of-term } \beta = \beta_G$
by (*simp add: β -def*)
ultimately have $gterm\text{-of-term } A \preceq_t gterm\text{-of-term } \beta$
by *argo*
hence $less\text{-B}^{==} A \beta$
by (*auto simp: less-B-def A-def β -def*)

show $\forall K \in\# C_0 \cdot Var. less\text{-B}^{==} (atm\text{-of } K) \beta$
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** $K\text{-def}: K = lit\text{-of-glit } K_G$
unfolding $C_0\text{-def cls-of-gcls-def}$ **by** *blast*

have $K_G \preceq_l Pos A_G$
using $\langle ord\text{-res.is-maximal-lit } (Pos A_G) C_{G0} \rangle \langle K_G \in\# C_{G0} \rangle$
unfolding *linorder-lit.is-maximal-in-mset-iff* **by** *fastforce*

hence $atm\text{-of } K_G \preceq_t A_G$
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 $less\text{-B}^{==} (atm\text{-of } K) A$
by (*auto simp: less-B-def K-def A-def atm-of-lit-of-glit-conv*)

then show $less\text{-B}^{==} (atm\text{-of } K) \beta$
using $\langle less\text{-B}^{==} A \beta \rangle$ **by** *order*
qed

show $\{\#K \in\# C_0. K \neq Pos A\# \} = \{\#K \in\# remove1\text{-mset } (Pos A) C_0. K \cdot l Var \neq Pos A \cdot l Var\# \}$
by *simp*

show $\{\#K \in\# remove1\text{-mset } (Pos A) C_0. K = Pos A\# \} = \{\#K \in\# remove1\text{-mset } (Pos A) C_0. K \cdot l Var = Pos A \cdot l Var\# \}$
by *simp*

have $trail\text{-false-cls } \Gamma_G (\{\#K_G \in\# C_{G0}. K_G \neq Pos A_G\# \})$
by (*smt (verit, ccfv-threshold) $\langle C_G = iefac \mathcal{F} C_{G0} \rangle \langle trail\text{-false-cls } \Gamma_G \{\#K \in\# C_G. K \neq Pos A_G\# \} \rangle iefac\text{-def mem-Collect-eq set-mset-efac set-mset-filter trail-false-cls-def$*)

moreover have ($cls\text{-of-gcls } \{\#K_G \in\# C_{G0}. K_G \neq Pos A_G\# \} :: ('f, 'v) term clause$) =
 $\{\#K \in\# C_0. K \neq Pos A\# \}$
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 ($\{\#K \in\# C_0. K \neq Pos A\# \} \cdot Var$)
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 ($\{\#K \in\# C_0. K \neq Pos A\# \} \cdot Var$)
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 $\forall x \in | trail-atms \Gamma_G. x \prec_t A_G$
using *step-hyps linorder-trm.is-least-in-ffilter-iff* **by** *simp*
hence $A_G \notin | trail-atms \Gamma_G$
by *blast*
hence $A_G \notin atm\text{-of } 'fst \text{ ' set } \Gamma_G$
unfolding *fset-trail-atms* .
hence *term-of-gterm* $A_G \notin \text{term-of-gterm } 'atm\text{-of } 'fst \text{ ' set } \Gamma_G$
using *inj-image-mem-iff inj-term-of-gterm* **by** *metis*
hence *term-of-gterm* $A_G \notin set (map (\lambda x. \text{term-of-gterm } (atm\text{-of } (fst x))))$

Γ_G)
unfolding *image-set list.map-comp comp-def* .
hence $A \notin set (map (\lambda x. atm\text{-of } (lit\text{-of-glit } (fst x)))) \Gamma_G$
unfolding *A-def atm-of-lit-of-glit-conv* .
hence $A \notin atm\text{-of } 'fst \text{ ' set } \Gamma$
unfolding *image-set list.map-comp comp-def* Γ -def *gtrailelem-of-trailelem-def*
fst-case-prod-simp .
hence $A \notin | trail-atms \Gamma$
unfolding *fset-trail-atms* .
hence $\neg trail\text{-defined-lit } \Gamma (Pos A \cdot l Var)$
by (*simp add: trail-defined-lit-iff-trail-defined-atm*)
thus $\neg SCL\text{-FOL.trail-defined-lit } \Gamma (Pos A \cdot l Var)$
by (*simp add: SCL-FOL.trail-defined-lit-def trail-defined-lit-def*)

