Typed Ordered Resolution

Adnan Mohammed Ahmed Balazs Toth

February 6, 2026

Abstract

Ordered Resolution is a proof calculus for reasoning about first-order logic that is implemented in many automatic theorem provers. It works by saturating the given set of clauses and is refutationally complete, meaning that if the set is inconsistent, the saturation will contain a contradiction. In this formalization, we restructured the completeness proof to cleanly separate the ground (i.e., variable-free) and nonground aspects. We also added a type system to the calculus. We relied on the library for first-order clauses and on the saturation framework.

Contents

theory *Ground-Ordered-Resolution*
imports
First-Order-Clause.Selection-Function
First-Order-Clause.Ground-Order
First-Order-Clause.Literal-Functor
begin

1 Resolution Calculus

locale *ground-ordered-resolution-calculus* =

ground-order **where** $less_t = less_t +$
selection-function select
for
$less_t :: 't \Rightarrow 't \Rightarrow bool$ **and**
select :: *'t clause* $\Rightarrow$ *'t clause*
begin

1.1 Resolution Calculus

inductive *resolution* :: *'t clause* $\Rightarrow$ *'t clause* $\Rightarrow$ *'t clause* $\Rightarrow$ *bool* **where**
resolutionI:
$E =$ *add-mset* l_1 $E' \Longrightarrow$
$D =$ *add-mset* l_2 $D' \Longrightarrow$
$l_1 =$ *Neg* $t \Longrightarrow$
$l_2 =$ *Pos* $t \Longrightarrow$
$C = (E' + D') \Longrightarrow$
resolution D E C
if
$D \prec_c E$
select $E = \{\#\} \wedge$ *is-maximal* l_1 $E \vee$ *is-maximal* l_1 (*select* E)
select $D = \{\#\}$
is-strictly-maximal l_2 D

inductive *factoring* :: *'t clause* $\Rightarrow$ *'t clause* $\Rightarrow$ *bool* **where**
factoringI:
$D =$ *add-mset* l (*add-mset* l D') $\Longrightarrow$
$l =$ *Pos* $t \Longrightarrow$
$C =$ *add-mset* l $D' \Longrightarrow$
factoring D C
if
select $D = \{\#\}$
is-maximal l D

1.2 Ground Layer

abbreviation *resolution-inferences* **where**
resolution-inferences $\equiv \{$*Infer* $[D, E]$ $C \mid D$ E C. *resolution* D E $C\}$

abbreviation *factoring-inferences* **where**
factoring-inferences $\equiv \{$*Infer* $[D]$ $C \mid D$ C. *factoring* D $C\}$

definition *G-Inf* :: *'t clause inference set* **where**
G-Inf =
$\{$*Infer* $[D, E]$ $C \mid D$ E C. *resolution* D E $C\} \cup$
$\{$*Infer* $[D]$ $C \mid D$ C. *factoring* D $C\}$

abbreviation *G-Bot* :: *'t clause set* **where**
G-Bot $\equiv \{\{\#\}\}$

definition *G-entails* :: *'t clause set* $\Rightarrow$ *'t clause set* $\Rightarrow$ *bool* **where**

G-entails $N_1\ N_2 \longleftrightarrow (\forall\ I.\ I \models s\ N_1 \longrightarrow I \models s\ N_2)$

1.3 Smaller Conclussions

lemma *ground-resolution-smaller-conclusion*:
  assumes
    step: *resolution D E C*
  shows $C \prec_c E$
  using *step*
proof (*cases D E C rule*: *resolution.cases*)
  case (*resolutionI* $l_1\ l_2\ E'\ D'\ t$)

  have $\forall k{\in}\#D'.\ k \prec_l Pos\ t$
    using ‹*is-strictly-maximal* $l_2\ D$› ‹$D =$ *add-mset* $l_2\ D'$›
    using *is-strictly-maximal-def resolutionI*(*8*)
    by *fastforce*

  moreover have $\bigwedge A.\ Pos\ A \prec_l Neg\ A$
    unfolding *literal.order.multiset-extension-def*
    by *auto*

  ultimately have $\forall k{\in}\#D'.\ k \prec_l Neg\ t$
    using *literal.order.order.strict-trans*
    by *blast*

  hence $D' \prec_c \{\#Neg\ t\#\}$
    using *one-step-implies-multp*[*of* $\{\#Neg\ t\#\}\ D'\ (\prec_l)\ \{\#\}$]
    by (*simp add*: $less_c$-*def*)

  hence $D' + E' \prec_c$ *add-mset* (*Neg t*) E'
    using *multp-cancel*[*of* $(\prec_l)\ E'\ D'\ \{\#Neg\ t\#\}$] $less_c$-*def*
    by *force*

  thus *?thesis*
    unfolding *resolutionI*
    by (*simp only*: *add.commute*)
qed

lemma *ground-factoring-smaller-conclusion*:
  assumes *step*: *factoring D C*
  shows $C \prec_c D$
  using *step*
proof (*cases D C rule*: *factoring.cases*)
  case (*factoringI l D' t*)

  have $C =$ *add-mset l D'*
    using *factoringI*
    by *argo*

then show *?thesis*
by (*metis* (*lifting*) *add.comm-neutral add-mset-add-single add-mset-not-empty clause.order.multiset-extension-def empty-iff local.factoringI*(*3*) *one-step-implies-multp set-mset-empty*)
qed

end

1.4 Redundancy Criterion

sublocale *ground-ordered-resolution-calculus* ⊆ *consequence-relation* **where**
Bot = *G-Bot* **and**
entails = *G-entails*
proof *unfold-locales*

show *G-Bot* ≠ {}
by *simp*
next

show ⋀*B N*. *B* ∈ *G-Bot* ⟹ *G-entails* {*B*} *N*
by (*simp add*: *G-entails-def*)
next

show ⋀*N2 N1*. *N2* ⊆ *N1* ⟹ *G-entails N1 N2*
by (*auto simp*: *G-entails-def elim*!: *true-clss-mono*[*rotated*])
next
fix *N1 N2*
assume *ball-G-entails*: ∀ *C* ∈ *N2*. *G-entails N1* {*C*}

show *G-entails N1 N2*
unfolding *G-entails-def*
proof (*intro allI impI*)
fix *I* :: *'t set*
assume *I* ⊨*s N1*

hence ∀ *C* ∈ *N2*. *I* ⊨*s* {*C*}
using *ball-G-entails*
by (*simp add*: *G-entails-def*)

then show *I* ⊨*s N2*
by (*simp add*: *true-clss-def*)
qed
next

show ⋀*N1 N2 N3*. *G-entails N1 N2* ⟹ *G-entails N2 N3* ⟹ *G-entails N1 N3*
using *G-entails-def*
by *simp*
qed

sublocale *ground-ordered-resolution-calculus* $\subseteq$ *calculus-with-finitary-standard-redundancy*
where
  Inf = *G-Inf* **and**
  Bot = *G-Bot* **and**
  entails = *G-entails* **and**
  less = $(\prec_c)$
  defines *GRed-I* = *Red-I* **and** *GRed-F* = *Red-F*
proof *unfold-locales*

  show *transp* $(\prec_c)$
    by *simp*
next

  show *wfP* $(\prec_c)$
    by *auto*
next

  show $\bigwedge\iota.\ \iota \in$ *G-Inf* $\Longrightarrow$ *prems-of* $\iota \neq []$
    by (*auto simp*: *G-Inf-def*)
next
  fix ι

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

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

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

end
theory *Relation-Extra*
  imports *Main*
begin

lemma *partition-set-around-element*:
  assumes *tot*: *totalp-on N R* **and** *x-in*: $x \in N$
  shows $N = \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
proof (*intro Set.equalityI Set.subsetI*)
  fix z **assume** $z \in N$
  hence $R\ z\ x \vee z = x \vee R\ x\ z$
    using *tot*[*THEN totalp-onD*] *x-in* **by** *auto*
  thus $z \in \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
    using ‹$z \in N$› **by** *auto*
next
  fix z **assume** $z \in \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
  hence $z \in N \vee z = x$
    by *auto*
  thus $z \in N$
    using *x-in* **by** *auto*
qed

end
theory *Ground-Ordered-Resolution-Completeness*
  imports
    Ground-Ordered-Resolution
    Relation-Extra
    First-Order-Clause.HOL-Extra
begin

1.5 Mode Construction

context *ground-ordered-resolution-calculus*
begin

context
  fixes $N :: 't\ clause\ set$
begin

function *epsilon* :: $'t\ clause \Rightarrow 't\ set$ **where**
  epsilon $C = \{A \mid A\ C'.$
    $C \in N \wedge$
    $C =$ *add-mset* (*Pos A*) $C' \wedge$
    select $C = \{\#\} \wedge$
    is-strictly-maximal (*Pos A*) $C \wedge$
    $\neg\ (\bigcup D \in \{D \in N.\ D \prec_c C\}.$ *epsilon D*$) \models C\}$
  by *auto*

termination *epsilon*
proof (*relation* $\{(x,\ y).\ x \prec_c y\}$)
  show *wf* $\{(x,\ y).\ x \prec_c y\}$
    using *wfp-def* **by** *blast*
next

show $\bigwedge C\ D.\ D \in \{D \in N.\ D \prec_c C\} \Longrightarrow (D,\ C) \in \{(x,\ y).\ x \prec_c y\}$
by *simp*
qed

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

end

lemma *epsilon-eq-empty-or-singleton*: *epsilon N C* = {} ∨ (∃ *A*. *epsilon N C* = {*A*})
proof −

have $\exists_{\leq 1} A.$ *is-strictly-maximal* (*Pos A*) *C*
by (*metis* (*mono-tags*, *lifting*) *Uniq-def literal.inject*(*1*) *literal.order.Uniq-is-strictly-maximal-in-mset*)

hence $\exists_{\leq 1} A.\ \exists\, C'.$
C = *add-mset* (*Pos A*) *C′* ∧ *is-strictly-maximal* (*Pos A*) *C*
by (*simp add*: *Uniq-def*)

hence *Uniq-epsilon*: $\exists_{\leq 1} A.\ \exists\, C'.$
C ∈ *N* ∧
C = *add-mset* (*Pos A*) *C′* ∧
select C = {#} ∧
is-strictly-maximal (*Pos A*) *C* ∧
¬ (⋃ *D* ∈ {*D* ∈ *N*. *D* ≺*c C*}. *epsilon N D*) ⊨ *C*
using *Uniq-antimono′*
by (*smt* (*verit*) *Uniq-def Uniq-prodI case-prod-conv*)

show *?thesis*
unfolding *epsilon.simps*[*of N C*]
using *Collect-eq-if-Uniq*[*OF Uniq-epsilon*]
by (*smt* (*verit*, *best*) *Collect-cong Collect-empty-eq Uniq-def Uniq-epsilon case-prod-conv insertCI mem-Collect-eq*)
qed

definition *rewrite-sys* **where**
rewrite-sys N C ≡ (⋃ *D* ∈ {*D* ∈ *N*. *D* ≺*c C*}. *epsilon N D*)

lemma *rewrite-sys-subset-if-less-cls*: *C* ≺*c D* ⟶ *rewrite-sys N C* ⊆ *rewrite-sys N D*
unfolding *rewrite-sys-def*
by *fastforce*

lemma *mem-epsilonE*:
assumes *rule-in*: *A* ∈ *epsilon N C*
obtains *C′* **where**
C ∈ *N* **and**

```
    C = add-mset (Pos A) C' and
    select C = {#} and
    is-strictly-maximal (Pos A) C and
    ¬ rewrite-sys N C ⊨ C
  using rule-in
  unfolding epsilon.simps[of N C] mem-Collect-eq Let-def rewrite-sys-def
  by (metis (no-types, lifting))

lemma epsilon-unfold: epsilon N C = {A | A C'.
    C ∈ N ∧
    C = add-mset (Pos A) C' ∧
    select C = {#} ∧
    is-strictly-maximal (Pos A) C ∧
    ¬ rewrite-sys N C ⊨ C}
  by (simp add: epsilon.simps[of N C] rewrite-sys-def)

lemma epsilon-subset-if-less-cls: C ≺c D ⟹ epsilon N C ⊆ rewrite-sys N D
  unfolding rewrite-sys-def
  using epsilon-unfold
  by blast

lemma
  assumes
    D ⪯c C and
    C-prod: A ∈ epsilon N C and
    L-in: L ∈# D
  shows
    lesseq-trm-if-pos: is-pos L ⟹ atm-of L ⪯t A and
    less-trm-if-neg: is-neg L ⟹ atm-of L ≺t A
proof −
  from C-prod obtain C' where
    C-def: C = add-mset (Pos A) C' and
    C-max-lit: is-strictly-maximal (Pos A) C
    by (auto elim: mem-epsilonE)

  have Pos A ≺l L if is-pos L and ¬ atm-of L ⪯t A
  proof −

    from that(2) have A ≺t atm-of L
      by order

    hence multp (≺t) {#A#} {#atm-of L#}
      by auto

    with that(1) show Pos A ≺l L
      by (metis (no-types, lifting) Pos-atm-of-iff less_l-def
          literal-to-mset.simps(1))
  qed
```

```
  moreover have Pos A ≺l L if is-neg L and ¬ atm-of L ≺t A
  proof −

    from that(2) have A ⪯t atm-of L
      by order

    hence multp (≺t) {#A#} {#atm-of L, atm-of L#}
      by auto

    with that(1) show Pos A ≺l L
      by (cases L) (simp-all add: lessl-def)
  qed

  moreover have False if Pos A ≺l L
  proof −

    have C ≺c D
      unfolding lessc-def
    proof (rule multp-if-maximal-of-lhs-is-less)

      show Pos A ∈# C
        by (simp add: C-def)
    next

      show L ∈# D
        using L-in
        by simp
    next

      show is-maximal (Pos A) C
        using C-max-lit
        by auto
    next

      show Pos A ≺l L
        using that
        by simp
    qed simp-all

    with ‹D ⪯c C› show False
      by order
  qed

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

lemma less-trm-iff-less-cls-if-mem-epsilon:
  assumes C-prod: A_C ∈ epsilon N C and D-prod: A_D ∈ epsilon N D
```

```
  shows A_C ≺_t A_D ⟷ C ≺_c D
proof −
  from C-prod
  obtain C′ where
    C ∈ N and
    C-def: C = add-mset (Pos A_C) C′ and
    is-strictly-maximal (Pos A_C) C
    by (auto elim!: mem-epsilonE)

  hence ∀ L ∈# C′. L ≺_l Pos A_C
    unfolding is-strictly-maximal-def
    by auto

  from D-prod
  obtain D′ where
    D ∈ N and
    D-def: D = add-mset (Pos A_D) D′ and
    is-strictly-maximal (Pos A_D) D
    by (auto elim!: mem-epsilonE)

  hence ∀ L ∈# D′. L≺_l Pos A_D
    unfolding is-strictly-maximal-def
    by auto

  show ?thesis
  proof (rule iffI)
    assume A_C ≺_t A_D

    hence Pos A_C ≺_l Pos A_D
      by (simp add: less_l-def)

    moreover hence ∀ L ∈# C′. L ≺_l Pos A_D
      using ‹∀ L ∈# C′. L ≺_l Pos A_C›
      by fastforce

    ultimately show C ≺_c D
      using one-step-implies-multp[of D C - {#}] less_c-def
      by (simp add: D-def C-def)
  next
    assume C ≺_c D

    hence epsilon N C ⊆ (⋃ (epsilon N ‘ {x ∈ N. x ≺_c D}))
      using ‹C ∈ N›
      by auto

    hence A_C ∈ (⋃ (epsilon N ‘ {x ∈ N. x ≺_c D}))
      using C-prod
      by auto
```

```
    hence A_C ≠ A_D
      by (metis D-prod rewrite-sys-def mem-epsilonE true-cls-add-mset true-lit-iff)

    moreover have ¬ (A_D ≺_t A_C)
    proof (rule notI)
      assume A_D ≺_t A_C

      then have Pos A_D ≺_l Pos A_C
        by (simp add: less_l-def)

      moreover have ∀ L ∈# D'. L ≺_l Pos A_C
        using ‹∀ L ∈# D'. L ≺_l Pos A_D›
        using calculation literal.order.order.strict-trans
        by blast

      ultimately have D ≺_c C
        using one-step-implies-multp[of C D - {#}] less_c-def
        by (simp add: D-def C-def)

      thus False
        using ‹C ≺_c D›
        by order
    qed

    ultimately show A_C ≺_t A_D
      by order
  qed
qed

lemma false-cls-if-productive-epsilon:
  assumes C-prod: A ∈ epsilon N C and D ∈ N and C ≺_c D
  shows ¬ rewrite-sys N D ⊨ C - {#Pos A#}
proof -

  from C-prod obtain C' where
    C-in: C ∈ N and
    C-def: C = add-mset (Pos A) C' and
    select C = {#} and
    Pox-A-max: is-strictly-maximal (Pos A) C and
    ¬ rewrite-sys N C ⊨ C
    by (rule mem-epsilonE) blast

  from ‹D ∈ N› ‹C ≺_c D› have A ∈ rewrite-sys N D
    using C-prod epsilon-subset-if-less-cls
    by auto

  from ‹D ∈ N› have rewrite-sys N D ⊆ (⋃ D ∈ N. epsilon N D)
    by (auto simp: rewrite-sys-def)
```

