

SCL(FOL) Can Simulate Ground Nonredundant Ordered Resolution

Martin Desharnais

March 10, 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	6
1.2	Ground Layer	6
1.3	Correctness	6
1.4	Redundancy Criterion	7
1.5	Refutational Completeness	7
1.6	Move to <i>HOL.Transitive-Closure</i>	20
2	Move to <i>HOL-Library.Multiset</i>	23
3	Move to <i>HOL-Library.FSet</i>	23
4	Move to <i>VeriComp.Simulation</i>	24
5	Move to <i>Simple-Clause-Learning.SCL-FOL</i>	25
6	Move to ground ordered resolution	27
7	Move somewhere?	28

8	Ground ordered resolution for ground terms	33
9	Common definitions and lemmas	33
10	Lemmas about going between ground and first-order terms	38
11	SCL(FOL) for first-order terms	39
12	ORD-RES	39
13	ORD-RES-1 (deterministic)	40
14	Function for full factorization	42
15	ORD-RES-2 (full factorization)	44
16	Function for full resolution	48
17	ORD-RES-3 (full resolve)	53
18	Function for implicit full factorization	54
19	ORD-RES-4 (implicit factorization)	57
20	ORD-RES-5 (explicit model construction)	59
21	ORD-RES-6 (model backjump)	64
22	ORD-RES-7 (clause-guided literal trail construction)	67
23	ORD-RES-8 (atom-guided literal trail construction)	74
24	ORD-RES-9 (factorize when propagating)	78
25	ORD-RES-10 (propagate iff a conflict is produced)	80
26	ORD-RES-11 (SCL strategy)	82
27	ORD-RES-1 (deterministic)	86
28	ORD-RES-2 (full factorization)	86
29	ORD-RES-3 (full resolve)	88
30	ORD-RES-4 (implicit factorization)	90
31	ORD-RES-5 (explicit model construction)	90

32 ORD-RES-6 (model backjump)	91
33 ORD-RES-7 (clause-guided literal trail construction)	92
34 ORD-RES-8 (atom-guided literal trail construction)	93
35 ORD-RES-9 (factorize when propagating)	95
36 ORD-RES-10 (propagate iff a conflict is produced)	96
37 ORD-RES-11 (SCL strategy)	97
38 ORD-RES-11 is a regular SCL strategy	98
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</i> where	
<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-ordered-resolution-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

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

<proof>

abbreviation *lesseq-trm* (**infix** \preceq_t 50) **where**

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

lemma *lesseq-trm-if-subtermeq: t* $\preceq_t ctxt\langle t \rangle_G$

<proof>

definition *less-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-lit* (**infix** \preceq_l 50) **where**

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

abbreviation *less-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-cls* (**infix** \preceq_c 50) **where**

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

lemma *transp-on-less-lit[simp]: transp-on A* (\prec_l)

<proof>

corollary *transp-less-lit: transp* (\prec_l)

<proof>

lemma *transp-less-cls[simp]*: *transp* (\prec_c)
<proof>

lemma *asympt-on-less-lit[simp]*: *asympt-on* A (\prec_l)
<proof>

corollary *asympt-less-lit[simp]*: *asympt* (\prec_l)
<proof>

lemma *asympt-less-cls[simp]*: *asympt* (\prec_c)
<proof>

lemma *irreflp-on-less-lit[simp]*: *irreflp-on* A (\prec_l)
<proof>

lemma *wfP-less-lit[simp]*: *wfP* (\prec_l)
<proof>

lemma *wfP-less-cls[simp]*: *wfP* (\prec_c)
<proof>

lemma *totalp-on-less-lit[simp]*: *totalp-on* A (\prec_l)
<proof>

corollary *totalp-less-lit*: *totalp* (\prec_l)
<proof>

lemma *totalp-less-cls[simp]*: *totalp* (\prec_c)
<proof>

interpretation *term-order*: *linorder lesseq-trm less-trm*
<proof>

interpretation *literal-order*: *linorder lesseq-lit less-lit*
<proof>

interpretation *clause-order*: *linorder lesseq-cls less-cls*
<proof>

lemma *less-lit-simps[simp]*:

Pos $A_1 \prec_l \text{Pos } A_2 \iff A_1 \prec_t A_2$

Pos $A_1 \prec_l \text{Neg } A_2 \iff A_1 \preceq_t A_2$

Neg $A_1 \prec_l \text{Neg } A_2 \iff A_1 \prec_t A_2$

Neg $A_1 \prec_l \text{Pos } A_2 \iff A_1 \prec_t A_2$

<proof>

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* (\prec_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* (\prec_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:

$P_1 = \text{add-mset } (\text{Neg } t) P_1' \Longrightarrow$

$P_2 = \text{add-mset } (\text{Pos } t) P_2' \Longrightarrow$

$P_2 \prec_c P_1 \Longrightarrow$

select $P_1 = \{\#\} \wedge$ *is-maximal-lit* ($\text{Neg } t$) $P_1 \vee \text{Neg } t \in \#$ *select* $P_1 \Longrightarrow$

select $P_2 = \{\#\} \Longrightarrow$

is-strictly-maximal-lit ($\text{Pos } t$) $P_2 \Longrightarrow$

$C = P_1' + P_2' \Longrightarrow$

ground-resolution $P_1 P_2 C$

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

ground-factoringI:

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

select $P = \{\#\} \Longrightarrow$

is-maximal-lit ($\text{Pos } t$) $P \Longrightarrow$

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

ground-factoring $P C$

1.2 Ground Layer

definition *G-Inf* :: 'f gterm atom clause inference set **where**

G-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-Bot* :: 'f gterm atom clause set **where**

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

definition *G-entails* :: 'f gterm atom clause set \Rightarrow 'f gterm atom clause set \Rightarrow bool **where**

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

1.3 Correctness

lemma *soundness-ground-resolution*:

assumes

step: *ground-resolution* $P_1 P_2 C$

shows G -entails $\{P1, P2\} \{C\}$
<proof>

lemma *soundness-ground-factoring:*
assumes *step: ground-factoring* $P C$
shows G -entails $\{P\} \{C\}$
<proof>

interpretation G : *sound-inference-system* G -Inf G -Bot G -entails
<proof>

1.4 Redundancy Criterion

lemma *ground-resolution-smaller-conclusion:*
assumes
 step: ground-resolution $P1 P2 C$
shows $C \prec_c P1$
<proof>

lemma *ground-factoring-smaller-conclusion:*
assumes *step: ground-factoring* $P C$
shows $C \prec_c P$
<proof>

interpretation G : *calculus-with-finitary-standard-redundancy* G -Inf G -Bot G -entails
 (\prec_c)
<proof>

1.5 Refutational Completeness

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

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

termination *production*
<proof>

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

end

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

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

$\langle \text{proof} \rangle$

lemma *production-eq-empty-or-singleton:*

$\text{production } N \ C = \{\} \vee (\exists A. \text{production } N \ C = \{A\})$

$\langle \text{proof} \rangle$

lemma *production-eq-singleton-if-atom-in-production:*

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

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

$\langle \text{proof} \rangle$

definition *interp where*

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

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

$\langle \text{proof} \rangle$

lemma *production-unfold:* $\text{production } N \ C = \{A \mid A \ C'\}$

$C \in N \wedge$

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

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

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

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

$\langle \text{proof} \rangle$

lemma *production-unfold':* $\text{production } N \ C = \{A \mid A.$

$C \in N \wedge$

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

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

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

$\langle \text{proof} \rangle$

lemma *mem-productionE:*

assumes $C\text{-prod}: A \in \text{production } N \ C$

obtains C' **where**

$C \in N$ **and**

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

$\text{select } C = \{\#\}$ **and**

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

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

$\langle \text{proof} \rangle$

lemma *production-subset-if-less-clc:* $C \prec_c D \implies \text{production } N \ C \subseteq \text{interp } N \ D$

$\langle \text{proof} \rangle$

lemma *Uniq-production-eq-singleton:* $\exists_{\leq 1} C. \text{production } N \ C = \{A\}$

$\langle \text{proof} \rangle$

lemma *singleton-eq-CollectD*: $\{x\} = \{y. P y\} \implies P x$
 ⟨proof⟩

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

lemma *interp-subset-if-less-cls*: $C \prec_c D \implies \text{interp } N C \subseteq \text{interp } N D$
 ⟨proof⟩

lemma *interp-subset-if-less-cls'*: $C \prec_c D \implies \text{interp } N C \subseteq \text{interp } N D \cup \text{production } N D$
 ⟨proof⟩

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⟩

lemma *split-Union-production'*:

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

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

⟨proof⟩

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⟩

lemma *less-imp-Interp-subseteq-interp*: $C \prec_c D \implies \text{interp } N C \cup \text{production } N C \subseteq \text{interp } N D$

⟨proof⟩

lemma *not-interp-to-Interp-imp-le*: $A \notin \text{interp } N C \implies A \in \text{interp } N D \cup \text{production } N D \implies C \preceq_c D$

⟨proof⟩

lemma *produces-imp-in-interp*:

assumes *Neg A* $A \in \# C$ **and** *D-prod*: $A \in \text{production } N D$

shows $A \in \text{interp } N C$

⟨proof⟩

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

$\text{Neg } A \in \# C \implies A \notin \text{interp } N D \cup \text{production } N D \implies C \preceq_c D \implies A \notin \text{production } N D''$

⟨proof⟩

lemma *lift-interp-entails*:

assumes

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

D-entailed: $\text{interp } N D \models D$ **and**

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

D-lt-C: $D \prec_c C$

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

<proof>

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>

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

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

<proof>

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-trm-if-neg: $\text{is-neg } L \implies \text{atm-of } L \prec_t A$

<proof>

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

assumes *C-prod*: $A_C \in \text{production } N C$ **and** *D-prod*: $A_D \in \text{production } N D$

shows $A_C \prec_t A_D \iff C \prec_c D$

<proof>

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 $\neg \text{interp } N D \models C - \{\#Pos A\}$

<proof>

lemma *production-subset-Union-production:*

$\bigwedge C N. C \in N \implies \text{production } N C \subseteq (\bigcup D \in N. \text{production } N D)$
 ⟨proof⟩

lemma *interp-subset-Union-production:*

$\bigwedge C N. C \in N \implies \text{interp } N C \subseteq (\bigcup D \in N. \text{production } N D)$
 ⟨proof⟩

lemma *model-construction:*

fixes

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

$C :: 'f \text{ gterm atom clause}$

assumes $G.\text{saturated } N$ **and** $\{\#\} \notin N$ **and** $C\text{-in: } C \in N$

shows

$\text{production } N C = \{\}$ \longleftrightarrow $\text{interp } N C \models C$

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

$D \in N \implies C \prec_c D \implies \text{interp } N D \models C$

⟨proof⟩

lemma

assumes $\text{clause-order.is-least-in-fset } N C$ **and** $A \in \text{production } (\text{fset } N) C$

shows $\bigwedge A'. A' \prec_t A \implies A' \notin (\bigcup D \in \text{fset } N. \text{production } (\text{fset } N) D)$

⟨proof⟩

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-productive:*

assumes $A \in \text{production } (\text{fset } N) C$

shows $\bigwedge A'. A' \prec_t A \implies A' \notin \text{interp } (\text{fset } N) C \implies A' \notin (\bigcup D \in \text{fset } N. \text{production } (\text{fset } N) D)$

⟨proof⟩

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp-if-not-productive:*

assumes $\text{literal-order.is-maximal-in-mset } C L$ **and** $\text{production } (\text{fset } N) C = \{\}$

shows $\bigwedge A'. A' \prec_t \text{atm-of } L \implies A' \notin \text{interp } (\text{fset } N) C \implies A' \notin (\bigcup D \in \text{fset } N. \text{production } (\text{fset } N) D)$

⟨proof⟩

lemma *lesser-atoms-not-in-previous-interp-are-not-in-final-interp:*

fixes A

assumes

$L\text{-max: literal-order.is-maximal-in-mset } C L$ **and**

$A\text{-less: } A \prec_t \text{atm-of } L$ **and**

$A\text{-no-in: } A \notin \text{interp } (\text{fset } N) C$

shows $A \notin (\bigcup D \in \text{fset } N. \text{production } (\text{fset } N) D)$

⟨proof⟩

lemma *lesser-atoms-in-previous-interp-are-in-final-interp:*

fixes A

assumes

$L\text{-max: literal-order.is-maximal-in-mset } C L$ **and**

A-less: $A \prec_t \text{atm-of } L$ **and**
A-in: $A \in \text{interp } N C$
shows $A \in (\bigcup D \in N. \text{production } N D)$
 ⟨*proof*⟩

lemma *interp-fixed-for-smaller-literals*:

fixes A
assumes
L-max: *literal-order.is-maximal-in-mset* $C L$ **and**
A-less: $A \prec_t \text{atm-of } L$ **and**
 $C \prec_c D$
shows $A \in \text{interp } N C \longleftrightarrow A \in \text{interp } N D$
 ⟨*proof*⟩

lemma *neg-lits-not-in-model-stay-out-of-model*:

assumes
L-in: $L \in \# C$ **and**
L-neg: *is-neg* L **and**
atm-L-not-in: $\text{atm-of } L \notin \text{interp } N C$
shows $\text{atm-of } L \notin (\bigcup D \in N. \text{production } N D)$
 ⟨*proof*⟩

lemma *neg-lits-already-in-model-stay-in-model*:

assumes
L-in: $L \in \# C$ **and**
L-neg: *is-neg* L **and**
atm-L-not-in: $\text{atm-of } L \in \text{interp } N C$
shows $\text{atm-of } L \in (\bigcup D \in N. \text{production } N D)$
 ⟨*proof*⟩

lemma *image-eq-imageI*:

assumes $\bigwedge x. x \in X \implies f x = g x$
shows $f ' X = g ' X$
 ⟨*proof*⟩

lemma *production-swap-clause-set*:

assumes
agree: $\{D \in N1. D \preceq_c C\} = \{D \in N2. D \preceq_c C\}$
shows $\text{production } N1 C = \text{production } N2 C$
 ⟨*proof*⟩

lemma *interp-swap-clause-set*:

assumes *agree*: $\{D \in N1. D \prec_c C\} = \{D \in N2. D \prec_c C\}$
shows $\text{interp } N1 C = \text{interp } N2 C$
 ⟨*proof*⟩

definition *interp'* **where**

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

lemma *interp-eq-interp'*: $\text{interp } N \ D = \text{interp}' \{C \in N. C \prec_c D\}$
 ⟨proof⟩

lemma *production-unfold''*: $\text{production } N \ C = \{A \mid A.$
 $C \in N \wedge \text{select } C = \{\#\} \wedge$
 $\text{is-strictly-maximal-lit } (\text{Pos } A) \ C \wedge$
 $\neg \text{interp}' \{B \in N. B \prec_c C\} \models C\}$
 ⟨proof⟩

lemma *Interp-swap-clause-set*:
assumes *agree*: $\{D \in N1. D \preceq_c C\} = \{D \in N2. D \preceq_c C\}$
shows $\text{interp } N1 \ C \cup \text{production } N1 \ C = \text{interp } N2 \ C \cup \text{production } N2 \ C$
 ⟨proof⟩

lemma *production-insert-greater-clause*:
assumes $C \prec_c D$
shows $\text{production } (\text{insert } D \ N) \ C = \text{production } N \ C$
 ⟨proof⟩

lemma *interp-insert-greater-clause-strong*:
assumes $C \preceq_c D$
shows $\text{interp } (\text{insert } D \ N) \ C = \text{interp } N \ C$
 ⟨proof⟩

lemma *interp-insert-greater-clause*:
assumes $C \prec_c D$
shows $\text{interp } (\text{insert } D \ N) \ C = \text{interp } N \ C$
 ⟨proof⟩

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⟩

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⟩

lemma *production-add-irrelevant-clause-to-set*:
assumes
fin: finite N **and**
C-in: $C \in N$ **and**
D-irrelevant: $\exists E \in N. E \subset \# \ D \wedge \text{set-mset } D = \text{set-mset } E$ **and**
no-select: $\bigwedge C. \text{select } C = \{\#\}$

shows $\text{production } (\text{insert } D \ N) \ C = \text{production } N \ C$
 ⟨proof⟩

lemma *production-add-irrelevant-clauses-to-set0*:

assumes

fin: $\text{finite } N \ \text{finite } N'$ **and**

D-in: $D \in N'$ **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 $\text{production } (N \cup N') \ D = \{\}$

⟨proof⟩

lemma *production-add-irrelevant-clauses-to-set*:

assumes

fin: $\text{finite } N \ \text{finite } N'$ **and**

C-in: $C \in N$ **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 $\text{production } (N \cup N') \ C = \text{production } N \ C$

⟨proof⟩

lemma *interp-add-irrelevant-clauses-to-set*:

assumes

fin: $\text{finite } N \ \text{finite } N'$ **and**

C-in: $C \in N$ **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 $\text{interp } (N \cup N') \ C = \text{interp } N \ C$

⟨proof⟩

lemma *interp-add-irrelevant-clauses-to-set'*:

assumes

fin: $\text{finite } N \ \text{finite } N'$ **and**

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

irrelevant: $\forall D \in N'. \exists E \in N. E \subseteq\# D \wedge \text{set-mset } D = \text{set-mset } E$ **and**

no-select: $\bigwedge C. \text{select } C = \{\#\}$

shows $\text{interp } (N \cup N') \ C = \text{interp } N \ C$

⟨proof⟩

lemma *lesser-entailed-clause-stays-entailed'*:

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

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

⟨proof⟩

lemma *lesser-entailed-clause-stays-entailed*:

assumes *C-le*: $C \preceq_c D$ **and** *D-lt*: $D \prec_c E$ **and** *C-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>

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

<proof>

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

<proof>

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

<proof>

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>

lemma

fixes

$P1 P2 :: 'f \text{ gterm and}$

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

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

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>

lemma

fixes

```

P1 P2 P3 P4 :: 'f gterm and
C1 C2 C3 C4 C5 :: 'f gterm clause and
N :: 'f gterm clause set
defines
C1 ≡ {#Neg P1, Neg P2#} and
C2 ≡ {#Pos P2, Neg P3#} and
C3 ≡ {#Pos P1, Pos P2, Pos P4#} and
C4 ≡ {#Pos P2, Pos P3, Pos P4#} and
C5 ≡ {#Pos P2, Neg P4#} and
N ≡ {C1, C2, C3, C4, C5}
assumes
term-order: P1 <t P2 P2 <t P3 P3 <t P4 and
no-select:  $\bigwedge C. \text{select } C = \{\#\}$ 
shows
C1 <c C2 C2 <c C3 C3 <c C4 C4 <c C5
⟨proof⟩

interpretation G: statically-complete-calculus G-Bot G-Inf G-entails G.Red-I G.Red-F
⟨proof⟩

end

end
theory Lower-Set
imports Main
begin

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

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

lemma is-lower-set-wrt-empty:
fixes X :: 'a set and R :: 'a ⇒ 'a ⇒ bool
assumes transp-on X R and asymp-on X R
shows is-lower-set-wrt R {} X
⟨proof⟩

lemma is-lower-set-wrt-refl:
fixes X :: 'a set and R :: 'a ⇒ 'a ⇒ bool
assumes transp-on X R and asymp-on X R
shows is-lower-set-wrt R X X
⟨proof⟩

```


lemma *is-lower-set-wrt-trans*:

fixes $X Y Z :: 'a \text{ set}$ **and** $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

assumes

transp-on $Z R$ **and** *asypm-on* $Z R$ **and**

is-lower-set-wrt $R X Y$ **and** *is-lower-set-wrt* $R Y Z$

shows *is-lower-set-wrt* $R X Z$

<proof>

lemma *is-lower-set-wrt-antisym*:

fixes $X Y :: 'a \text{ set}$ **and** $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$

assumes

transp-on $Y R$ **and** *asypm-on* $Y R$ **and**

is-lower-set-wrt $R X Y$ **and** *is-lower-set-wrt* $R Y X$

shows $X = Y$

<proof>

lemma *order-is-lower-set-wrt*:

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

assumes *transp* R **and** *asypm* R

shows *class.order* (*is-lower-set-wrt* R) (*is-strict-lower-set-wrt* R)

<proof>

lemma *is-lower-set-wrt-insertI*:

assumes *transp-on* (*insert* $x X$) R **and** *asypm-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>

lemma *lower-set-wrt-appendI*:

assumes

trans: *transp-on* (*set* ($xs @ ys$)) R **and**

asym: *asypm-on* (*set* ($xs @ ys$)) R **and**

sorted: *sorted-wrt* R ($xs @ ys$)

shows *is-lower-set-wrt* R (*set* xs) (*set* ($xs @ ys$))

<proof>

lemma *sorted-and-lower-set-wrt-appendD-left*:

assumes *transp-on* $A R$ **and** *asypm-on* $A R$ **and**

sorted-wrt R ($xs @ ys$) **and** *is-lower-set-wrt* R (*set* ($xs @ ys$)) A

shows *sorted-wrt* R xs **and** *is-lower-set-wrt* R (*set* xs) A

<proof>

lemma *sorted-and-lower-set-wrt-appendD-right*:

assumes *transp-on* $A R$ **and** *asypm-on* $A R$ **and**

sorted-wrt $(\lambda x y. R y x)$ ($xs @ ys$) **and** *is-lower-set-wrt* R (*set* ($xs @ ys$)) A

shows *sorted-wrt* $(\lambda x y. R y x)$ ys **and** *is-lower-set-wrt* R (*set* ys) A