have *set-mset* (*add-mset* (*Pos A*) $\{\#K \in\# remove1\text{-mset } (Pos A) C_0. K = Pos A\#\}$) =
 $\{Pos A\}$
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*) $\{\#K \in\# remove1\text{-mset } (Pos A) C_0. K = Pos A\#\}$))
by (*metis* (*no-types*, *lifting*) *finite-imageI finite-set-mset image-empty*
image-insert
is-unifier-alt singletonD)
hence *is-unifiers* *Var* $\{atm\text{-of } ' set\text{-mset}$
(*add-mset* (*Pos A*) $\{\#K \in\# remove1\text{-mset } (Pos A) C_0. K = Pos A\#\}\}$
unfolding *SCL-FOL.is-unifiers-def* **by** *simp*
thus *SCL-FOL.is-imgu* *Var* $\{atm\text{-of } ' set\text{-mset}$
(*add-mset* (*Pos A*) $\{\#K \in\# remove1\text{-mset } (Pos A) C_0. K = Pos A\#\}\}$

```

unfolding SCL-FOL.is-imagu-def by simp

have  $\{\#K \in\# C_{G0}. K \neq Pos A_G\# \} = \{\#K \in\# C_G. K \neq Pos A_G\# \}$ 
using  $\langle ord-res.is-maximal-lit (Pos A_G) C_G \rangle$ 
using  $\langle ord-res.is-maximal-lit (Pos A_G) C_{G0} \rangle$ 
unfolding  $\langle C_G = iefac \mathcal{F} C_{G0} \rangle$ 
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  $\{\#K \in\# C_0. K \neq Pos A\# \} = \{\#K \in\# C. K \neq Pos A\# \}$ 
unfolding C0-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  $\Gamma' = trail-propagate \Gamma (Pos A \cdot l Var) (\{\#K \in\# C_0. K \neq Pos A\# \} \cdot$ 
Var) Var
unfolding subst-cls-id-subst subst-lit-id-subst
using  $\langle \Gamma' = propagate-lit (Pos A) \{\#L \in\# C. L \neq Pos A\# \} Var \# \Gamma \rangle$ 
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, C) (Γ', U', C') rule: scl-fol.decide.cases)
case (decideI L γ)
show False
using  $\langle \Gamma' = decide-lit (L \cdot l \gamma) \# \Gamma \rangle$ 
using  $\langle \Gamma' = propagate-lit (Pos A) \{\#L \in\# C. L \neq Pos A\# \} Var \# \Gamma \rangle$ 
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 CG)

have  $\Gamma' = \Gamma$ 
unfolding Γ'-def  $\langle \Gamma_{G'} = \Gamma_G \rangle$  Γ-def ..

have  $U' = U$ 
unfolding U'-def  $\langle U_{G'} = U_G \rangle$  U-def ..

```

have
 $C_G\text{-in}: C_G \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)$ **and**
 $C_G\text{-false}: \text{trail-false-cls } \Gamma_G \ C_G$ **and**
 $C_G\text{-lt}: \forall E_G \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G). E_G \neq C_G \longrightarrow \text{trail-false-cls } \Gamma_G \ E_G$
 $\longrightarrow C_G \prec_c E_G$
using $\langle \text{linorder-cls.is-least-in-fset} - C_G \rangle$
unfolding $\text{atomize-conj linorder-cls.is-least-in-filter-iff}$ **by** argo

have $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L \ C_G$
proof (rule notI , elim exE conjE)
fix $L :: 'f \text{ gliteral}$
assume $\text{is-pos } L$ **and** $C_G\text{-max-lit}: \text{ord-res.is-maximal-lit } L \ C_G$

have $\{ \# \} \mid \notin \mid \text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)$
using $C_G\text{-lt}$
by ($\text{metis (full-types) } C_G\text{-max-lit bot-fset.rep-eq fBex-fempty linorder-cls.leD}$
 $\text{linorder-lit.is-maximal-in-mset-iff mempty-lesseq-cls set-mset-empty}$
 $\text{trail-false-cls-mempty}$)

have $\text{trail-false-lit } \Gamma_G \ L$
using $C_G\text{-max-lit } C_G\text{-false}$
unfolding $\text{linorder-lit.is-maximal-in-mset-iff trail-false-cls-def}$
by metis