have $\neg$ *rewrite-sys N D* $\Vdash$ C'
unfolding *true-cls-def Set.bex-simps*
proof (*intro ballI*)
fix L
assume *L-in*: $L \in\# C'$

hence $L \in\# C$
by (*simp add*: *C-def*)

have $C' \prec_c C$
by (*metis* (*mono-tags*, *lifting*) *C-def add.comm-neutral*
add-mset-add-single add-mset-not-empty
clause.order.multiset-extension-def empty-iff
one-step-implies-multp set-mset-empty)

hence $C' \preceq_c C$
by *order*

show $\neg$ *rewrite-sys N D* $\Vdash l$ L
proof (*cases L*)
case (*Pos* A_L)

moreover have $A_L \notin$ *rewrite-sys N D*
proof $-$

have $\forall y \in\# C'$. $y \prec_l$ *Pos A*
using *Pox-A-max C-def is-strictly-maximal-def*
by *auto*

with *Pos* **have** $A_L \notin$ *insert A* (*rewrite-sys N C*)
using *L-in* ‹$\neg$ *rewrite-sys N C* $\Vdash$ *C*› *C-def*
by *blast*

moreover have $A_L \notin (\bigcup D' \in \{D' \in N.\ C \prec_c D' \wedge D' \prec_c D\}.$ *epsilon N* $D')$
proof $-$
have $A_L \preceq_t A$
using *Pos lesseq-trm-if-pos*[*OF* ‹$C' \preceq_c C$› *C-prod* ‹$L \in\# C'$›]
by *simp*

thus *?thesis*
using *less-trm-iff-less-cls-if-mem-epsilon*
using *C-prod calculation rewrite-sys-def*
by *fastforce*
qed

moreover have *rewrite-sys N D* $=$
insert A (*rewrite-sys N C*) $\cup$ $(\bigcup D' \in \{D' \in N.\ C \prec_c D' \wedge D' \prec_c D\}.$ *epsilon N* $D')$

proof −

have *rewrite-sys N D* = (⋃ *D′* ∈ {*D′* ∈ *N*. *D′* ≺$_c$ *D*}. *epsilon N D′*)
by (*simp only*: *rewrite-sys-def*)

also have
. . . = (⋃ *D′* ∈ {*D′* ∈ {*y* ∈ *N*. *y* ≺$_c$ *C*} ∪ {*C*} ∪ {*y* ∈ *N*. *C* ≺$_c$ *y*}. *D′* ≺$_c$ *D*}. *epsilon N D′*)
using *C-in clause.order.antisym-conv3*
by *auto*

also have
. . . = (⋃ *D′* ∈ {*y* ∈ *N*. *y* ≺$_c$ *C* ∧ *y* ≺$_c$ *D*} ∪ {*C*} ∪ {*y* ∈ *N*. *C* ≺$_c$ *y* ∧ *y* ≺$_c$ *D*}. *epsilon N D′*)
using ‹*C* ≺$_c$ *D*›
by *auto*

also have . . . = (⋃ *D′* ∈ {*y* ∈ *N*. *y* ≺$_c$ *C*} ∪ {*C*} ∪ {*y* ∈ *N*. *C* ≺$_c$ *y* ∧ *y* ≺$_c$ *D*}. *epsilon N D′*)
by (*metis* (*lifting*) *assms*(*3*) *clause.order.order.strict-trans*)

also have
. . . = *rewrite-sys N C* ∪ *epsilon N C* ∪ (⋃ *D′* ∈ {*y* ∈ *N*. *C* ≺$_c$ *y* ∧ *y* ≺$_c$ *D*}. *epsilon N D′*)
by (*auto simp*: *rewrite-sys-def*)

finally show *?thesis*
using *C-prod*
by (*metis* (*no-types*, *lifting*) *empty-iff insertE insert-is-Un epsilon-eq-empty-or-singleton sup-commute*)
qed

ultimately show *?thesis*
by *simp*
qed
ultimately show *?thesis*
by *simp*
next
case (*Neg* A_L)

moreover have A_L ∈ *rewrite-sys N D*
using *Neg* ‹*L* ∈# *C*› ‹*C* ≺$_c$ *D*› ‹¬ *rewrite-sys N C* ⊨ *C*› *rewrite-sys-subset-if-less-cls*
by *blast*

ultimately show *?thesis*
by *simp*
qed
qed

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

lemma *neg-notin-Interp-not-produce*:
Neg A ∈# *C* ⟹ *A* ∉ *rewrite-sys N D* ∪ *epsilon N D* ⟹ *C* ≼*c D* ⟹ *A* ∉ *epsilon N D*″
by (*smt* (*verit*, *del-insts*) *Neg-atm-of-iff UN-I Un-iff clause.order.order.strict-trans1 clause.order.not-less less-trm-if-neg literal.sel*(2) *mem-Collect-eq mem-epsilonE*

rewrite-sys-def term.order.order.asym)

lemma *lift-interp-entails*:
assumes
D-in: *D* ∈ *N* **and**
D-entailed: *rewrite-sys N D* ⊨ *D* **and**
C-in: *C* ∈ *N* **and**
D-lt-C: *D* ≺*c C*
shows *rewrite-sys N C* ⊨ *D*
proof −

from *D-entailed* **obtain** *L A* **where**
L-in: *L* ∈# *D* **and**
L-eq-disj-L-eq: *L* = *Pos A* ∧ *A* ∈ *rewrite-sys N D* ∨ *L* = *Neg A* ∧ *A* ∉ *rewrite-sys N D*
unfolding *true-cls-def true-lit-iff*
by *metis*

have *rewrite-sys N D* ⊆ *rewrite-sys N C*
using *D-lt-C rewrite-sys-subset-if-less-cls*
by *auto*

from *L-eq-disj-L-eq*
show *rewrite-sys N C* ⊨ *D*
proof (*elim disjE conjE*)
assume *L* = *Pos A* **and** *A* ∈ *rewrite-sys N D*

thus *rewrite-sys N C* ⊨ *D*
using *L-in* ‹*rewrite-sys N D* ⊆ *rewrite-sys N C*›
by *auto*
next
assume *L*: *L* = *Neg A* **and** *A*: *A* ∉ *rewrite-sys N D*

have *A* ∉ *rewrite-sys N C*
proof (*cases A* ∈ *epsilon N C*)
case *True*

then show *?thesis*
by (*meson mem-epsilonE pos-literal-in-imp-true-cls strictly-maximal-in-clause*)

```
    next
      case False

      then have A ∉ epsilon N D
        using D-entailed mem-epsilonE
        by blast

      then show ?thesis
        using neg-notin-Interp-not-produce A L-in
        unfolding rewrite-sys-def L
        by blast
    qed

    thus rewrite-sys N C ⊨ D
      using L-in ‹L = Neg A›
      by blast
  qed
qed

lemma produces-imp-in-interp:
  assumes Neg A ∈# C and D-prod: A ∈ epsilon N D
  shows A ∈ rewrite-sys N C
proof −
  from D-prod have Pos A ∈# D and is-strictly-maximal (Pos A) D
    by (auto elim: mem-epsilonE)

  have D ≺c C
    unfolding less_c-def
  proof (rule multp-if-maximal-of-lhs-is-less)

    show Pos A ∈# D
      using ‹Pos A ∈# D› .
  next

    show Neg A ∈# C
      using ‹Neg A ∈# C› .
  next

    show Pos A ≺l Neg A
      by (simp add: less_l-def)
  next

    show is-maximal (Pos A) D
      using ‹is-strictly-maximal (Pos A) D›
      by auto
  qed simp-all

  hence ¬ (≺c)== C D
    using clause.order.not-less
```

by *blast*

thus *?thesis*
proof (*rule contrapos-np*)

from *D-prod* **show** $A \notin$ *rewrite-sys N C* $\Longrightarrow (\prec_c)^{==}$ *C D*
using ‹*D* $\prec_c$ *C*› *epsilon-subset-if-less-cls*
by *blast*
qed
qed

lemma *split-Union-epsilon*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.$ *epsilon N C*$) =$
rewrite-sys N D $\cup$ *epsilon N D* $\cup$ $(\bigcup C \in \{C \in N.\ D \prec_c C\}.$ *epsilon N C*$)$
proof −

have $N = \{C \in N.\ C \prec_c D\} \cup \{D\} \cup \{C \in N.\ D \prec_c C\}$
proof (*rule partition-set-around-element*)

show *totalp-on N* $(\prec_c)$
using *clause.order.totalp-on-less* **.**
next

show $D \in N$
using *D-in*
by *simp*
qed

hence $(\bigcup C \in N.$ *epsilon N C*$) =$
$(\bigcup C \in \{C \in N.\ C \prec_c D\}.$ *epsilon N C*$) \cup$ *epsilon N D* $\cup$ $(\bigcup C \in \{C \in N.\ D \prec_c C\}.$ *epsilon N C*$)$
by *auto*

thus $(\bigcup C \in N.$ *epsilon N C*$) =$
rewrite-sys N D $\cup$ *epsilon N D* $\cup$ $(\bigcup C \in \{C \in N.\ D \prec_c C\}.$ *epsilon N C*$)$
by (*simp add*: *rewrite-sys-def*)
qed

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

lemma *lift-entailment-to-Union*:
fixes *N D*
assumes

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

lemma *true-cls-if-productive-epsilon*:
assumes $A \in epsilon\ N\ C\ C \prec_c D$
shows *rewrite-sys N D* $\models$ *C*
by (*meson assms epsilon-subset-if-less-cls mem-epsilonE in-mono is-strictly-maximal-def pos-literal-in-imp-true-cls*)

lemma *model-preconstruction*:
fixes
N :: *′t clause set* **and**
C :: *′t clause*
defines
$entails \equiv \lambda E\ C.\ E \models C$
assumes *saturated N* **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
shows
$epsilon\ N\ C = \{\} \longleftrightarrow entails\ (rewrite\text{-}sys\ N\ C)\ C$
$(\bigcup D \in N.\ epsilon\ N\ D) \models C$
$D \in N \Longrightarrow C \prec_c D \Longrightarrow entails\ (rewrite\text{-}sys\ N\ D)\ C$
unfolding *atomize-all atomize-conj atomize-imp*
using *clause.order.wfp C-in*
proof (*induction C arbitrary*: *D rule*: *wfp-induct-rule*)
case (*less C*)
note *IH* = *less.IH*

from ‹$\{\#\} \notin N$› ‹$C \in N$› **have** $C \neq \{\#\}$
by *metis*

define *I* :: *′t set* **where**
$I = rewrite\text{-}sys\ N\ C$

have *i*: $(epsilon\ N\ C = \{\}) \longleftrightarrow entails\ (rewrite\text{-}sys\ N\ C)\ C$
proof (*rule iffI*)

show $entails\ (rewrite\text{-}sys\ N\ C)\ C \Longrightarrow epsilon\ N\ C = \{\}$
unfolding *entails-def rewrite-sys-def*
by (*metis* (*no-types*) *empty-iff equalityI mem-epsilonE rewrite-sys-def subsetI*)
next
assume $epsilon\ N\ C = \{\}$

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

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

then obtain A **where**
Neg $A \in\# C$ **and**
sel-or-max: *select* $C = \{\#\} \wedge$ *is-maximal* (*Neg* A) $C \vee$ *is-maximal* (*Neg* A) (*select* C)
by (*metis* (*lifting*) *Neg-atm-of-iff empty-iff*
literal.order.ex-maximal-in-mset maximal-in-clause
mset-subset-eqD select-negative-literals select-subset
set-mset-empty)

then obtain C' **where**
C-def: $C =$ *add-mset* (*Neg* A) C'
by (*metis mset-add*)

show *?thesis*
proof (*cases* $A \in$ *rewrite-sys* N C)
case *True*

then obtain D **where**
$A \in$ *epsilon* N D **and** $D \in N$ **and** $D \prec_c C$
unfolding *rewrite-sys-def*
by *auto*

then obtain D' **where**
D-def: $D =$ *add-mset* (*Pos* A) D' **and**
sel-D: *select* $D = \{\#\}$ **and**
max-t-t': *is-strictly-maximal* (*Pos* A) D **and**
$\neg$ *entails* (*rewrite-sys* N D) D
by (*metis* (*lifting*) *IH empty-iff mem-epsilonE*)

define ι :: *'t clause inference* **where**
$\iota =$ *Infer* [D, C] ($C' + D'$)

have *resolution*: *resolution* D C ($C' + D'$)
proof (*rule resolutionI*)

show $C =$ *add-mset* (*Neg* A) C'
by (*simp add*: *C-def*)
next

show $D =$ *add-mset* (*Pos* A) D'
by (*simp add*: *D-def*)
next

show $D \prec_c C$
using ‹$D \prec_c C$› **.**
next

show *select* $C = \{\#\} \wedge$ *is-maximal* (*Neg* A) $C \vee$ *is-maximal* (*Neg* A) (*select* C)
using *sel-or-max*
by *auto*
next

show *select* $D = \{\#\}$
using *sel-D* **by** *blast*
next

show *is-strictly-maximal* (*Pos* A) D
using *max-t-t'* **.**
qed *simp-all*

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

moreover have $\bigwedge t.\ t \in$ *set* (*prems-of* ι) $\longrightarrow t \in N$
using ‹$C \in N$› ‹$D \in N$›
by (*auto simp*: ι*-def*)

ultimately have $\iota \in$ *Inf-from* N
by (*auto simp*: *Inf-from-def*)

hence $\iota \in$ *Red-I* N
using ‹*saturated* N›
by (*auto simp*: *saturated-def*)

then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-CD: *G-entails* (*insert* D *DD*) $\{C' + D'\}$ **and**
ball-DD-lt-C: $\forall\, D \in DD.\ D \prec_c C$
unfolding *Red-I-def redundant-infer-def mem-Collect-eq*
by (*auto simp*: ι*-def*)

moreover have $\forall\, D \in$ *insert* D *DD*. *entails* (*rewrite-sys* N C) D
using *IH*[*THEN conjunct2*, *THEN conjunct2*, *rule-format*, *of* - C]
using ‹$C \in N$› ‹$D \in N$› ‹$D \prec_c C$ › *DD-subset ball-DD-lt-C*
by *blast*