<proof>

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

```

fixes  $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
assumes  $\text{trans: trans-on } Y \ R$  and  $\text{asym: asymp-on } Y \ R$ 
shows  $\text{is-lower-set-wrt } R \ X \ Y \Longrightarrow y \notin X \Longrightarrow y \in Y \Longrightarrow R \ y \ z \Longrightarrow z \notin X$ 
<proof>

abbreviation (in preorder)  $\text{is-lower-set where}$ 
 $\text{is-lower-set} \equiv \text{is-lower-set-wrt } (<)$ 

lemmas (in preorder)  $\text{is-lower-set-iff} =$ 
 $\text{is-lower-set-wrt-def}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

context  $\text{linorder begin}$ 

sublocale  $\text{is-lower-set: order is-lower-set-wrt } (<) \ \text{is-strict-lower-set-wrt } (<)$ 
<proof>

end

lemmas (in preorder)  $\text{is-lower-set-empty}[simp] =$ 
 $\text{is-lower-set-wrt-empty}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

lemmas (in preorder)  $\text{is-lower-set-insertI} =$ 
 $\text{is-lower-set-wrt-insertI}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

lemmas (in preorder)  $\text{lower-set-appendI} =$ 
 $\text{lower-set-wrt-appendI}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

lemmas (in preorder)  $\text{sorted-and-lower-set-appendD-left} =$ 
 $\text{sorted-and-lower-set-wrt-appendD-left}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

lemmas (in preorder)  $\text{sorted-and-lower-set-appendD-right} =$ 
 $\text{sorted-and-lower-set-wrt-appendD-right}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

lemmas (in preorder)  $\text{not-in-lower-setI} =$ 
 $\text{not-in-lower-set-wrtI}[OF \ \text{trans-on-less} \ \text{asymp-on-less}]$ 

end
theory  $\text{HOL-Extra-Extra}$ 
imports  $\text{First-Order-Clause.HOL-Extra}$ 
begin

no-notation  $\text{restrict-map (infixl } |' \ 110)$ 

lemma
assumes  $\exists_{\leq 1} x. P \ x$ 
shows  $\text{finite } \{x. P \ x\}$ 
<proof>

```

lemma *finite-if-Uniq-Uniq*:

assumes

$\exists_{\leq 1} x. P x$

$\forall x. \exists_{\leq 1} y. Q x y$

shows $finite \{y. \exists x. P x \wedge Q x y\}$

<proof>

lemma *finite-if-finite-finite*:

assumes

$finite \{x. P x\}$

$\forall x. finite \{y. Q x y\}$

shows $finite \{y. \exists x. P x \wedge Q x y\}$

<proof>

lemma (in *order*) *greater-wfp-on-finite-set*: $finite \mathcal{X} \implies Wellfounded.wfp-on \mathcal{X}$

($>$)

<proof>

lemma (in *order*) *less-wfp-on-finite-set*: $finite \mathcal{X} \implies Wellfounded.wfp-on \mathcal{X}$ ($<$)

<proof>

lemma *distinct-if-sorted-wrt-asymp*:

assumes *asymp-on* (*set xs*) *R* **and** *sorted-wrt* *R xs*

shows *distinct xs*

<proof>

lemma *dropWhile-append-eq-rhs*:

fixes *xs ys* :: 'a list **and** *P* :: 'a \implies bool

assumes

$\bigwedge x. x \in set\ xs \implies P x$ **and**

$\bigwedge y. y \in set\ ys \implies \neg P y$

shows $dropWhile\ P\ (xs\ @\ ys) = ys$

<proof>

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

fixes *R* :: 'a \implies 'a \implies bool **and** *xs* :: 'a list **and** *P* :: 'a \implies bool

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

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

<proof>

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

fixes *R* :: 'a \implies 'a \implies bool **and** *xs* :: 'a list **and** *P* :: 'a \implies bool

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

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

<proof>

lemma *filter-set-eq-filter-set-minus-singleton*:

assumes $\neg P y$

shows $\{x \in X. P x\} = \{x \in X - \{y\}. P x\}$
 ⟨proof⟩

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 $\exists! X'. X' \subseteq X \wedge Y' = f ` X'$
 ⟨proof⟩

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 $\{y. P y\} = f ` \{x. x \in X \wedge P (f x)\}$
 ⟨proof⟩

lemma *restrict-map-ident-if-dom-subset*: $\text{dom } \mathcal{M} \subseteq A \Longrightarrow \text{restrict-map } \mathcal{M} A = \mathcal{M}$
 ⟨proof⟩

1.6 Move to HOL.Transitive-Closure

lemma *relpowp-right-unique*:
fixes $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$ **and** $n :: \text{nat}$ **and** $x y z :: 'a$
assumes *runiq*: $\bigwedge x y z. R x y \Longrightarrow R x z \Longrightarrow y = z$
shows $(R \overset{\sim}{\sim} n) x y \Longrightarrow (R \overset{\sim}{\sim} n) x z \Longrightarrow y = z$
 ⟨proof⟩

lemma *Uniq-relpowp*:
fixes $n :: \text{nat}$ **and** $R :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
assumes *runiq*: $\forall x. \exists_{\leq 1} y. R x y$
shows $\exists_{\leq 1} y. (R \overset{\sim}{\sim} n) x y$
 ⟨proof⟩

lemma *relpowp-plus-of-right-unique*:
assumes
 right-unique R
 $(R \overset{\sim}{\sim} m) x y$ **and**
 $(R \overset{\sim}{\sim} (m + n)) x z$
shows $(R \overset{\sim}{\sim} n) y z$
 ⟨proof⟩

lemma *relpowp-plusD*:
assumes $(R \overset{\sim}{\sim} (m + n)) x z$
shows $\exists y. (R \overset{\sim}{\sim} m) x y \wedge (R \overset{\sim}{\sim} n) y z$
 ⟨proof⟩

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

$R\ x\ y$ **and**
 $(R \overset{\sim}{\sim} \text{Suc } n)\ x\ z$
shows $(R \overset{\sim}{\sim} n)\ y\ z$
 $\langle \text{proof} \rangle$

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

lemma *transp-on-singleton[simp]*: *transp-on* $\{x\}\ R$
 $\langle \text{proof} \rangle$

lemma *rtranclp-rtranclp-compose-if-right-unique*:
assumes *runique*: *right-unique* R **and** $R^{**}\ a\ b$ **and** $R^{**}\ a\ c$
shows $R^{**}\ a\ b \wedge R^{**}\ b\ c \vee R^{**}\ a\ c \wedge R^{**}\ c\ b$
 $\langle \text{proof} \rangle$

lemma *right-unique-terminating-rtranclp*:
assumes *right-unique* R
shows *right-unique* $(\lambda x\ y.\ R^{**}\ x\ y \wedge (\nexists z.\ R\ y\ z))$
 $\langle \text{proof} \rangle$

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* $P = \text{Some } x \implies P\ x$
 $\langle \text{proof} \rangle$

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

lemma *The-optional-eq-NoneD*: *The-optional* $P = \text{None} \implies \nexists!x.\ P\ x$
 $\langle \text{proof} \rangle$

lemma *None-eq-The-optionalD*: *None* = *The-optional* $P \implies \nexists!x.\ P\ x$
 $\langle \text{proof} \rangle$

lemma *The-optional-eq-SomeI*:
assumes $\exists_{\leq 1}x.\ P\ x$ **and** $P\ x$
shows *The-optional* $P = \text{Some } x$
 $\langle \text{proof} \rangle$

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 $\exists_{\leq 1} y. \text{full-run } R x y$
 $\langle \text{proof} \rangle$

lemma *ex1-full-run*:
assumes *Uniq-R*: $\bigwedge x. \exists_{\leq 1} y. R x y$ **and** *wfP-R*: $\text{wfP } R^{-1-1}$
shows $\exists! y. \text{full-run } R x y$
 $\langle \text{proof} \rangle$

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 $P y$
 $\langle \text{proof} \rangle$

end

theory *Background*

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

begin

lemma $I \models l L \longleftrightarrow (\text{is-pos } L \longleftrightarrow \text{atm-of } L \in I)$
 $\langle \text{proof} \rangle$

2 Move to *HOL-Library.Multiset*

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

hide-fact *subset-implies-multp*

lemma *subset-implies-reflclp-multp*: $A \subseteq\# B \implies (\text{multp } R)^{==} A B$
 ⟨*proof*⟩

lemma *member-mset-repeat-msetD*: $L \in\# \text{repeat-mset } n M \implies L \in\# M$
 ⟨*proof*⟩

lemma *member-mset-repeat-mset-Suc[simp]*: $L \in\# \text{repeat-mset } (\text{Suc } n) M \longleftrightarrow L \in\# M$
 ⟨*proof*⟩

lemma *image-msetI*: $x \in\# M \implies f x \in\# \text{image-mset } f M$
 ⟨*proof*⟩

lemma *inj-image-mset-mem-iff*: $\text{inj } f \implies f x \in\# \text{image-mset } f M \longleftrightarrow x \in\# M$
 ⟨*proof*⟩

3 Move to *HOL-Library.FSet*

declare *wfP-pfsubset[intro]*

syntax

-FFilter :: *pttrn* \Rightarrow 'a *fset* \Rightarrow *bool* \Rightarrow 'a *fset* ((1{|- | \in | -/ -|}))

translations

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

lemma *fimage-ffUnion*: $f \mid\uparrow \text{ffUnion } SS = \text{ffUnion } ((\mid\uparrow) f \mid\uparrow SS)$
 ⟨*proof*⟩

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

assumes $\neg P y$

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

⟨*proof*⟩

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

⟨*proof*⟩

lemma *ffilter-fempty[simp]*: $\text{ffilter } P \{\mid\} = \{\mid\}$

⟨*proof*⟩

lemma *fstrict-subset-iff-fset-strict-subset-fset*:

fixes $\mathcal{X} \ \mathcal{Y} :: \text{-fset}$

shows $\mathcal{X} \mid\subset\} \mathcal{Y} \longleftrightarrow \text{fset } \mathcal{X} \subset \text{fset } \mathcal{Y}$

<proof>

lemma (in *linorder*) *ex1-sorted-list-for-fset*:
 $\exists !xs. \text{sorted-wrt } (<) \text{ } xs \wedge \text{fset-of-list } xs = X$
<proof>

lemma (in *linorder*) *is-least-in-fset-ffilterD*:
assumes *is-least-in-fset-wrt* (<) (*ffilter P X*) *x*
shows $x \in X \wedge P x$
<proof>

4 Move to *VeriComp.Simulation*

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

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 ⇒ 'state2 ⇒ bool and
    final1 :: 'state1 ⇒ bool and
    final2 :: 'state2 ⇒ bool +
fixes
    match :: 'state1 ⇒ 'state2 ⇒ bool and
    measure :: 'state2 ⇒ 'index and
    order :: 'index ⇒ 'index ⇒ bool (infix □ 70)
assumes
    wfp-order:
    wfp (□) and
    match-final:
    match s1 s2 ⇒⇒ final2 s2 ⇒⇒ final1 s1 and
    simulation:
    match s1 s2 ⇒⇒ step2 s2 s2' ⇒⇒
    (∃ s1'. step1++ s1 s1' ∧ match s1' s2') ∨ (match s1 s2' ∧ measure s2' □
    measure s2)
begin

sublocale backward-simulation where
    step1 = step1 and step2 = step2 and final1 = final1 and final2 = final2 and
    order = order and
    match = λi x y. i = measure y ∧ match x y
    ⟨proof⟩