then obtain $L_n \ \Gamma_{G_0}$ **where** $\Gamma_G = L_n \# \Gamma_{G_0}$
unfolding $\text{trail-false-lit-def}$
by ($\text{metis (no-types) List.insert-def image-iff insert-Nil neq-Nil-conv}$)

moreover have
 $AAA: \forall x \ xs. \Gamma_G = x \# xs \longrightarrow$
 $((\text{snd } x \neq \text{None}) \longleftrightarrow \text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } (x \#$
 $xs))) \wedge$
 $(\text{snd } x \neq \text{None} \longrightarrow \text{is-pos } (\text{fst } x)) \wedge$
 $(\forall x \in \text{set } xs. \text{snd } x = \text{None})$ **and**
 $BBB: (\forall \Gamma_1 \ L_n \ \Gamma_0. \Gamma_G = \Gamma_1 \ @ \ L_n \ # \ \Gamma_0 \longrightarrow \text{snd } L_n = \text{None} \longrightarrow$
 $\neg \text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } (L_n \ # \ \Gamma_0)))$ **and**
 $\Gamma_G\text{-sorted}: \text{sorted-wrt } (\lambda x \ y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \ \Gamma_G$
using $\langle \text{ord-res-11-invars } N_G \ S_G \rangle [\text{unfolded } S_G\text{-def ord-res-11-invars.simps}$
 $\text{ord-res-10-invars.simps}]$
by simp-all

moreover have $\text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } \Gamma_G)$
using $C_G\text{-in } C_G\text{-false}$ **by** metis

ultimately have $\text{snd } L_n \neq \text{None}$ **and** $\text{is-pos } (\text{fst } L_n)$ **and** $\forall x \in \text{set } \Gamma_{G_0}. \text{snd } x = \text{None}$
unfolding atomize-conj **by** metis

have $\neg \text{fBex } (\text{iefac } \mathcal{F} \mid \uparrow \mid (N_G \mid \cup \mid U_G)) (\text{trail-false-cls } \Gamma_{G_0})$

proof (*cases* Γ_{G0})
case *Nil*
then show *?thesis*
using $\langle \{\#\} \mid \notin \text{iefac } \mathcal{F} \mid \uparrow (N_G \mid \cup \mid U_G) \rangle$
unfolding *trail-false-cls-def trail-false-lit-def*
by *simp*
next
case (*Cons* x xs)
then show *?thesis*
using $\langle \Gamma_G = Ln \# \Gamma_{G0} \rangle$
using $\langle \forall x \in \text{set } \Gamma_{G0}. \text{snd } x = \text{None} \rangle$
using *BBB[rule-format, of [Ln], unfolded append-Cons append-Nil]*
by *simp*
qed

hence $\neg \text{trail-false-cls } \Gamma_{G0} C_G$
using *C_G-in* **by** *metis*

hence *fst Ln = - L*
using *C_G-false C_G-max-lit Γ_G -sorted[unfolded $\langle \Gamma_G = Ln \# \Gamma_{G0} \rangle$ sorted-wrt.simps]*
by (*smt (verit, ccfv-SIG) Neg-atm-of-iff $\langle \Gamma_G = Ln \# \Gamma_{G0} \rangle$ 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 $\neg \text{is-pos } L$
using $\langle \text{is-pos } (\text{fst } Ln) \rangle$
by (*simp add: is-pos-neg-not-is-pos*)

thus *False*
using $\langle \text{is-pos } L \rangle$ **by** *contradiction*

qed

hence $C_G \mid \in \mid N_G \mid \cup \mid U_G$

proof –

obtain C **where** $C \mid \in \mid N_G \mid \cup \mid U_G$ **and** $C_G = \text{iefac } \mathcal{F} C$
using *C_G-in* **by** *blast*

hence $C_G = C$
using $\langle \# L. \text{is-pos } L \wedge \text{ord-res.is-maximal-lit } L C_G \rangle$
by (*metis clauses-in- \mathcal{F} -have-pos-max-lit ex1-efac-eq-factoring-chain iefac-def*
ord-res.ground-factorings-preserves-maximal-literal)

thus *?thesis*
using $\langle C \mid \in \mid N_G \mid \cup \mid U_G \rangle$ **by** *simp*