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

moreover have ¬ *entails* (*rewrite-sys* N D) D'
using ‹¬ *entails* (*rewrite-sys* N D) D›
using *D-def entails-def*

by *fastforce*

moreover have $D' \prec_c D$
by (*metis* (*lifting*) *D-def add.comm-neutral add-mset-add-single add-mset-not-empty clause.order.multiset-extension-def empty-iff one-step-implies-multp set-mset-empty*)

moreover have ¬ *entails* (*rewrite-sys N C*) *D'*
using *D-def* ‹$A \in$ *epsilon N D*› ‹$D \prec_c C$› *entails-def false-cls-if-productive-epsilon less.prems*
by *fastforce*

then show *entails* (*rewrite-sys N C*) *C*
using *C-def entails-def*
using *calculation*(*1*) **by** *fastforce*
next
case *False*

thus *?thesis*
using ‹*Neg A* ∈# *C*›
by (*auto simp add*: *entails-def true-cls-def*)
qed
next
case *False*

hence *select C* = {#}
using *select-subset select-negative-literals*
by (*metis* (*no-types*, *opaque-lifting*) *Neg-atm-of-iff mset-subset-eqD multiset-nonemptyE*)

from *False* **obtain** *A* **where** *Pos-A-in*: *Pos A* ∈# *C* **and** *max-Pos-A*: *is-maximal* (*Pos A*) *C*
by (*metis Neg-atm-of-iff* ‹*C* ≠ {#}› ‹*select C* = {#}› *literal.collapse*(*1*) *literal.order.ex-maximal-in-mset maximal-in-clause*)

then obtain *C'* **where** *C-def*: *C* = *add-mset* (*Pos A*) *C'*
by (*meson mset-add*)

show *?thesis*
proof (*cases entails* (*rewrite-sys N C*) *C'*)
case *True*

then show *?thesis*
using *C-def entails-def*
by *force*
next
case *False*

show *?thesis*

```
    proof (cases is-strictly-maximal (Pos A) C)
      case strictly-maximal: True
      then show ?thesis
        using ‹epsilon N C = {}› ‹select C = {#}› less.prems
        unfolding epsilon-unfold[of N C] Collect-empty-eq
        unfolding C-def entails-def
        by blast
    next
      case False

      hence count C (Pos A) ≥ 2
        using max-Pos-A
        using C-def is-maximal-def is-strictly-maximal-def
        by fastforce

      then obtain C' where C-def: C = add-mset (Pos A) (add-mset (Pos A)
C')
        by (metis two-le-countE)

      define ι :: 't clause inference where
        ι = Infer [C] (add-mset (Pos A) C')

      have eq-fact: factoring C (add-mset (Pos A) C')
      proof (rule factoringI)

        show C = add-mset (Pos A) (add-mset (Pos A) C')
          by (simp add: C-def)
      next

        show select C = {#}
          using ‹select C = {#}› .
      next

        show is-maximal (Pos A) C
          using max-Pos-A .
      qed simp-all

      hence ι ∈ G-Inf
        by (auto simp: ι-def G-Inf-def)

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

      ultimately have ι ∈ Inf-from N
        by (auto simp: Inf-from-def)

      hence ι ∈ Red-I N
        using ‹saturated N›
```

```
      by (auto simp: saturated-def)

    then obtain DD where
      DD-subset: DD ⊆ N and
      finite DD and
      DD-entails-concl: G-entails DD {add-mset (Pos A) C'} and
      ball-DD-lt-C: ∀ D ∈ DD. D ≺c C
      unfolding Red-I-def redundant-infer-def
      by (auto simp: ι-def)

    moreover have ∀ D ∈ DD. entails (rewrite-sys N C) D
      using IH[THEN conjunct2, rule-format, of - C]
      using ‹C ∈ N › DD-subset ball-DD-lt-C
      by blast

    ultimately have entails (rewrite-sys N C) (add-mset (Pos A) C')
      using DD-entails-concl
      unfolding G-entails-def entails-def
      by (simp add: I-def true-clss-def)

    then show ?thesis
      using C-def entails-def
      by fastforce
   qed
  qed
 qed
qed

moreover have iia: entails (⋃ (epsilon N ' N)) C
 using epsilon-eq-empty-or-singleton[of N C]
proof (elim disjE exE)
 assume epsilon N C = {}

 hence entails (rewrite-sys N C) C
  by (simp only: i)

 thus ?thesis
  using lift-entailment-to-Union[OF ‹C ∈ N›] entails-def
  by argo
next
 fix A
 assume epsilon N C = {A}

 hence eps: A ∈ epsilon N C
  by simp

 from eps have Pos A ∈# C
  unfolding epsilon.simps[of N C] mem-Collect-eq
  by force
```

```
    moreover from eps have A ∈ ⋃ (epsilon N `N)
      using ‹C ∈ N› UN-upper
      by fast

    ultimately show ?thesis
      using entails-def
      by blast
  qed

  moreover have iib: entails (rewrite-sys N D) C if D ∈ N and C ≺_c D
    using epsilon-eq-empty-or-singleton[of N C]
  proof (elim disjE exE)
    assume epsilon N C = {}

    hence entails (rewrite-sys N C) C
      unfolding i
      by simp

    thus ?thesis
      using lift-interp-entails[OF ‹C ∈ N› - that] entails-def
      by argo
  next
    fix A assume epsilon N C = {A}

    thus ?thesis
      by (simp add: entails-def that(1,2) true-cls-if-productive-epsilon)
  qed

  ultimately show ?case
    by (simp add: entails-def)
qed

lemma model-construction:
  fixes
    N :: 't clause set and
    C :: 't clause
  defines entails ≡ λE C. E ⊨ C
  assumes saturated N and {#} ∉ N and C-in: C ∈ N
  shows entails (⋃ D ∈ N. epsilon N D) C
  unfolding atomize-conj atomize-imp
  using epsilon-eq-empty-or-singleton[of N C]
proof (elim disjE exE)
  assume epsilon N C = {}

  hence entails (rewrite-sys N C) C
    using model-preconstruction(1)[OF assms(2,3,4)]
    by (metis entails-def)
```

thus *?thesis*
using *lift-entailment-to-Union*(*1*)[*OF* ‹*C* ∈ *N*›]
by (*simp only*: *entails-def*)
next
fix *A* **assume** *epsilon N C* = {*A*}

thus *?thesis*
using *C-in assms*(*2*,*3*) *entails-def model-preconstruction*(*2*)
by *blast*
qed

1.6 Static Refutational Completeness

lemma *statically-complete*:
fixes *N* :: *'t clause set*
assumes *saturated N* **and** *G-entails N* {{#}}
shows {#} ∈ *N*
using ‹*G-entails N* {{#}}›
proof (*rule contrapos-pp*)
assume {#} ∉ *N*

define *I* :: *'t set* **where**
I = (⋃ *D* ∈ *N*. *epsilon N D*)

show ¬ *G-entails N G-Bot*
unfolding *G-entails-def not-all not-imp*
proof (*intro exI conjI*)

show *I* ⊨*s N*
unfolding *I-def*
using *model-construction*[*OF* ‹*saturated N*› ‹{#} ∉ *N*›]
by (*simp add*: *true-clss-def*)
next

show ¬ *I* ⊨*s G-Bot*
by *simp*
qed
qed

sublocale *statically-complete-calculus* **where**
Bot = *G-Bot* **and**
Inf = *G-Inf* **and**
entails = *G-entails* **and**
Red-I = *Red-I* **and**
Red-F = *Red-F*
using *statically-complete*
by *unfold-locales simp*

end

```
end
theory Ground-Ordered-Resolution-Soundness
  imports Ground-Ordered-Resolution
begin

context ground-ordered-resolution-calculus
begin

lemma soundness-ground-resolution:
  assumes
    step: resolution D E C
  shows G-entails {D, E} {C}
  using step
proof (cases D E C rule: resolution.cases)
  case (resolutionI l₁ l₂ E' D' t)

  show ?thesis
    unfolding G-entails-def true-clss-singleton
    unfolding true-clss-insert
  proof (intro allI impI, elim conjE)
    fix I :: 't set
    assume I ⊨ E and I ⊨ D

    then obtain k₁ k₂ :: 't literal where
      k₁ ∈# E and I ⊨l k₁ and k₂ ∈# D and I ⊨l k₂
      by (auto simp: true-cls-def)

    show I ⊨ C
    proof (cases k₁ = l₁)
      case K1-def: True

      hence I ⊨l l₁
        using ‹I ⊨l k₁›
        by simp

      show ?thesis
      proof (cases k₂ = l₂)
        case K2-def: True

        hence I ⊨l l₂
          using ‹I ⊨l k₂›
          by simp

        hence False
          using ‹I ⊨l l₁› local.resolutionI(7,8) by auto

        thus ?thesis ..
      next
```

```
      case False

      hence k₂ ∈# D'
        using ‹k₂ ∈# D›
        unfolding resolutionI
        by simp

      hence I ⊨ D'
        using ‹I ⊨l k₂›
        by blast

      thus ?thesis
        unfolding resolutionI
        by simp
    qed
  next
    case False

    hence k₁ ∈# E'
      using ‹k₁ ∈# E›
      unfolding resolutionI
      by simp

    hence I ⊨ E'
      using ‹I ⊨l k₁›
      by blast

    thus ?thesis
      unfolding resolutionI
      by simp
  qed
 qed
qed

lemma soundness-ground-factoring:
 assumes step: factoring D C
 shows G-entails {D} {C}
 using step
proof (cases D C rule: factoring.cases)
 case (factoringI l D' t)

 show ?thesis
   unfolding G-entails-def true-clss-singleton
 proof (intro allI impI)
   fix I :: 't set
   assume I ⊨ D

   then obtain k :: 't literal where
     k ∈# D and I ⊨l k
```

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

    show I ⊨ C
    proof (cases k = l)
      case True

      hence I ⊨l l
        using ‹I ⊨l k›
        by metis

      thus ?thesis
        unfolding factoringI
        by (metis true-cls-add-mset)
    next
      case False

      hence k ∈# D'
        using ‹k ∈# D›
        unfolding factoringI
        by auto

      hence k ∈# C
        unfolding factoringI
        by simp

      thus ?thesis
        using ‹I ⊨l k›
        by blast
    qed
  qed
qed

sublocale sound-inference-system where
  Inf = G-Inf and
  Bot = G-Bot and
  entails = G-entails
proof unfold-locales
  fix ι
  assume ι ∈ G-Inf

  then show G-entails (set (prems-of ι)) {concl-of ι}
    unfolding G-Inf-def
    using soundness-ground-resolution soundness-ground-factoring
    by auto
qed