end

```

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

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

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

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

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

lemma *trail-false-cls-empty[simp]*: *trail-false-cls* Γ {#}
 ⟨proof⟩

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

lemma *trail-defined-lit-iff*: *trail-defined-lit* Γ L ↔ atm-of L ∈ atm-of ' fst ' set
 Γ
 ⟨proof⟩

definition *trail-defined-cls* :: (*- literal* × *- option*) *list* ⇒ *- clause* ⇒ *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 $\Gamma L \longleftrightarrow \text{trail-true-lit } \Gamma L \vee \text{trail-false-lit } \Gamma L$
 ⟨*proof*⟩

lemma *trail-true-or-false-cls-if-defined*:
trail-defined-cls $\Gamma C \implies \text{trail-true-cls } \Gamma C \vee \text{trail-false-cls } \Gamma C$
 ⟨*proof*⟩

lemma *subtrail-falseI*:
assumes *tr-false*: *trail-false-cls* ((*L*, *Cl*) # Γ) *C* **and** *L-not-in*: $-L \notin \# C$
shows *trail-false-cls* ΓC
 ⟨*proof*⟩

inductive *trail-consistent* :: (*'a literal* × *'b option*) *list* ⇒ *bool* **where**
Nil[simp]: *trail-consistent* [] |
Cons: $\neg \text{trail-defined-lit } \Gamma L \implies \text{trail-consistent } \Gamma \implies \text{trail-consistent } ((L, u) \# \Gamma)$

lemma *distinct-lits-if-trail-consistent*:
trail-consistent $\Gamma \implies \text{distinct } (\text{map } \text{fst } \Gamma)$
 ⟨*proof*⟩

lemma *trail-true-lit-if-trail-true-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-true-lit } \Gamma' K \implies \text{trail-true-lit } \Gamma K$
 ⟨*proof*⟩

lemma *trail-true-cls-if-trail-true-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-true-cls } \Gamma' C \implies \text{trail-true-cls } \Gamma C$
 ⟨*proof*⟩

lemma *trail-false-lit-if-trail-false-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-false-lit } \Gamma' K \implies \text{trail-false-lit } \Gamma K$
 ⟨*proof*⟩

lemma *trail-false-cls-if-trail-false-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-false-cls } \Gamma' C \implies \text{trail-false-cls } \Gamma C$
 ⟨*proof*⟩

lemma *trail-defined-lit-if-trail-defined-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-defined-lit } \Gamma' K \implies \text{trail-defined-lit } \Gamma K$
 ⟨*proof*⟩

lemma *trail-defined-cls-if-trail-defined-suffix*:
suffix $\Gamma' \Gamma \implies \text{trail-defined-cls } \Gamma' C \implies \text{trail-defined-cls } \Gamma C$
 ⟨*proof*⟩

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 $\text{trail-consistent } \Gamma \implies \neg (\text{trail-true-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma L)$
 $\langle \text{proof} \rangle$

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 $\text{trail-consistent } \Gamma \implies \neg (\text{trail-true-cls } \Gamma C \wedge \text{trail-false-cls } \Gamma C)$
 $\langle \text{proof} \rangle$

lemma *not-lit-and-comp-lit-false-if-trail-consistent*:
assumes $\text{trail-consistent } \Gamma$
shows $\neg (\text{trail-false-lit } \Gamma L \wedge \text{trail-false-lit } \Gamma (\neg L))$
 $\langle \text{proof} \rangle$

lemma *not-both-lit-and-comp-lit-in-false-clause-if-trail-consistent*:
assumes $\Gamma\text{-consistent}$: $\text{trail-consistent } \Gamma$ **and** $C\text{-false}$: $\text{trail-false-cls } \Gamma C$
shows $\neg (L \in \# C \wedge \neg L \in \# C)$
 $\langle \text{proof} \rangle$

6 Move to ground ordered resolution

lemma (*in ground-ordered-resolution-calculus*) *unique-ground-resolution*:
shows $\exists_{\leq 1} C. \text{ground-resolution } P1 P2 C$
 $\langle \text{proof} \rangle$

lemma (*in ground-ordered-resolution-calculus*) *unique-ground-factoring*:
shows $\exists_{\leq 1} C. \text{ground-factoring } P C$
 $\langle \text{proof} \rangle$

lemma (*in ground-ordered-resolution-calculus*) *termination-ground-factoring*:
shows $wfP \text{ground-factoring}^{-1-1}$
 $\langle \text{proof} \rangle$

lemma (*in ground-ordered-resolution-calculus*) *atms-of-concl-subset-if-ground-resolution*:
assumes $\text{ground-resolution } P_1 P_2 C$
shows $\text{atms-of } C \subseteq \text{atms-of } P_1 \cup \text{atms-of } P_2$
 $\langle \text{proof} \rangle$

lemma (*in ground-ordered-resolution-calculus*) *strict-subset-mset-if-ground-factoring*:
assumes $\text{ground-factoring } P C$
shows $C \subset \# P$
 $\langle \text{proof} \rangle$

lemma (*in ground-ordered-resolution-calculus*) *set-mset-eq-set-mset-if-ground-factoring*:
assumes $\text{ground-factoring } P C$
shows $\text{set-mset } P = \text{set-mset } C$
 $\langle \text{proof} \rangle$

lemma (in *ground-ordered-resolution-calculus*) *atms-of-concl-eq-if-ground-factoring*:
assumes *ground-factoring* $P\ C$
shows $\text{atms-of } C = \text{atms-of } P$
<proof>

lemma (in *ground-ordered-resolution-calculus*) *ground-factoring-preserves-maximal-literal*:
assumes *ground-factoring* $P\ C$
shows $\text{is-maximal-lit } L\ P = \text{is-maximal-lit } L\ C$
<proof>

lemma (in *ground-ordered-resolution-calculus*) *ground-factorings-preserves-maximal-literal*:
assumes *ground-factoring*** $P\ C$
shows $\text{is-maximal-lit } L\ P = \text{is-maximal-lit } L\ C$
<proof>

lemma (in *ground-ordered-resolution-calculus*) *ground-factoring-reduces-maximal-pos-lit*:
assumes *ground-factoring* $P\ C$ **and** *is-pos* L **and**
is-maximal-lit $L\ P$ **and** $\text{count } P\ L = \text{Suc } (\text{Suc } n)$
shows *is-maximal-lit* $L\ C$ **and** $\text{count } C\ L = \text{Suc } n$
<proof>

lemma (in *ground-ordered-resolution-calculus*) *ground-factorings-reduces-maximal-pos-lit*:
assumes (*ground-factoring* $\widehat{\sim} m$) $P\ C$ **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** $\text{count } C\ L = \text{Suc } (\text{Suc } n - m)$
<proof>

lemma (in *ground-ordered-resolution-calculus*) *full-ground-factorings-reduces-maximal-pos-lit*:
assumes *steps*: (*ground-factoring* $\widehat{\sim} \text{Suc } n$) $P\ C$ **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** $\text{count } C\ L = \text{Suc } 0$
<proof>

7 Move somewhere?

lemma *true-cls-imp-neq-mempty*: $\mathcal{I} \models C \implies C \neq \{\#\}$
<proof>

lemma *lift-tranclp-to-pairs-with-constant-fst*:
 $(R\ x)^{++}\ y\ z \implies (\lambda(x, y)\ (x', z).\ x = x' \wedge R\ x\ y\ z)^{++}\ (x, y)\ (x, z)$
<proof>

abbreviation (in *preorder*) *is-lower-fset* **where**
is-lower-fset $X\ Y \equiv \text{is-lower-set-wrt } (<) (fset\ X)\ (fset\ Y)$

lemma *lower-set-wrt-prefixI*:
assumes
trans: *transp-on* (*set* zs) R **and**

asym: *asympt-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*⟩

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

lemma *lower-set-wrt-suffixI*:
assumes
trans: *transp-on* (set *zs*) *R* **and**
asym: *asympt-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*⟩

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

lemma *true-cls-repeat-mset-Suc*[*simp*]: $I \models \text{repeat-mset } (Suc\ n)\ C \longleftrightarrow I \models C$
 ⟨*proof*⟩

lemma (in *backward-simulation*)
assumes *match* *i S1 S2* **and** $\neg L1.\text{inf-step } S1$
shows $\neg L2.\text{inf-step } S2$
 ⟨*proof*⟩

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

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

lemma (in *scl-fol-calculus*) *decideI'*:
 $\text{is-ground-lit } (L \cdot l\ \gamma) \Longrightarrow \neg SCL-FOL.\text{trail-defined-lit } \Gamma (L \cdot l\ \gamma) \Longrightarrow \text{atm-of } L \cdot a$

$\gamma \preceq_B \beta \implies$
 $\Gamma' = \text{trail-decide } \Gamma (L \cdot l \gamma) \implies$
 $\text{decide } N \beta (\Gamma, U, \text{None}) (\Gamma', U, \text{None})$
 $\langle \text{proof} \rangle$

lemma *ground-iff-vars-term-empty*: $\text{ground } t \iff \text{vars-term } t = \{\}$
 $\langle \text{proof} \rangle$

lemma *is-ground-atm-eq-ground[iff]*: $\text{is-ground-atm} = \text{ground}$
 $\langle \text{proof} \rangle$

definition *lit-of-glit* :: 'f gterm literal \Rightarrow ('f, 'v) term literal **where**
 $\text{lit-of-glit} = \text{map-literal term-of-gterm}$

definition *glit-of-lit* **where**
 $\text{glit-of-lit} = \text{map-literal gterm-of-term}$

definition *cls-of-gcls* **where**
 $\text{cls-of-gcls} = \text{image-mset lit-of-glit}$

definition *gcls-of-cls* **where**
 $\text{gcls-of-cls} = \text{image-mset glit-of-lit}$

lemma *inj-lit-of-glit*: inj lit-of-glit
 $\langle \text{proof} \rangle$

lemma *atm-of-lit-of-glit-conv*: $\text{atm-of (lit-of-glit } L) = \text{term-of-gterm (atm-of } L)$
 $\langle \text{proof} \rangle$

lemma *ground-atm-of-lit-of-glit[simp]*: $\text{Term-Context.ground (atm-of (lit-of-glit } L))$
 $\langle \text{proof} \rangle$

lemma *is-ground-lit-lit-of-glit[simp]*: $\text{is-ground-lit (lit-of-glit } L)$
 $\langle \text{proof} \rangle$

lemma *is-ground-cls-cls-of-gcls[simp]*: $\text{is-ground-cls (cls-of-gcls } C)$
 $\langle \text{proof} \rangle$

lemma *glit-of-lit-lit-of-glit[simp]*: $\text{glit-of-lit (lit-of-glit } L) = L$
 $\langle \text{proof} \rangle$

lemma *gcls-of-cls-cls-of-gcls[simp]*: $\text{gcls-of-cls (cls-of-gcls } L) = L$
 $\langle \text{proof} \rangle$

lemma *lit-of-glit-glit-of-lit-ident[simp]*: $\text{is-ground-lit } L \implies \text{lit-of-glit (glit-of-lit } L) = L$
 $\langle \text{proof} \rangle$

lemma *cls-of-gcls-gcls-of-cls-ident[simp]*: $\text{is-ground-cls } D \implies \text{cls-of-gcls (gcls-of-cls } D)$

$D) = D$
 $\langle proof \rangle$

lemma *vars-lit-lit-of-glit[simp]*: $vars\text{-}lit (lit\text{-of-glit } L) = \{\}$
 $\langle proof \rangle$

lemma *vars-cls-cls-of-gcls[simp]*: $vars\text{-}cls (cls\text{-of-gcls } C) = \{\}$
 $\langle proof \rangle$

definition *atms-of-cls* :: 'a clause \Rightarrow 'a fset **where**
 $atms\text{-of-cls } C = atm\text{-of } |\cdot| fset\text{-mset } C$

definition *atms-of-cls* :: 'a clause fset \Rightarrow 'a fset **where**
 $atms\text{-of-cls } N = ffUnion (atms\text{-of-cls } |\cdot| N)$

lemma *atms-of-cls-empty[simp]*: $atms\text{-of-cls } \{\|\} = \{\|\}$
 $\langle proof \rangle$

lemma *atms-of-cls-finsert[simp]*:
 $atms\text{-of-cls } (finsert C N) = atm\text{-of-cls } C \cup atm\text{-of-cls } N$
 $\langle proof \rangle$

definition *lits-of-cls* :: 'a clause fset \Rightarrow 'a literal fset **where**
 $lits\text{-of-cls } N = ffUnion (fset\text{-mset } |\cdot| N)$

definition *lit-occures-in-cls* **where**
 $lit\text{-occures-in-cls } L N \longleftrightarrow fBex N (\lambda C. L \in \# C)$

inductive *constant-context* **for** R **where**
 $R C \mathcal{D} \mathcal{D}' \Longrightarrow constant\text{-context } R (C, \mathcal{D}) (C, \mathcal{D}')$

lemma *rtranclp-constant-context*: $(R C)^{**} \mathcal{D} \mathcal{D}' \Longrightarrow (constant\text{-context } R)^{**} (C, \mathcal{D}) (C, \mathcal{D}')$
 $\langle proof \rangle$

lemma *tranclp-constant-context*: $(R C)^{++} \mathcal{D} \mathcal{D}' \Longrightarrow (constant\text{-context } R)^{++} (C, \mathcal{D}) (C, \mathcal{D}')$
 $\langle proof \rangle$

lemma *right-unique-constant-context*:
assumes $R\text{-ru}$: $\bigwedge C. right\text{-unique } (R C)$
shows $right\text{-unique } (constant\text{-context } R)$
 $\langle proof \rangle$

lemma *safe-state-constant-context-if-invars*:
fixes $N s$
assumes
 $\mathcal{R}\text{-preserves-}\mathcal{I}$:
 $\bigwedge N s s'. \mathcal{R} N s s' \Longrightarrow \mathcal{I} N s \Longrightarrow \mathcal{I} N s'$ **and**

ex- \mathcal{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 \mathcal{R}) $\mathcal{F}(N, s)$
 \langle *proof* \rangle

primrec *trail-atms* :: $(- \text{ literal } \times -) \text{ list } \Rightarrow - \text{ fset}$ **where**
trail-atms [] = {} |
trail-atms (Ln # Γ) = *finset* (*atm-of* (*fst* Ln)) (*trail-atms* Γ)

lemma *fset-trail-atms*: $\text{fset} (\text{trail-atms } \Gamma) = \text{atm-of } \text{'fst' set } \Gamma$
 \langle *proof* \rangle

lemma *trail-defined-lit-iff-trail-defined-atm*:
 $\text{trail-defined-lit } \Gamma L \longleftrightarrow \text{atm-of } L \in | \text{trail-atms } \Gamma$
 \langle *proof* \rangle

lemma *trail-atms-subset-if-suffix*:
assumes *suffix* $\Gamma' \Gamma$
shows $\text{trail-atms } \Gamma' \subseteq | \text{trail-atms } \Gamma$
 \langle *proof* \rangle

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

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* :: $'v \text{ set } \Rightarrow 'v \Rightarrow 'v +$
fixes
 $\text{less-trm} :: 'f \text{ gterm } \Rightarrow 'f \text{ gterm } \Rightarrow \text{bool}$ (**infix** \prec_t 50)
assumes
transp-less-trm[*simp*]: *transp* (\prec_t) **and**
asyp-less-trm[*intro*]: *asyp* (\prec_t) **and**
wfP-less-trm[*intro*]: *wfP* (\prec_t) **and**
totalp-less-trm[*intro*]: *totalp* (\prec_t) **and**
finite-less-trm: $\bigwedge \beta. \text{finite } \{x. x \prec_t \beta\}$ **and**
less-trm-compatible-with-gctxt[*simp*]: $\bigwedge \text{ctxt } t t'. t \prec_t t' \implies \text{ctxt}(t)_G \prec_t \text{ctxt}(t')_G$
and

less-trm-if-subterm[simp]: $\bigwedge t \text{ ctxt}. \text{ ctxt} \neq \square_G \implies t \prec_t \text{ ctxt}(t)_G$

8 Ground ordered resolution for ground terms

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

sublocale *ord-res: ground-ordered-resolution-calculus* $(\prec_t) \lambda\cdot \{\#\}$
 $\langle \text{proof} \rangle$

sublocale *linorder-trm: linorder* $(\preceq_t) (\prec_t)$
 $\langle \text{proof} \rangle$

sublocale *linorder-lit: linorder* $(\preceq_l) (\prec_l)$
 $\langle \text{proof} \rangle$

sublocale *linorder-cls: linorder* $(\preceq_c) (\prec_c)$
 $\langle \text{proof} \rangle$

declare *linorder-trm.is-least-in-fset-filterD*[no-atp]

declare *linorder-lit.is-least-in-fset-filterD*[no-atp]

declare *linorder-cls.is-least-in-fset-filterD*[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-cls.is-least-in-fset $\{ \mid C \mid \in \mid N. \neg \text{ord-res-Interp } (\text{fset } N) \ C \models C \} \ C$

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

shows *is-least-false-clause* $(\text{finsert } C \ N) \ C$

$\langle \text{proof} \rangle$

lemma *is-least-false-clause-swap-swap-compatible-fsets:*

assumes $\{ \mid x \mid \in \mid N1. x \preceq_c C \} = \{ \mid x \mid \in \mid N2. x \preceq_c C \}$

shows *is-least-false-clause* $N1 \ C \longleftrightarrow \text{is-least-false-clause } N2 \ C$

$\langle \text{proof} \rangle$

lemma *Uniq-is-least-false-clause*: $\exists_{\leq 1} C. \text{is-least-false-clause } N \ C$
(proof)

lemma *empty-lesseq-cl[simp]*: $\{\#\} \preceq_c C$ for C
(proof)

lemma *is-least-false-clause-empty*: $\{\#\} \in N \implies \text{is-least-false-clause } N \ \{\#\}$
(proof)

lemma *production-union-unproductive-strong*:

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 $\text{ord-res.production } (N1 \cup N2) \ C = \text{ord-res.production } N1 \ C$

(proof)

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 $\text{ord-res.production } (N1 \cup N2) \ C = \text{ord-res.production } N1 \ C$

(proof)

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 $\text{ord-res.interp } (N1 \cup N2) = \text{ord-res.interp } N1$

(proof)

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 $\text{ord-res.Interp } (N1 \cup N2) \ C = \text{ord-res.Interp } N1 \ C$

(proof)

lemma *Interp-insert-unproductive*:

assumes

fin: finite $N1$ **and**

x-unproductive: $\text{ord-res.production } (\text{insert } x \ N1) \ x = \{\}$

shows $\text{ord-res.Interp } (\text{insert } x \ N1) \ C = \text{ord-res.Interp } N1 \ C$

(proof)

lemma *extended-partial-model-entails-iff-partial-model-entails*:

assumes

fin: finite N finite N' **and**

irrelevant: $\forall D \in N'. \exists E \in N. E \subset\# D \wedge \text{set-mset } D = \text{set-mset } E$ **and**
C-in: $C \in N$
shows $\text{ord-res-Interp } (N \cup N') C \models C \longleftrightarrow \text{ord-res-Interp } N C \models C$
<proof>

lemma *nex-strictly-maximal-pos-lit-if-factorizable*:
assumes *ord-res.ground-factoring* $C C'$
shows $\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L C$
<proof>

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 $\text{ord-res.production } N C = \{\}$
<proof>

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 $\forall C \in N'. \text{ord-res.production } (N \cup N') C = \{\}$
<proof>

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: $\exists D \in N. D \prec_c C \wedge \{D\} \models_e \{C\}$
shows $\text{is-least-false-clause } (\text{finsert } C N) = \text{is-least-false-clause } N$
<proof>

lemma *is-least-false-clause-funion-cancel-right-strong*:
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 $\text{is-least-false-clause } (N1 \mid\cup\mid N2) = \text{is-least-false-clause } N1$
<proof>

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 $\text{is-least-false-clause } (N1 \mid\cup\mid N2) = \text{is-least-false-clause } N1$
<proof>

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

lemma *obtains-least-false-clause-if-ex-false-clause*:
assumes *ex-false-clause* $(\text{fset } N)$
obtains C **where** $\text{is-least-false-clause } N C$
<proof>

lemma *ex-false-clause-if-least-false-clause*:

assumes *is-least-false-clause* $N C$

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

<proof>

lemma *ex-false-clause-iff*: *ex-false-clause* ($\text{fset } N$) \longleftrightarrow $(\exists C. \text{is-least-false-clause } N C)$

<proof>

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>

lemma *ord-res-model-entails-clauses-if-nex-false-clause*:

assumes *finite* N **and** $N \neq \{\}$ **and** $\neg \text{ex-false-clause } N$

shows *ord-res-model* $N \models_s N$

<proof>

lemma *pos-lit-not-greatest-if-maximal-in-least-false-clause*:

assumes

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

C-max-lit: *ord-res.is-maximal-lit* $(\text{Pos } A) C$

shows $\neg \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C$

<proof>

lemma *ex-ground-factoringI*:

assumes

C-max-lit: *ord-res.is-maximal-lit* $(\text{Pos } A) C$ **and**

C-not-max-lit: $\neg \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) C$

shows $\exists C'. \text{ord-res.ground-factoring } C C'$

<proof>

lemma *true-cls-if-true-lit-in*: $L \in \# C \implies I \models_l L \implies I \models C$

<proof>

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* $(\text{Neg } A) C$

shows $\text{fBex } N (\lambda D. D \prec_c C \wedge \text{ord-res.is-strictly-maximal-lit } (\text{Pos } A) D \wedge$

$\text{ord-res.production } (\text{fset } N) D = \{A\})$

<proof>

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 $(\text{fset } N) D = \{\text{atm-of } L\})$

<proof>

lemma *ex-ground-resolutionI*:

assumes

C-max-lit: *ord-res.is-maximal-lit* $(\text{Neg } A)$ C **and**

D-lt: $D \prec_c C$ **and**

D-max-lit: *ord-res.is-strictly-maximal-lit* $(\text{Pos } A)$ D

shows $\exists CD. \text{ord-res.ground-resolution } C D CD$

<proof>

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 $\neg \text{ord-res.interp } (N \cup N') C \cup \text{ord-res.production } (N \cup N') C \models C$

<proof>

lemma *trail-consistent-if-sorted-wrt-atoms*:

assumes *sorted-wrt* $(\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$

shows *trail-consistent* Γ

<proof>

lemma *mono-atms-lt*: *monotone-on* $(\text{set } \Gamma) (\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) (\lambda x y. y \leq x)$

$(\lambda x. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } x))$ **for** K

<proof>

lemma *in-trail-atms-dropWhileI*:

assumes

sorted-wrt R Γ **and**

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

$\neg P A$ **and**

$A \in | \text{trail-atms } \Gamma$

shows $A \in | \text{trail-atms } (\text{dropWhile } (\lambda Ln. P (\text{atm-of } (\text{fst } Ln)))) \Gamma$

<proof>

lemma *trail-defined-lit-dropWhileI*:

assumes

sorted-wrt $R \Gamma$ **and**
monotone-on (*set* Γ) $R (\geq) (\lambda x. P (fst\ x))$ **and**
 $\neg P\ L \wedge \neg P\ (\neg\ L)$ **and**
trail-defined-lit $\Gamma\ L$
shows *trail-defined-lit* (*dropWhile* ($\lambda Ln. P (fst\ Ln)$) Γ) L
 $\langle proof \rangle$

lemma *trail-defined-cls-dropWhileI*:

assumes
sorted-wrt $R \Gamma$ **and**
monotone-on (*set* Γ) $R (\geq) (\lambda x. P (fst\ x))$ **and**
 $\forall L \in \# C. \neg P\ L \wedge \neg P\ (\neg\ L)$ **and**
trail-defined-cls $\Gamma\ C$
shows *trail-defined-cls* (*dropWhile* ($\lambda Ln. P (fst\ Ln)$) Γ) C
 $\langle proof \rangle$

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** $X\ x$
 $\langle proof \rangle$

lemma *clause-le-if-lt-least-greater*:

fixes $N\ U_{er}\ \mathcal{F}\ C\ D$
defines
 $C \equiv The_optional\ (linorder_cls.is_least_in_fset\ (ffilter\ ((\prec_c)\ D)\ N))$
assumes
 $C\text{-lt}$: $\bigwedge E. C = Some\ E \implies C \prec_c E$ **and**
 $C\text{-in}$: $C \mid \in \mid N$
shows $C \preceq_c D$
 $\langle proof \rangle$

end

10 Lemmas about going between ground and first-order terms

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

lemma *ex1-gterm-of-term*:

fixes $t :: 'f\ gterm$
shows $\exists!(t' :: ('f, 'v)\ term). ground\ t' \wedge t = gterm\text{-of-term}\ t'$
 $\langle proof \rangle$

lemma *binj-betw-gterm-of-term*: *binj-betw* *gterm-of-term* $\{t. ground\ t\}\ UNIV$
 $\langle proof \rangle$

end

11 SCL(FOL) for first-order terms

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

definition *less-B* **where**

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

sublocale *order-less-B*: $order\ less-B = less-B$

<proof>

sublocale *scl-fol*: *scl-fol-calculus renaming-vars less-B*

<proof>

end

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

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

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

<proof>

lemma *trail-false-lit-eq-trail-false-lit-if-same-lits*:

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

shows $trail-false-lit\ \Gamma_9 = trail-false-lit\ \Gamma_{10}$

<proof>

lemma *trail-false-cls-eq-trail-false-cls-if-same-lits*:

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

shows $trail-false-cls\ \Gamma_9 = trail-false-cls\ \Gamma_{10}$

<proof>

lemma *trail-defined-lit-eq-trail-defined-lit-if-same-lits*:

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

shows $trail-defined-lit\ \Gamma_9 = trail-defined-lit\ \Gamma_{10}$

<proof>

lemma *trail-defined-cls-eq-trail-defined-cls-if-same-lits*:

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

shows $trail-defined-cls\ \Gamma_9 = trail-defined-cls\ \Gamma_{10}$

<proof>

end

theory *ORD-RES*

imports *Background*

begin

12 ORD-RES

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

lemma *ex-false-clause* $N \longleftrightarrow \neg (\forall C \in N. \text{ord-res-Interp } N C \models C)$
 ⟨*proof*⟩

lemma $\neg \text{ex-false-clause } N \longleftrightarrow (\forall C \in N. \text{ord-res-Interp } N C \models C)$
 ⟨*proof*⟩

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

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

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

sublocale *ord-res-antics: semantics* **where**
 $\text{step} = \text{ord-res}$ **and**
 $\text{final} = \text{ord-res-final}$
 ⟨*proof*⟩

sublocale *ord-res-language: language* **where**
 $\text{step} = \text{ord-res}$ **and**
 $\text{final} = \text{ord-res-final}$ **and**
 $\text{load} = \text{ord-res-load}$
 ⟨*proof*⟩

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 $L \implies$
 $D \mid \in \mid N \implies$
 $D \prec_c C \implies$
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 $D \prec_c C$
 $\langle \text{proof} \rangle$

lemma *right-unique-ord-res-1: right-unique ord-res-1*
 $\langle \text{proof} \rangle$

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-antics: semantics* **where**

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

$\langle \text{proof} \rangle$

sublocale *ord-res-1-language: language* **where**

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

$\langle \text{proof} \rangle$

lemma *ex-ord-res-1-if-not-final:*

assumes $\neg \text{ord-res-1-final } N$
shows $\exists N'. \text{ord-res-1 } N N'$

$\langle \text{proof} \rangle$

corollary *ord-res-1-safe: ord-res-1-final* $N \vee (\exists N'. \text{ord-res-1 } N N')$

<proof>
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 $C = (\text{THE } C'. \text{ord-res.ground-factoring}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factoring } C' C''))$

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

lemma *ex1-efac-eq-factoring-chain*:
 $\exists! C'. \text{efac } C = C' \wedge \text{ord-res.ground-factoring}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factoring } C' C'')$
 <proof>

lemma *efac-eq-disj*:
 $\text{efac } C = C \vee (\exists! C'. \text{efac } C = C' \wedge \text{ord-res.ground-factoring}^{**} C C' \wedge (\nexists C''. \text{ord-res.ground-factoring } C' C''))$
 <proof>

lemma *member-mset-if-count-eq-Suc*: $\text{count } X x = \text{Suc } n \implies x \in\# X$