qed

```

have scl-fol.conflict  $N \beta S S'$ 
  unfolding  $S\text{-def } S'\text{-def } \langle \Gamma' = \Gamma \rangle \langle U' = U \rangle \mathcal{C}\text{-def } \mathcal{C}'\text{-def } \langle C_G = \text{None} \rangle \langle C_G' = \text{Some } C_G \rangle$  option.map
= Some } C_G } option.map
proof (rule scl-fol.conflictI)
  show cls-of-gcls  $C_G \mid \in \mid N \mid \cup \mid U$ 
    unfolding  $N\text{-def } U\text{-def}$ 
    using  $\langle C_G \mid \in \mid N_G \mid \cup \mid U_G \rangle$  by auto
next
  show is-ground-cls (cls-of-gcls  $C_G \cdot (\text{Var}::'v \Rightarrow (f, -) \text{Term.term})$ )
    by simp
next
  have trail-false-cls  $\Gamma_G C_G$ 
    using  $\langle \text{trail-false-cls } \Gamma_G C_G \rangle$  .

  hence trail-false-cls  $\Gamma (\text{cls-of-gcls } C_G \cdot \text{Var})$ 
    unfolding  $\Gamma\text{-def } \text{subst-cls-id-subst}$ 
    using trail-false-cls-nonground-iff-trail-false-cls-ground by metis

  thus SCL-FOL.trail-false-cls  $\Gamma (\text{cls-of-gcls } C_G \cdot \text{Var})$ 
    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  $\Gamma = \text{gtrailelem-of-trailelem } (L_G, n_G) \# \Gamma'$ 
    unfolding  $\Gamma\text{-def } \Gamma'\text{-def } \langle \Gamma_G = (L_G, n_G) \# \Gamma_G' \rangle$  by simp

  have  $U' = U$ 
    unfolding  $U'\text{-def } \langle U_G' = U_G \rangle U\text{-def}$  ..

  have  $\neg \text{Ex } (\text{scl-fol.conflict } N \beta S)$ 
    using  $\langle C_G = \text{Some } C_G \rangle \text{nex-conflict-if-alread-in-conflict}$  by metis

  moreover have scl-fol.reasonable-scl  $N \beta S S'$ 
    unfolding scl-fol.reasonable-scl-def
  proof (intro conjI impI notI ; (elim exE) ?)
    have scl-fol.skip  $N \beta S S'$ 
      unfolding  $S\text{-def } S'\text{-def } \langle U' = U \rangle \mathcal{C}\text{-def } \mathcal{C}'\text{-def } \langle C_G = \text{Some } C_G \rangle \langle C_G' = \text{Some } C_G \rangle$  option.map
    unfolding  $\langle \Gamma = - \# \Gamma' \rangle \text{gtrailelem-of-trailelem-def prod.case}$ 
  proof (rule scl-fol.skipI)
    have  $- \text{lit-of-glit } L_G = \text{lit-of-glit } (- L_G)$ 
      by (cases  $L_G$ ) (simp-all add: lit-of-glit-def)
    show  $- \text{lit-of-glit } L_G \notin \# \text{cls-of-gcls } C_G \cdot \text{Var}$ 

```



```

unfolding subst-cls-id-subst
unfolding ⟨- lit-of-glit  $L_G = \text{lit-of-glit } (- L_G)$ ⟩
unfolding cls-of-gcls-def
using ⟨-  $L_G \notin \# C_G$ ⟩ inj-image-mset-mem-iff[OF inj-lit-of-glit]
by metis
qed

thus scl-fol.scl  $N \beta S S'$ 
unfolding scl-fol.scl-def by argo
next
fix  $S'' :: ('f, 'v) \text{SCL-FOL.state}$ 
assume scl-fol.conflict  $N \beta S' S''$ 

moreover have  $\nexists S''. \text{scl-fol.conflict } N \beta S' S''$ 
unfolding  $S'\text{-def } C'\text{-def } \langle C_G' = \text{Some } C_G \rangle$  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  $L_G C_G \Gamma_G'' D_G$ )

have  $\Gamma' = \Gamma$ 
unfolding  $\Gamma'\text{-def } \langle \Gamma_G' = \Gamma_G \rangle \Gamma\text{-def} ..$ 

have  $U' = U$ 
unfolding  $U'\text{-def } \langle U_G' = U_G \rangle U\text{-def} ..$ 

have  $C = \text{Some } (\text{cls-of-gcls } D_G, \text{Var})$ 
unfolding  $C\text{-def } \langle C_G = \text{Some } D_G \rangle \text{option.map} ..$ 

hence  $C\text{-eq: } C = \text{Some}$ 
  (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( $\gamma$ )
  uminus-not-id')

have  $C' = \text{Some } ($ 
  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  $C'\text{-def } \langle C_G' = \text{Some } \rightarrow \text{option.map}$ 
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 mset-map-invr)