end

end
```

theory *Ordered-Resolution*
imports
First-Order-Clause.Nonground-Order
First-Order-Clause.Nonground-Selection-Function
First-Order-Clause.Nonground-Typing
First-Order-Clause.Typed-Tiebreakers
Saturation-Framework.Lifting-to-Non-Ground-Calculi

Ground-Ordered-Resolution
begin

locale *ordered-resolution-calculus* =
witnessed-nonground-typing **where**
welltyped = *welltyped* **and** *term-to-ground* = *term-to-ground* :: $'t \Rightarrow 't_G$ **and**
id-subst = *id-subst* :: *'subst* +

nonground-order **where** $less_t = less_t$ +

nonground-selection-function **where**
select = *select* **and** *atom-subst* = $(\cdot t)$ **and** *atom-vars* = *term.vars* **and**
atom-from-ground = *term.from-ground* **and** *atom-to-ground* = *term.to-ground* +
tiebreakers tiebreakers
for
select :: *'t select* **and**
$less_t$ :: $'t \Rightarrow 't \Rightarrow$ *bool* **and**
tiebreakers :: $('t_G, 't)$ *tiebreakers* **and**
welltyped :: $('v$ :: *infinite*, *'ty*$)$ *var-types* $\Rightarrow 't \Rightarrow$ *'ty* $\Rightarrow$ *bool*
begin

inductive *factoring* :: $('t, 'v,$ *'ty*$)$ *typed-clause* $\Rightarrow$ $('t, 'v,$ *'ty*$)$ *typed-clause* $\Rightarrow$ *bool*
where
factoringI:
D = *add-mset* l_1 (*add-mset* l_2 D') $\Longrightarrow$
l_1 = *Pos* t_1 $\Longrightarrow$
l_2 = *Pos* t_2 $\Longrightarrow$
$C = (\textit{add-mset}\ l_1\ D') \cdot \mu \Longrightarrow$
factoring $(\mathcal{V}, D)$ $(\mathcal{V}, C)$
if
select $D = \{\#\}$
is-maximal $(l_1 \cdot l\ \mu)$ $(D \cdot \mu)$
type-preserving-on (*clause.vars* D) $\mathcal{V}$ μ
term.is-imgu μ $\{\{t_1, t_2\}\}$

inductive *resolution* ::
$('t, 'v,$ *'ty*$)$ *typed-clause* $\Rightarrow$
$('t, 'v,$ *'ty*$)$ *typed-clause* $\Rightarrow$
$('t, 'v,$ *'ty*$)$ *typed-clause* $\Rightarrow$ *bool* **where**
resolutionI:
E = *add-mset* l_1 $E' \Longrightarrow$

$D = \mathit{add\text{-}mset}\ l_2\ D' \Longrightarrow$
$l_1 = \mathit{Neg}\ t_1 \Longrightarrow$
$l_2 = \mathit{Pos}\ t_2 \Longrightarrow$
$C = (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$
$\mathit{resolution}\ (\mathcal{V}_2,\ D)\ (\mathcal{V}_1,\ E)\ (\mathcal{V}_3,\ C)$
if
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_1$
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_2$
$\mathit{term.is\text{-}renaming}\ \varrho_1$
$\mathit{term.is\text{-}renaming}\ \varrho_2$
$\mathit{clause.vars}\ (E \cdot \varrho_1) \cap \mathit{clause.vars}\ (D \cdot \varrho_2) = \{\}$
$\mathit{type\text{-}preserving\text{-}on}\ (\mathit{clause.vars}\ (E \cdot \varrho_1) \cup \mathit{clause.vars}\ (D \cdot \varrho_2))\ \mathcal{V}_3\ \mu$
$\mathit{term.is\text{-}imgu}\ \mu\ \{\{t_1 \cdot t\ \varrho_1,\ t_2 \cdot t\ \varrho_2\}\}$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu)$
$\mathit{select}\ E = \{\#\} \Longrightarrow \mathit{is\text{-}maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)$
$\mathit{select}\ E \neq \{\#\} \Longrightarrow \mathit{is\text{-}maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ ((\mathit{select}\ E) \cdot \varrho_1 \odot \mu)$
$\mathit{select}\ D = \{\#\}$
$\mathit{is\text{-}strictly\text{-}maximal}\ (l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu)$
$\forall\, x \in \mathit{clause.vars}\ E.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (\mathit{term.rename}\ \varrho_1\ x)$
$\forall\, x \in \mathit{clause.vars}\ D.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (\mathit{term.rename}\ \varrho_2\ x)$
$\mathit{type\text{-}preserving\text{-}on}\ (\mathit{clause.vars}\ E)\ \mathcal{V}_1\ \varrho_1$
$\mathit{type\text{-}preserving\text{-}on}\ (\mathit{clause.vars}\ D)\ \mathcal{V}_2\ \varrho_2$

abbreviation *factoring-inferences* **where**
$\mathit{factoring\text{-}inferences} \equiv \{\ \mathit{Infer}\ [D]\ C \mid D\ C.\ \mathit{factoring}\ D\ C\ \}$

abbreviation *resolution-inferences* **where**
$\mathit{resolution\text{-}inferences} \equiv \{\ \mathit{Infer}\ [D,\ E]\ C \mid D\ E\ C.\ \mathit{resolution}\ D\ E\ C\ \}$

definition *inferences* :: $('t,\ 'v,\ 'ty)$ *typed-clause inference set* **where**
$\mathit{inferences} \equiv \mathit{resolution\text{-}inferences} \cup \mathit{factoring\text{-}inferences}$

abbreviation bottom_F :: $('t,\ 'v,\ 'ty)$ *typed-clause set* $(\bot_F)$ **where**
$\mathit{bottom}_F \equiv \{(\mathcal{V},\ \{\#\}) \mid \mathcal{V}.\ \mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}\ \}$

end

end
theory *Grounded-Ordered-Resolution*
imports
Ordered-Resolution
Ground-Ordered-Resolution

First-Order-Clause.Grounded-Selection-Function
First-Order-Clause.Nonground-Inference
Saturation-Framework.Lifting-to-Non-Ground-Calculi

Polynomial-Factorization.Missing-List
begin

locale *grounded-ordered-resolution-calculus* =
ordered-resolution-calculus **where**
select = *select* **and** *welltyped* = *welltyped* **and**
term-from-ground = *term-from-ground* :: $'t_G \Rightarrow 't$ **and** *id-subst* = *id-subst* :: *'subst* +

grounded-selection-function **where**
select = *select* **and** *atom-subst* = $(\cdot t)$ **and** *atom-vars* = *term.vars* **and**
atom-from-ground = *term.from-ground* **and** *atom-to-ground* = *term.to-ground* **and**
is-ground-instance = *is-ground-instance*
for
select :: *'t select* **and**
welltyped :: *('v :: infinite, 'ty) var-types* $\Rightarrow$ *'t* $\Rightarrow$ *'ty* $\Rightarrow$ *bool*
begin

sublocale *ground*: *ground-ordered-resolution-calculus* **where**
$less_t = (\prec_{tG})$ **and** $select = select_G$
rewrites
multiset-extension.multiset-extension $(\prec_{tG})$ *ground.literal-to-mset* = $(\prec_{lG})$ **and**
multiset-extension.multiset-extension $(\prec_{lG})$ $(\lambda x.\ x) = (\prec_{cG})$ **and**
$\bigwedge l_G\ C_G$. *ground.is-maximal* $l_G\ C_G \longleftrightarrow$ *ground-is-maximal* $l_G\ C_G$ **and**
$\bigwedge l_G\ C_G$. *ground.is-strictly-maximal* $l_G\ C_G \longleftrightarrow$ *ground-is-strictly-maximal* $l_G\ C_G$
unfolding *is-maximal-rewrite*[*symmetric*] *is-strictly-maximal-rewrite*[*symmetric*]
by *unfold-locales simp-all*

abbreviation *is-inference-ground-instance-one-premise* **where**
is-inference-ground-instance-one-premise $D\ C\ \iota_G\ \gamma \equiv$
case $(D,\ C)$ *of* $((\mathcal{V}',\ D),\ (\mathcal{V},\ C)) \Rightarrow$
inference.is-ground (*Infer* $[D]\ C \cdot\iota\ \gamma$) $\wedge$
ι_G = *inference.to-ground* (*Infer* $[D]\ C \cdot\iota\ \gamma$) $\wedge$
type-preserving-on (*clause.vars* C) $\mathcal{V}\ \gamma\ \wedge$
$\mathcal{V} = \mathcal{V}' \wedge$
infinite-variables-per-type $\mathcal{V}$

abbreviation *is-inference-ground-instance-two-premises* **where**
is-inference-ground-instance-two-premises $D\ E\ C\ \iota_G\ \gamma\ \varrho_1\ \varrho_2 \equiv$
case $(D,\ E,\ C)$ *of* $((\mathcal{V}_2,\ D),\ (\mathcal{V}_1,\ E),\ (\mathcal{V}_3,\ C)) \Rightarrow$
term.is-renaming $\varrho_1\ \wedge$
term.is-renaming $\varrho_2\ \wedge$
clause.vars $(E \cdot \varrho_1) \cap$ *clause.vars* $(D \cdot \varrho_2) = \{\}\ \wedge$
inference.is-ground (*Infer* $[D \cdot \varrho_2,\ E \cdot \varrho_1]\ C \cdot\iota\ \gamma$) $\wedge$
ι_G = *inference.to-ground* (*Infer* $[D \cdot \varrho_2,\ E \cdot \varrho_1]\ C \cdot\iota\ \gamma$) $\wedge$
type-preserving-on (*clause.vars* C) $\mathcal{V}_3\ \gamma\ \wedge$
infinite-variables-per-type $\mathcal{V}_1\ \wedge$
infinite-variables-per-type $\mathcal{V}_2\ \wedge$
infinite-variables-per-type $\mathcal{V}_3$

abbreviation *is-inference-ground-instance* **where**
is-inference-ground-instance ι ι_G γ $\equiv$
(*case* ι *of*
Infer [D] C $\Rightarrow$ *is-inference-ground-instance-one-premise* D C ι_G γ
| *Infer* [D, E] C $\Rightarrow$ $\exists\, \varrho_1$ ϱ_2. *is-inference-ground-instance-two-premises* D E C ι_G γ ϱ_1 ϱ_2
| - $\Rightarrow$ *False*)
$\wedge$ $\iota_G \in$ *ground.G-Inf*

definition *inference-ground-instances* **where**
inference-ground-instances ι = { ι_G | ι_G γ. *is-inference-ground-instance* ι ι_G γ }

lemma *is-inference-ground-instance*:
is-inference-ground-instance ι ι_G γ $\Longrightarrow$ $\iota_G \in$ *inference-ground-instances* ι
unfolding *inference-ground-instances-def*
by *blast*

lemma *is-inference-ground-instance-one-premise*:
assumes *is-inference-ground-instance-one-premise* D C ι_G γ $\iota_G \in$ *ground.G-Inf*
shows $\iota_G \in$ *inference-ground-instances* (*Infer* [D] C)
using *assms*
unfolding *inference-ground-instances-def*
by *auto*

lemma *is-inference-ground-instance-two-premises*:
assumes *is-inference-ground-instance-two-premises* D E C ι_G γ ϱ_1 ϱ_2 $\iota_G \in$ *ground.G-Inf*
shows $\iota_G \in$ *inference-ground-instances* (*Infer* [D, E] C)
using *assms*
unfolding *inference-ground-instances-def*
by *auto*

lemma *ground-inference_concl-in-ground-instances*:
assumes $\iota_G \in$ *inference-ground-instances* ι
shows *concl-of* $\iota_G \in$ *uncurried-ground-instances* (*concl-of* ι)
proof −
obtain *premises* C $\mathcal{V}$ **where**
ι: ι = *Infer premises* (C, $\mathcal{V}$)
using *Calculus.inference.exhaust*
by (*metis prod.collapse*)

show *?thesis*
using *assms*
unfolding ι *inference-ground-instances-def ground-instances-def*
by (*cases premises rule*: *list-4-cases*) *auto*
qed

lemma *ground-inference-red-in-ground-instances-of-concl*:

assumes $\iota_G \in$ *inference-ground-instances* ι
shows $\iota_G \in$ *ground.Red-I* (*uncurried-ground-instances* (*concl-of* ι))
proof −
from *assms* **have** $\iota_G \in$ *ground.G-Inf*
unfolding *inference-ground-instances-def*
by *blast*

moreover have *concl-of* $\iota_G \in$ *uncurried-ground-instances* (*concl-of* ι)
using *assms ground-inference_concl-in-ground-instances*
by *auto*

ultimately show $\iota_G \in$ *ground.Red-I* (*uncurried-ground-instances* (*concl-of* ι))
using *ground.Red-I-of-Inf-to-N*
by *blast*
qed

sublocale *lifting*:
tiebreaker-lifting
$\bot_F$
inferences
ground.G-Bot
ground.G-entails
ground.G-Inf
ground.GRed-I
ground.GRed-F
uncurried-ground-instances
Some ∘ *inference-ground-instances*
typed-tiebreakers
proof (*unfold-locales*; (*intro impI typed-tiebreakers.wfp typed-tiebreakers.transp*)*?*)

show $\bot_F \neq \{\}$
using *obtain-infinite-variables-per-type-on''*[*of* {}]
by *auto*
next
fix *bottom*
assume *bottom* $\in \bot_F$

then show *uncurried-ground-instances bottom* $\neq \{\}$
unfolding *ground-instances-def*
by *fastforce*
next
fix *bottom*
assume *bottom* $\in \bot_F$

then show *uncurried-ground-instances bottom* $\subseteq$ *ground.G-Bot*
unfolding *ground-instances-def*
by *auto*
next
fix C :: ('*t*, '*v*, '*ty*) *typed-clause*

assume *uncurried-ground-instances* $C \cap$ *ground.G-Bot* $\neq \{\}$

moreover then have *snd* $C = \{\#\}$
unfolding *ground-instances-def*
by *simp*

then have $C = (fst\ C, \{\#\})$
by (*metis split-pairs*)

ultimately show $C \in \bot_F$
unfolding *ground-instances-def*
by *blast*
next
fix ι :: ('*t*, '*v*, '*ty*) *typed-clause inference*

show *the* ((*Some* $\circ$ *inference-ground-instances*) ι) $\subseteq$
ground.GRed-I (*uncurried-ground-instances* (*concl-of* ι))
using *ground-inference-red-in-ground-instances-of-concl*
by *auto*
qed
end

context *ordered-resolution-calculus*
begin

abbreviation *grounded-inference-ground-instances* **where**
grounded-inference-ground-instances select$_G \equiv$
grounded-ordered-resolution-calculus.inference-ground-instances
($\odot$) *apply-subst* ($\cdot t$) *term.vars term.to-ground* ($\prec_t$) *id-subst term.from-ground*
select$_G$ *welltyped*

sublocale
lifting-intersection
inferences
{{#}}
select$_{Gs}$
ground-ordered-resolution-calculus.G-Inf ($\prec_{tG}$)
λ-. *ground-ordered-resolution-calculus.G-entails*
ground-ordered-resolution-calculus.GRed-I ($\prec_{tG}$)
λ-. *ground-ordered-resolution-calculus.GRed-F* ($\prec_{tG}$)
$\bot_F$
λ-. *uncurried-ground-instances*
λ*select*$_G$. *Some* $\circ$ *grounded-inference-ground-instances select*$_G$
typed-tiebreakers
proof (*unfold-locales*; (*intro ballI*)*?*)

show *select*$_{Gs} \neq \{\}$
using *select*$_G$*-simple*

unfolding *select$_{Gs}$-def*
by *blast*
next
fix *select$_G$*
assume *select$_G$* $\in$ *select$_{Gs}$*

then interpret *grounded-ordered-resolution-calculus*
where *select$_G$* = *select$_G$*
by *unfold-locales* (*simp add*: *select$_{Gs}$-def*)

show *consequence-relation ground.G-Bot ground.G-entails*
using *ground.consequence-relation-axioms* **.**

show *tiebreaker-lifting*
$\bot_F$
inferences
ground.G-Bot
ground.G-entails
ground.G-Inf
ground.GRed-I
ground.GRed-F
uncurried-ground-instances
(*Some* $\circ$ *inference-ground-instances*)
typed-tiebreakers
by *unfold-locales*
qed

end

end
theory *Ordered-Resolution-Soundness*
imports *Grounded-Ordered-Resolution*
begin

1.7 Soundness

context *grounded-ordered-resolution-calculus*
begin

notation *lifting.entails-$\mathcal{G}$* (**infix** $\models\!\models_F$ *50*)

lemma *factoring-sound*:
assumes *factoring*: *factoring D C*
shows $\{D\} \models\!\models_F \{C\}$
using *factoring*
proof (*cases D C rule*: *factoring.cases*)
case (*factoringI D* l_1 μ $\mathcal{V}$ t_1 t_2 l_2 D' *C*)

{

fix $I :: 't_G\ set$ **and** $\gamma :: 'subst$

assume
entails-ground-instances: $\forall\, D_G \in ground\text{-}instances\ \mathcal{V}\ D.\ I \models D_G$ **and**
C-is-ground: $clause.is\text{-}ground\ (C \cdot \gamma)$ **and**
type-preserving-γ: $type\text{-}preserving\text{-}on\ (clause.vars\ C)\ \mathcal{V}\ \gamma$ **and**
$\mathcal{V}$: $infinite\text{-}variables\text{-}per\text{-}type\ \mathcal{V}$

obtain γ' **where**
γ'*-is-ground-subst*: $term.is\text{-}ground\text{-}subst\ \gamma'$ **and**
type-preserving-γ': $type\text{-}preserving\ \mathcal{V}\ \gamma'$ **and**
γ'*-γ*: $\forall\, x \in clause.vars\ C.\ x \cdot v\ \gamma = x \cdot v\ \gamma'$
using *clause.type-preserving-ground-subst-extension*[*OF C-is-ground type-preserving-γ*]
.

let $?D_G = clause.to\text{-}ground\ (D \cdot \mu \cdot \gamma')$
let $?D_G' = clause.to\text{-}ground\ (D' \cdot \mu \cdot \gamma')$
let $?l_{G1} = literal.to\text{-}ground\ (l_1 \cdot l\ \mu \cdot l\ \gamma')$
let $?l_{G2} = literal.to\text{-}ground\ (l_2 \cdot l\ \mu \cdot l\ \gamma')$
let $?t_{G1} = term.to\text{-}ground\ (t_1 \cdot t\ \mu \cdot t\ \gamma')$
let $?t_{G2} = term.to\text{-}ground\ (t_2 \cdot t\ \mu \cdot t\ \gamma')$
let $?C_G = clause.to\text{-}ground\ (C \cdot \gamma')$

have *type-preserving-μ*: $type\text{-}preserving\text{-}on\ (clause.vars\ D)\ \mathcal{V}\ \mu$
using *factoringI*(5)
by *blast*

have [*simp*]: $?t_{G2} = ?t_{G1}$
using *factoringI*(6) *term.is-imgu-unifies-pair*
by *metis*

have [*simp*]: $t_1 \cdot t\ \mu \cdot t\ \gamma' = t_1 \cdot t\ \mu \cdot t\ \gamma$
using γ'*-γ term.subst-eq*
unfolding *factoringI*
by *fastforce*

have $?D_G \in ground\text{-}instances\ \mathcal{V}\ D$
proof(*unfold ground-instances-def mem-Collect-eq fst-conv snd-conv*,
intro exI, *intro conjI* $\mathcal{V}$)

show $clause.to\text{-}ground\ (D \cdot \mu \cdot \gamma') = clause.to\text{-}ground\ (D \cdot \mu \odot \gamma')$
by *simp*
next

show $clause.is\text{-}ground\ (D \cdot \mu \odot \gamma')$
using γ'*-is-ground-subst clause.is-ground-subst-is-ground*
by *auto*
next

```
    show type-preserving-on (clause.vars D) 𝒱 (μ ⊙ γ')
      using
        type-preserving-μ
        type-preserving-γ'
        γ'-is-ground-subst
        term.type-preserving-ground-compose-ground-subst
      by presburger
  qed

  then have I ⊨ ?D_G
    using entails-ground-instances
    by blast

  then have I ⊨ clause.to-ground (C · γ)
    using clause.subst-eq[OF γ'-γ[rule-format]]
    unfolding factoringI
    by auto
 }

 then show ?thesis
  unfolding
    factoringI(1, 2)
    ground.G-entails-def
    true-clss-def
    ground-instances-def
  by auto
qed

lemma resolution-sound:
 assumes resolution: resolution D E C
 shows {E, D} ⊨_F {C}
 using resolution
proof (cases D E C rule: resolution.cases)
 case (resolutionI 𝒱₁ 𝒱₂ ϱ₁ ϱ₂ E D 𝒱₃ μ t₁ t₂ l₁ l₂ E' D' C)