 <proof>

lemmas *member-fsetE = mset-add*

lemma *ord-res-ground-factoring-iff*: $\text{ord-res.ground-factoring } C C' \longleftrightarrow$
 $(\exists A. \text{ord-res.is-maximal-lit } (\text{Pos } A) C \wedge (\exists n. \text{count } C (\text{Pos } A) = \text{Suc } (\text{Suc } n) \wedge C' = C - \{\#\text{Pos } A\}))$
 <proof>

lemma *trancpl-ord-res-ground-factoring-iff*:
 $\text{ord-res.ground-factoring}^{++} C C' \wedge (\nexists C''. \text{ord-res.ground-factoring } C' C'') \longleftrightarrow$
 $(\exists A. \text{ord-res.is-maximal-lit } (\text{Pos } A) C \wedge (\exists n. \text{count } C (\text{Pos } A) = \text{Suc } (\text{Suc } n) \wedge C' = C - \text{replicate-mset } (\text{Suc } n) (\text{Pos } A))$
 <proof>

lemma *minus-mset-replicate-mset-eq-add-mset-filter-mset*:
assumes $\text{count } X x = \text{Suc } n$
shows $X - \text{replicate-mset } n x = \text{add-mset } x \{\#y \in\# X. y \neq x\}$
 <proof>

lemma *minus-mset-replicate-mset-eq-add-mset-add-mset-filter-mset*:
assumes $\text{count } X \ x = \text{Suc } (\text{Suc } n)$
shows $X - \text{replicate-mset } n \ x = \text{add-mset } x \ (\text{add-mset } x \ \{\#y \in\# \ X. \ y \neq x\#})$
 $\langle \text{proof} \rangle$

lemma *rtrancl-ground-factoring-iff*:
shows $\text{ord-res.ground-factoring}^{**} \ C \ C' \wedge (\exists C''. \ \text{ord-res.ground-factoring } C' \ C'')$
 \longleftrightarrow
 $((\exists A. \ \text{ord-res.is-maximal-lit } (\text{Pos } A) \ C \wedge \text{count } C \ (\text{Pos } A) \geq 2) \wedge C = C' \vee$
 $(\exists A. \ \text{ord-res.is-maximal-lit } (\text{Pos } A) \ C \wedge C' = \text{add-mset } (\text{Pos } A) \ \{\#L \in\# \ C.$
 $L \neq \text{Pos } A\#}))$
 $\langle \text{proof} \rangle$

lemma *efac-spec*: $\text{efac } C = C \vee$
 $(\exists A. \ \text{ord-res.is-maximal-lit } (\text{Pos } A) \ C \wedge \text{efac } C = \text{add-mset } (\text{Pos } A) \ \{\#L \in\# \ C.$
 $L \neq \text{Pos } A\#})$
 $\langle \text{proof} \rangle$

lemma *efac-spec-if-pos-lit-is-maximal*:
assumes $L\text{-pos}: \text{is-pos } L$ **and** $L\text{-max}: \text{ord-res.is-maximal-lit } L \ C$
shows $\text{efac } C = \text{add-mset } L \ \{\#K \in\# \ C. \ K \neq L\#}$
 $\langle \text{proof} \rangle$

lemma *efac-mempty[simp]*: $\text{efac } \{\#\} = \{\#\}$
 $\langle \text{proof} \rangle$

lemma *set-mset-efac[simp]*: $\text{set-mset } (\text{efac } C) = \text{set-mset } C$
 $\langle \text{proof} \rangle$

lemma *efac-subset*: $\text{efac } C \subseteq\# \ C$
 $\langle \text{proof} \rangle$

lemma *true-cls-efac-iff[simp]*:
fixes $\mathcal{I} :: 'f \ \text{gterm set}$ **and** $C :: 'f \ \text{gclause}$
shows $\mathcal{I} \models \text{efac } C \longleftrightarrow \mathcal{I} \models C$
 $\langle \text{proof} \rangle$

lemma *obtains-positive-greatest-lit-if-efac-not-ident*:
assumes $\text{efac } C \neq C$
obtains L **where** $\text{is-pos } L$ **and** $\text{linorder-lit.is-greatest-in-mset } (\text{efac } C) \ L$
 $\langle \text{proof} \rangle$

lemma *mempty-in-image-efac-iff[simp]*: $\{\#\} \in \text{efac } \text{' } N \longleftrightarrow \{\#\} \in N$
 $\langle \text{proof} \rangle$

lemma *greatest-literal-in-efacI*:
assumes $\text{is-pos } L$ **and** $C\text{-max-lit}: \text{linorder-lit.is-maximal-in-mset } C \ L$
shows $\text{linorder-lit.is-greatest-in-mset } (\text{efac } C) \ L$
 $\langle \text{proof} \rangle$

lemma *linorder-lit.is-maximal-in-mset* (efac C) L \longleftrightarrow *linorder-lit.is-maximal-in-mset*
C L
⟨proof⟩

lemma
assumes *is-pos* L
shows *linorder-lit.is-greatest-in-mset* (efac C) L \longleftrightarrow *linorder-lit.is-maximal-in-mset*
C L
⟨proof⟩

lemma *factorizable-if-neq-efac*:
assumes C \neq efac C
shows \exists C'. *ord-res.ground-factoring* C C'
⟨proof⟩

lemma *nex-strictly-maximal-pos-lit-if-neq-efac*:
assumes C \neq efac C
shows \nexists L. *is-pos* L \wedge *ord-res.is-strictly-maximal-lit* L C
⟨proof⟩

lemma *efac-properties-if-not-ident*:
assumes efac C \neq C
shows efac C \prec_c C **and** {efac C} \models_e {C}
⟨proof⟩

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 \cup U_r \cup U_{ef}) C \implies

linorder-lit.is-maximal-in-mset C L \implies

is-pos L \implies

U_{ef}' = *fininsert* (efac C) U_{ef} \implies

ord-res-2 N (U_r, U_{ef}) (U_r, U_{ef}') |

resolution:

is-least-false-clause (N \cup U_r \cup U_{ef}) C \implies

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

inductive ord-res-2-step where

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

inductive ord-res-2-final where

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

inductive ord-res-2-load where

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

sublocale ord-res-2-semantic: semantics where

$step = ord\text{-}res\text{-}2\text{-}step$ **and**

$final = ord\text{-}res\text{-}2\text{-}final$

$\langle proof \rangle$

sublocale ord-res-2-language: language where

$step = ord\text{-}res\text{-}2\text{-}step$ **and**

$final = ord\text{-}res\text{-}2\text{-}final$ **and**

$load = ord\text{-}res\text{-}2\text{-}load$

$\langle proof \rangle$

lemma is-least-in-fset-with-irrelevant-clauses-if-is-least-in-fset:

assumes

$irrelevant: \forall D \mid\in\ N'. \exists E \mid\in\ N. E \subset\# D \wedge set\text{-}mset\ D = set\text{-}mset\ E$ **and**

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

shows $linorder\text{-}cls.is\text{-}least\text{-}in\text{-}fset\ \{\mid C \mid\in\ N \mid\cup\ N'. \neg ord\text{-}res\text{-}Interp\ (fset\ (N \mid\cup\ N'))\ C \models C\} C$

$\langle proof \rangle$

primrec fset-upto :: nat \Rightarrow nat \Rightarrow nat fset where

$fset\text{-}upto\ i\ 0 = (if\ i = 0\ then\ \{\mid 0\mid\}\ else\ \{\mid\}) \mid$

$fset\text{-}upto\ i\ (Suc\ n) = (if\ i \leq Suc\ n\ then\ finsert\ (Suc\ n)\ (fset\text{-}upto\ i\ n)\ else\ \{\mid\})$

lemma

$fset\text{-}upto\ 0\ 0 = \{\mid 0\mid\}$

$fset\text{-}upto\ 0\ 1 = \{\mid 0, 1\mid\}$

$fset\text{-}upto\ 0\ 2 = \{\mid 0, 1, 2\mid\}$

$fset\text{-}upto\ 0\ 3 = \{\mid 0, 1, 2, 3\mid\}$

$fset\text{-}upto\ 1\ 3 = \{\mid 1, 2, 3\mid\}$

$fset\text{-}upto\ 2\ 3 = \{\mid 2, 3\mid\}$

$fset\text{-upto } 3 \ 3 = \{|3|\}$
 $fset\text{-upto } 4 \ 3 = \{|\}$
 <proof>

lemma $i \leq 1 + j \implies List.\text{upto } i \ (1 + j) = List.\text{upto } i \ j \ @ \ [1 + j]$
 <proof>

lemma *fset-of-append-singleton*: $fset\text{-of-list } (xs \ @ \ [x]) = finsert \ x \ (fset\text{-of-list } xs)$
 <proof>

lemma *fset-of-list* $(List.\text{upto } (int \ i) \ (int \ j)) = int \ |^| \ fset\text{-upto } i \ j$
 <proof>

lemma *fset-fset-upto[simp]*: $fset \ (fset\text{-upto } m \ n) = \{m..n\}$
 <proof>

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

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

lemma
assumes
 $linorder\text{-cls.is-least-in-fset } \{|x \ | \in \ N1. \ P \ N1 \ x|\} \ x$ **and**
 $linorder\text{-cls.is-least-in-fset } N2 \ y$ **and**
 $\forall z \ | \in \ N2. \ z \preceq_c \ x$ **and**
 $P \ (N1 \ | \cup \ N2) \ y$ **and**
 $\forall z \ | \in \ N1. \ z \prec_c \ x \longrightarrow \neg \ P \ (N1 \ | \cup \ N2) \ z$
shows $linorder\text{-cls.is-least-in-fset } \{|x \ | \in \ N1 \ | \cup \ N2. \ P \ (N1 \ | \cup \ N2) \ x|\} \ y$
 <proof>

lemma *ground-factoring-preserves-efac*:
assumes $ord\text{-res.ground-factoring } P \ C$
shows $efac \ P = efac \ C$
 <proof>

lemma *ground-factorings-preserves-efac*:
assumes $ord\text{-res.ground-factoring}^{**} \ P \ C$
shows $efac \ P = efac \ C$
 <proof>

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>

corollary *ord-res-2-step-safe*: $\text{ord-res-2-final } S \vee (\exists S'. \text{ord-res-2-step } S S')$

<proof>

lemma *is-least-false-clause-if-is-least-false-clause-in-union-unproductive*:

assumes

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

and

C-in: $C \in N1$ **and**

C-least-false: $\text{is-least-false-clause } (N1 \cup N2) C$

shows $\text{is-least-false-clause } N1 C$

<proof>

lemma *ground-factoring-replicate-max-pos-lit*:

ord-res.ground-factoring

$(C_0 + \text{replicate-mset } (Suc (Suc n)) (Pos A))$

$(C_0 + \text{replicate-mset } (Suc n) (Pos A))$

if *ord-res.is-maximal-lit* $(Pos A) (C_0 + \text{replicate-mset } (Suc (Suc n)) (Pos A))$

for $A C_0 n$

<proof>

lemma *ground-factorings-replicate-max-pos-lit*:

assumes

ord-res.is-maximal-lit $(Pos A) (C_0 + \text{replicate-mset } (Suc (Suc n)) (Pos A))$

shows $m \leq Suc n \implies (\text{ord-res.ground-factoring } \widehat{m})$

$(C_0 + \text{replicate-mset } (Suc (Suc n)) (Pos A))$

$(C_0 + \text{replicate-mset } (Suc (Suc n - m)) (Pos A))$

<proof>

lemma *ord-res-Interp-entails-if-greatest-lit-is-pos*:

assumes *C-in*: $C \in N$ **and** *L-greatest*: $\text{linorder-lit.is-greatest-in-mset } C L$ **and**

L-pos: $\text{is-pos } L$

shows $\text{ord-res-Interp } N C \models C$

<proof>

lemma *right-unique-ord-res-2*: $\text{right-unique } (\text{ord-res-2 } N)$

<proof>

lemma *right-unique-ord-res-2-step*: $\text{right-unique } \text{ord-res-2-step}$

<proof>

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 $D C CD = \text{ord-res.} \text{ground-resolution } C D CD$

lemma *Uniq-ground-resolution*: $\exists_{\leq 1} DC. \text{ground-resolution } D C DC$
<proof>

lemma *ground-resolution-terminates*: $\text{wfP } (\text{ground-resolution } D)^{-1-1}$
<proof>

lemma *not-ground-resolution-empty-left*: $\neg \text{ground-resolution } \{\#\} C x$
<proof>

lemma *not-ground-resolution-empty-right*: $\neg \text{ground-resolution } C \{\#\} x$
<proof>

lemma *not-tranclp-ground-resolution-empty-left*: $\neg (\text{ground-resolution } \{\#\})^{++} C x$
<proof>

lemma *not-tranclp-ground-resolution-empty-right*: $\neg (\text{ground-resolution } C)^{++} \{\#\} x$
<proof>

lemma *left-premise-lt-right-premise-if-ground-resolution*:
 $\text{ground-resolution } D C DC \implies D \prec_c C$
<proof>

lemma *left-premise-lt-right-premise-if-tranclp-ground-resolution*:
 $(\text{ground-resolution } D)^{++} C DC \implies D \prec_c C$
<proof>

lemma *resolvent-lt-right-premise-if-ground-resolution*:
 $\text{ground-resolution } D C DC \implies DC \prec_c C$
<proof>

lemma *resolvent-lt-right-premise-if-tranclp-ground-resolution*:
 $(\text{ground-resolution } D)^{++} C DC \implies DC \prec_c C$
<proof>

Exhaustive resolution

definition *eres* **where**

$eres\ D\ C = (THE\ DC.\ full-run\ (ground-resolution\ D)\ C\ DC)$

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

lemma *eres-ident-iff*: $eres\ D\ C = C \longleftrightarrow (\#DC.\ ground-resolution\ D\ C\ DC)$
 ⟨proof⟩

lemma

assumes

step1: $ground-resolution\ D\ C\ DC$ **and**

stuck: $\#DDC.\ ground-resolution\ D\ DC\ DDC$

shows $eres\ D\ C = DC$

⟨proof⟩

lemma

assumes

step1: $ground-resolution\ D\ C\ DC$ **and**

step2: $ground-resolution\ D\ DC\ DDC$ **and**

stuck: $\#DDDC.\ ground-resolution\ D\ DDC\ DDDC$

shows $eres\ D\ C = DDC$

⟨proof⟩

lemma

assumes

step1: $ground-resolution\ D\ C\ DC$ **and**

step2: $ground-resolution\ D\ DC\ DDC$ **and**

step3: $ground-resolution\ D\ DDC\ DDDC$ **and**

stuck: $\#DDDDC.\ ground-resolution\ D\ DDDC\ DDDDC$

shows $eres\ D\ C = DDDC$

⟨proof⟩

lemma *eres-mempty-left[simp]*: $eres\ \{\#\}\ C = C$

⟨proof⟩

lemma *eres-mempty-right[simp]*: $eres\ C\ \{\#\} = \{\#\}$

⟨proof⟩

lemma *ex1-eres-eq-full-run-ground-resolution*: $\exists!DC.\ eres\ D\ C = DC \wedge full-run\ (ground-resolution\ D)\ C\ DC$

⟨proof⟩

lemma *eres-le*: $eres\ D\ C \preceq_c\ C$

⟨proof⟩

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)

lemma *eres-lt-if*:

assumes *E-max-lit*: *ord-res.is-maximal-lit* $L E$ **and** *L-neg*: *is-neg* L **and**
D-max-lit: *linorder-lit.is-greatest-in-mset* $D (- L)$

shows $eres D E \prec_c E$
(proof)

lemma *eres-eq-after-ground-resolution*:

assumes *ground-resolution* $D C DC$

shows $eres D C = eres D DC$

(proof)

lemma *eres-eq-after-rtranclp-ground-resolution*:

assumes $(ground-resolution D)^{**} C DC$

shows $eres D C = eres D DC$

(proof)

lemma *eres-eq-after-tranclp-ground-resolution*:

assumes $(ground-resolution D)^{++} C DC$

shows $eres D C = eres D DC$

(proof)

lemma *resolvable-if-neq-eres*:

assumes $C \neq eres D C$

shows $\exists! DC. ground-resolution D C DC$

(proof)

lemma *nex-maximal-pos-lit-if-resolvable*:

assumes *ground-resolution* $D C DC$

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

(proof)

corollary *nex-strictly-maximal-pos-lit-if-resolvable*:

assumes *ground-resolution* $D C DC$

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

(proof)

corollary *nex-maximal-pos-lit-if-neq-eres*:

assumes $C \neq eres D C$

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

(proof)

corollary *nex-strictly-maximal-pos-lit-if-neq-eres*:

assumes $C \neq eres D C$

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

(proof)

lemma *ground-resolutionD*:

assumes *ground-resolution D C DC*

shows $\exists m A D' C'$.

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

DC = D' + replicate-mset m (Neg A) + C'

<proof>

lemma *relpowp-ground-resolutionD*:

assumes $n \neq 0$ **and** (*ground-resolution D \rightsquigarrow n*) *C DnC*

shows $\exists m A D' C'$. *Suc m \geq 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 n D' + replicate-mset (Suc m - n) (Neg A) + C'

<proof>

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>

lemma *eres-not-identD*:

assumes *eres D C \neq C*

shows $\exists m A D' C'$.

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

eres D C = repeat-mset (Suc m) D' + C'

<proof>

lemma *lit-in-one-of-resolvents-if-in-eres*:

fixes *L :: 'f gterm literal and C D :: 'f gclause*

assumes *L $\in\#$ eres C D*

shows *L $\in\#$ C \vee L $\in\#$ D*

<proof>

lemma *strong-lit-in-one-of-resolvents-if-in-eres*:

fixes $L :: 'f \text{ gterm literal}$ **and** $C D :: 'f \text{ gclause}$
assumes
D-max-lit: linorder-lit.is-maximal-in-mset $D L$ **and**
K-in: $K \in\# \text{ eres } C D$
shows $K \in\# C \wedge K \neq -L \vee K \in\# D$
 <proof>

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: $K \in\# \text{ eres } C D$
shows $K \in\# C \wedge K \neq -L \vee K \in\# D \wedge K \neq L$
 <proof>

lemma *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-max-lit: linorder-lit.is-maximal-in-mset $D L$ **and**
 $-L \notin\# D$ **and**
K-in-eres: $K \in\# \text{ eres } C D$
shows $\text{atm-of } K \prec_t \text{ atm-of } L$
 <proof>

lemma *eres-entails-resolvent:*
fixes $C D :: 'f \text{ gterm clause}$
assumes $(\text{ground-resolution } C)^{++} D_0 D$
shows $\{\text{eres } C D_0\} \models_e \{D\}$
 <proof>

lemma *clause-true-if-resolved-true:*
assumes
 $(\text{ground-resolution } D)^{++} C DC$ **and**
D-productive: ord-res.production $N D \neq \{\}$ **and**
C-true: ord-res-Interp $N DC \models DC$
shows $\text{ord-res-Interp } N C \models C$
 <proof>

lemma *clause-true-if-eres-true:*
assumes
 $(\text{ground-resolution } D1)^{++} D2 C$ **and**
 $C \neq \text{eres } D1 C$ **and**
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>

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:

$$\begin{aligned}
 & \text{is-least-false-clause } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) \ C \implies \\
 & \text{linorder-lit.is-maximal-in-mset } C \ L \implies \\
 & \text{is-pos } L \implies \\
 & U_{ef}' = \text{finsert } (\text{efac } C) \ U_{ef} \implies \\
 & \text{ord-res-3 } N \ (U_{er}, U_{ef}) \ (U_{er}, U_{ef}') \mid
 \end{aligned}$$

resolution:

$$\begin{aligned}
 & \text{is-least-false-clause } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) \ C \implies \\
 & \text{linorder-lit.is-maximal-in-mset } C \ L \implies \\
 & \text{is-neg } L \implies \\
 & D \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef} \implies \\
 & D \prec_c C \implies \\
 & \text{ord-res.production } (\text{fset } (N \mid \cup \mid U_{er} \mid \cup \mid U_{ef})) \ D = \{\text{atm-of } L\} \implies \\
 & U_{er}' = \text{finsert } (\text{eres } D \ C) \ U_{er} \implies \\
 & \text{ord-res-3 } N \ (U_{er}, U_{ef}) \ (U_{er}', U_{ef})
 \end{aligned}$$

inductive ord-res-3-step where

$$\text{ord-res-3 } N \ s \ s' \implies \text{ord-res-3-step } (N, s) \ (N, s')$$

inductive ord-res-3-final where

$$\text{ord-res-final } (N \mid \cup \mid U_{rr} \mid \cup \mid U_{ef}) \implies \text{ord-res-3-final } (N, (U_{rr}, U_{ef}))$$

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>

sublocale ord-res-3-language: language where

step = ord-res-3-step **and**

final = ord-res-3-final **and**

load = ord-res-3-load

<proof>

lemma *is-least-false-clause-conv-if-partial-resolution-invariant:*

assumes $\forall C \mid \in \mid U_{pr}. \exists D1 \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}. \exists D2 \mid \in \mid N \mid \cup \mid U_{er} \mid \cup \mid U_{ef}.$
 $(\text{ground-resolution } D1)^{++} D2 C \wedge C \neq \text{eres } D1 D2 \wedge \text{eres } D1 D2 \mid \in \mid U_{er}$

shows *is-least-false-clause* $(N \mid \cup \mid U_{pr} \mid \cup \mid U_{er} \mid \cup \mid U_{ef}) = \text{is-least-false-clause}$
 $(N \mid \cup \mid U_{er} \mid \cup \mid U_{ef})$

<proof>

lemma *right-unique-ord-res-3: right-unique (ord-res-3 N)*

<proof>

lemma *right-unique-ord-res-3-step: right-unique ord-res-3-step*

<proof>

lemma *ex-ord-res-3-if-not-final:*

assumes $\neg \text{ord-res-3-final } S$

shows $\exists S'. \text{ord-res-3-step } S S'$

<proof>

corollary *ord-res-3-step-safe: ord-res-3-final S \vee ($\exists S'. \text{ord-res-3-step } S S'$)*

<proof>

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* $\mathcal{F} \{\#\} = \{\#\}$

<proof>

lemma *fset-mset-iefac[simp]:*

fixes $\mathcal{F} :: 'f \text{ gclause fset}$ **and** $C :: 'f \text{ gclause}$

shows *fset-mset* $(\text{iefac } \mathcal{F} C) = \text{fset-mset } C$

<proof>

lemma *atms-of-cls-iefac[simp]*:

fixes $\mathcal{F} :: 'f\ gclause\ fset$ **and** $C :: 'f\ gclause$
shows $atms-of-cls\ (iefac\ \mathcal{F}\ C) = atms-of-cls\ C$
<proof>

lemma *iefac-le*:

fixes $\mathcal{F} :: 'f\ gclause\ fset$ **and** $C :: 'f\ gclause$
shows $iefac\ \mathcal{F}\ C \preceq_c C$
<proof>

lemma *true-cls-iefac-iff[simp]*:

fixes $\mathcal{I} :: 'f\ gterm\ set$ **and** $\mathcal{F} :: 'f\ gclause\ fset$ **and** $C :: 'f\ gclause$
shows $\mathcal{I} \models iefac\ \mathcal{F}\ C \longleftrightarrow \mathcal{I} \models C$
<proof>

lemma *funion-funion-eq-funion-funion-fimage-iefac-if*:

assumes $U_{ef}\text{-eq}: U_{ef} = iefac\ \mathcal{F}\ |\uparrow\ \{C \mid C \in N \mid\cup\ U_{er}. iefac\ \mathcal{F}\ C \neq C\}$
shows $N \mid\cup\ U_{er} \mid\cup\ U_{ef} = N \mid\cup\ U_{er} \mid\cup\ (iefac\ \mathcal{F}\ |\uparrow\ (N \mid\cup\ U_{er}))$
<proof>

lemma *clauses-for-iefac-are-unproductive*:

$\forall C \in N \mid\cup\ iefac\ \mathcal{F}\ |\uparrow\ N. \forall U. ord\text{-res.}\text{production}\ U\ C = \{\}$
<proof>

lemma *clauses-for-iefac-have-smaller-entailing-clause*:

$\forall C \in N \mid\cup\ iefac\ \mathcal{F}\ |\uparrow\ N. \exists D \in iefac\ \mathcal{F}\ |\uparrow\ N. D \prec_c C \wedge \{D\} \models_e \{C\}$
<proof>

lemma *is-least-false-clause-with-iefac-conv*:

$is\text{-least-false-clause}\ (N \mid\cup\ U_{er} \mid\cup\ iefac\ \mathcal{F}\ |\uparrow\ (N \mid\cup\ U_{er})) =$
 $is\text{-least-false-clause}\ (iefac\ \mathcal{F}\ |\uparrow\ (N \mid\cup\ U_{er}))$
<proof>

lemma *MAGIC4*:

fixes $N\ \mathcal{F}\ \mathcal{F}'\ U_{er}\ U_{er}'$

defines

$N1 \equiv iefac\ \mathcal{F}\ |\uparrow\ (N \mid\cup\ U_{er})$ **and**
 $N2 \equiv iefac\ \mathcal{F}'\ |\uparrow\ (N \mid\cup\ U_{er}')$

assumes

$subsets\text{-agree}: \{x \mid x \in N1. x \prec_c C\} = \{x \mid x \in N2. x \prec_c C\}$ **and**

$is\text{-least-false-clause}\ N1\ D$ **and**

$is\text{-least-false-clause}\ N2\ E$ **and**

$C \prec_c D$

shows $C \preceq_c E$

<proof>

lemma *atms-of-cls-fimage-iefac[simp]*:

$atms-of-clss (iefac \mathcal{F} \mid \uparrow N) = atms-of-clss N$
 ⟨proof⟩

lemma *atm-of-in-atms-of-clssI*:
assumes *L-in*: $L \in \# C$ **and** *C-in*: $C \in \mid iefac \mathcal{F} \mid \uparrow N$
shows *atm-of* $L \in \mid atms-of-clss N$
 ⟨proof⟩

lemma *clause-almost-almost-definedI*:
fixes $\Gamma D K$
assumes
D-in: $D \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$ **and**
D-max-lit: *ord-res.is-maximal-lit* $K D$ **and**
no-undef-atm: $\neg (\exists A \in \mid atms-of-clss (N \mid \cup U_{er}). A \prec_t atm-of K \wedge A \notin \mid$
trail-atms $\Gamma)$
shows *trail-defined-cl* $\Gamma \{ \#L \in \# D. L \neq K \wedge L \neq -K \# \}$
 ⟨proof⟩

lemma *clause-almost-definedI*:
fixes $\Gamma D K$
assumes
D-in: $D \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$ **and**
D-max-lit: *ord-res.is-maximal-lit* $K D$ **and**
no-undef-atm: $\neg (\exists A \in \mid atms-of-clss (N \mid \cup U_{er}). A \prec_t atm-of K \wedge A \notin \mid$
trail-atms $\Gamma)$ **and**
K-defined: *trail-defined-lit* ΓK
shows *trail-defined-cl* $\Gamma \{ \#Ka \in \# D. Ka \neq K \# \}$
 ⟨proof⟩

lemma *eres-not-in-known-clauses-if-trail-false-cl*:
fixes
 $\mathcal{F} :: 'f gclause fset$ **and**
 $\Gamma :: ('f gliteral \times 'f gclause option) list$
assumes
 Γ -*consistent*: *trail-consistent* Γ **and**
clauses-lt-E-true: $\forall C \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er}). C \prec_c E \longrightarrow trail-true-cl \Gamma$
 C **and**
eres $D E \prec_c E$ **and**
trail-false-cl $\Gamma (eres D E)$
shows *eres* $D E \notin \mid N \mid \cup U_{er}$
 ⟨proof⟩

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* $(N \mid \cup U_{er})$)
and
D-in: $D \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup U_{er})$ **and**
D-false: *trail-false-cl* ΓD **and**
D-max-lit: *linorder-lit.is-maximal-in-mset* $D L$

shows $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \preceq_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma)$
 ⟨proof⟩

lemma *trail-defined-if-no-undef-atom-le-max-lit*:

assumes

C-in: $C | \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ **and**

C-max-lit: *linorder-lit.is-maximal-in-mset* $C K$ **and**

no-undef-atom-le-K:

$\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \preceq_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma)$

shows *trail-defined-clc* ΓC

⟨proof⟩

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* ($N \cup U_{er}$))

and

D-in: $D | \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ **and**

D-false: *trail-false-clc* ΓD **and**

D-max-lit: *linorder-lit.is-maximal-in-mset* $D L$ **and**

C-in: $C | \in | \text{iefac } \mathcal{F} \uparrow (N \cup U_{er})$ **and**

C-max-lit: *linorder-lit.is-maximal-in-mset* $C K$ **and**

C-lt: $C \prec_c D$

shows $\neg (\exists A | \in | \text{atms-of-clss } (N \cup U_{er}). A \preceq_t \text{atm-of } K \wedge A \notin | \text{trail-atms } \Gamma)$

⟨proof⟩

lemma *bex-trail-false-clc-simp*:

fixes $\mathcal{F} N \Gamma$

shows $fBex (\text{iefac } \mathcal{F} \uparrow N) (\text{trail-false-clc } \Gamma) \longleftrightarrow fBex N (\text{trail-false-clc } \Gamma)$

⟨proof⟩

end

end

theory *ORD-RES-4*

imports

ORD-RES

Implicit-Exhaustive-Factorization

Exhaustive-Resolution

begin

19 ORD-RES-4 (implicit factorization)

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

inductive *ord-res-4* **where**

factoring:

$NN = \text{iefac } \mathcal{F} \uparrow (N \cup U_{er}) \implies$

is-least-false-clause $NN C \implies$

linorder-lit.is-maximal-in-mset $C L \implies$

$is\text{-}pos\ L \implies$
 $\mathcal{F}' = \text{finsert}\ C\ \mathcal{F} \implies$
 $ord\text{-}res\text{-}4\ N\ (U_{er}, \mathcal{F})\ (U_{er}, \mathcal{F}')\ |$

resolution:

$NN = \text{iefac}\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}) \implies$
 $is\text{-}least\text{-}false\text{-}clause\ NN\ C \implies$
 $linorder\text{-}lit.\text{is}\text{-}maximal\text{-}in\text{-}mset\ C\ L \implies$
 $is\text{-}neg\ L \implies$
 $D\ |\in|\ NN \implies$
 $D\ \prec_c\ C \implies$
 $ord\text{-}res.\text{production}\ (fset\ NN)\ D = \{atm\text{-}of\ L\} \implies$
 $U_{er}' = \text{finsert}\ (eres\ D\ C)\ U_{er} \implies$
 $ord\text{-}res\text{-}4\ N\ (U_{er}, \mathcal{F})\ (U_{er}', \mathcal{F})$

inductive *ord-res-4-step* **where**

$ord\text{-}res\text{-}4\ N\ s\ s' \implies ord\text{-}res\text{-}4\text{-}step\ (N, s)\ (N, s')$

inductive *ord-res-4-final* **where**

$ord\text{-}res\text{-}final\ (\text{iefac}\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er})) \implies ord\text{-}res\text{-}4\text{-}final\ (N, U_{er}, \mathcal{F})$

sublocale *ord-res-4-semantic: semantics* **where**

$step = ord\text{-}res\text{-}4\text{-}step$ **and**

$final = ord\text{-}res\text{-}4\text{-}final$

$\langle proof \rangle$

lemma *right-unique-ord-res-4: right-unique* ($ord\text{-}res\text{-}4\ N$)

$\langle proof \rangle$

lemma *right-unique-ord-res-4-step: right-unique* *ord-res-4-step*

$\langle proof \rangle$

lemma *ex-ord-res-4-if-not-final:*

assumes $\neg ord\text{-}res\text{-}4\text{-}final\ S$