```

hence $C' \text{-eq: } C' = \text{Some } ($
 $(\text{remove1-mset } (- (\text{lit-of-glit } L_G)) (\text{cls-of-gcls } D_G) \cdot \text{Var} +$
 $\text{remove1-mset } (\text{lit-of-glit } L_G) (\text{cls-of-gcls } C_G) \cdot \text{Var}) \cdot \text{Var}, \text{Var})$
by *simp*

have *linorder-lit.is-greatest-in-mset* $C_G L_G$
using $\langle \text{ord-res-11-invars } N_G S_G \rangle [\text{unfolded } S_G \text{-def } \langle \Gamma_G = (L_G, \text{Some } C_G) \#$
 $\Gamma_G'' \rangle]$
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* $(C_G - \{ \#L_G \# \}) = \text{cls-of-gcls } \{ \#K \in \# C_G. K \neq L_G \# \}$
by *argo*

moreover have *cls-of-gcls* $(C_G - \{ \#L_G \# \}) = \text{cls-of-gcls } C_G - \text{cls-of-gcls}$
 $\{ \#L_G \# \}$
unfolding *cls-of-gcls-def*
proof (*rule image-mset-Diff*)
show $\{ \#L_G \# \} \subseteq \# C_G$
by (*metis* $\langle \text{ord-res.is-strictly-maximal-lit } L_G C_G \rangle$ *linorder-lit.is-greatest-in-mset-iff*
single-subset-iff)

qed

ultimately have *cls-of-gcls* $C_G - \{ \# \text{lit-of-glit } L_G \# \} = \text{cls-of-gcls } \{ \#K \in \#$
 $C_G. K \neq L_G \# \}$
by (*metis* $\langle \text{add-mset } L_G \{ \#y \in \# C_G. y \neq L_G \# \} = C_G \rangle$ *cls-of-gcls-def*
image-mset-remove1-mset-if
union-single-eq-member)

have $\neg \text{Ex } (\text{scl-fol.conflict } N \beta S)$
using $\langle C_G = \text{Some } \rightarrow \text{nex-conflict-if-alread-in-conflict} \rangle$ **by** *metis*

moreover have *scl-fol.reasonable-scl* $N \beta S S'$
unfolding *scl-fol.reasonable-scl-def*
proof (*intro conjI impI notI ; (elim exE) ?*)
have *scl-fol.resolve* $N \beta S S'$
unfolding *S-def S'-def* $\langle \Gamma' = \Gamma \rangle \langle U' = U \rangle$
unfolding *C-eq C'-eq*
proof (*rule scl-fol.resolveI*)
show $\Gamma = \text{trail-propagate } (\text{map } \text{gtrailelem-of-trailelem } \Gamma_G'')$
 $(\text{lit-of-glit } L_G) (\text{remove1-mset } (\text{lit-of-glit } L_G) (\text{cls-of-gcls } C_G)) \text{Var}$
unfolding $\Gamma \text{-def } \langle \Gamma_G = (L_G, \text{Some } C_G) \# \Gamma_G'' \rangle$ *gtrailelem-of-trailelem-def*
list.map prod.case

```

      unfolding propagate-lit-def subst-lit-id-subst option.map
      unfolding ⟨remove1-mset (lit-of-glit LG) (cls-of-gcls CG) = cls-of-gcls
{#K ∈# CG. K ≠ LG#}⟩
      by argo
    next
      show lit-of-glit LG ·l Var = - (- lit-of-glit LG ·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 LG)
        (remove1-mset (- lit-of-glit LG) (cls-of-gcls DG)) · Var) ∩
      vars-cls (add-mset (lit-of-glit LG)
        (remove1-mset (lit-of-glit LG) (cls-of-gcls CG)) · 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-ingu Var
      {{atm-of (- lit-of-glit LG) ·a Var, atm-of (lit-of-glit LG) ·a Var}}
      by (simp add: SCL-FOL.is-ingu-def SCL-FOL.is-unifiers-def SCL-FOL.is-unifier-def)
    next
      show is-grounding-merge Var
      (vars-cls
        (add-mset (- lit-of-glit LG) (remove1-mset (- lit-of-glit LG) (cls-of-gcls
DG)) · Var))
      (rename-subst-domain Var Var)
      (vars-cls
        (add-mset (lit-of-glit LG) (remove1-mset (lit-of-glit LG) (cls-of-gcls CG))
· 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 ⟨CG' = 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  $\langle U_G' = \text{finsert } C_G \ U_G \rangle$ 
  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 C = Some (add-mset K D, Var)
  by (smt (verit) D-def K-def 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 C' = None
  unfolding C'-def  $\langle C_G' = \text{None} \rangle$  option.map ..

have  $\neg$  Ex (scl-fol.conflict N  $\beta$  S)
  using  $\langle C_G = \text{Some } \rightarrow \text{nex-conflict-if-alread-in-conflict} \text{ by } \text{metis} \rangle$ 

moreover have scl-fol.reasonable-scl N  $\beta$  S S'
  unfolding scl-fol.reasonable-scl-def
  proof (intro conjI impI notI ; (elim exE) ?)
  have scl-fol.backtrack N  $\beta$  S S'
    unfolding S-def S'-def
    unfolding  $\langle U' = \text{finsert } (\text{add-mset } K \ D) \ U \rangle$   $\langle C = \text{Some } (\text{add-mset } K \ D,$ 
  Var)  $\rangle$   $\langle C' = \text{None} \rangle$ 
    proof (rule scl-fol.backtrackI)
    show  $\Gamma = \text{trail-decide } ([ ] \ @ \ \Gamma')$  (lit-of-glit  $L_G$ )
      unfolding append-Nil
      unfolding decide-lit-def
      unfolding  $\Gamma$ -def  $\langle \Gamma_G = - \ # \ \rightarrow \ \text{list.map } \Gamma'$ -def[symmetric]