 {
  fix I :: 't_G set and γ :: 'subst

  assume
    E-entails-ground-instances: ∀ E_G ∈ ground-instances 𝒱₁ E. I ⊨ E_G and
    D-entails-ground-instances: ∀ D_G ∈ ground-instances 𝒱₂ D. I ⊨ D_G and
    C-is-ground: clause.is-ground (C · γ) and
    type-preserving-γ: type-preserving-on (clause.vars C) 𝒱₃ γ

  obtain γ' where
    γ'-is-ground-subst: term.is-ground-subst γ' and
    type-preserving-γ': type-preserving 𝒱₃ γ' and
    γ'-γ: ∀ x ∈ clause.vars C. x ·v γ = x ·v γ'
   using clause.type-preserving-ground-subst-extension[OF C-is-ground type-preserving-γ]
```

.

let *?E*$_G$ = *clause.to-ground* (*E* · ϱ_1 · μ · γ')
let *?D*$_G$ = *clause.to-ground* (*D* · ϱ_2 · μ · γ')

let *?l*$_{G1}$ = *literal.to-ground* (*l*$_1$ ·*l* ϱ_1 ·*l* μ ·*l* γ')
let *?l*$_{G2}$ = *literal.to-ground* (*l*$_2$ ·*l* ϱ_2 ·*l* μ ·*l* γ')

let *?E*$_G$' = *clause.to-ground* (*E*' · ϱ_1 · μ · γ')
let *?D*$_G$' = *clause.to-ground* (*D*' · ϱ_2 · μ · γ')

let *?t*$_{G1}$ = *term.to-ground* (*t*$_1$ ·*t* ϱ_1 ·*t* μ ·*t* γ')
let *?t*$_{G2}$ = *term.to-ground* (*t*$_2$ ·*t* ϱ_2 ·*t* μ ·*t* γ')

let *?C*$_G$ = *clause.to-ground* (*C* · γ')

have μ-γ'-*is-ground-subst*: *term.is-ground-subst* (μ ⊙ γ')
using *term.is-ground-subst-comp-right*[*OF* γ'-*is-ground-subst*] **.**

have *type-preserving-*μ: *type-preserving-on* (*clause.vars* (*E* · ϱ_1) ∪ *clause.vars* (*D* · ϱ_2)) $\mathcal{V}_3$ μ
using *resolutionI*(*9*)
by *blast*

have *type-preserving-*μ-γ:
type-preserving-on (*clause.vars* (*E* · ϱ_1) ∪ *clause.vars* (*D* · ϱ_2)) $\mathcal{V}_3$ (μ ⊙ γ')
using
type-preserving-γ'
type-preserving-μ
γ'-*is-ground-subst*
term.type-preserving-ground-compose-ground-subst
by *presburger*

note *type-preserving-*ϱ-μ-γ = *term.renaming-ground-subst*[*OF* - μ-γ'-*is-ground-subst*]

have *?E*$_G$ ∈ *ground-instances* $\mathcal{V}_1$ *E*
proof(
unfold ground-instances-def mem-Collect-eq fst-conv snd-conv,
intro exI, *intro conjI resolutionI*)

show *clause.to-ground* (*E* · ϱ_1 · μ · γ') = *clause.to-ground* (*E* · ϱ_1 ⊙ μ ⊙ γ')
by *simp*
next

show *clause.is-ground* (*E* · ϱ_1 ⊙ μ ⊙ γ')
using γ'-*is-ground-subst clause.is-ground-subst-is-ground*
by *auto*
next

```
  show type-preserving-on (clause.vars E) 𝒱₁ (ϱ₁ ⊙ μ ⊙ γ')
    using
      type-preserving-μ-γ
      type-preserving-ϱ-μ-γ[OF resolutionI(6, 18) - resolutionI(16)]
    by (simp add: term.assoc clause.vars-subst)
qed

then have entails-E_G: I ⊨ ?E_G
  using E-entails-ground-instances
  by blast

have ?D_G ∈ ground-instances 𝒱₂ D
proof(
    unfold ground-instances-def mem-Collect-eq fst-conv snd-conv,
    intro exI, intro conjI resolutionI)

  show clause.to-ground (D · ϱ₂ · μ · γ') = clause.to-ground (D · ϱ₂ ⊙ μ ⊙ γ')
    by simp
next

  show clause.is-ground (D · ϱ₂ ⊙ μ ⊙ γ')
    using γ'-is-ground-subst clause.is-ground-subst-is-ground
    by auto
next

  show type-preserving-on (clause.vars D) 𝒱₂ (ϱ₂ ⊙ μ ⊙ γ')
    using
      type-preserving-μ-γ
      type-preserving-ϱ-μ-γ[OF resolutionI(7, 19) - resolutionI(17)]
    by (simp add: term.assoc clause.vars-subst)
qed

then have entails-D_G: I ⊨ ?D_G
  using D-entails-ground-instances
  by blast

have I ⊨ clause.to-ground (C · γ')
proof −
  have [simp]: ?t_G1 = ?t_G2
    using resolutionI(10) term.is-imgu-unifies-pair
    by metis

  have [simp]: ?l_G1 = Neg ?t_G1
    unfolding resolutionI
    by simp

  have [simp]: ?l_G2 = Pos ?t_G2
    unfolding resolutionI
    by simp
```

```
    have [simp]: ?E_G = add-mset ?l_G1 ?E_G'
      unfolding resolutionI
      by simp

    have [simp]: ?D_G = add-mset ?l_G2 ?D_G'
      unfolding resolutionI
      by simp

    have ¬ I ⊨l ?l_G1 ∨ ¬ I ⊨l ?l_G2
      by simp

    then have I ⊨ ?E_G' ∨ I ⊨ ?D_G'
      using entails-E_G entails-D_G
      by force

    moreover have ?C_G = ?E_G' + ?D_G'
      unfolding resolutionI
      by simp

    ultimately show ?thesis
      by auto
  qed

  then have I ⊨ clause.to-ground (C · γ)
    by (metis γ'-γ clause.subst-eq)
 }

 then show ?thesis
  unfolding ground.G-entails-def ground-instances-def true-clss-def resolutionI(1–3)
   by auto
qed

sublocale sound-inference-system inferences ⊥_F (⊨_F)
proof unfold-locales
fix ι

 assume ι ∈ inferences

 then show set (prems-of ι) ⊨_F {concl-of ι}
   using
     factoring-sound
     resolution-sound
   unfolding inferences-def ground.G-entails-def
   by auto
qed

end
```

sublocale *ordered-resolution-calculus* $\subseteq$ *sound-inference-system inferences* $\bot_F$ *entails-*$\mathcal{G}$
proof *unfold-locales*
obtain $select_G$ **where** $select_G$: $select_G \in select_{Gs}$
using *Q-nonempty* **by** *blast*

then interpret *grounded-ordered-resolution-calculus*
where $select_G = select_G$
by *unfold-locales* (*simp add*: $select_{Gs}$*-def*)

fix ι
assume $\iota \in inferences$

then show *entails-*$\mathcal{G}$ (*set* (*prems-of* ι)) {*concl-of* ι}
unfolding *entails-def*
using *sound*
by *blast*
qed

end
theory *Ordered-Resolution-Completeness*
imports
Grounded-Ordered-Resolution
Ground-Ordered-Resolution-Completeness
begin

2 Completeness

context *grounded-ordered-resolution-calculus*
begin

2.1 Liftings

lemma *factoring-lifting*:
fixes
D_G C_G :: $'t_G$ *clause* **and**
D C :: $'t$ *clause* **and**
γ :: $'subst$
defines
[*simp*]: $D_G \equiv clause.to\text{-}ground\ (D \cdot \gamma)$ **and**
[*simp*]: $C_G \equiv clause.to\text{-}ground\ (C \cdot \gamma)$
assumes
ground-factoring: *ground.factoring* D_G C_G **and**
D-grounding: *clause.is-ground* $(D \cdot \gamma)$ **and**
C-grounding: *clause.is-ground* $(C \cdot \gamma)$ **and**
select: *clause.from-ground* $(select_G\ D_G) = (select\ D) \cdot \gamma$ **and**
type-preserving-γ: *type-preserving-on* (*clause.vars* D) $\mathcal{V}$ γ **and**
$\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$
obtains C'

where
factoring $(\mathcal{V}, D)$ $(\mathcal{V}, C')$
Infer $[D_G]$ $C_G \in$ *inference-ground-instances* (*Infer* $[(\mathcal{V}, D)]$ $(\mathcal{V}, C')$)
$C' \cdot \gamma = C \cdot \gamma$
using *ground-factoring*
proof(*cases* D_G C_G *rule*: *ground.factoring.cases*)
case *ground-factoringI*: (*factoringI* l_G $D_G{}'$ t_G)

have $D \neq \{\#\}$
using *ground-factoringI*(*3*)
by *auto*

then obtain l_1 **where**
l_1-*is-maximal*: *is-maximal* l_1 D **and**
l_1-γ-*is-maximal*: *is-maximal* $(l_1 \cdot l \ \gamma)$ $(D \cdot \gamma)$
using *that obtain-maximal-literal D-grounding*
by *blast*

obtain t_1 **where**
l_1: $l_1 = (Pos\ t_1)$ **and**
l_1-γ: $l_1 \cdot l \ \gamma = (Pos\ (term.from\text{-}ground\ t_G))$ **and**
t_1-γ: $t_1 \cdot t \ \gamma = term.from\text{-}ground\ t_G$
proof−
have *is-maximal* (*literal.from-ground* l_G) $(D \cdot \gamma)$
using *D-grounding ground-factoringI*(*2*)
by *auto*

then have $l_1 \cdot l \ \gamma = (Pos\ (term.from\text{-}ground\ t_G))$
unfolding *ground-factoringI*(*4*)
using *unique-maximal-in-ground-clause*[*OF D-grounding* l_1-γ-*is-maximal*]
by *simp*

then show *?thesis*
using *that*
by (*metis Neg-atm-of-iff clause-safe-unfolds*(*9*) *literal.collapse*(*1*) *literal.sel*(*1*)
subst-polarity-stable(*2*))
qed

obtain l_2 D' **where**
l_2-γ: $l_2 \cdot l \ \gamma = Pos\ (term.from\text{-}ground\ t_G)$ **and**
D: $D = add\text{-}mset\ l_1\ (add\text{-}mset\ l_2\ D')$
proof−

obtain D'' **where** D: $D = add\text{-}mset\ l_1\ D''$
using *maximal-in-clause*[*OF* l_1-*is-maximal*]
by (*meson multi-member-split*)

moreover have $D \cdot \gamma = clause.from\text{-}ground\ (add\text{-}mset\ l_G\ (\ add\text{-}mset\ l_G\ D_G{}'))$
using *ground-factoringI*(*3*) C_G-*def*

by (*metis* D_G*-def D-grounding clause.to-ground-inverse*)

ultimately have $D'' \cdot \gamma$ = *add-mset* (*literal.from-ground* l_G) (*clause.from-ground* D_G')
using l_1-γ
by (*simp add*: *ground-factoringI*(*4*))

then obtain l_2 **where** $l_2 \cdot l \ \gamma$ = *Pos* (*term.from-ground* t_G) $l_2 \in\#$ D''
unfolding *clause.subst-def ground-factoringI*
using *msed-map-invR*
by *force*

then show *?thesis*
using *that*
unfolding D
by (*metis mset-add*)
qed

then obtain t_2 **where**
l_2: l_2 = (*Pos* t_2) **and**
t_2-γ: $t_2 \cdot t \ \gamma$ = *term.from-ground* t_G
unfolding *ground-factoringI*(*2*)
by (*metis clause-safe-unfolds*(*9*) *is-pos-def literal.sel*(*1*) *subst-polarity-stable*(*2*))

have D'-γ: $D' \cdot \gamma$ = *clause.from-ground* D_G'
using D *D-grounding ground-factoringI* l_1-γ l_2-γ
by *force*

obtain μ **where**
type-preserving-μ: *type-preserving-on* (*clause.vars* D) $\mathcal{V}$ μ **and**
imgu: *term.is-imgu* μ $\{\{t_1, t_2\}\}$
proof (*rule obtain-type-preserving-on-imgu*[*OF - that*], *intro conjI*)

show $t_1 \cdot t \ \gamma = t_2 \cdot t \ \gamma$
using t_1-γ t_2-γ
by *argo*
next

show *type-preserving-on* (*term.vars* $t_1 \cup$ *term.vars* t_2) $\mathcal{V}$ γ
using *type-preserving-*γ
unfolding D l_1 l_2
by *auto*
qed

obtain σ **where** γ: $\gamma = \mu \odot \sigma$
using *term.obtain-imgu-absorption*[*of* γ, *OF - imgu*] t_1-γ t_2-γ
by *auto*

let *?C''* = *add-mset* l_1 D'

let *?C′ = ?C″ · μ*

show *?thesis*
proof(*rule that*)

show *factoring*: *factoring* (*V*, *D*) (*V*, *?C′*)
proof (*rule factoringI*; (*rule D imgu type-preserving-μ refl V*)*?*)

show *select D* = {#}
using *ground-factoringI*(*1*) *select*
by *simp*
next

have l_1 ·*l* *μ* ∈# *D* · *μ*
using l_1*-is-maximal clause.subst-in-to-set-subst maximal-in-clause*
by *blast*

then show *is-maximal* (l_1 ·*l* *μ*) (*D* · *μ*)
using *is-maximal-if-grounding-is-maximal D-grounding* l_1*-γ-is-maximal*
unfolding *γ*
by *auto*
next

show *D* = *add-mset* l_1 (*add-mset* (*Pos* t_2) *D′*)
unfolding *D* l_2 **..**
next
show l_1 = *Pos* t_1
using l_1 **.**
qed

show *C′-γ*: *?C′* · *γ* = *C* · *γ*
proof−
have *term.is-idem μ*
using *imgu*
unfolding *term.is-imgu-iff-is-idem-and-is-mgu*
by *blast*

then have *μ-γ*: *μ* ⊙ *γ* = *γ*
unfolding *γ term.is-idem-def term.assoc*[*symmetric*]
by *argo*

have *C* · *γ* = *clause.from-ground* (*add-mset* (*Pos* t_G) D_G′)
using *ground-factoringI*(*5*) *clause.to-ground-eq*[*OF C-grounding clause.ground-is-ground*]
unfolding C_G*-def*
by (*metis clause.from-ground-inverse ground-factoringI*(*4*))

also have ... = *?C″* · *γ*
using t_1*-γ D′-γ* l_1*-γ*
by *auto*

```
      also have ... = ?C' · γ
        unfolding clause.subst-comp-subst[symmetric] μ-γ ..

      finally show ?thesis ..
    qed

    show Infer [D_G] C_G ∈ inference-ground-instances (Infer [(V, D)] (V, ?C'))
    proof (rule is-inference-ground-instance-one-premise)
      show is-inference-ground-instance-one-premise (V, D) (V, ?C') (Infer [D_G]
C_G) γ
      proof(unfold split, intro conjI; (rule refl V)?)