shows $\exists S'. ord\text{-}res\text{-}4\text{-}step\ S\ S'$

$\langle proof \rangle$

corollary *ord-res-4-step-safe: ord-res-4-final* $S \vee (\exists S'. ord\text{-}res\text{-}4\text{-}step\ S\ S')$

$\langle proof \rangle$

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 ⇒
is-pos L ⇒
linorder-lit.is-greatest-in-mset C L ⇒
M' = M(atm-of L := Some 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') |

factoring:

¬ (dom M) ⊨ C ⇒
linorder-lit.is-maximal-in-mset C L ⇒
is-pos L ⇒
¬ linorder-lit.is-greatest-in-mset C L ⇒
F' = finsert C F ⇒
M' = (λ-. None) ⇒
C' = The-optional (linorder-cls.is-least-in-fset (iefac F' |' (N |∪| U_{er}))) ⇒
ord-res-5 N (U_{er}, F, M, Some C) (U_{er}, F', M', C') |

resolution:

¬ (dom M) ⊨ C ⇒
linorder-lit.is-maximal-in-mset C L ⇒
is-neg L ⇒
M (atm-of L) = Some D ⇒
U_{er}' = finsert (eres D C) U_{er} ⇒
M' = (λ-. None) ⇒
C' = The-optional (linorder-cls.is-least-in-fset (iefac F |' (N |∪| U_{er}'))) ⇒
ord-res-5 N (U_{er}, F, M, Some C) (U_{er}', F, M', C')

inductive ord-res-5-step :: 'f ord-res-5 ⇒ 'f ord-res-5 ⇒ bool **where**

ord-res-5 N s s' ⇒ 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' ⇒ ord-res-5-step⁺⁺ (N, s) (N, s')

<proof>

inductive *ord-res-5-final* :: 'f *ord-res-5* \Rightarrow bool **where**

model-found:

ord-res-5-final (*N*, *U_{er}*, *F*, *M*, None) |

contradiction-found:

ord-res-5-final (*N*, *U_{er}*, *F*, *M*, Some {#})

sublocale *ord-res-5-antics*: *semantics* **where**

step = *ord-res-5-step* **and**

final = *ord-res-5-final*

<proof>

lemma *right-unique-ord-res-5*: *right-unique* (*ord-res-5* *N*)

<proof>

lemma *right-unique-ord-res-5-step*: *right-unique* *ord-res-5-step*

<proof>

definition *next-clause-in-factorized-clause* **where**

next-clause-in-factorized-clause *N s* \longleftrightarrow

$(\forall U_{er} \mathcal{F} \mathcal{M} \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \longrightarrow C \in |iefac \mathcal{F} | \uparrow (N \cup U_{er}))$

lemma *next-clause-in-factorized-clause*:

assumes *step*: *ord-res-5* *N s s'*

shows *next-clause-in-factorized-clause* *N s'*

<proof>

definition *implicitly-factorized-clauses-subset* **where**

implicitly-factorized-clauses-subset *N s* \longleftrightarrow

$(\forall U_{er} \mathcal{F} \mathcal{M} \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow \mathcal{F} \subseteq |N \cup U_{er}$)

lemma *ord-res-5-preserves-implicitly-factorized-clauses-subset*:

assumes

step: *ord-res-5* *N s s'* **and**

invars:

implicitly-factorized-clauses-subset *N s* **and**

next-clause-in-factorized-clause *N s*

shows *implicitly-factorized-clauses-subset* *N s'*

<proof>

lemma *interp-eq-Interp-if-least-greater*:

assumes

C-in: *C* \in | *NN* **and**

D-least-gt-C: *linorder-cls.is-least-in-fset* (*ffilter* ((\prec_c) *C*) *NN*) *D*

shows *ord-res.interp* (*fset* *NN*) *D* = *ord-res.interp* (*fset* *NN*) *C* \cup *ord-res.production* (*fset* *NN*) *C*

<proof>

lemma *interp-eq-empty-if-least-in-set:*
assumes *linorder-clc.is-least-in-set* $N C$
shows $\text{ord-res.interp } N C = \{\}$
 $\langle \text{proof} \rangle$

definition *model-eq-interp-upto-next-clause* **where**
 $\text{model-eq-interp-upto-next-clause } N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \mathcal{M} C. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) \longrightarrow$
 $\text{dom } \mathcal{M} = \text{ord-res.interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C)$

lemma *model-eq-interp-upto-next-clause:*
assumes *step: ord-res-5* $N s s'$ **and**
invars:
 $\text{model-eq-interp-upto-next-clause } N s$
 $\text{next-clause-in-factorized-clause } N s$
shows $\text{model-eq-interp-upto-next-clause } N s'$
 $\langle \text{proof} \rangle$

definition *all-smaller-clauses-true-wrt-respective-Interp* **where**
 $\text{all-smaller-clauses-true-wrt-respective-Interp } N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \mathcal{M} C. s = (U_{er}, \mathcal{F}, \mathcal{M}, C) \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). (\forall D. C = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$
 $\text{ord-res-Interp } (\text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}))) C \models C))$

lemma *all-smaller-clauses-true-wrt-respective-Interp:*
assumes *step: ord-res-5* $N s s'$ **and**
invars:
 $\text{all-smaller-clauses-true-wrt-respective-Interp } N s$
 $\text{model-eq-interp-upto-next-clause } N s$
 $\text{next-clause-in-factorized-clause } N s$
shows $\text{all-smaller-clauses-true-wrt-respective-Interp } N s'$
 $\langle \text{proof} \rangle$

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 $\forall U_{er} \mathcal{F} \mathcal{M} D. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D) \longrightarrow$
 $(\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). C \prec_c D \longrightarrow \text{dom } \mathcal{M} \models C)$
 $\langle \text{proof} \rangle$

definition *model-eq-sublocale* **where**
 $\text{model-eq-sublocale } N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \mathcal{M}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{None}) \longrightarrow$
 $(\text{let } NN = \text{fset } (\text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \text{ in } \text{dom } \mathcal{M} = \cup (\text{ord-res.production } NN \text{ ' } NN)))$

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 $\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow$

$(\forall C\ |\in|\ \text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}). (\forall D. \mathcal{C} = \text{Some}\ D \longrightarrow C \prec_c D) \longrightarrow \text{dom}\ \mathcal{M} \Vdash C)$

$\langle \text{proof} \rangle$

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 $\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some}\ C) \longrightarrow$

$(\forall D. \text{is-least-false-clause}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}))\ D \longrightarrow C \preceq_c D)$

$\langle \text{proof} \rangle$

definition *atoms-in-model-were-produced where*

atoms-in-model-were-produced $N\ s \longleftrightarrow$

$(\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ \mathcal{C}. s = (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \longrightarrow (\forall A\ C. \mathcal{M}\ A = \text{Some}\ C \longrightarrow$

$A \in \text{ord-res.production}\ (\text{fset}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ C))$

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* $N\ s'$

$\langle \text{proof} \rangle$

definition *all-produced-atoms-in-model where*

all-produced-atoms-in-model $N\ s \longleftrightarrow$

$(\forall U_{er}\ \mathcal{F}\ \mathcal{M}\ D. s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some}\ D) \longrightarrow (\forall C\ A. C \prec_c D \longrightarrow$

$A \in \text{ord-res.production}\ (\text{fset}\ (\text{iefac}\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er})))\ C \longrightarrow \mathcal{M}\ A = \text{Some}\ C))$

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* $N\ s'$

$\langle \text{proof} \rangle$

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* $\mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$) C

shows *ord-res-5-invars* $N\ (U_{er}, \mathcal{F}, \text{Map.empty}, \text{Some } C)$

<proof>

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>

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* $N\ s'$

<proof>

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* $N\ s'$

<proof>

lemma *le-least-false-clause*:

fixes $N\ s\ U_{er}\ \mathcal{F}\ \mathcal{M}\ C\ D$

assumes

invars: *ord-res-5-invars* $N\ s$ **and**

s-def: $s = (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C)$ **and**

D-least-false: *is-least-false-clause* (*iefac* $\mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$) D

shows $C \preceq_c D$

<proof>

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>

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>

lemma *MAGIC1*:

assumes *invars*: *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$
shows $\exists M' C'. (ord-res-5 N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, M', C') \wedge$
 $(\nexists M'' C''. ord-res-5 N (U_{er}, \mathcal{F}, M', C') (U_{er}, \mathcal{F}, M'', C''))$
 <proof>

lemma *MAGIC2*:

assumes *invars*: *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C)$
assumes $C \neq \{\#\}$
shows $\exists s'. ord-res-5 N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) s'$
 <proof>

lemma *MAGIC3*:

assumes *invars*: *ord-res-5-invars* $N (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$ **and**
steps: $(ord-res-5 N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, M', C')$ **and**
no-more-steps: $(\nexists M'' C''. ord-res-5 N (U_{er}, \mathcal{F}, M', C') (U_{er}, \mathcal{F}, M'', C''))$
shows $(\forall C. C' = \text{Some } C \longleftrightarrow is-least-false-clause (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C)$
 <proof>

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 $\exists M' C'. (ord-res-5 N)^{**} (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) (U_{er}, \mathcal{F}, M', C') \wedge$
 $(\forall C. C' = \text{Some } C \longleftrightarrow is-least-false-clause (iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) C)$
 <proof>

end

end

theory *ORD-RES-6*

imports

ORD-RES-5

begin

21 ORD-RES-6 (model backjump)

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

inductive *ord-res-6* **where**

skip:

$(dom \mathcal{M}) \models C \implies$

$C' = \text{The-optional} (\text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid iefac \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$

$C \prec_c D\}) \implies$

$ord-res-6 N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, C') \mid$

production:

$$\begin{aligned}
& \neg (\text{dom } \mathcal{M}) \models C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \text{is-pos } L \implies \\
& \text{linorder-lit.is-greatest-in-mset } C L \implies \\
& \mathcal{M}' = \mathcal{M}(\text{atm-of } L := \text{Some } C) \implies \\
& C' = \text{The-optional } (\text{linorder-cls.is-least-in-fset } \{|D | \in | \text{iefac } \mathcal{F} | \uparrow (N \cup | U_{er}). \\
C \prec_c D\}) \implies \\
& \text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}', C') |
\end{aligned}$$

factoring:

$$\begin{aligned}
& \neg (\text{dom } \mathcal{M}) \models C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \text{is-pos } L \implies \\
& \neg \text{linorder-lit.is-greatest-in-mset } C L \implies \\
& \mathcal{F}' = \text{finsert } C \mathcal{F} \implies \\
& \text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}, \mathcal{F}', \mathcal{M}, \text{Some } (\text{efac } C)) |
\end{aligned}$$

resolution-bot:

$$\begin{aligned}
& \neg (\text{dom } \mathcal{M}) \models C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \text{is-neg } L \implies \\
& \mathcal{M} (\text{atm-of } L) = \text{Some } D \implies \\
& U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies \\
& \text{eres } D C = \{\#\} \implies \\
& \mathcal{M}' = (\lambda-. \text{None}) \implies \\
& \text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } \{\#\}) |
\end{aligned}$$

resolution-pos:

$$\begin{aligned}
& \neg (\text{dom } \mathcal{M}) \models C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \text{is-neg } L \implies \\
& \mathcal{M} (\text{atm-of } L) = \text{Some } D \implies \\
& U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies \\
& \text{eres } D C \neq \{\#\} \implies \\
& \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \implies \\
& \text{linorder-lit.is-maximal-in-mset } (\text{eres } D C) K \implies \\
& \text{is-pos } K \implies \\
& \text{ord-res-6 } N (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } C) (U_{er}', \mathcal{F}, \mathcal{M}', \text{Some } (\text{eres } D C)) |
\end{aligned}$$

resolution-neg:

$$\begin{aligned}
& \neg (\text{dom } \mathcal{M}) \models C \implies \\
& \text{linorder-lit.is-maximal-in-mset } C L \implies \\
& \text{is-neg } L \implies \\
& \mathcal{M} (\text{atm-of } L) = \text{Some } D \implies \\
& U_{er}' = \text{finsert } (\text{eres } D C) U_{er} \implies \\
& \text{eres } D C \neq \{\#\} \implies \\
& \mathcal{M}' = \text{restrict-map } \mathcal{M} \{A. A \prec_t \text{atm-of } K\} \implies \\
& \text{linorder-lit.is-maximal-in-mset } (\text{eres } D C) K \implies
\end{aligned}$$

$is\text{-}neg\ K \implies$
 $\mathcal{M}\ (atm\text{-}of\ K) = Some\ E \implies$
 $ord\text{-}res\text{-}6\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, Some\ C)\ (U_{er}', \mathcal{F}, \mathcal{M}', Some\ E)$

inductive *ord-res-6-step* **where**

$ord\text{-}res\text{-}6\ N\ s\ s' \implies ord\text{-}res\text{-}6\text{-}step\ (N, s)\ (N, s')$

lemma *tranclp-ord-res-6-step-if-tranclp-ord-res-6*:

$(ord\text{-}res\text{-}6\ N)^{++}\ s\ s' \implies ord\text{-}res\text{-}6\text{-}step^{++}\ (N, s)\ (N, s')$
 $\langle proof \rangle$

lemma *right-unique-ord-res-6*: *right-unique* $(ord\text{-}res\text{-}6\ N)$

$\langle proof \rangle$

lemma *right-unique-ord-res-6-step*: *right-unique* *ord-res-6-step*

$\langle proof \rangle$

inductive *ord-res-6-final* **where**

model-found:

$ord\text{-}res\text{-}6\text{-}final\ (N, U_{er}, \mathcal{F}, \mathcal{M}, None) \mid$

contradiction-found:

$ord\text{-}res\text{-}6\text{-}final\ (N, U_{er}, \mathcal{F}, \mathcal{M}, Some\ \{\#\})$

sublocale *ord-res-6-antics*: *antics* **where**

step = *ord-res-6-step* **and**

final = *ord-res-6-final*

$\langle proof \rangle$

lemma *ord-res-6-preserves-invars*:

assumes *step*: $ord\text{-}res\text{-}6\ N\ s\ s'$ **and** *invars*: *ord-res-5-invars* $N\ s$

shows *ord-res-5-invars* $N\ s'$

$\langle proof \rangle$

lemma *rtranclp-ord-res-6-preserves-invars*:

assumes *steps*: $(ord\text{-}res\text{-}6\ N)^{**}\ s\ s'$ **and** *invars*: *ord-res-5-invars* $N\ s$

shows *ord-res-5-invars* $N\ s'$

$\langle proof \rangle$

lemma *ex-ord-res-6-if-not-final*:

assumes

not-final: $\neg\ ord\text{-}res\text{-}6\text{-}final\ S$ **and**

invars: $\forall\ N\ s.\ S = (N, s) \longrightarrow ord\text{-}res\text{-}5\text{-}invars\ N\ s$

shows $\exists\ S'.\ ord\text{-}res\text{-}6\text{-}step\ S\ S'$

$\langle proof \rangle$

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>

lemma *ex-model-build-from-least-clause-to-any-less-than-least-false:*

assumes

F-subset: $\mathcal{F} \mid\subseteq\mid N \mid\cup\mid U_{er}$ **and**

C-least: *linorder-cls.is-least-in-fset* (*iefac* $\mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$) *C* **and**

D-in: $D \mid\in\mid \text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})$ **and**

D-lt-least-false: $\forall E. \text{is-least-false-clause } (\text{iefac } \mathcal{F} \mid\uparrow\mid (N \mid\cup\mid U_{er})) E \longrightarrow D \preceq_c$

E **and**

$C \preceq_c D$

shows $\exists \mathcal{M}. (\text{ord-res-5 } N)^{**} (U_{er}, \mathcal{F}, \text{Map.empty}, \text{Some } C) (U_{er}, \mathcal{F}, \mathcal{M}, \text{Some } D)$

<proof>

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-cls.is-least-in-fset* (*iefac* $\mathcal{F}' \mid\uparrow\mid (N \mid\cup\mid U_{er}')$) *C*

shows $(\text{ord-res-5 } N)^{**} (U_{er}', \mathcal{F}', \text{Map.empty}, \text{Some } C) (U_{er}', \mathcal{F}', \mathcal{M}', \text{Some } D)$

<proof>

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:

$\neg \text{trail-false-cls } \Gamma C \Longrightarrow$

linorder-lit.is-maximal-in-mset $C L \Longrightarrow$

linorder-trm.is-least-in-fset $\{|A \mid\in\mid \text{atms-of-clss } (N \mid\cup\mid U_{er})\}.$

$A \prec_t \text{atm-of } L \wedge A \notin \text{trail-atms } \Gamma\} A \Longrightarrow$

$\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \Longrightarrow$

ord-res-7 $N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma', \text{Some } C) \mid$

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:

$$\begin{aligned}
& \text{trail-false-cls } \Gamma E \implies \\
& \text{linorder-lit.is-maximal-in-mset } E L \implies \\
& \text{is-neg } L \implies \\
& \text{map-of } \Gamma (- L) = \text{Some } (\text{Some } D) \implies \\
& U_{er}' = \text{finsert } (\text{eres } D E) U_{er} \implies \\
& \text{eres } D E \neq \{\#\} \implies \\
& \Gamma' = \text{dropWhile } (\lambda Ln. \text{atm-of } K \preceq_t \text{atm-of } (\text{fst } Ln)) \Gamma \implies \\
& \text{linorder-lit.is-maximal-in-mset } (\text{eres } D E) K \implies \\
& \text{is-neg } K \implies \\
& \text{map-of } \Gamma (- K) = \text{Some } (\text{Some } C) \implies
\end{aligned}$$

$ord\text{-}res\text{-}7\ N\ (U_{er}, \mathcal{F}, \Gamma, \text{Some } E)\ (U_{er}', \mathcal{F}, \Gamma', \text{Some } C)$

lemma *right-unique-ord-res-7*:

fixes $N :: 'f\ gclause\ fset$

shows *right-unique* ($ord\text{-}res\text{-}7\ N$)

$\langle proof \rangle$

inductive *ord-res-7-final* **where**

model-found:

$ord\text{-}res\text{-}7\text{-}final\ (N, U_{er}, \mathcal{F}, \Gamma, None) \mid$

contradiction-found:

$ord\text{-}res\text{-}7\text{-}final\ (N, U_{er}, \mathcal{F}, \Gamma, \text{Some } \{\#\})$

sublocale *ord-res-7-antics: semantics* **where**

step = *constant-context ord-res-7* **and**

final = *ord-res-7-final*

$\langle proof \rangle$

inductive *ord-res-7-invars* **for** N **where**

ord-res-7-invars $N\ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ **if**

$\mathcal{F} \mid\subseteq\ N \mid\cup\ U_{er}$ **and**

$(\forall C. \mathcal{C} = \text{Some } C \longrightarrow C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}))$ **and**

$(\forall D. \mathcal{C} = \text{Some } D \longrightarrow$

$(\forall C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}). C \prec_c D \longrightarrow$

$(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C\ L_C \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma\ L_C \longrightarrow (\text{trail-true-cls } \Gamma\ \{\#K \in\# C. K \neq L_C\# \}$

$\wedge \text{is-pos } L_C)))$ **and**

$(\forall C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$\text{trail-true-cls } \Gamma\ C)$ **and**

$(\forall C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}). (\forall D. \mathcal{C} = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C\ K \longrightarrow$

$\neg (\exists A \mid\in\ \text{atms-of-clss } (N \mid\cup\ U_{er}). A \prec_t \text{atm-of } K \wedge A \not\in\ \text{trail-atms}$

$\Gamma)))$ **and**

sorted-wrt $(\lambda x\ y. \text{atm-of } (fst\ y) \prec_t \text{atm-of } (fst\ x))\ \Gamma$ **and**

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**

$(\forall Ln \in\ \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$

$(\forall C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}). C \prec_c \{\#fst\ Ln\# \} \longrightarrow \text{trail-true-cls } \Gamma\ C))$

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**

$(\forall Ln \in\ \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid\in\ iefac\ \mathcal{F} \mid\uparrow\ (N \mid\cup\ U_{er}))$ **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

$$\begin{aligned} & (\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow \\ & \quad (\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C)) \end{aligned}$$

lemma *clause-almost-defined-if-lt-next-clause:*

assumes *ord-res-7-invars* $N (U_{er}, \mathcal{F}, \Gamma, C)$

shows $\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). (\forall D. C = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow \text{trail-defined-cls } \Gamma \{\#L \in \# C. L \neq K\#})$

<proof>

lemma *ord-res-7-invars-def:*

ord-res-7-invars $N s \longleftrightarrow$

$(\forall U_{er} \mathcal{F} \Gamma C. s = (U_{er}, \mathcal{F}, \Gamma, C) \longrightarrow$

$\mathcal{F} \subseteq | N \cup U_{er} \wedge$

$(\forall C. C = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er})) \wedge$

$(\forall D. C = \text{Some } D \longrightarrow$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). C \prec_c D \longrightarrow$

$(\forall L_C. \text{linorder-lit.is-maximal-in-mset } C L_C \longrightarrow$

$\neg \text{trail-defined-lit } \Gamma L_C \longrightarrow (\text{trail-true-cls } \Gamma \{\#K \in \# C. K \neq L_C\#}$

$\wedge \text{is-pos } L_C)))) \wedge$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). (\forall D. C = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$\text{trail-true-cls } \Gamma C) \wedge$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). (\forall D. C = \text{Some } D \longrightarrow C \prec_c D) \longrightarrow$

$(\forall K. \text{linorder-lit.is-maximal-in-mset } C K \longrightarrow$

$\neg (\exists A \in | \text{atms-of-clss } (N \cup U_{er}). A \prec_t \text{atm-of } K \wedge A \notin | \text{trail-atms}$

$\Gamma)))) \wedge$

$\text{sorted-wrt } (\lambda x y. \text{atm-of } (fst y) \prec_t \text{atm-of } (fst x)) \Gamma \wedge$

$\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \cup U_{er})) \wedge$

$(C = \text{None} \longrightarrow \text{trail-atms } \Gamma = \text{atms-of-clss } (N \cup U_{er})) \wedge$

$(\forall C. C = \text{Some } C \longrightarrow$

$(\forall A \in | \text{trail-atms } \Gamma. \exists L. \text{linorder-lit.is-maximal-in-mset } C L \wedge A \preceq_t \text{atm-of}$

$L)) \wedge$

$(\forall Ln \in \text{set } \Gamma. \text{is-neg } (fst Ln) \longleftrightarrow \text{snd } Ln = \text{None}) \wedge$

$(\forall Ln \in \text{set } \Gamma. \text{snd } Ln = \text{None} \longrightarrow$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). C \prec_c \{\#fst Ln\#} \longrightarrow \text{trail-true-cls } \Gamma C))$

\wedge

$(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C$

$(fst Ln)) \wedge$

$(\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er})) \wedge$

$(\forall \Gamma_1 L C \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } C) \# \Gamma_0 \longrightarrow \text{trail-false-cls } \Gamma_0 \{\#K \in \#$

$C. K \neq L\#\}) \wedge$

$(\forall \Gamma_1 L D \Gamma_0. \Gamma = \Gamma_1 @ (L, \text{Some } D) \# \Gamma_0 \longrightarrow$

$(\forall C \in | \text{iefac } \mathcal{F} \uparrow | (N \cup U_{er}). C \prec_c D \longrightarrow \text{trail-true-cls } \Gamma_0 C))$

(is ?NICE $N s \longleftrightarrow ?UGLY N s)$

<proof>

lemma *ord-res-7-invars-implies-trail-consistent:*

assumes *ord-res-7-invars* $N (U_{er}, \mathcal{F}, \Gamma, C)$

shows *trail-consistent* Γ
 ⟨*proof*⟩

lemma *ord-res- γ -invars-implies-propagated-clause-almost-false*:
assumes *invars*: *ord-res- γ -invars* N ($U_{er}, \mathcal{F}, \Gamma, \mathcal{C}$) **and** ($L, \text{Some } C$) \in *set* Γ
shows *trail-false-cls* Γ $\{\#K \in\# C. K \neq L\#$
 ⟨*proof*⟩

lemma *ord-res- γ -preserves-invars*:
assumes *step*: *ord-res- γ* N s s' **and** *invar*: *ord-res- γ -invars* N s
shows *ord-res- γ -invars* N s'
 ⟨*proof*⟩

lemma *rtranclp-ord-res- γ -preserves-ord-res- γ -invars*:
assumes
 step: (*ord-res- γ* N)^{**} s s' **and**
 invars: *ord-res- γ -invars* N s
shows *ord-res- γ -invars* N s'
 ⟨*proof*⟩

lemma *tranclp-ord-res- γ -preserves-ord-res- γ -invars*:
assumes
 step: (*ord-res- γ* N)⁺⁺ s s' **and**
 invars: *ord-res- γ -invars* N s
shows *ord-res- γ -invars* N s'
 ⟨*proof*⟩

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 *set* Γ
shows *trail-false-cls* Γ $\{\#K \in\# C. K \neq L\#$
 ⟨*proof*⟩

lemma *ex-ord-res- γ -if-not-final*:
assumes
 not-final: \neg *ord-res- γ -final* (N, s) **and**
 invars: *ord-res- γ -invars* N s
shows $\exists s'. \text{ord-res-}\gamma$ N s s'
 ⟨*proof*⟩

lemma *ord-res- γ -safe-state-if-invars*:
fixes $N :: 'f$ *gclause* *fset* **and** s
assumes *invars*: *ord-res- γ -invars* N s
shows *safe-state* (*constant-context* *ord-res- γ*) *ord-res- γ -final* (N, s)
 ⟨*proof*⟩

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  $C L \wedge \text{trail-false-clc } \Gamma \{\#K \in\# C. K \neq L\# \}$ 

lemma trail-false-if-could-have-propagated:
  clause-could-propagate  $\Gamma C L \implies \text{trail-false-clc } ((- L, n) \# \Gamma) C$ 
  <proof>

lemma atoms-of-trail-lt-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>

lemma trail-false-clc-filter-mset-iff:
  trail-false-clc  $\Gamma \{\#Ka \in\# C. Ka \neq K\# \} \longleftrightarrow (\forall L \in\# C. L \neq K \longrightarrow \text{trail-false-lit } \Gamma L)$ 
  <proof>

lemma clause-could-propagate-iff: clause-could-propagate  $\Gamma C K \longleftrightarrow$ 
   $\neg \text{trail-defined-lit } \Gamma K \wedge \text{ord-res.is-maximal-lit } K C \wedge (\forall L \in\# C. L \neq K \longrightarrow$ 
  trail-false-lit  $\Gamma L)$ 
  <proof>

lemma clause-could-propagate-efac: clause-could-propagate  $\Gamma (\text{efac } C) = \text{clause-could-propagate } \Gamma C$ 
  <proof>

lemma bex-clause-could-propagate-simp:
fixes  $\mathcal{F} N \Gamma L$ 
shows  $fBex (\text{iefac } \mathcal{F} \mid \uparrow N) (\lambda C. \text{clause-could-propagate } \Gamma C L) \longleftrightarrow$ 
   $fBex N (\lambda C. \text{clause-could-propagate } \Gamma C L)$ 
sketch (rule iffI; elim bexE)
  <proof>

end

end

theory ORD-RES-8
imports