```

```

      unfolding gtrailelem-of-trailelem-def prod.case option.map
    ..
next
  show lit-of-glit  $L_G = - (K \cdot l \text{ Var})$ 
    unfolding K-def by simp
next
  have sorted-wrt  $(\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma_G$ 
    using  $\langle \text{ord-res-11-invars } N_G \ S_G \rangle [\text{unfolded } S_G\text{-def}]$ 
    unfolding ord-res-11-invars.simps ord-res-10-invars.simps
    by fast

  hence trail-consistent  $\Gamma_G$ 
    using trail-consistent-if-sorted-wrt-atoms by metis

  hence  $\neg$  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  $\neg$  trail-false-lit  $\Gamma_G' (- L_G)$ 
    using trail-defined-lit-iff-true-or-false by metis

  hence  $\neg$  trail-false-cls  $\Gamma_G' C_G$ 
    using  $\langle - L_G \in\# C_G \rangle$ 
    unfolding trail-false-cls-def by metis

  hence  $\neg$  trail-false-cls  $\Gamma' (\text{add-mset } K \ D)$ 
    unfolding  $\Gamma'\text{-def } \langle \text{add-mset } K \ D = \text{cls-of-gcls } C_G \rangle$ 
    unfolding trail-false-cls-nonground-iff-trail-false-cls-ground .

  moreover have is-ground-cls  $(\text{add-mset } K \ D)$ 
    using C-def  $\langle C = \text{Some } (\text{add-mset } K \ D, \text{Var}) \rangle$  step-hyps(1) by auto

  ultimately have  $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } K \ D \cdot \gamma) \wedge \text{trail-false-cls } \Gamma'$ 
     $(\text{add-mset } K \ D \cdot \gamma)$ 
    by simp

  thus  $\nexists \gamma. \text{is-ground-cls } (\text{add-mset } K \ D \cdot \gamma) \wedge \text{SCL-FOL.trail-false-cls } \Gamma'$ 
     $(\text{add-mset } K \ D \cdot \gamma)$ 
    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  $N \ \beta \ S \ S'$ 
    unfolding scl-fol.scl-def by argo
next
  fix  $S'' :: ('f, 'v) \text{SCL-FOL.state}$ 
  assume scl-fol.decide  $N \ \beta \ S \ S'$ 
  thus False