        show inference.is-ground (Infer [D] ?C' ·ι γ)
          using C-grounding D-grounding C'-γ
          by auto
      next

        show Infer [D_G] C_G = inference.to-ground (Infer [D] ?C' ·ι γ)
          using C'-γ
          by simp
      next

        have clause.vars ?C' ⊆ clause.vars D
          using clause.variables-in-base-imgu[OF imgu, of ?C'']
          unfolding D l_1 l_2
          by auto

        then show type-preserving-on (clause.vars ?C') V γ
          using type-preserving-γ
          by blast
      qed

      show Infer [D_G] C_G ∈ ground.G-Inf
        unfolding ground.G-Inf-def
        using ground-factoring
        by blast
    qed
  qed
qed

lemma resolution-lifting:
  fixes
    E_G D_G C_G :: 't_G clause and
    E D C :: 't clause and
    γ ϱ_1 ϱ_2 :: 'subst and
    V_1 V_2 :: ('v, 'ty) var-types
  defines
    [simp]: E_G ≡ clause.to-ground (E · ϱ_1 ⊙ γ) and
```

[*simp*]: $D_G \equiv$ *clause.to-ground* $(D \cdot \varrho_2 \odot \gamma)$ **and**
[*simp*]: $C_G \equiv$ *clause.to-ground* $(C \cdot \gamma)$ **and**
[*simp*]: $N_G \equiv$ *ground-instances* $\mathcal{V}_1$ $E \cup$ *ground-instances* $\mathcal{V}_2$ D **and**
[*simp*]: $\iota_G \equiv$ *Infer* $[D_G, E_G]$ C_G
assumes
ground-resolution: *ground.resolution* D_G E_G C_G **and**
ϱ_1: *term.is-renaming* ϱ_1 **and**
ϱ_2: *term.is-renaming* ϱ_2 **and**
rename-apart: *clause.vars* $(E \cdot \varrho_1) \cap$ *clause.vars* $(D \cdot \varrho_2) = \{\}$ **and**
E-grounding: *clause.is-ground* $(E \cdot \varrho_1 \odot \gamma)$ **and**
D-grounding: *clause.is-ground* $(D \cdot \varrho_2 \odot \gamma)$ **and**
C-grounding: *clause.is-ground* $(C \cdot \gamma)$ **and**
select-from-E: *clause.from-ground* $(select_G\ E_G) = (select\ E) \cdot \varrho_1 \odot \gamma$ **and**
select-from-D: *clause.from-ground* $(select_G\ D_G) = (select\ D) \cdot \varrho_2 \odot \gamma$ **and**
type-preserving-ϱ_1*-*γ: *type-preserving-on* (*clause.vars* E) $\mathcal{V}_1$ $(\varrho_1 \odot \gamma)$ **and**
type-preserving-ϱ_2*-*γ: *type-preserving-on* (*clause.vars* D) $\mathcal{V}_2$ $(\varrho_2 \odot \gamma)$ **and**
type-preserving-ϱ_1: *type-preserving-on* (*clause.vars* E) $\mathcal{V}_1$ ϱ_1 **and**
type-preserving-ϱ_2: *type-preserving-on* (*clause.vars* D) $\mathcal{V}_2$ ϱ_2 **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$
obtains C' $\mathcal{V}_3$
where
resolution $(\mathcal{V}_2, D)$ $(\mathcal{V}_1, E)$ $(\mathcal{V}_3, C')$
$\iota_G \in$ *inference-ground-instances* (*Infer* $[(\mathcal{V}_2, D), (\mathcal{V}_1, E)]$ $(\mathcal{V}_3, C')$)
$C' \cdot \gamma = C \cdot \gamma$
using *ground-resolution*
proof(*cases* D_G E_G C_G *rule*: *ground.resolution.cases*)
case *ground-resolutionI*: (*resolutionI* l_{G1} l_{G2} $E_G{}'$ $D_G{}'$ t_G)

have *E-*γ: $E \cdot \varrho_1 \odot \gamma =$ *clause.from-ground* (*add-mset* l_{G1} $E_G{}'$)
using *ground-resolutionI*(5)
unfolding E_G*-def*
by (*metis E-grounding clause.to-ground-inverse*)

have *D-*γ: $D \cdot \varrho_2 \odot \gamma =$ *clause.from-ground* (*add-mset* l_{G2} $D_G{}'$)
using *ground-resolutionI*(6) D_G*-def*
by (*metis D-grounding clause.to-ground-inverse*)

let *?select*$_G$*-empty* $=$ *select*$_G$ (*clause.to-ground* $(E \cdot \varrho_1 \odot \gamma)) = \{\#\}$
let *?select*$_G$*-not-empty* $=$ *select*$_G$ (*clause.to-ground* $(E \cdot \varrho_1 \odot \gamma)) \neq \{\#\}$

obtain l_1 **where**
l_1*-*γ: $l_1 \cdot l\ \varrho_1 \odot \gamma =$ *literal.from-ground* l_{G1} **and**
l_1*-is-maximal*: *?select*$_G$*-empty* $\Longrightarrow$ *is-maximal* l_1 E **and**
l_1*-*γ*-is-maximal*: *?select*$_G$*-empty* $\Longrightarrow$ *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \gamma)$ $(E \cdot \varrho_1 \odot \gamma)$
and
l_1*-selected*: *?select*$_G$*-not-empty* $\Longrightarrow$ *is-maximal* l_1 (*select* E) **and**
l_1*-*γ*-selected*: *?select*$_G$*-not-empty* $\Longrightarrow$*is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \gamma)$ (*select* $E \cdot \varrho_1 \odot \gamma$) **and**

```
    l₁-in-E: l₁ ∈# E
  proof −
    have E-not-empty: E ≠ {#}
      using ground-resolutionI(5)
      by auto

    then obtain max-l where
      is-maximal max-l E and
      is-max-in-E-γ: is-maximal (max-l ·l ϱ₁ ⊙ γ) (E · ϱ₁ ⊙ γ)
      using that E-grounding obtain-maximal-literal E-not-empty
      by blast

    moreover then have max-l ∈# E
      unfolding is-maximal-def
      by blast

    moreover have max-l ·l ϱ₁ ⊙ γ = literal.from-ground l_G1 if ?select_G-empty
    proof −
      have ground-is-maximal l_G1 E_G
        using ground-resolutionI(2) that
        unfolding is-maximal-def
        by simp

      then show ?thesis
          using unique-maximal-in-ground-clause[OF E-grounding is-max-in-E-γ]
E-grounding
        unfolding ground-resolutionI(3)
        by simp
    qed

    moreover then obtain selected-l where
      is-maximal selected-l (select E)
      is-maximal (selected-l ·l ϱ₁ ⊙ γ) ((select E) · ϱ₁ ⊙ γ)
      selected-l ·l ϱ₁ ⊙ γ = literal.from-ground l_G1
    if ?select_G-not-empty
    proof −

      have is-maximal (literal.from-ground l_G1) (select E · ϱ₁ ⊙ γ)
        if ?select_G-not-empty
        using ground-resolutionI(2) that
        unfolding ground-resolutionI(3)
        by (metis E_G-def select-from-E)

      then show ?thesis
        using
          that
          obtain-maximal-literal[OF - select-ground-subst[OF E-grounding]]
          unique-maximal-in-ground-clause[OF select-ground-subst[OF E-grounding]]
      by (metis (no-types, lifting) clause.magma-subst-empty(1) is-maximal-not-empty)
```

qed

moreover then have *selected-l* $\in\#$ E **if** *?select*$_G$*-not-empty*
using *that maximal-in-clause mset-subset-eqD select-subset*
by *meson*

ultimately show *?thesis*
using *that ground-resolutionI*
by *blast*
qed

obtain E' **where** E: $E = add\text{-}mset\ l_1\ E'$
by (*meson* l_1*-in-E multi-member-split*)

then have E'-γ: $E' \cdot \varrho_1 \odot \gamma = clause.from\text{-}ground\ E_G{}'$
using l_1-γ E-γ
by *auto*

obtain t_1 **where**
l_1: $l_1 = Neg\ t_1$ **and**
t_1-γ: $t_1 \cdot t\ \varrho_1 \odot \gamma = term.from\text{-}ground\ t_G$
using l_1-γ
by (*metis Neg-atm-of-iff literal-from-ground-atom-from-ground*(*1*) *clause-safe-unfolds*(*9*)
ground-resolutionI(*7*) *literal.sel*(*2*) *subst-polarity-stable*(*2*))

obtain l_2 **where**
l_2-γ: $l_2 \cdot l\ \varrho_2 \odot \gamma = literal.from\text{-}ground\ l_{G2}$ **and**
l_2*-is-strictly-maximal*: *is-strictly-maximal* l_2 D
proof−
have *is-strictly-maximal* (*literal.from-ground* l_{G2}) ($D \cdot \varrho_2 \odot \gamma$)
using *ground-resolutionI*(*4*) *D-grounding*
by *simp*

then show *?thesis*
using *obtain-strictly-maximal-literal*[*OF D-grounding*] *that*
by *force*
qed

then have l_2*-in-D*: $l_2 \in\#$ D
using *strictly-maximal-in-clause*
by *blast*

from l_2-γ **have** l_2-γ: $l_2 \cdot l\ \varrho_2 \odot \gamma = (Pos\ (term.from\text{-}ground\ t_G))$
unfolding *ground-resolutionI*
by *simp*

then obtain t_2 **where**
l_2: $l_2 = Pos\ t_2$ **and**
t_2-γ: $t_2 \cdot t\ \varrho_2 \odot \gamma = term.from\text{-}ground\ t_G$

by (*metis clause-safe-unfolds*(*9*) *literal.collapse*(*1*) *literal.disc*(*1*) *literal.sel*(*1*) *subst-polarity-stable*(*2*))

obtain D' **where** *D*: $D = \textit{add-mset}\ l_2\ D'$
by (*meson* l_2*-in-D multi-member-split*)

then have D'-γ: $D' \cdot \varrho_2 \odot \gamma = \textit{clause.from-ground}\ {D_G}'$
using *D*-γ l_2-γ
unfolding *ground-resolutionI*
by *auto*

obtain $\mathcal{V}_3$ **where**
$\mathcal{V}_3$: *infinite-variables-per-type* $\mathcal{V}_3$ **and**
$\mathcal{V}_1$-$\mathcal{V}_3$: $\forall\, x{\in}\textit{clause.vars}\ E.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (\textit{term.rename}\ \varrho_1\ x)$ **and**
$\mathcal{V}_2$-$\mathcal{V}_3$: $\forall\, x{\in}\textit{clause.vars}\ D.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (\textit{term.rename}\ \varrho_2\ x)$
using *clause.obtain-merged-*$\mathcal{V}$[*OF* ϱ_1 ϱ_2 *rename-apart clause.finite-vars clause.finite-vars infinite-UNIV*] **.**

have *type-preserving-*γ: *type-preserving-on* (*clause.vars* $(E \cdot \varrho_1) \cup$ *clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3$ γ
proof(*unfold Set.ball-Un*, *intro conjI*)

show *type-preserving-on* (*clause.vars* $(E \cdot \varrho_1)$) $\mathcal{V}_3$ γ
using *clause.renaming-grounding*[*OF* ϱ_1 *type-preserving-*ϱ_1*-*γ *E-grounding* $\mathcal{V}_1$-$\mathcal{V}_3$] **.**
next

show *type-preserving-on* (*clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3$ γ
using *clause.renaming-grounding*[*OF* ϱ_2 *type-preserving-*ϱ_2*-*γ *D-grounding* $\mathcal{V}_2$-$\mathcal{V}_3$] **.**
qed

obtain μ **where**
type-preserving-μ: *type-preserving-on* (*clause.vars* $(E \cdot \varrho_1) \cup$ *clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3$ μ **and**
imgu: *term.is-imgu* μ $\{\{t_1 \cdot t\ \varrho_1,\ t_2 \cdot t\ \varrho_2\}\}$
proof (*rule obtain-type-preserving-on-imgu*[*OF - that*], *intro conjI*)

show $t_1 \cdot t\ \varrho_1 \cdot t\ \gamma = t_2 \cdot t\ \varrho_2 \cdot t\ \gamma$
using t_1-γ t_2-γ
by *simp*
next

show *type-preserving-on* (*term.vars* $(t_1 \cdot t\ \varrho_1) \cup$ *term.vars* $(t_2 \cdot t\ \varrho_2)$) $\mathcal{V}_3$ γ
using *type-preserving-*γ
unfolding *E D* l_1 l_2
by *auto*
qed

obtain σ **where** γ: $\gamma = \mu \odot \sigma$
using *term.obtain-imgu-absorption*[*of* γ, *OF - imgu*] t_1-γ t_2-γ
by *auto*

define C' **where**
C': $C' = (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu$

show *?thesis*
proof(*rule that*)

show *resolution*: *resolution* ($\mathcal{V}_2$, D) ($\mathcal{V}_1$, E) ($\mathcal{V}_3$, C')
proof (*rule resolutionI*; ((*rule* ϱ_1 ϱ_2 E D l_1 l_2 *imgu type-preserving-*μ *rename-apart*
type-preserving-ϱ_1 *type-preserving-*ϱ_2 $\mathcal{V}_1$ $\mathcal{V}_2$ C' $\mathcal{V}_1$-$\mathcal{V}_3$ $\mathcal{V}_2$-$\mathcal{V}_3$)+)*?*)

show $\neg$ $E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu$
proof(*rule clause.order.ground-less-not-less-eq*)

show *clause.vars* ($D \cdot \varrho_2 \odot \mu \cdot \sigma$) = {}
using *D-grounding*
unfolding γ
by *simp*

show *clause.vars* ($E \cdot \varrho_1 \odot \mu \cdot \sigma$) = {}
using *E-grounding*
unfolding γ
by *simp*

show $D \cdot \varrho_2 \odot \mu \cdot \sigma \prec_c E \cdot \varrho_1 \odot \mu \cdot \sigma$
using *ground-resolutionI*(*1*) *D-grounding E-grounding*
unfolding E_G*-def* D_G*-def clause.order.less*$_G$*-def* γ
by *simp*
qed
next
assume *select* E = {#}

moreover then have *?select*$_G$*-empty*
using *is-maximal-not-empty* l_1*-selected*
by *blast*

moreover have $l_1 \cdot l \varrho_1 \odot \mu \in\# E \cdot \varrho_1 \odot \mu$
using l_1*-in-E*
by *blast*

ultimately show *is-maximal* ($l_1 \cdot l \varrho_1 \odot \mu$) ($E \cdot \varrho_1 \odot \mu$)
using l_1-γ*-is-maximal is-maximal-if-grounding-is-maximal E-grounding*
unfolding γ
by *force*

next
assume *select* $E \neq \{\#\}$

then have $\neg$*?select*$_G$*-empty*
using *is-maximal-not-empty* l_1*-selected select-from-E*
by *auto*

moreover have $l_1 \cdot l\ \varrho_1 \odot \mu \in\#$ *select* $E \cdot \varrho_1 \odot \mu$
using l_1*-selected maximal-in-clause calculation*
by *blast*

ultimately show *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \mu)$ $(\textit{select}\ E \cdot \varrho_1 \odot \mu)$
using *select-ground-subst*[*OF E-grounding*] *is-maximal-if-grounding-is-maximal* l_1*-*γ*-selected*
unfolding γ
by *fastforce*
next

show *select* $D = \{\#\}$
using *ground-resolutionI*(*3*) *select-from-D*
by *fastforce*
next

show *is-strictly-maximal* $(l_2 \cdot l\ \varrho_2 \odot \mu)$ $(D \cdot \varrho_2 \odot \mu)$
proof(*rule is-strictly-maximal-if-grounding-is-strictly-maximal*)

show $l_2 \cdot l\ \varrho_2 \odot \mu \in\#\ D \cdot \varrho_2 \odot \mu$
using l_2*-in-D*
by *blast*

show *clause.is-ground* $(D \cdot \varrho_2 \odot \mu \cdot \sigma)$
using *D-grounding*[*unfolded* γ]
by *simp*

show *is-strictly-maximal* $(l_2 \cdot l\ \varrho_2 \odot \mu \cdot l\ \sigma)$ $(D \cdot \varrho_2 \odot \mu \cdot \sigma)$
using l_2*-*γ *D-*γ *ground-resolutionI*(*4*)
unfolding γ *ground-resolutionI*
by *fastforce*
qed
qed

show C'*-*γ: $C' \cdot \gamma = C \cdot \gamma$
proof $-$

have *term.is-idem* μ
using *imgu term.is-imgu-iff-is-idem-and-is-mgu*
by *blast*

then have μ*-*γ: $\mu \odot \gamma = \gamma$