```

Background
 Implicit-Exhaustive-Factorization
 Exhaustive-Resolution
 Clause-Could-Propagate

begin

23 ORD-RES-8 (atom-guided literal trail construction)

type-synonym *'f ord-res-8-state* =
'f gclause fset × *'f gclause fset* × *'f gclause fset* × (*'f gliteral* × *'f gclause option*)
list

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

inductive *ord-res-8* **where**

decide-neg:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C (Pos A)) \implies \\ & \Gamma' = (Neg A, None) \# \Gamma \implies \\ & \text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid \end{aligned}$$

propagate:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A)\} C \implies \\ & \text{linorder-lit.is-greatest-in-mset } C (Pos A) \implies \\ & \Gamma' = (Pos A, Some C) \# \Gamma \implies \\ & \text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid \end{aligned}$$

factorize:

$$\begin{aligned} & \neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies \\ & \text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}). \\ & \quad \forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies \\ & \text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \\ & \quad \text{clause-could-propagate } \Gamma C (Pos A)\} C \implies \\ & \neg \text{linorder-lit.is-greatest-in-mset } C (Pos A) \implies \\ & \mathcal{F}' = \text{finsert } C \mathcal{F} \implies \\ & \text{ord-res-8 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma) \mid \end{aligned}$$

resolution:

$$\begin{aligned} & \text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma D\} \\ & D \implies \\ & \text{linorder-lit.is-maximal-in-mset } D (Neg A) \implies \end{aligned}$$

$map-of \Gamma (Pos A) = Some (Some C) \implies$
 $U_{er}' = finsert (eres C D) U_{er} \implies$
 $\Gamma' = dropWhile (\lambda Ln. \forall K.$
 $linorder-lit.is-maximal-in-mset (eres C D) K \longrightarrow atm-of K \preceq_t atm-of (fst$
 $Ln)) \Gamma \implies$
 $ord-res-8 N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

lemma *right-unique-ord-res-8*:

fixes $N :: 'f gclause fset$

shows *right-unique (ord-res-8 N)*

<proof>

inductive *ord-res-8-final* :: *'f ord-res-8-state \Rightarrow bool where*

model-found:

$\neg (\exists A | \in | atms-of-cls (N | \cup | U_{er}). A | \notin | trail-atms \Gamma) \implies$

$\neg (\exists C | \in | iefac \mathcal{F} | \uparrow | (N | \cup | U_{er}). trail-false-cls \Gamma C) \implies$

ord-res-8-final (N, U_{er}, \mathcal{F} , Γ) |

contradiction-found:

$\{\#\} | \in | iefac \mathcal{F} | \uparrow | (N | \cup | U_{er}) \implies$

ord-res-8-final (N, U_{er}, \mathcal{F} , Γ)

sublocale *ord-res-8-antics: semantics where*

step = constant-context ord-res-8 and

final = ord-res-8-final

<proof>

definition *trail-is-sorted where*

trail-is-sorted N s \longleftrightarrow

$(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow$

sorted-wrt ($\lambda x y. atm-of (fst y) \prec_t atm-of (fst x)$) Γ)

lemma *ord-res-8-preserves-trail-is-sorted*:

assumes

step: ord-res-8 N s s' and

invar: trail-is-sorted N s

shows *trail-is-sorted N s'*

<proof>

inductive *trail-annotations-invars*

for $N :: 'f gterm literal multiset fset$

where

Nil:

trail-annotations-invars N (U_{er}, \mathcal{F} , \square) |

Cons-None:

trail-annotations-invars N (U_{er}, \mathcal{F} , (L, None) # Γ)

if *trail-annotations-invars N (U_{er}, \mathcal{F} , Γ) |*

Cons-Some:

trail-annotations-invars N (U_{er}, \mathcal{F} , (L, Some D) # Γ)

if *linorder-lit.is-greatest-in-mset* $D L$ **and**
 $D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$ **and**
trail-false-cls $\Gamma \{ \#K \in \# D. K \neq L \# \}$ **and**
linorder-cls.is-least-in-fset
 $\{ \mid D \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ clause-could-propagate } \Gamma D L \}$ 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* $\Gamma C L$
 $\langle \text{proof} \rangle$

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 $C \in \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})$
 $\langle \text{proof} \rangle$

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* $N (U_{er}, \mathcal{F}, \Gamma')$
 $\langle \text{proof} \rangle$

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* $N s'$
 $\langle \text{proof} \rangle$

definition *trail-is-lower-set* **where**

trail-is-lower-set $N s \iff$

$(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \implies$

$\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-cls } (N \mid \cup \mid U_{er})))$

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 $\forall A. A \notin \text{atms-of-cls } (N \mid \cup \mid U_{er}) \implies A \notin \text{trail-atms } \Gamma$
 $\langle \text{proof} \rangle$

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* $N s'$
 ⟨*proof*⟩

definition *false-cls-is-mempty-or-has-neg-max-lit* **where**

false-cls-is-mempty-or-has-neg-max-lit $N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow (\forall C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{trail-false-cls } \Gamma C \longrightarrow C = \{\#\} \vee (\exists A. \text{linorder-lit.is-maximal-in-mset } C$
 $(\text{Neg } A))))$

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* $N s'$
 ⟨*proof*⟩

definition *decided-literals-all-neg* **where**

decided-literals-all-neg $N s \longleftrightarrow$
 $(\forall U_{er} \mathcal{F} \Gamma. s = (U_{er}, \mathcal{F}, \Gamma) \longrightarrow$
 $(\forall Ln \in \text{set } \Gamma. \forall L. Ln = (L, \text{None}) \longrightarrow \text{is-neg } L))$

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* $N s'$
 ⟨*proof*⟩

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* $N s'$
 ⟨*proof*⟩

lemma *rtranclp-ord-res-8-preserves-invars*:

assumes

step: $(ord-res-8\ N)^{**}\ s\ s'$ **and**

invars: $ord-res-8-invars\ N\ s$

shows $ord-res-8-invars\ N\ s'$

<proof>

lemma *tranclp-ord-res-8-preserves-invars*:

assumes

step: $(ord-res-8\ N)^{++}\ s\ s'$ **and**

invars: $ord-res-8-invars\ N\ s$

shows $ord-res-8-invars\ N\ s'$

<proof>

lemma *ex-ord-res-8-if-not-final*:

assumes

not-final: $\neg\ ord-res-8-final\ (N,\ s)$ **and**

invars: $ord-res-8-invars\ N\ s$

shows $\exists\ s'.\ ord-res-8\ N\ s\ s'$

<proof>

lemma *ord-res-8-safe-state-if-invars*:

fixes $N\ s$

assumes *invars*: $ord-res-8-invars\ N\ s$

shows *safe-state* (*constant-context* $ord-res-8$) $ord-res-8-final\ (N,\ s)$

<proof>

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:

$\neg\ (\exists\ C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}).\ trail-false-cls\ \Gamma\ C) \implies$

$linorder-trm.is-least-in-fset\ \{|A_2\ |\in|\ atms-of-class\ (N\ |\cup|\ U_{er}).$

$\forall\ A_1\ |\in|\ trail-atms\ \Gamma.\ A_1\ \prec_t\ A_2|\}\ A \implies$

$\neg\ (\exists\ C\ |\in|\ iefac\ \mathcal{F}\ |\uparrow|\ (N\ |\cup|\ U_{er}).\ clause-could-propagate\ \Gamma\ C\ (Pos\ A)) \implies$

$\Gamma' = (Neg\ A,\ None)\ \#\ \Gamma \implies$

$ord-res-9\ N\ (U_{er},\ \mathcal{F},\ \Gamma)\ (U_{er},\ \mathcal{F},\ \Gamma')\ |$

propagate:

$\neg (\exists C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \in | \text{atms-of-cls } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-cls.is-least-in-fset } \{|C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{clause-could-propagate } \Gamma C (\text{Pos } A)\} C \implies$
 $\Gamma' = (\text{Pos } A, \text{Some } (\text{efac } C)) \# \Gamma \implies$
 $\mathcal{F}' = (\text{if } \text{linorder-lit.is-greatest-in-mset } C (\text{Pos } A) \text{ then } \mathcal{F} \text{ else } \text{finsert } C \mathcal{F}) \implies$
 $\text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid$

resolution:

$\text{linorder-cls.is-least-in-fset } \{|D \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma D\}$
 $D \implies$
 $\text{linorder-lit.is-maximal-in-mset } D (\text{Neg } A) \implies$
 $\text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C) \implies$
 $U_{er}' = \text{finsert } (\text{eres } C D) U_{er} \implies$
 $\Gamma' = \text{dropWhile } (\lambda Ln. \forall K.$
 $\text{linorder-lit.is-maximal-in-mset } (\text{eres } C D) K \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst}$
 $\text{Ln})) \Gamma \implies$
 $\text{ord-res-9 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

lemma *right-unique-ord-res-9:*

fixes $N :: 'f \text{ gclause fset}$
shows *right-unique* ($\text{ord-res-9 } N$)
 $\langle \text{proof} \rangle$

lemma *ord-res-9-is-one-or-two-ord-res-9-steps:*

fixes $N s s'$
assumes *step:* $\text{ord-res-9 } N s s'$
shows $\text{ord-res-8 } N s s' \vee (\text{ord-res-8 } N \text{ OO } \text{ord-res-8 } N) s s'$
 $\langle \text{proof} \rangle$

lemma *ord-res-9-preserves-invars:*

assumes
step: $\text{ord-res-9 } N s s'$ **and**
invars: $\text{ord-res-8-invars } N s$
shows $\text{ord-res-8-invars } N s'$
 $\langle \text{proof} \rangle$

lemma *rtranclp-ord-res-9-preserves-ord-res-8-invars:*

assumes
step: $(\text{ord-res-9 } N)^{**} s s'$ **and**
invars: $\text{ord-res-8-invars } N s$
shows $\text{ord-res-8-invars } N s'$
 $\langle \text{proof} \rangle$

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>

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)
 <proof>

sublocale *ord-res-9-semantic*: *semantic* **where**
step = *constant-context ord-res-9* **and**
final = *ord-res-8-final*
 <proof>

end

end

theory *ORD-RES-10*
imports *ORD-RES-8*
begin

25 ORD-RES-10 (propagate iff a conflict is produced)

type-synonym *'f ord-res-10-state* =
'f gclause fset \times *'f gclause fset* \times *'f gclause fset* \times (*'f gliteral* \times *'f gclause option*)
list