```

```

    unfolding S-def ⟨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 ⇒ 'b and
    R :: 'b ⇒ 'b ⇒ bool and Q :: 'a ⇒ 'a ⇒ bool
  assumes
    wf: wfp-on B R and
    sub: f ' A ⊆ B and
    mono: ∧x y. x ∈ A ⇒ y ∈ A ⇒ Q x y ⇒ 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' ⊆ A and A' ≠ {}
  have f ' A' ⊆ B
    using ⟨A' ⊆ A⟩ sub by blast
  moreover have f ' A' ≠ {}
    using ⟨A' ≠ {}⟩ by blast
  ultimately have ∃z∈f ' A'. ∀y. R y z → y ∉ f ' A'
    using wf wfp-on-iff-ex-minimal by blast
  hence ∃z∈A'. ∀y. R (f y) (f z) → y ∉ A'
    by blast
  thus ∃z∈A'. ∀y. Q y z → y ∉ A'
    using ⟨A' ⊆ A⟩ mono by blast
qed

For AFP-devel, delete  $\llbracket wfp\text{-on } ?B \ ?R; ?f \ ' \ ?A \subseteq \ ?B; \bigwedge x \ y. \llbracket x \in \ ?A; \ y \in \ ?A; \ ?Q \ x \ y \rrbracket \implies \ ?R \ (\ ?f \ x) \ (\ ?f \ y) \rrbracket \implies wfp\text{-on } \ ?A \ ?Q$  as it is available in HOL.Wellfounded.

corollary (in scl-fol-calculus) termination-projectable-strategy:
  fixes
    N :: ('f, 'v) Term.term clause fset and
    β :: ('f, 'v) Term.term and
    strategy and strategy-init and proj
  assumes 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

```

```

shows wfp-on {S. strategy** strategy-init S} strategy-1-1
proof (rule wfp-on-antimono-stronger)
show wfp-on {proj S | S. strategy** strategy-init S} (regular-scl N β)-1-1
proof (rule wfp-on-subset)
  show wfp-on {S. (regular-scl N β)** initial-state S} (regular-scl N β)-1-1
    using termination-regular-scl by metis
next
  show {proj S | S. strategy** strategy-init S} ⊆ {S. (regular-scl N β)** initial-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 $\llbracket scl\text{-fol}\text{-calculus } ?renaming\text{-vars } ?less\text{-B}; \wedge S S'. \llbracket ?strategy^{**} ?strategy\text{-init } S; ?strategy S S' \rrbracket \Longrightarrow scl\text{-fol}\text{-calculus.regular-scl } ?less\text{-B } ?N ?\beta (?proj S) (?proj S'); ?proj ?strategy\text{-init} = initial\text{-state} \rrbracket \Longrightarrow wfp\text{-on } \{S. ?strategy^{**} ?strategy\text{-init } S\} ?strategy^{-1-1}$ 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  $\beta :: 'f$  gterm where
   $\beta = (\text{THE } A. \text{linorder-trm.is-greatest-in-fset } (\text{atms-of-clss } N) A)$ 

show scl-fol.regular-scl (cls-of-gcls |4  $N$ ) (term-of-gterm  $\beta$ ) (state-of-gstate  $S$ )
(state-of-gstate  $S'$ )
proof (rule ord-res-11-is-strategy-for-regular-scl)
  show  $\forall A_G \in | \text{atms-of-clss } N. A_G \preceq_t \beta$ 
  proof (cases atms-of-clss  $N = \{\{\}\}$ )
    case True
    thus ?thesis
    by simp
  next
  case False
  then show ?thesis
  unfolding  $\beta$ -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  $N$ )** ( $\{\{\}\}, \{\{\}\}, [], \text{None}$ )  $S$ 
  using run .
next
  show ord-res-11  $N$   $S$   $S'$ 
  using step .
  qed
next
  show state-of-gstate ( $\{\{\}\}, \{\{\}\}, [], \text{None}$ ) = 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  $N$   $\beta$  (proj  $S$ )  $S'$  and
  strategy-restricts-regular-scl:
   $\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta$  (proj
 $S$ ) (proj  $S'$ ) and
  initial-state: proj strategy-init = initial-state
defines
   $U \equiv \text{state-learned } (\text{proj } S)$ 
shows  $\exists C \gamma. \text{state-conflict } (\text{proj } S) = \text{Some } (C, \gamma) \wedge \neg (\exists D \in | N \cup | U.$ 
subsumes  $D$   $C$ )
proof –
  have (regular-scl  $N$   $\beta$ )** 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 $\llbracket scl\text{-fol}\text{-calculus } ?renaming\text{-vars } ?less\text{-B}; ?strategy^{**} ?strategy\text{-init } ?S; scl\text{-fol}\text{-calculus.backtrack } ?N ?\beta (?proj ?S) ?S'; \bigwedge S S'. \llbracket ?strategy^{**} ?strategy\text{-init } S; ?strategy S S' \rrbracket \implies scl\text{-fol}\text{-calculus.regular}\text{-scl } ?less\text{-B } ?N ?\beta (?proj S) (?proj S'); ?proj ?strategy\text{-init} = initial\text{-state} \rrbracket \implies \exists C \gamma. state\text{-conflict } (?proj ?S) = Some (C, \gamma) \wedge \neg (\exists D | \in | ?N | \cup | state\text{-learned } (?proj ?S). subsumes D C)$ as it is available in *Simple-Clause-Learning.Non-Redundancy*.

corollary (in *simulation-SCLFOL-ground-ordered-resolution*) *ord-res-11-non-subsumption*:

```

fixes  $N_G :: 'f gclause fset \mathbf{and} s :: - \times - \times - \times -$ 
defines
   $\beta \equiv (THE A. linorder\text{-trm.is}\text{-greatest}\text{-in}\text{-fset } (atms\text{-of}\text{-class } N_G) A)$ 
assumes
  run: (ord-res-11 NG)** ({||}, {||}, [], None) s and
  step: scl-fol.backtrack (cls-of-gcls |q NG) (term-of-gterm β) (state-of-gstate s)
s'
shows  $\exists U_{er} \mathcal{F} \Gamma D. s = (U_{er}, \mathcal{F}, \Gamma, Some D) \wedge \neg (\exists C | \in | N_G | \cup | U_{er}. C \subseteq \# D)$ 
proof –
  have  $\exists C \gamma. state\text{-conflict } (state\text{-of}\text{-gstate } s) = Some (C, \gamma) \wedge \neg (\exists D | \in | cls\text{-of}\text{-gcls } |^q N_G | \cup | state\text{-learned } (state\text{-of}\text{-gstate } s). subsumes D C)$ 
proof (rule scl-fol.static-non-subsumption-projectable-strategy[ of ord-res-11 NG - - - state-of-gstate , OF run step])
  fix  $S S'$ 
  assume run: (ord-res-11 NG)** ({||}, {||}, [], None) S and step: ord-res-11 NG S S'
show scl-fol.regular-scl (cls-of-gcls |q NG) (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  $A_G | \in | atms\text{-of}\text{-class } N_G$ 
  show  $A_G \preceq_t \beta$ 

```

```

proof (cases atms-of-cls  $N_G = \{\|\}$ )
  case True
  thus ?thesis
    using ⟨ $A_G \mid \in \mid$  atms-of-cls  $N_G$ ⟩
    by simp
next
  case False
  then show ?thesis
    using ⟨ $A_G \mid \in \mid$  atms-of-cls  $N_G$ ⟩
    unfolding  $\beta$ -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  $U_G \mathcal{F} \Gamma D$  where  $s = (U_G, \mathcal{F}, \Gamma, \text{Some } D)$ 
proof atomize-elim
  obtain  $U_G \mathcal{F} \Gamma \mathcal{C}$  where  $s = (U_G, \mathcal{F}, \Gamma, \mathcal{C})$ 
    by (metis prod.exhaust)

  moreover obtain  $D$  where  $\mathcal{C} = \text{Some } D$ 
    using step
    by (auto simp: ⟨ $s = (U_G, \mathcal{F}, \Gamma, \mathcal{C})$ ⟩ elim: scl-fol.backtrack.cases)

  ultimately show  $\exists U_{er} \mathcal{F} \Gamma D. s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D)$ 
    by metis
qed

ultimately have  $\neg (\exists C \mid \in \mid N_G \mid \cup \mid U_G. \text{subsumes } (\text{cls-of-gcls } C :: ('f, 'v) \text{ term clause}) (\text{cls-of-gcls } D))$ 
  by auto

hence  $\neg (\exists C \mid \in \mid N_G \mid \cup \mid U_G. (\text{cls-of-gcls } C :: ('f, 'v) \text{ term clause}) \subseteq\# (\text{cls-of-gcls } D))$ 
  by (simp add: subsumes-def)

hence  $\neg (\exists C \mid \in \mid N_G \mid \cup \mid U_G. C \subseteq\# D)$ 
  by (metis cls-of-gcls-def image-mset-subseteq-mono)

```

thus *?thesis*
unfolding $\langle s = (U_G, \mathcal{F}, \Gamma, \text{Some } D) \rangle$ **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.