```
      unfolding γ term.is-idem-def
      by (metis term.assoc)

    have C · γ = (clause.from-ground E_G' + clause.from-ground D_G')
      using ground-resolutionI(8, 9) clause.to-ground-inverse[OF C-grounding]
      by auto

    then show ?thesis
      unfolding
        C'
        E'-γ[symmetric]
        D'-γ[symmetric]
        clause.subst-comp-subst[symmetric]
        μ-γ
      by simp
  qed

  show ι_G ∈ inference-ground-instances (Infer [(𝒱_2, D), (𝒱_1, E)] (𝒱_3, C'))
  proof (rule is-inference-ground-instance-two-premises)

    show is-inference-ground-instance-two-premises (𝒱_2, D) (𝒱_1, E) (𝒱_3, C') ι_G
γ ϱ_1 ϱ_2
    proof(unfold split, intro conjI;
        (rule ϱ_1 ϱ_2 rename-apart refl 𝒱_1 𝒱_2 𝒱_3)?)

      show inference.is-ground (Infer [D · ϱ_2, E · ϱ_1] C' ·ι γ)
        using D-grounding E-grounding C-grounding C'-γ
        by auto
    next

      show ι_G = inference.to-ground (Infer [D · ϱ_2, E · ϱ_1] C' ·ι γ)
        using C'-γ
        by simp
    next

      show type-preserving-on (clause.vars C') 𝒱_3 γ
      proof(rule type-preserving-on-subset[OF type-preserving-γ])

        show clause.vars C' ⊆ clause.vars (E · ϱ_1) ∪ clause.vars (D · ϱ_2)
        proof (unfold subset-eq, intro ballI)
          fix x

          have is-imgu: term.is-imgu μ {{t_1 ·t ϱ_1, t_2 ·t ϱ_2}}
            using imgu
            by blast

          assume x ∈ clause.vars C'

          then consider
```

```
        (E') x ∈ clause.vars (E' · ϱ1 ⊙ μ) |
        (D') x ∈ clause.vars (D' · ϱ2 ⊙ μ)
        unfolding C'
        by auto

      then show x ∈ clause.vars (E · ϱ1) ∪ clause.vars (D · ϱ2)
      proof cases
        case E'

        then show ?thesis
          using clause.variables-in-base-imgu[OF is-imgu]
          unfolding E l1 D l2
          by auto
      next
        case D'

        then show ?thesis
          using clause.variables-in-base-imgu[OF is-imgu]
          unfolding E l1 D l2
          by auto
      qed
    qed
  qed
  show ιG ∈ ground.G-Inf
    unfolding ground.G-Inf-def
    using ground-resolution
    by simp
 qed
qed
qed
```

2.2 Ground instances

```
context
  fixes ιG N
  assumes
    subst-stability: subst-stability-on N and
    ιG-Inf-from: ιG ∈ ground.Inf-from-q select_G (⋃(uncurried-ground-instances ‘
N))
begin

lemma factoring-ground-instance:
  assumes ground-factoring: ιG ∈ ground.factoring-inferences
  obtains ι where
    ι ∈ Inf-from N
    ιG ∈ inference-ground-instances ι
proof −
```

obtain D_G C_G **where**
ι_G: $\iota_G = Infer\ [D_G]\ C_G$ **and**
ground-inference: *ground.factoring* D_G C_G
using *ground-factoring*
by *blast*

have D_G*-in-groundings*: $D_G \in \bigcup(uncurried\text{-}ground\text{-}instances\ `\ N)$
using ι_G*-Inf-from*
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by *simp*

obtain $D\ \gamma\ \mathcal{V}$ **where**
D-grounding: *clause.is-ground* $(D \cdot \gamma)$ **and**
type-preserving-γ: *type-preserving-on* $(clause.vars\ D)\ \mathcal{V}\ \gamma$ **and**
$\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$ **and**
D-in-N: $(\mathcal{V},\ D) \in N$ **and**
$select_G\ D_G = clause.to\text{-}ground\ (select\ D \cdot \gamma)$
$D \cdot \gamma = clause.from\text{-}ground\ D_G$
using *subst-stability*[*rule-format*, *OF* D_G*-in-groundings*]
by *blast*

then have
D_G: $D_G = clause.to\text{-}ground\ (D \cdot \gamma)$ **and**
select: *clause.from-ground* $(select_G\ D_G) = select\ D \cdot \gamma$
by (*simp-all add*: *select-ground-subst*)

obtain C **where**
C_G: $C_G = clause.to\text{-}ground\ (C \cdot \gamma)$ **and**
C-grounding: *clause.is-ground* $(C \cdot \gamma)$
by (*metis clause.all-subst-ident-iff-ground clause.from-ground-inverse*
clause.ground-is-ground)

obtain C' **where**
factoring: *factoring* $(\mathcal{V},\ D)$ $(\mathcal{V},\ C')$ **and**
inference-ground-instances: $\iota_G \in inference\text{-}ground\text{-}instances\ (Infer\ [(\mathcal{V},\ D)]\ (\mathcal{V},\ C'))$ **and**
C'*-C*: $C' \cdot \gamma = C \cdot \gamma$
using
factoring-lifting[*OF*
ground-inference[*unfolded* D_G C_G]
D-grounding
C-grounding
select[*unfolded* D_G]
type-preserving-γ
$\mathcal{V}$]
unfolding D_G C_G ι_G **.**

let $?\iota = Infer\ [(\mathcal{V},\ D)]\ (\mathcal{V},\ C')$

show *?thesis*
proof(*rule that*[*OF - inference-ground-instances*])

show *?ι ∈ Inf-from N*
using *D-in-N factoring*
unfolding *Inf-from-def inferences-def inference-system.Inf-from-def*
by *auto*
qed
qed

lemma *resolution-ground-instance*:
assumes *ground-resolution*: $\iota_G \in$ *ground.resolution-inferences*
obtains ι **where**
$\iota \in$ *Inf-from N*
$\iota_G \in$ *inference-ground-instances* ι
proof−
obtain E_G D_G C_G **where**
$\iota_G : \iota_G =$ *Infer* $[D_G, E_G]$ C_G **and**
ground-resolution: *ground.resolution* D_G E_G C_G
using *assms*(*1*)
by *blast*

have
E_G*-in-groundings*: $E_G \in \bigcup$ (*uncurried-ground-instances* ‘ *N*) **and**
D_G*-in-groundings*: $D_G \in \bigcup$ (*uncurried-ground-instances* ‘ *N*)
using ι_G*-Inf-from*
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by *simp-all*

obtain *E* $\mathcal{V}_1$ γ_1 **where**
E-grounding: *clause.is-ground* $(E \cdot \gamma_1)$ **and**
type-preserving-γ_1: *type-preserving-on* (*clause.vars E*) $\mathcal{V}_1$ γ_1 **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
E-in-N: $(\mathcal{V}_1, E) \in N$ **and**
$select_G$ E_G = *clause.to-ground* (*select E* $\cdot$ γ_1)
$E \cdot \gamma_1$ = *clause.from-ground* E_G
using *subst-stability*[*rule-format*, *OF* E_G*-in-groundings*]
by *blast*

then have
E_G: E_G = *clause.to-ground* $(E \cdot \gamma_1)$ **and**
select-from-E: *clause.from-ground* $(select_G\ E_G)$ = *select E* $\cdot$ γ_1
by (*simp-all add*: *select-ground-subst*)

obtain *D* $\mathcal{V}_2$ γ_2 **where**
D-grounding: *clause.is-ground* $(D \cdot \gamma_2)$ **and**
type-preserving-γ_2: *type-preserving-on* (*clause.vars D*) $\mathcal{V}_2$ γ_2 **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**

D-in-N: $(\mathcal{V}_2,\ D) \in N$ **and**
$select_G\ D_G = clause.to\text{-}ground\ (select\ D \cdot \gamma_2)$
$D \cdot \gamma_2 = clause.from\text{-}ground\ D_G$
using *subst-stability*[*rule-format*, *OF* D_G*-in-groundings*]
by *blast*

then have
D_G: $D_G = clause.to\text{-}ground\ (D \cdot \gamma_2)$ **and**
select-from-D: $clause.from\text{-}ground\ (select_G\ D_G) = select\ D \cdot \gamma_2$
by (*simp-all add*: *select-ground-subst*)

obtain $\varrho_1\ \varrho_2\ \gamma$ **where**
ϱ_1: *term.is-renaming* ϱ_1 **and**
ϱ_2: *term.is-renaming* ϱ_2 **and**
rename-apart: $clause.vars\ (E \cdot \varrho_1) \cap clause.vars\ (D \cdot \varrho_2) = \{\}$ **and**
type-preserving-ϱ_1: *type-preserving-on* $(clause.vars\ E)\ \mathcal{V}_1\ \varrho_1$ **and**
type-preserving-ϱ_2: *type-preserving-on* $(clause.vars\ D)\ \mathcal{V}_2\ \varrho_2$ **and**
γ_1-γ: $\forall x \in clause.vars\ E.\ x \cdot_v \gamma_1 = x \cdot_v \varrho_1 \odot \gamma$ **and**
γ_2-γ: $\forall x \in clause.vars\ D.\ x \cdot_v \gamma_2 = x \cdot_v \varrho_2 \odot \gamma$
using *clause.obtain-merged-grounding*[*OF*
type-preserving-γ_1 *type-preserving-*γ_2 *E-grounding D-grounding* $\mathcal{V}_2$ *clause.finite-vars*]
.

have *E-grounding*: $clause.is\text{-}ground\ (E \cdot \varrho_1 \odot \gamma)$
using *clause.subst-eq* γ_1-γ *E-grounding*
by *fastforce*

have E_G: $E_G = clause.to\text{-}ground\ (E \cdot \varrho_1 \odot \gamma)$
using *clause.subst-eq* γ_1-γ E_G
by *fastforce*

have *D-grounding*: $clause.is\text{-}ground\ (D \cdot \varrho_2 \odot \gamma)$
using *clause.subst-eq* γ_2-γ *D-grounding*
by *fastforce*

have D_G: $D_G = clause.to\text{-}ground\ (D \cdot \varrho_2 \odot \gamma)$
using *clause.subst-eq* γ_2-γ D_G
by *fastforce*

have *type-preserving-*ϱ_1-γ: *type-preserving-on* $(clause.vars\ E)\ \mathcal{V}_1\ (\varrho_1 \odot \gamma)$
using *type-preserving-*γ_1 γ_1-γ
by *fastforce*

have *type-preserving-*ϱ_2-γ: *type-preserving-on* $(clause.vars\ D)\ \mathcal{V}_2\ (\varrho_2 \odot \gamma)$
using *type-preserving-*γ_2 γ_2-γ
by *fastforce*

have *select-from-E*:
$clause.from\text{-}ground\ (select_G\ (clause.to\text{-}ground\ (E \cdot \varrho_1 \odot \gamma))) = select\ E \cdot \varrho_1 \odot$

γ
proof −
have $E \cdot \gamma_1 = E \cdot \varrho_1 \odot \gamma$
using γ_1-γ *clause.subst-eq*
by *fast*

moreover have *select* $E \cdot \gamma_1 =$ *select* $E \cdot \varrho_1 \cdot \gamma$
using *clause.subst-eq* γ_1-γ *select-vars-subset*
by (*metis* (*no-types*, *lifting*) *clause.comp-subst.left.monoid-action-compatibility in-mono*)

ultimately show *?thesis*
using *select-from-E*
unfolding E_G
by *simp*
qed

have *select-from-D*:
clause.from-ground (*select*$_G$ (*clause.to-ground* ($D \cdot \varrho_2 \odot \gamma$))) = *select* $D \cdot \varrho_2 \odot \gamma$
proof −
have $D \cdot \gamma_2 = D \cdot \varrho_2 \odot \gamma$
using γ_2-γ *clause.subst-eq*
by *fast*

moreover have *select* $D \cdot \gamma_2 =$ *select* $D \cdot \varrho_2 \cdot \gamma$
using *clause.subst-eq* γ_2-γ *select-vars-subset*[*of D*]
by (*metis* (*no-types*, *lifting*) *clause.comp-subst.left.monoid-action-compatibility in-mono*)

ultimately show *?thesis*
using *select-from-D*
unfolding D_G
by *simp*
qed

obtain C **where**
C-grounding: *clause.is-ground* ($C \cdot \gamma$) **and**
C_G: $C_G =$ *clause.to-ground* ($C \cdot \gamma$)
by (*metis clause.all-subst-ident-if-ground clause.from-ground-inverse clause.ground-is-ground*)

have *ground-instances* $\mathcal{V}_1$ $E \cup$ *ground-instances* $\mathcal{V}_2$ $D \subseteq \bigcup$ (*uncurried-ground-instances* ‘ N)
using *E-in-N D-in-N*
by *force*

obtain C' $\mathcal{V}_3$ **where**
resolution: *resolution* ($\mathcal{V}_2$, D) ($\mathcal{V}_1$, E) ($\mathcal{V}_3$, C') **and**
inference-groundings: $\iota_G \in$ *inference-ground-instances* (*Infer* [($\mathcal{V}_2$, D), ($\mathcal{V}_1$, E)]

```
(𝒱₃, C′)) and
    C′-γ-C-γ: C′ · γ = C · γ
    using resolution-lifting[OF ground-resolution[unfolded E_G D_G C_G]
        ϱ₁ ϱ₂
        rename-apart
        E-grounding D-grounding C-grounding
        select-from-E select-from-D
        type-preserving-ϱ₁-γ type-preserving-ϱ₂-γ
        type-preserving-ϱ₁ type-preserving-ϱ₂
        𝒱₁ 𝒱₂]
    unfolding ι_G C_G E_G D_G .

  let ?ι = Infer [(𝒱₂, D), (𝒱₁, E)] (𝒱₃, C′)

  show ?thesis
  proof(rule that[OF - inference-groundings])

    show ?ι ∈ Inf-from N
      using E-in-N D-in-N resolution
      unfolding Inf-from-def inferences-def inference-system.Inf-from-def
      by auto
  qed
qed

lemma ground-instances:
  obtains ι where
    ι ∈ Inf-from N
    ι_G ∈ inference-ground-instances ι
proof −

  consider
    (resolution) ι_G ∈ ground.resolution-inferences |
    (factoring) ι_G ∈ ground.factoring-inferences
    using ι_G-Inf-from
    unfolding
      ground.Inf-from-q-def
      ground.G-Inf-def
      inference-system.Inf-from-def
    by fastforce

  then show ?thesis
  proof cases
    case resolution

    then show ?thesis
      using that resolution-ground-instance
      by blast
  next
    case factoring
```

```
    then show ?thesis
      using that factoring-ground-instance
      by blast
  qed
qed

end

end

context ordered-resolution-calculus
begin

lemma overapproximation:
  obtains select_G where
    ground-Inf-overapproximated select_G premises
    is-grounding select_G
proof−
  obtain select_G where
    subst-stability: select-subst-stability-on select select_G premises and
    is-grounding select_G
    using obtain-subst-stable-on-select-grounding
    by blast

  then interpret grounded-ordered-resolution-calculus
    where select_G = select_G
    by unfold-locales

  show thesis
  proof(rule that[OF - select_G])

    show ground-Inf-overapproximated select_G premises
      using ground-instances[OF subst-stability]
      by auto
  qed
qed

sublocale statically-complete-calculus ⊥_F inferences entails-G Red-I-G Red-F-G
proof (unfold static-empty-ord-inter-equiv-static-inter,
    rule stat-ref-comp-to-non-ground-fam-inter,
    rule ballI)
  fix select_G
  assume select_G ∈ select_Gs

  then interpret grounded-ordered-resolution-calculus
    where select_G = select_G
    by unfold-locales (simp add: select_Gs-def)
```