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

inductive *ord-res-10* **where**

decide-neg:
 $\neg (\exists C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\neg (\exists C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C (\text{Pos } A)) \implies$
 $\Gamma' = (\text{Neg } A, \text{None}) \# \Gamma \implies$
 $\text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}, \Gamma') \mid$

decide-pos:
 $\neg (\exists C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \in | \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \in | \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-clc.is-least-in-fset } \{|C \in | \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{clause-could-propagate } \Gamma C (\text{Pos } A)\} C \implies$
 $\Gamma' = (\text{Pos } A, \text{None}) \# \Gamma \implies$

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma' C) \implies$
 $\mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (\text{Pos } A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies$
 $\text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid$

propagate:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\} A \implies$
 $\text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{clause-could-propagate } \Gamma C (\text{Pos } A)\} C \implies$
 $\Gamma' = (\text{Pos } A, \text{Some } (\text{efac } C)) \# \Gamma \implies$
 $(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma' C) \implies$
 $\mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (\text{Pos } A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies$
 $\text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}, \mathcal{F}', \Gamma') \mid$

resolution:

$\text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma D\}$
 $D \implies$
 $\text{linorder-lit.is-maximal-in-mset } D (\text{Neg } A) \implies$
 $\text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C) \implies$
 $U_{er}' = \text{finsert } (\text{eres } C D) U_{er} \implies$
 $\Gamma' = \text{dropWhile } (\lambda Ln. \forall K.$
 $\text{linorder-lit.is-maximal-in-mset } (\text{eres } C D) K \longrightarrow \text{atm-of } K \preceq_t \text{atm-of } (\text{fst}$
 $\text{Ln})) \Gamma \implies$
 $\text{ord-res-10 } N (U_{er}, \mathcal{F}, \Gamma) (U_{er}', \mathcal{F}, \Gamma')$

lemma *right-unique-ord-res-10:*

fixes $N :: 'f \text{ gclause } fset$
shows *right-unique (ord-res-10 N)*
 $\langle \text{proof} \rangle$

sublocale *ord-res-10-semantics: semantics where*

$\text{step} = \text{constant-context ord-res-10}$ **and**
 $\text{final} = \text{ord-res-8-final}$
 $\langle \text{proof} \rangle$

inductive *ord-res-10-invars for N where*

$\text{ord-res-10-invars } N (U_{er}, \mathcal{F}, \Gamma)$ **if**
 $\text{sorted-wrt } (\lambda x y. \text{atm-of } (\text{fst } y) \prec_t \text{atm-of } (\text{fst } x)) \Gamma$ **and**
 $\forall Ln \in \text{set } \Gamma. \forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{linorder-lit.is-greatest-in-mset } C (\text{fst}$
 $\text{Ln})$ **and**
 $\text{linorder-trm.is-lower-fset } (\text{trail-atms } \Gamma) (\text{atms-of-clss } (N \mid \cup \mid U_{er}))$ **and**
 $\forall Ln \Gamma'. \Gamma = Ln \# \Gamma' \longrightarrow$
 $(\text{snd } Ln \neq \text{None} \longleftrightarrow (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{ trail-false-cls } \Gamma C))$
 \wedge
 $(\text{snd } Ln \neq \text{None} \longrightarrow \text{is-pos } (\text{fst } Ln)) \wedge$
 $(\forall C. \text{snd } Ln = \text{Some } C \longrightarrow C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er})) \wedge$
 $(\forall C. \text{snd } Ln = \text{Some } C \longrightarrow \text{clause-could-propagate } \Gamma' C (\text{fst } Ln)) \wedge$
 $(\forall x \in \text{set } \Gamma'. \text{snd } x = \text{None})$ **and**

$\forall \Gamma_1 Ln \Gamma_0. \Gamma = \Gamma_1 @ Ln \# \Gamma_0 \longrightarrow$
 $snd Ln = None \longrightarrow \neg(\exists C | \in | iefac \mathcal{F} | \uparrow (N | \cup | U_{er}). trail-false-cls (Ln \#$
 $\Gamma_0) C)$

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'*

<proof>

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'*

<proof>

lemma *ex-ord-res-10-if-not-final*:

assumes

not-final: \neg *ord-res-8-final* (*N, s*) **and**

invars: *ord-res-10-invars* *N s*

shows $\exists s'. \text{ord-res-10 } N s s'$

<proof>

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*)

<proof>

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} F Γ A*

assumes

no-false-cls: $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C)$ **and**
A-least: $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\}$ **A and**
C-least: $\text{linorder-cls.is-least-in-fset } \{|C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}).$
 $\text{clause-could-propagate } \Gamma C (Pos A)\}$ **C**

defines

$\Gamma' \equiv (Pos A, None) \# \Gamma$ **and**
 $\mathcal{F}' \equiv (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F})$

shows

$(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C) \longleftrightarrow$
 $(\exists C \mid \in \mid \text{iefac } \mathcal{F}' \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C)$
<proof>

inductive ord-res-11 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
 $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{clause-could-propagate } \Gamma C (Pos A)) \implies$
 $\Gamma' = (Neg A, None) \# \Gamma \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}, \Gamma', None) \mid$

decide-pos:

$\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma C) \implies$
 $\text{linorder-trm.is-least-in-fset } \{|A_2 \mid \in \mid \text{atms-of-clss } (N \mid \cup \mid U_{er}).$
 $\forall A_1 \mid \in \mid \text{trail-atms } \Gamma. A_1 \prec_t A_2\}$ **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
 $\Gamma' = (Pos A, None) \# \Gamma \implies$
 $\neg (\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C) \implies$
 $\mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}', \Gamma', None) \mid$

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
 $\Gamma' = (Pos A, Some (\text{efac } C)) \# \Gamma \implies$
 $(\exists C \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma' C) \implies$
 $\mathcal{F}' = (\text{if linorder-lit.is-greatest-in-mset } C (Pos A) \text{ then } \mathcal{F} \text{ else finsert } C \mathcal{F}) \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}', \Gamma', None) \mid$

conflict:

$\text{linorder-cls.is-least-in-fset } \{|D \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow (N \mid \cup \mid U_{er}). \text{trail-false-cls } \Gamma D\}$
 $D \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, None) (U_{er}, \mathcal{F}, \Gamma, Some D) \mid$

skip: $- L \notin \# C \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, (L, n) \# \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) \mid$

resolution:

$\Gamma = (L, \text{Some } D) \# \Gamma' \implies - L \in \# C \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (U_{er}, \mathcal{F}, \Gamma, \text{Some } ((C - \{\# - L\# \}) + (D$
 $- \{\# L\# \}))) \mid$

backtrack:

$\Gamma = (L, \text{None}) \# \Gamma' \implies - L \in \# C \implies$
 $\text{ord-res-11 } N (U_{er}, \mathcal{F}, \Gamma, \text{Some } C) (\text{finsert } C U_{er}, \mathcal{F}, \Gamma', \text{None})$

lemma *right-unique-ord-res-11*:

fixes $N :: 'f \text{ gclause } fset$

shows *right-unique* ($\text{ord-res-11 } N$)

$\langle \text{proof} \rangle$

inductive *ord-res-11-final* :: $'f \text{ ord-res-11-state} \Rightarrow \text{bool}$ **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*: *semantic* **where**

step = *constant-context ord-res-11* **and**

final = *ord-res-11-final*

$\langle \text{proof} \rangle$

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**

$\{\#\} \mid \in \mid N \mid \cup \mid U_{er} \longrightarrow \Gamma = []$ **and**

$\forall C. \mathcal{C} = \text{Some } C \longrightarrow \text{atms-of-cls } C \mid \subseteq \mid \text{atms-of-clss } (N \mid \cup \mid U_{er})$ **and**

$\forall C. \mathcal{C} = \text{Some } C \longrightarrow \text{trail-false-cls } \Gamma C$ **and**

atms-of-clss $U_{er} \mid \subseteq \mid \text{atms-of-clss } N$ **and**

$\forall C \mid \in \mid \mathcal{F}. \exists L. \text{is-pos } L \wedge \text{linorder-lit.is-maximal-in-mset } C L$

lemma *ord-res-11-invars-initial-state*: *ord-res-11-invars* $N (\{\mid\}, \{\mid\}, [], \text{None})$

$\langle \text{proof} \rangle$

lemma *mempty-in-fimage-iefac[simp]*: $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid \uparrow \mid N \longleftrightarrow \{\#\} \mid \in \mid N$

$\langle \text{proof} \rangle$

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'*

<proof>

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'*

<proof>

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'*

<proof>

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 $\exists s'. \text{ord-res-11 } N s s'$

<proof>

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*)

<proof>

lemma *rtrancl-ord-res-11-all-resolution-steps*:

assumes *C-max-lit*: *ord-res.is-strictly-maximal-lit* *K C*

shows (*ord-res-11 N*)^{**} (*U, F, (K, Some C) # Γ, Some D*) (*U, F, (K, Some C) # Γ, Some (eres C D)*)

<proof>

lemma *rtrancl-ord-res-11-all-skip-steps*:

(*ord-res-11 N*)^{**} (*U, F, Γ, Some C*) (*U, F, dropWhile* ($\lambda Ln. - \text{fst } Ln \notin C$) *Γ, Some C*)

<proof>

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>

end

28 ORD-RES-2 (full factorization)

type-synonym *'f ord-res-2-state = 'f gclause fset × 'f gclause fset × 'f gclause fset*

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

fun *ord-res-1-matches-ord-res-2*

:: 'f ord-res-1-state ⇒ - ⇒ bool **where**

ord-res-1-matches-ord-res-2 S1 (N, (U_r, U_{ef})) ⟷ (∃ U_f.

S1 = N |∪| U_r |∪| U_{ef} |∪| U_f ∧

*(∀ C_f |∈| U_f. ∃ C |∈| N |∪| U_r |∪| U_{ef}. *ord-res.ground-factoring*⁺⁺ C C_f ∧*

$C_f \neq \text{efac } C_f \wedge$
 $(\text{efac } C_f \mid \in \mid U_{ef} \vee \text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef}) C))$

lemma *ord-res-1-matches-ord-res-2-simps'*:

ord-res-1-matches-ord-res-2 $S1 (N, (U_r, U_{ef})) \longleftrightarrow$
 $(\exists U_f. S1 = N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f \wedge$
 $(\forall C_f \mid \in \mid U_f. C_f \neq \text{efac } C_f \wedge (\exists C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef}. \text{ord-res.ground-factoring}^{++}$
 $C C_f \wedge$
 $(\text{efac } C_f \mid \in \mid U_{ef} \vee \text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef}) C))))$
 $\langle \text{proof} \rangle$

lemma *ord-res-1-matches-ord-res-2-simps''*:

ord-res-1-matches-ord-res-2 $S1 (N, (U_r, U_{ef})) \longleftrightarrow$
 $(\exists U_f. S1 = N \mid \cup \mid U_r \mid \cup \mid U_{ef} \mid \cup \mid U_f \wedge$
 $(\forall C_f \mid \in \mid U_f. C_f \neq \text{efac } C_f \wedge (\exists C \mid \in \mid N \mid \cup \mid U_r \mid \cup \mid U_{ef}. \text{ord-res.ground-factoring}^{++}$
 $C C_f \wedge$
 $(\text{efac } C \mid \in \mid U_{ef} \vee \text{is-least-false-clause } (N \mid \cup \mid U_r \mid \cup \mid U_{ef}) C))))$
 $\langle \text{proof} \rangle$

lemma *ord-res-1-final-iff-ord-res-2-final*:

assumes *match*: *ord-res-1-matches-ord-res-2* $S_1 S_2$
shows *ord-res-1-final* $S_1 \longleftrightarrow \text{ord-res-2-final } S_2$
 $\langle \text{proof} \rangle$

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* $S_1 \wedge \text{safe-state ord-res-2-step ord-res-2-final } S_2$
 $\langle \text{proof} \rangle$

definition *ord-res-1-measure* **where**

ord-res-1-measure $s1 =$
 $(\text{if } \exists C. \text{is-least-false-clause } s1 C \text{ then}$
 $\quad \text{The } (\text{is-least-false-clause } s1)$
 else
 $\quad \{\#\})$

lemma *forward-simulation*:

assumes *match*: *ord-res-1-matches-ord-res-2* $s1 s2$ **and**
 $\text{step1: ord-res-1 } s1 s1'$
shows $(\exists s2'. \text{ord-res-2-step}^{++} s2 s2' \wedge \text{ord-res-1-matches-ord-res-2 } s1' s2') \vee$
 $\text{ord-res-1-matches-ord-res-2 } s1' s2 \wedge \text{ord-res-1-measure } s1' \subset \# \text{ ord-res-1-measure } s1$
 $\langle \text{proof} \rangle$

theorem *bisimulation-ord-res-1-ord-res-2*:

defines *match* $\equiv \lambda i s1 s2. i = \text{ord-res-1-measure } s1 \wedge \text{ord-res-1-matches-ord-res-2 } s1 s2$
shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ ord-res-1-state} \Rightarrow 'f \text{ ord-res-2-state} \Rightarrow \text{bool})$

$\mathcal{R}_f \mathcal{R}_b$.
bisimulation ord-res-1 ord-res-1-final ord-res-2-step ord-res-2-final MATCH \mathcal{R}_f
 \mathcal{R}_b
 ⟨proof⟩
 end

29 ORD-RES-3 (full resolve)

type-synonym 'f ord-res-3-state = 'f gclause fset × 'f gclause fset × 'f gclause fset

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

inductive *ord-res-2-matches-ord-res-3* :: - ⇒ 'f ord-res-3-state ⇒ bool **where**
 (∀ C |∈| U_{pr}. ∃ D1 |∈| N |∪| U_{er} |∪| U_{ef}. ∃ D2 |∈| N |∪| U_{er} |∪| U_{ef}.
 (ground-resolution D1)⁺⁺ D2 C ∧ C ≠ eres D1 D2 ∧ eres D1 D2 |∈| U_{er})
 ⇒
ord-res-2-matches-ord-res-3 (N, (U_{pr} |∪| U_{er}, U_{ef})) (N, (U_{er}, U_{ef}))

lemma *ord-res-2-final-iff-ord-res-3-final*:
assumes *match*: *ord-res-2-matches-ord-res-3* S₂ S₃
shows *ord-res-2-final* S₂ ↔ *ord-res-3-final* S₃
 ⟨proof⟩

definition *ord-res-2-measure* **where**
ord-res-2-measure S1 =
 (let (N, (U_r, U_{ef})) = S1 in
 (if ∃ C. *is-least-false-clause* (N |∪| U_r |∪| U_{ef}) C then
 The (*is-least-false-clause* (N |∪| U_r |∪| U_{ef}))
 else
 {#}))

definition *resolvent-at* **where**
resolvent-at C D i = (THE CD. (ground-resolution C $\overset{\sim}{\sim}$ i) D CD)

lemma *resolvent-at-0[simp]*: *resolvent-at* C D 0 = D
 ⟨proof⟩

lemma *resolvent-at-less-cls-resolvent-at*:
assumes *reso-at*: (ground-resolution C $\overset{\sim}{\sim}$ n) D CD
assumes i < j **and** j ≤ n
shows *resolvent-at* C D j <_c *resolvent-at* C D i
 ⟨proof⟩

lemma
assumes *reso-at*: (ground-resolution C $\overset{\sim}{\sim}$ n) D CD **and** i < n
shows
left-premise-lt-resolvent-at: C <_c *resolvent-at* C D i **and**

max-lit-resolvent-at:

ord-res.is-maximal-lit $L D \implies \text{ord-res.is-maximal-lit } L (\text{resolvent-at } C D i)$

and

nex-pos-strictly-max-lit-in-resolvent-at:

$\nexists L. \text{is-pos } L \wedge \text{ord-res.is-strictly-maximal-lit } L (\text{resolvent-at } C D i)$ **and**

ground-resolution-resolvent-at-resolvent-at-Suc:

ground-resolution $C (\text{resolvent-at } C D i) (\text{resolvent-at } C D (\text{Suc } i))$ **and**

relpowp-to-resolvent-at: $(\text{ground-resolution } C \overset{\sim}{\sim} i) D (\text{resolvent-at } C D i)$
<proof>

definition *resolvents-upto where*

resolvents-upto $C D n = \text{resolvent-at } C D \mid^{\dagger} \text{fset-upto } (\text{Suc } 0) n$

lemma *resolvents-upto-0[simp]:*

resolvents-upto $C D 0 = \{\mid\}$

<proof>

lemma *resolvents-upto-Suc[simp]:*

resolvents-upto $C D (\text{Suc } n) = \text{finsert } (\text{resolvent-at } C D (\text{Suc } n)) (\text{resolvents-upto } C D n)$

<proof>

lemma *resolvent-at-fmember-resolvents-upto:*

assumes $k \neq 0$

shows *resolvent-at* $C D k \mid \in \mid \text{resolvents-upto } C D k$

<proof>

lemma *backward-simulation-2-to-3:*

fixes *match measure less*

defines *match* $\equiv \text{ord-res-2-matches-ord-res-3}$

assumes

match: *match* $S2 S3$ **and**

step2: *ord-res-3-step* $S3 S3'$

shows $(\exists S2'. \text{ord-res-2-step}^{++} S2 S2' \wedge \text{match } S2' S3')$

<proof>

lemma *safe-states-if-ord-res-2-matches-ord-res-3:*

assumes *match:* *ord-res-2-matches-ord-res-3* $S2 S3$

shows

safe-state *ord-res-2-step* *ord-res-2-final* $S2$

safe-state *ord-res-3-step* *ord-res-3-final* $S3$

<proof>

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 $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-2-state} \Rightarrow 'f \text{ord-res-3-state} \Rightarrow \text{bool})$

$\mathcal{R}_f \mathcal{R}_b.$

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>

end

30 ORD-RES-4 (implicit factorization)

type-synonym 'f ord-res-4-state = 'f gclause fset × 'f gclause fset × 'f gclause fset

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

inductive ord-res-3-matches-ord-res-4 :: 'f ord-res-3-state ⇒ 'f ord-res-4-state ⇒ bool **where**

$\mathcal{F} \mid \subseteq \mid N \mid \cup \mid U_{er} \implies U_{ef} = \text{iefac } \mathcal{F} \mid \{ \mid C \mid \in \mid N \mid \cup \mid U_{er} . \text{iefac } \mathcal{F} \ C \neq C \mid \} \implies$
ord-res-3-matches-ord-res-4 (N, (U_{er}, U_{ef})) (N, U_{er}, 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

<proof>

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 $(\exists S4' . \text{ord-res-4-step}^{++} S4 S4' \wedge \text{ord-res-3-matches-ord-res-4 } S3' S4')$

<proof>

theorem bisimulation-ord-res-3-ord-res-4:

defines match $\equiv \lambda . S3 S4 . \text{ord-res-3-matches-ord-res-4 } S3 S4$

shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-3-state} \Rightarrow 'f \text{ord-res-4-state} \Rightarrow \text{bool})$
 $\mathcal{R}_f \ \mathcal{R}_b .$

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>

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**

$ord\text{-}res\text{-}5\text{-}invars\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \implies$
 $(\forall C. C = Some\ C \longleftrightarrow is\text{-}least\text{-}false\text{-}clause\ (iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup\ U_{er}))\ C) \implies$
 $ord\text{-}res\text{-}4\text{-}matches\text{-}ord\text{-}res\text{-}5\ (N, U_{er}, \mathcal{F})\ (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$

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* $S_4 \longleftrightarrow ord\text{-}res\text{-}5\text{-}final\ S_5$
 $\langle proof \rangle$

lemma *forward-simulation-between-4-and-5*:
fixes $S_4\ S_4'\ S_5$
assumes *match*: *ord-res-4-matches-ord-res-5* $S_4\ S_5$ **and** *step*: *ord-res-4-step* $S_4\ S_4'$
shows $\exists S_5'. ord\text{-}res\text{-}5\text{-}step^{++}\ S_5\ S_5' \wedge ord\text{-}res\text{-}4\text{-}matches\text{-}ord\text{-}res\text{-}5\ S_4'\ S_5'$
 $\langle proof \rangle$

theorem *bisimulation-ord-res-4-ord-res-5*:
defines *match* $\equiv \lambda\cdot. ord\text{-}res\text{-}4\text{-}matches\text{-}ord\text{-}res\text{-}5$
shows $\exists (MATCH :: nat \times nat \Rightarrow 'f\ ord\text{-}res\text{-}4\text{-}state \Rightarrow 'f\ ord\text{-}res\text{-}5\text{-}state \Rightarrow bool)$
 $\mathcal{R}_f\ \mathcal{R}_b.$
bisimulation ord-res-4-step ord-res-4-final ord-res-5-step ord-res-5-final MATCH
 $\mathcal{R}_f\ \mathcal{R}_b$
 $\langle proof \rangle$

end

32 ORD-RES-6 (model backjump)

type-synonym *'f ord-res-6-state* $= 'f\ gclause\ fset \times 'f\ gclause\ fset \times 'f\ gclause\ fset \times$
 $('f\ gterm \Rightarrow 'f\ gclause\ option) \times 'f\ gclause\ option$

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

inductive *ord-res-5-matches-ord-res-6* $:: 'f\ ord\text{-}res\text{-}5\text{-}state \Rightarrow 'f\ ord\text{-}res\text{-}6\text{-}state \Rightarrow$
 $bool$ **where**

$ord\text{-}res\text{-}5\text{-}invars\ N\ (U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C}) \implies$
 $ord\text{-}res\text{-}5\text{-}matches\text{-}ord\text{-}res\text{-}6\ (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})\ (N, U_{er}, \mathcal{F}, \mathcal{M}, \mathcal{C})$

lemma *ord-res-5-final-iff-ord-res-6-final*:
fixes $i\ S_5\ S_6$
assumes *match*: *ord-res-5-matches-ord-res-6* $S_5\ S_6$
shows *ord-res-5-final* $S_5 \longleftrightarrow ord\text{-}res\text{-}6\text{-}final\ S_6$
 $\langle proof \rangle$

lemma *backward-simulation-between-5-and-6*:
fixes $S_5\ S_6\ S_6'$
assumes *match*: *ord-res-5-matches-ord-res-6* $S_5\ S_6$ **and** *step*: *ord-res-6-step* $S_6\ S_6'$

shows $\exists S5'. \text{ord-res-5-step}^{++} S5 S5' \wedge \text{ord-res-5-matches-ord-res-6} S5' S6'$
 ⟨proof⟩

theorem *bisimulation-ord-res-5-ord-res-6*:

defines *match* $\equiv \lambda-. \text{ord-res-5-matches-ord-res-6}$

shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-5-state} \Rightarrow 'f \text{ord-res-6-state} \Rightarrow \text{bool})$
 $\mathcal{R}_f \mathcal{R}_b.$

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⟩

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 $N (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \Longrightarrow$

$(\forall A C. \mathcal{M} A = \text{Some } C \longleftrightarrow \text{map-of } \Gamma (\text{Pos } A) = \text{Some } (\text{Some } C)) \Longrightarrow$

$(\forall A. \mathcal{M} A = \text{None} \longleftrightarrow \text{map-of } \Gamma (\text{Neg } A) \neq \text{None} \vee A \notin \text{trail-atms } \Gamma) \Longrightarrow$

$i = \text{atms-of-clss } (N \cup U_{er}) - \text{trail-atms } \Gamma \Longrightarrow$

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* $S6 \longleftrightarrow \text{ord-res-7-final } S7$

⟨proof⟩

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

$(\exists i' S6'. \text{ord-res-6-step}^{++} S6 S6' \wedge \text{ord-res-6-matches-ord-res-7 } i' S6' S7') \vee$

$(\exists i'. \text{ord-res-6-matches-ord-res-7 } i' S6 S7' \wedge i' \mid \mathcal{C} \mid i)$

⟨proof⟩

theorem *bisimulation-ord-res-6-ord-res-7*:

defines $match \equiv ord\text{-}res\text{-}6\text{-}matches\text{-}ord\text{-}res\text{-}7$
shows $\exists (MATCH :: nat \times nat \Rightarrow 'f\text{-}ord\text{-}res\text{-}6\text{-}state \Rightarrow 'f\text{-}ord\text{-}res\text{-}7\text{-}state \Rightarrow bool)$
 $\mathcal{R}_f \mathcal{R}_b$.
bisimulation $ord\text{-}res\text{-}6\text{-}step\ ord\text{-}res\text{-}6\text{-}final\ (constant\text{-}context\ ord\text{-}res\text{-}7)\ ord\text{-}res\text{-}7\text{-}final$
 $MATCH \mathcal{R}_f \mathcal{R}_b$
 $\langle proof \rangle$
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**

$\neg trail\text{-}false\text{-}cls\ \Gamma\ C \Longrightarrow$
 $linorder\text{-}lit.\text{is-maximal-in-mset}\ C\ L \Longrightarrow$
 $linorder\text{-}trm.\text{is-least-in-fset}\ \{|A|\ \in\ |atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er})\}.$
 $A \prec_t atm\text{-}of\ L \wedge A \notin trail\text{-}atms\ \Gamma \} A \Longrightarrow$
 $ord\text{-}res\text{-}8\text{-}can\text{-}decide\text{-}neg\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

inductive *ord-res-8-can-skip-undefined-neg* **where**

$\neg trail\text{-}false\text{-}cls\ \Gamma\ C \Longrightarrow$
 $linorder\text{-}lit.\text{is-maximal-in-mset}\ C\ L \Longrightarrow$
 $\neg(\exists A \in |atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}). A \prec_t atm\text{-}of\ L \wedge A \notin trail\text{-}atms\ \Gamma) \Longrightarrow$
 $\neg trail\text{-}defined\text{-}lit\ \Gamma\ L \Longrightarrow$
 $is\text{-}neg\ L \Longrightarrow$
 $ord\text{-}res\text{-}8\text{-}can\text{-}skip\text{-}undefined\text{-}neg\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

inductive *ord-res-8-can-skip-undefined-pos-ultimate* **where**

$\neg trail\text{-}false\text{-}cls\ \Gamma\ C \Longrightarrow$
 $linorder\text{-}lit.\text{is-maximal-in-mset}\ C\ L \Longrightarrow$
 $\neg(\exists A \in |atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}). A \prec_t atm\text{-}of\ L \wedge A \notin trail\text{-}atms\ \Gamma) \Longrightarrow$
 $\neg trail\text{-}defined\text{-}lit\ \Gamma\ L \Longrightarrow$
 $is\text{-}pos\ L \Longrightarrow$
 $\neg trail\text{-}false\text{-}cls\ \Gamma\ \{\#K \in \# C. K \neq L\# \} \Longrightarrow$
 $\neg(\exists D \in |iefac\ \mathcal{F}\ |\uparrow\ (N\ |\cup|\ U_{er}). C \prec_c D) \Longrightarrow$
 $ord\text{-}res\text{-}8\text{-}can\text{-}skip\text{-}undefined\text{-}pos\text{-}ultimate\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

inductive *ord-res-8-can-produce* **where**

$\neg trail\text{-}false\text{-}cls\ \Gamma\ C \Longrightarrow$
 $linorder\text{-}lit.\text{is-maximal-in-mset}\ C\ L \Longrightarrow$
 $\neg(\exists A \in |atms\text{-}of\text{-}clss\ (N\ |\cup|\ U_{er}). A \prec_t atm\text{-}of\ L \wedge A \notin trail\text{-}atms\ \Gamma) \Longrightarrow$
 $\neg trail\text{-}defined\text{-}lit\ \Gamma\ L \Longrightarrow$
 $is\text{-}pos\ L \Longrightarrow$
 $trail\text{-}false\text{-}cls\ \Gamma\ \{\#K \in \# C. K \neq L\# \} \Longrightarrow$
 $linorder\text{-}lit.\text{is-greatest-in-mset}\ C\ L \Longrightarrow$
 $ord\text{-}res\text{-}8\text{-}can\text{-}produce\ N\ U_{er}\ \mathcal{F}\ \Gamma\ C$

inductive *ord-res-8-can-factorize* **where**

$\neg \text{trail-false-cls } \Gamma \ C \implies$
 $\text{linorder-lit.is-maximal-in-mset } C \ L \implies$
 $\neg (\exists A \ | \in | \text{atms-of-clss } (N \ | \cup | \ U_{er}). A \prec_t \text{atm-of } L \wedge A \notin | \text{trail-atms } \Gamma) \implies$
 $\neg \text{trail-defined-lit } \Gamma \ L \implies$
 $\text{is-pos } L \implies$
 $\text{trail-false-cls } \Gamma \ \{\#K \in \# \ C. K \neq L\# \} \implies$
 $\neg \text{linorder-lit.is-greatest-in-mset } C \ L \implies$
 $\text{ord-res-8-can-factorize } N \ U_{er} \ \mathcal{F} \ \Gamma \ C$

definition *is-least-nonskipped-clause* **where**

$\text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ C \longleftrightarrow$
 $\text{linorder-cls.is-least-in-fset } \{|C \ | \in | \ \text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er}).$
 $\text{trail-false-cls } \Gamma \ C \vee$
 $\text{ord-res-8-can-decide-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ C \vee$
 $\text{ord-res-8-can-skip-undefined-neg } N \ U_{er} \ \mathcal{F} \ \Gamma \ C \vee$
 $\text{ord-res-8-can-skip-undefined-pos-ultimate } N \ U_{er} \ \mathcal{F} \ \Gamma \ C \vee$
 $\text{ord-res-8-can-produce } N \ U_{er} \ \mathcal{F} \ \Gamma \ C \vee$
 $\text{ord-res-8-can-factorize } N \ U_{er} \ \mathcal{F} \ \Gamma \ C\} \ C$

lemma *is-least-nonskipped-clause-empty*:

assumes *bot-in*: $\{\#\} \ | \in | \ \text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er})$
shows *is-least-nonskipped-clause* $N \ U_{er} \ \mathcal{F} \ \Gamma \ \{\#\}$
 $\langle \text{proof} \rangle$

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: $\neg \text{fBex } (\text{iefac } \mathcal{F} \ | \uparrow (N \ | \cup | \ U_{er})) (\text{trail-false-cls } \Gamma)$
shows $\# \ C. \ \text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ C$
 $\langle \text{proof} \rangle$

lemma *MAGIC5*:

assumes *invars*: *ord-res-7-invars* $N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$ **and**
no-more-steps: $\# \ C'. \ \text{ord-res-7 } N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}')$
shows $(\forall C. \ C = \text{Some } C \longleftrightarrow \text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ C)$
 $\langle \text{proof} \rangle$

lemma *MAGIC6*:

assumes *invars*: *ord-res-7-invars* $N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C})$
shows $\exists C'. \ (\text{ord-res-7 } N)^{**} \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}') \wedge$
 $(\# \ C''. \ \text{ord-res-7 } N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}') \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}''))$
 $\langle \text{proof} \rangle$

inductive *ord-res-7-matches-ord-res-8* $:: 'f \ \text{ord-res-7-state} \Rightarrow 'f \ \text{ord-res-8-state} \Rightarrow$
bool **where**

$\text{ord-res-7-invars } N \ (U_{er}, \mathcal{F}, \Gamma, \mathcal{C}) \implies$
 $\text{ord-res-8-invars } N \ (U_{er}, \mathcal{F}, \Gamma) \implies$

$(\forall C. C = \text{Some } C \iff \text{is-least-nonskipped-clause } N \ U_{er} \ \mathcal{F} \ \Gamma \ C) \implies$
 $\text{ord-res-7-matches-ord-res-8 } (N, U_{er}, \mathcal{F}, \Gamma, C) (N, U_{er}, \mathcal{F}, \Gamma)$

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* $S7 \iff \text{ord-res-8-final}$ $S8$
 $\langle \text{proof} \rangle$

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 $\exists S7'. (\text{constant-context ord-res-7})^{++} S7 \ S7' \wedge \text{ord-res-7-matches-ord-res-8}$
 $S7' \ S8'$
 $\langle \text{proof} \rangle$

theorem *bisimulation-ord-res-7-ord-res-8*:
defines *match* $\equiv \lambda-. \text{ord-res-7-matches-ord-res-8}$
shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-7-state} \Rightarrow 'f \text{ord-res-8-state} \Rightarrow \text{bool})$
 $\mathcal{R}_f \ \mathcal{R}_b.$
bisimulation
(constant-context ord-res-7) ord-res-7-final
(constant-context ord-res-8) ord-res-8-final
 $\text{MATCH } \mathcal{R}_f \ \mathcal{R}_b$
 $\langle \text{proof} \rangle$

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 $N \ (U_{er}, \mathcal{F}, \Gamma) \implies$
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* $S8 \iff \text{ord-res-8-final}$ $S9$
 $\langle \text{proof} \rangle$

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 $\exists S8'. (constant-context\ ord-res-8)^{++}\ S8\ S8' \wedge ord-res-8-matches-ord-res-9\ S8'\ S9'$
 $\langle proof \rangle$

theorem $bisimulation-ord-res-8-ord-res-9$:
defines $match \equiv \lambda-. ord-res-8-matches-ord-res-9$
shows $\exists (MATCH :: nat \times nat \Rightarrow 'f\ ord-res-8-state \Rightarrow 'f\ ord-res-9-state \Rightarrow bool)$
 $\mathcal{R}_f\ \mathcal{R}_b.$
 $bisimulation$
 $(constant-context\ ord-res-8)\ ord-res-8-final$
 $(constant-context\ ord-res-9)\ ord-res-8-final$
 $MATCH\ \mathcal{R}_f\ \mathcal{R}_b$
 $\langle proof \rangle$

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**
 $ord-res-8-invars\ N\ (U_{er}, \mathcal{F}, \Gamma_9) \Longrightarrow$
 $ord-res-10-invars\ N\ (U_{er}, \mathcal{F}, \Gamma_{10}) \Longrightarrow$
 $list-all2\ (\lambda x\ y. fst\ x = fst\ y)\ \Gamma_9\ \Gamma_{10} \Longrightarrow$
 $list-all2\ (\lambda x\ y. snd\ y \neq None \longrightarrow x = y)\ \Gamma_9\ \Gamma_{10} \Longrightarrow$
 $ord-res-9-matches-ord-res-10\ (N, U_{er}, \mathcal{F}, \Gamma_9)\ (N, U_{er}, \mathcal{F}, \Gamma_{10})$

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 $ord-res-8-final\ S9 \longleftrightarrow ord-res-8-final\ S10$
 $\langle proof \rangle$

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 $\exists S9'. (constant-context\ ord-res-9)^{++}\ S9\ S9' \wedge ord-res-9-matches-ord-res-10\ S9'\ S10'$
 $\langle proof \rangle$

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 $\mathcal{R}_f \mathcal{R}_b$
 $\langle \text{proof} \rangle$
end

37 ORD-RES-11 (SCL strategy)

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

inductive *ord-res-10-matches-ord-res-11* $:: 'f \text{ord-res-10-state} \Rightarrow 'f \text{ord-res-11-state} \Rightarrow \text{bool}$ **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 $\{\#\} \mid \in \mid \text{iefac } \mathcal{F} \mid ' (N \mid \cup \mid U_{er10})$ *then* $\Gamma = [] \wedge \mathcal{C} = \text{Some } \{\#\}$ *else* $\mathcal{C} = \text{None} \Longrightarrow$
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* $S10 \longleftrightarrow \text{ord-res-11-final } S11$

$\langle \text{proof} \rangle$

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 $\exists S11'. (\text{constant-context ord-res-11})^{++} S11 S11' \wedge \text{ord-res-10-matches-ord-res-11 } S10' S11'$

$\langle \text{proof} \rangle$

theorem *bisimulation-ord-res-10-ord-res-11*:

defines *match* $\equiv \lambda-. \text{ord-res-10-matches-ord-res-11}$

shows $\exists (\text{MATCH} :: \text{nat} \times \text{nat} \Rightarrow 'f \text{ord-res-10-state} \Rightarrow 'f \text{ord-res-11-state} \Rightarrow \text{bool}) \mathcal{R}_f \mathcal{R}_b.$

bisimulation

(constant-context ord-res-10) ord-res-8-final

(constant-context ord-res-11) ord-res-11-final

MATCH $\mathcal{R}_f \mathcal{R}_b$

$\langle \text{proof} \rangle$

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

$\mathcal{R}_f \mathcal{R}_b$:: *bisim-index-1-11* ⇒ *bisim-index-1-11* ⇒ *bool*

where

bisimulation

ord-res-1 ord-res-1-final

(*constant-context ord-res-11*) *ord-res-11-final*

MATCH $\mathcal{R}_f \mathcal{R}_b$

(*proof*)

theorem

obtains

MATCH :: *bisim-index-1-11* ⇒ 'f *ord-res-1-state* ⇒ 'f *ord-res-11-state* ⇒ *bool*

and

$\mathcal{R}_f \mathcal{R}_b$:: *bisim-index-1-11* ⇒ *bisim-index-1-11* ⇒ *bool*

where

bisimulation

ord-res-1 ord-res-1-final

(*constant-context ord-res-11*) *ord-res-11-final*

MATCH $\mathcal{R}_f \mathcal{R}_b$ **and**

$\bigwedge_j S1 S11. \text{MATCH } j S1 S11 \implies \text{ord-res-1-final } S1 \longleftrightarrow \text{ord-res-11-final } S11$

(*proof*)

38 ORD-RES-11 is a regular SCL strategy

definition *gtrailelem-of-trailelem* **where**

gtrailelem-of-trailelem ≡ λ(*L, opt*).

(*lit-of-glit L, map-option* (λ*C. (cls-of-gcls* {#*K* ∈ # *C. K* ≠ *L*#}, *lit-of-glit L, Var*)) *opt*)

fun *state-of-gstate* :: - ⇒ ('f, 'v) *SCL-FOL.state* **where**

state-of-gstate (*U_G, -, Γ_G, C_G*) =

(let
 $\Gamma = \text{map } \text{gtrailelem-of-trailelem } \Gamma_G;$
 $U = \text{cls-of-gcls } |\cdot| U_G;$
 $C = \text{map-option } (\lambda C_G. (\text{cls-of-gcls } C_G, \text{Var})) C_G$
in (Γ, U, C))

lemma *fst-case-prod-simp*: $\text{fst } (\text{case } p \text{ of } (x, y) \Rightarrow (f x, g x y)) = f (\text{fst } p)$
⟨proof⟩

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 $\text{trail-false-cls } \Gamma D \longleftrightarrow \text{trail-false-cls } \Gamma_G D_G$
⟨proof⟩

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
 $\text{ball-le-}\beta_G: \forall A_G \mid \in \mid \text{atms-of-cls } N_G. A_G \preceq_t \beta_G$ **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⟩

end

lemma *wfp-on-antimono-stronger*:
fixes
 $A :: 'a \text{ set}$ **and** $B :: 'b \text{ set}$ **and**
 $f :: 'a \Rightarrow 'b$ **and**
 $R :: 'b \Rightarrow 'b \Rightarrow \text{bool}$ **and** $Q :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
assumes
 $\text{wf}: \text{wfp-on } B R$ **and**
 $\text{sub}: f ' A \subseteq B$ **and**

mono: $\bigwedge x y. x \in A \implies y \in A \implies Q x y \implies R (f x) (f y)$
shows *wfp-on* $A Q$
 ⟨*proof*⟩

For AFP-devel, delete $\llbracket \text{wfp-on } ?B ?R; ?f ' ?A \subseteq ?B; \bigwedge x y. \llbracket x \in ?A; y \in ?A; ?Q x y \rrbracket \implies ?R (?f x) (?f y) \rrbracket \implies \text{wfp-on } ?A ?Q$ as it is available in *HOL.Wellfounded*.

corollary (in *scl-fol-calculus*) *termination-projectable-strategy*:

fixes

$N :: (f, 'v)$ *Term.term clause fset* **and**

$\beta :: (f, 'v)$ *Term.term* **and**

strategy **and** *strategy-init* **and** *proj*

assumes *strategy-restricts-regular-scl*:

$\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta (\text{proj } S)$

(*proj* S') **and**

initial-state: *proj strategy-init* = *initial-state*

shows *wfp-on* $\{S. \text{strategy}^{**} \text{strategy-init } S\} \text{strategy}^{-1-1}$

⟨*proof*⟩

For AFP-devel, delete $\llbracket \text{scl-fol-calculus } ?renaming-vars ?less-B; \bigwedge S S'. \llbracket ?\text{strategy}^{**} ?\text{strategy-init } S; ?\text{strategy } S S' \rrbracket \implies \text{scl-fol-calculus.regular-scl } ?less-B ?N ?\beta (?proj S) (?proj S'); ?proj ?\text{strategy-init} = \text{initial-state} \rrbracket \implies \text{wfp-on } \{S. ?\text{strategy}^{**} ?\text{strategy-init } S\} ?\text{strategy}^{-1-1}$ 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* $\{S. (\text{ord-res-11 } N)^{**} (\{\}, \{\}, [], \text{None}) S\} (\text{ord-res-11 } N)^{-1-1}$

⟨*proof*⟩

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* $N \beta (\text{proj } S) S'$ **and**

strategy-restricts-regular-scl:

$\bigwedge S S'. \text{strategy}^{**} \text{strategy-init } S \implies \text{strategy } S S' \implies \text{regular-scl } N \beta (\text{proj } S)$

(*proj* S') **and**

initial-state: *proj strategy-init* = *initial-state*

defines

$U \equiv \text{state-learned } (\text{proj } S)$

shows $\exists C \gamma. \text{state-conflict } (\text{proj } S) = \text{Some } (C, \gamma) \wedge \neg (\exists D |\in| N |\cup| U. \text{subsumes } D C)$

⟨*proof*⟩

For AFP-devel, delete $\llbracket \text{scl-fol-calculus } ?renaming-vars ?less-B; ?\text{strategy}^{**} ?\text{strategy-init } ?S; \text{scl-fol-calculus.backtrack } ?N ?\beta (?proj ?S) ?S'; \bigwedge S S'. \llbracket ?\text{strategy}^{**} ?\text{strategy-init } S; ?\text{strategy } S S' \rrbracket \implies \text{scl-fol-calculus.regular-scl } ?less-B ?N ?\beta (?proj S) (?proj S'); ?proj ?\text{strategy-init} = \text{initial-state} \rrbracket$

$\implies \exists C \gamma. \text{state-conflict } (?proj \ ?S) = \text{Some } (C, \gamma) \wedge \neg (\exists D | \in | ?N \ | \cup | \text{state-learned } (?proj \ ?S). \text{subsumes } D \ C)$ as it is available in *Simple-Clause-Learning.Non-Redundancy*.

corollary (in *simulation-SCLFOL-ground-ordered-resolution*) *ord-res-11-non-subsumption*:

fixes $N_G :: 'f \ gclause \ fset$ **and** $s :: - \times - \times - \times -$

defines

$\beta \equiv (\text{THE } A. \text{linorder-trm.is-greatest-in-fset } (\text{atms-of-cls } N_G) \ A)$

assumes

run: $(\text{ord-res-11 } N_G)^{**} (\{\|\}, \{\|\}, [], \text{None}) \ s$ **and**

step: $\text{scl-fol.backtrack } (\text{cls-of-gcls } | \ ^i \ N_G) (\text{term-of-gterm } \beta) (\text{state-of-gstate } \ s)$
 s'

shows $\exists U_{er} \ \mathcal{F} \ \Gamma \ D. \ s = (U_{er}, \mathcal{F}, \Gamma, \text{Some } D) \wedge \neg (\exists C \ | \in | \ N_G \ | \cup | \ U_{er}. \ C \subseteq \#$
 $D)$

$\langle \text{proof} \rangle$

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.