```
  show statically-complete-calculus
       ground.G-Bot
       ground.G-Inf
       ground.G-entails
       ground.Red-I
       ground.Red-F
    by unfold-locales
next

  show ⋀N. ∃ select_G ∈ select_Gs. ground-Inf-overapproximated select_G N
    using overapproximation
    unfolding select_Gs-def
    by (smt (verit, best) mem-Collect-eq)
qed

end

end
theory Ordered-Resolution-Welltypedness-Preservation
  imports Grounded-Ordered-Resolution
begin

context ordered-resolution-calculus
begin

lemma factoring-preserves-typing:
  assumes factoring: factoring (𝒱, D) (𝒱, C)
  shows clause.is-welltyped 𝒱 D ⟷ clause.is-welltyped 𝒱 C
  using assms
proof (cases (𝒱, D) (𝒱, C) rule: factoring.cases)
  case (factoringI l₁ μ t₁ t₂ l₂ D')

  show ?thesis
  proof (rule iffI)
    assume clause.is-welltyped 𝒱 D
    then show clause.is-welltyped 𝒱 C
      using factoringI
      by simp
  next
    assume C-is-welltyped: clause.is-welltyped 𝒱 C

    note imgu = factoringI(3, 4)

    have clause.is-welltyped 𝒱 (add-mset l₁ D')
      using C-is-welltyped imgu
      unfolding factoringI
      by simp

    moreover have literal.is-welltyped 𝒱 l₂
```

```
    using C-is-welltyped term.imgu-same-type[OF imgu] imgu
    unfolding factoringI
    by force

  ultimately show clause.is-welltyped 𝒱 D
    unfolding factoringI
    by simp
 qed
qed

lemma resolution-preserves-typing:
 assumes
  resolution: resolution (𝒱₂, D) (𝒱₁, E) (𝒱₃, C) and
  D-is-welltyped: clause.is-welltyped 𝒱₂ D and
  E-is-welltyped: clause.is-welltyped 𝒱₁ E
 shows clause.is-welltyped 𝒱₃ C
 using resolution
proof (cases (𝒱₂, D) (𝒱₁, E) (𝒱₃, C) rule: resolution.cases)
 case (resolutionI ϱ₁ ϱ₂ μ t₁ t₂ l₁ l₂ E' D')

 note μ-type-preserving = resolutionI(6)

 have clause.is-welltyped 𝒱₃ (E · ϱ₁)
   using E-is-welltyped clause.welltyped-renaming[OF resolutionI(3, 13)]
   by blast

 then have Eμ-is-welltyped: clause.is-welltyped 𝒱₃ (E · ϱ₁ ⊙ μ)
   using μ-type-preserving
   by simp

 moreover have clause.is-welltyped 𝒱₃ (D · ϱ₂)
   using D-is-welltyped clause.welltyped-renaming[OF resolutionI(4, 14)]
   by blast

 then have Dμ-is-welltyped: clause.is-welltyped 𝒱₃ (D · ϱ₂ ⊙ μ)
   using μ-type-preserving
   by simp

 ultimately show ?thesis
   unfolding resolutionI
   by auto
qed

end

end
theory Untyped-Ordered-Resolution
  imports
   First-Order-Clause.Nonground-Order
```

First-Order-Clause.Nonground-Selection-Function
First-Order-Clause.Tiebreakers

Fresh-Identifiers.Fresh
begin

locale *untyped-ordered-resolution-calculus* =
nonground-order **where**
$less_t = less_t$ **and** *id-subst* = *id-subst* **and** *term-from-ground* = *term-from-ground* :: $'t_G \Rightarrow 't$ **and**
term-vars = *term-vars* +

nonground-selection-function **where**
select = *select* **and** *atom-subst* = $(\cdot t)$ **and** *atom-vars* = *term.vars* **and** *term-vars* = *term-vars* **and**
atom-from-ground = *term.from-ground* **and** *atom-to-ground* = *term.to-ground* **and** *id-subst* = *id-subst* +

tiebreakers tiebreakers +
term: *exists-imgu* **where** *vars* = *term-vars* **and** *subst* = $(\cdot t)$ **and** *id-subst* = *id-subst*
for
select :: $'t$ *select* **and**
$less_t$ :: $'t \Rightarrow 't \Rightarrow$ *bool* **and**
tiebreakers :: $('t_G, 't)$ *tiebreakers* **and**
id-subst :: $'subst$ **and**
term-vars :: $'t \Rightarrow ('v ::$ *infinite*$)$ *set*
begin

inductive *factoring* :: $'t$ *clause* $\Rightarrow$ $'t$ *clause* $\Rightarrow$ *bool* **where**
factoringI:
$D =$ *add-mset* l_1 (*add-mset* l_2 D') $\Longrightarrow$
$l_1 =$ *Pos* $t_1 \Longrightarrow$
$l_2 =$ *Pos* $t_2 \Longrightarrow$
$C = ($*add-mset* l_1 $D'$$) \cdot \mu \Longrightarrow$
factoring D C
if
select $D = \{\#\}$
is-maximal $(l_1 \cdot l \ \mu)$ $(D \cdot \mu)$
term.is-imgu μ $\{\{t_1, t_2\}\}$

inductive *resolution* :: $'t$ *clause* $\Rightarrow$ $'t$ *clause* $\Rightarrow$ $'t$ *clause* $\Rightarrow$ *bool* **where**
resolutionI:
$E =$ *add-mset* l_1 $E' \Longrightarrow$
$D =$ *add-mset* l_2 $D' \Longrightarrow$
$l_1 =$ *Neg* $t_1 \Longrightarrow$
$l_2 =$ *Pos* $t_2 \Longrightarrow$
$C = (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$
resolution D E C

if
term.is-renaming ϱ_1
term.is-renaming ϱ_2
clause.vars $(E \cdot \varrho_1) \cap$ *clause.vars* $(D \cdot \varrho_2) = \{\}$
term.is-imgu μ $\{\{t_1 \cdot t\ \varrho_1,\ t_2 \cdot t\ \varrho_2\}\}$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu)$
select $E = \{\#\} \Longrightarrow$ *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)$
select $E \neq \{\#\} \Longrightarrow$ *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \mu)\ (\textit{select}\ E \cdot \varrho_1 \odot \mu)$
select $D = \{\#\}$
is-strictly-maximal $(l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu)$

abbreviation *factoring-inferences* **where**
factoring-inferences $\equiv \{$ *Infer* $[D]$ $C \mid D\ C.$ *factoring* $D\ C\ \}$

abbreviation *resolution-inferences* **where**
resolution-inferences $\equiv \{$ *Infer* $[D,\ E]$ $C \mid D\ E\ C.$ *resolution* $D\ E\ C\ \}$

definition *inferences* :: *'t clause inference set* **where**
inferences $\equiv$ *resolution-inferences* $\cup$ *factoring-inferences*

abbreviation *bottom* :: *'t clause set* **where**
bottom $\equiv \{\{\#\}\}$

end

end
theory *Untyped-Ordered-Resolution-Inference-System*
imports
Untyped-Ordered-Resolution
First-Order-Clause.Untyped-Calculus
Grounded-Ordered-Resolution
begin

context *untyped-ordered-resolution-calculus*
begin

sublocale *typed*: *ordered-resolution-calculus* **where**
welltyped $= \lambda$- - (). *True*
by
unfold-locales
(*auto intro*: *term.ground-exists simp*: *term.exists-imgu right-unique-def split*: *unit.splits*)

declare
typed.term.welltyped-renaming [*simp del*]
typed.term.welltyped-subst-stability [*simp del*]
typed.term.welltyped-subst-stability' [*simp del*]

abbreviation *entails* **where**
  entails $N\ N' \equiv$ *typed.entails-*$\mathcal{G}$ (*empty-typed* ‘ N) (*empty-typed* ‘ N')

sublocale *untyped-consequence-relation* **where**
  typed-bottom $= \bot_F$ **and** *typed-entails* $=$ *typed.entails-*$\mathcal{G}$ **and**
  bottom $=$ *bottom* **and** *entails* $=$ *entails*
proof *unfold-locales*

  have *bottom* $=$ *snd* ‘ $\bot_F$
    using *image-iff typed.bot-not-empty*
    by *fastforce*

  then show *bottom* $\equiv$ *snd* ‘ $\bot_F$
    by *argo*
qed

sublocale *untyped-inference-system* **where**
  inferences $=$ *inferences* **and** *typed-inferences* $=$ *typed.inferences*
proof *unfold-locales*

  {
    fix $\mathcal{V}\ D\ \mathcal{V}'\ C$
    have *typed.factoring* ($\mathcal{V}$, D) ($\mathcal{V}'$, C) $\longleftrightarrow$ *factoring* $D\ C$
      unfolding $\mathcal{V}$*-all-same*[*of* $\mathcal{V}$] $\mathcal{V}$*-all-same*[*of* $\mathcal{V}'$]
    proof (*rule iffI*)
      assume *typed.factoring* (*empty-typed* D) (*empty-typed* C)

      then show *factoring* $D\ C$
      proof (*cases rule*: *typed.factoring.cases*)
        case *typed-factoringI*: *factoringI*

        then show *?thesis*
          by (*intro factoringI*; (*rule typed-factoringI*) *?*)
      qed
    next
      assume *factoring* $D\ C$

      then show *typed.factoring* (*empty-typed* D) (*empty-typed* C)
      proof (*cases rule*: *factoring.cases*)
        case *factoringI*

        then show *?thesis*
          by (*intro typed.factoringI*) (*auto simp*: *case-unit-Unity*)
      qed
    qed
  }

  then have *remove-types* ‘ *typed.factoring-inferences* $=$ *factoring-inferences*
    by *auto*

```
  moreover {
    fix 𝒱₁ D 𝒱₂ E 𝒱₃ C
    have typed.resolution (𝒱₁, D) (𝒱₂, E) (𝒱₃, C) ⟷ resolution D E C
      unfolding 𝒱-all-same[of 𝒱₁] 𝒱-all-same[of 𝒱₂] 𝒱-all-same[of 𝒱₃]
    proof (rule iffI)
      assume typed.resolution (empty-typed D) (empty-typed E) (empty-typed C)

      then show resolution D E C
      proof (cases rule: typed.resolution.cases)
        case typed-resolutionI: resolutionI

        then show ?thesis
          by (intro resolutionI; (rule typed-resolutionI)?)
      qed
    next
      assume resolution D E C

       then show typed.resolution (empty-typed D) (empty-typed E) (empty-typed
C)
      proof (cases rule: resolution.cases)
        case resolutionI

        show ?thesis
          by
            (intro typed.resolutionI; (rule resolutionI)?)
            (auto simp: case-unit-Unity)
      qed
    qed
  }

  then have remove-types ‘ typed.resolution-inferences = resolution-inferences
    by auto

  ultimately show inferences ≡ remove-types ‘ typed.inferences
    unfolding inferences-def typed.inferences-def image-Un
    by auto
qed

end

end
theory Untyped-Ordered-Resolution-Completeness
  imports
    Untyped-Ordered-Resolution-Inference-System
    Ordered-Resolution-Completeness
begin

context untyped-ordered-resolution-calculus
```

begin

abbreviation *Red-F* **where**
Red-F N ≡ *snd* ‘ *typed.Red-F-*$\mathcal{G}$ (*empty-typed* ‘ *N*)

abbreviation *Red-I* **where**
Red-I N ≡ *remove-types* ‘ *typed.Red-I-*$\mathcal{G}$ (*empty-typed* ‘ *N*)

sublocale *untyped-complete-calculus* **where**
typed-bottom = $\bot_F$ **and** *typed-entails* = *typed.entails-*$\mathcal{G}$ **and**
typed-inferences = *typed.inferences* **and** *typed-Red-I* = *typed.Red-I-*$\mathcal{G}$ **and**
typed-Red-F = *typed.Red-F-*$\mathcal{G}$ **and** *bottom* = *bottom* **and** *inferences* = *inferences*
and *Red-F* = *Red-F* **and**
Red-I = *Red-I* **and** *entails* = *entails*
by *unfold-locales*

end

end
theory *Untyped-Ordered-Resolution-Soundness*
imports
Untyped-Ordered-Resolution-Inference-System
Ordered-Resolution-Soundness
begin

context *untyped-ordered-resolution-calculus*
begin

sublocale *untyped-sound-inference-system* **where**
typed-bottom = $\bot_F$ **and** *typed-entails* = *typed.entails-*$\mathcal{G}$ **and**
typed-inferences = *typed.inferences* **and** *bottom* = *bottom* **and** *inferences* = *inferences* **and**
entails = *entails*
by *unfold-locales*

end

end
theory *Monomorphic-Ordered-Resolution*
imports
Ordered-Resolution

First-Order-Clause.IsaFoR-Nonground-Clause
First-Order-Clause.Monomorphic-Typing
begin

locale *monomorphic-ordered-resolution-calculus* =
monomorphic-term-typing +

ordered-resolution-calculus **where**
comp-subst = ($\circ_s$) **and** *id-subst* = *Var* **and** *term-subst* = ($\cdot$) **and** *term-vars* = *term.vars* **and**
apply-subst = *apply-subst* **and** *subst-update* = *fun-upd* **and** *subst-updates* = *subst-updates* **and**
term-from-ground = *term.from-ground* **and** *term-to-ground* = *term.to-ground* **and**
welltyped = *welltyped*

end
theory *Ordered-Resolution-Example*
imports
Monomorphic-Ordered-Resolution
First-Order-Clause.IsaFoR-KBO
begin

hide-type *Uprod-Literal-Functor.clause*

abbreviation *trivial-tiebreakers* ::
$'f$ *gterm clause* $\Rightarrow$ ($'f$,$'v$) *term clause* $\Rightarrow$ ($'f$,$'v$) *term clause* $\Rightarrow$ *bool* **where**
trivial-tiebreakers $\equiv \bot$

abbreviation *trivial-select* :: $'a$ *clause* $\Rightarrow$ $'a$ *clause* **where**
trivial-select - $\equiv \{\#\}$

abbreviation *unit-typing* **where**
unit-typing - - $\equiv$ *Some* ([], ())

interpretation *unit-types*: *monomorphic-term-typing* **where** $\mathcal{F}$ = *unit-typing*
by *unfold-locales*

interpretation *example1*: *monomorphic-ordered-resolution-calculus* **where**
select = *trivial-select* :: (($'f$ :: *weighted* , $'v$ :: *infinite*) *term*) *select* **and**
$less_t$ = *less-kbo* **and**
$\mathcal{F}$ = *unit-typing* **and**
tiebreakers = *trivial-tiebreakers*
by *unfold-locales* (*auto intro*: *unit-types.exists-witness-if-exists-const-for-all-types*)

instantiation *nat* :: *infinite*
begin

instance
by *intro-classes simp*

end

datatype *type* = *A* | *B*

```
abbreviation types :: nat ⇒ nat ⇒ (type list × type) option where
  types f n ≡
    let type = if even f then A else B
    in Some (replicate n type, type)

interpretation example-types: monomorphic-term-typing where 𝓕 = types
  by unfold-locales


interpretation example2: monomorphic-ordered-resolution-calculus where
    select = trivial-select :: (nat, nat) term select and
    less_t = less-kbo and
    𝓕 = types and
    tiebreakers = trivial-tiebreakers
proof (unfold-locales, rule example-types.exists-witness-if-exists-const-for-all-types)
  fix τ

  show ∃f. types f 0 = Some ([], τ)
  proof (cases τ)
    case A
    show ?thesis
      unfolding A
      by (rule exI[of - 0]) auto
  next
    case B
    show ?thesis
      unfolding B
      by (rule exI[of - 1]) auto
  qed
qed

end
```