# A Modular Formalization of Superposition

Martin Desharnais    Balazs Toth

March 17, 2025

**Abstract**

Superposition is an efficient proof calculus for reasoning about first-order logic with equality 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 relied on the IsaFoR library for first-order terms and on the Isabelle saturation framework. A paper describing this formalization was published at the 15th International Conference on Interactive Theorem Proving (ITP 2024) [1].

## Contents



**theory** *Ground-Critical-Pairs*
**imports** *First-Order-Clause.Term-Rewrite-System*
**begin**

**definition** *ground-critical-pairs* :: *'f gterm rel* ⇒ *'f gterm rel* **where**
*ground-critical-pairs R* = {(*ctxt*⟨$r_2$⟩$_G$, $r_1$) | *ctxt l* $r_1$ $r_2$. (*ctxt*⟨*l*⟩$_G$, $r_1$) ∈ *R* ∧ (*l*, $r_2$) ∈ *R*}

**locale** *ground-critical-pair-theorem* =
**fixes** *f-type* :: *'f itself*
**assumes** *ground-critical-pair-theorem*:
⋀*R* :: *'f gterm rel*.
*WCR* (*rewrite-inside-gctxt R*) ⟷ *ground-critical-pairs R* ⊆ (*rewrite-inside-gctxt R*)$^{\downarrow}$

**end**
**theory** *Ground-Superposition*
**imports**
*Ground-Critical-Pairs*
*First-Order-Clause.Selection-Function*
*First-Order-Clause.Ground-Order*
*First-Order-Clause.Ground-Clause*
**begin**

# 1 Superposition Calculus

**locale** *ground-superposition-calculus* =
*ground-order-with-equality* **where** $less_t$ = $less_t$ +
*selection-function select* +
*ground-critical-pair-theorem TYPE*(*'f*)
**for**
$less_t$ :: *'f gterm* ⇒ *'f gterm* ⇒ *bool* **and**
*select* :: *'f gatom clause* ⇒ *'f gatom clause*
**begin**

## 1.1 Ground Rules

**inductive** *superposition* ::
*'f gatom clause* ⇒ *'f gatom clause* ⇒ *'f gatom clause* ⇒ *bool*
**where**
*superpositionI*:
$E$ = *add-mset* $L_E$ $E'$ ⟹
$D$ = *add-mset* $L_D$ $D'$ ⟹
$D \prec_c E$ ⟹
$\mathcal{P} \in \{Pos, Neg\}$ ⟹
$L_E = \mathcal{P}$ (*Upair* $\kappa\langle t\rangle_G$ $u$) ⟹
$L_D = t \approx t'$ ⟹
$u \prec_t \kappa\langle t\rangle_G$ ⟹
$t' \prec_t t$ ⟹
($\mathcal{P}$ = *Pos* ∧ *select E* = {#} ∧ *is-strictly-maximal* $L_E$ $E$) ∨
($\mathcal{P}$ = *Neg* ∧ (*select E* = {#} ∧ *is-maximal* $L_E$ $E$ ∨ *is-maximal* $L_E$ (*select E*)))
⟹
*select D* = {#} ⟹

$is\text{-}strictly\text{-}maximal\ L_D\ D \Longrightarrow$
$C = add\text{-}mset\ (\mathcal{P}\ (Upair\ \kappa\langle t'\rangle_G\ u))\ (E' + D') \Longrightarrow$
$superposition\ D\ E\ C$

**inductive** *eq-resolution* :: *'f gatom clause* ⇒ *'f gatom clause* ⇒ *bool* **where**
*eq-resolutionI*:
$D = add\text{-}mset\ L\ D' \Longrightarrow$
$L = t\ !\!\approx t \Longrightarrow$
$select\ D = \{\#\} \wedge is\text{-}maximal\ L\ D \vee is\text{-}maximal\ L\ (select\ D) \Longrightarrow$
$C = D' \Longrightarrow$
$eq\text{-}resolution\ D\ C$

**inductive** *eq-factoring* :: *'f gatom clause* ⇒ *'f gatom clause* ⇒ *bool* **where**
*eq-factoringI*:
$D = add\text{-}mset\ L_1\ (add\text{-}mset\ L_2\ D') \Longrightarrow$
$L_1 = t \approx t' \Longrightarrow$
$L_2 = t \approx t'' \Longrightarrow$
$select\ D = \{\#\} \Longrightarrow$
$is\text{-}maximal\ L_1\ D \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$C = add\text{-}mset\ (t'\ !\!\approx t'')\ (add\text{-}mset\ (t \approx t'')\ D') \Longrightarrow$
$eq\text{-}factoring\ D\ C$

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

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

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

### 1.1.1 Alternative Specification of the Superposition Rule

**inductive** *superposition'* ::
*'f gatom clause* ⇒ *'f gatom clause* ⇒ *'f gatom clause* ⇒ *bool*
**where**
*superposition'I*:
$P_1 = add\text{-}mset\ L_1\ P_1' \Longrightarrow$
$P_2 = add\text{-}mset\ L_2\ P_2' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$\mathcal{P} \in \{Pos, Neg\} \Longrightarrow$
$L_1 = \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s') \Longrightarrow$
$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$(\mathcal{P} = Pos \longrightarrow select\ P_1 = \{\#\} \wedge is\text{-}strictly\text{-}maximal\ L_1\ P_1) \Longrightarrow$
$(\mathcal{P} = Neg \longrightarrow (select\ P_1 = \{\#\} \wedge is\text{-}maximal\ L_1\ P_1 \vee is\text{-}maximal\ L_1\ (select\ P_1))) \Longrightarrow$

$select\ P_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\ L_2\ P_2 \Longrightarrow$
$C = add\text{-}mset\ (\mathcal{P}\ (Upair\ s\langle t'\rangle_G\ s'))\ (P_1{}' + P_2{}') \Longrightarrow$
$superposition'\ P_2\ P_1\ C$

**lemma** *superposition′ = superposition*
**proof** (*intro ext iffI*)
**fix** *P1 P2 C*
**assume** *superposition′ P2 P1 C*
**thus** *superposition P2 P1 C*
**proof** (*cases P2 P1 C rule*: *superposition′.cases*)
**case** (*superposition′I* $L_1\ P_1{}'\ L_2\ P_2{}'\ \mathcal{P}\ s\ t\ s'\ t'$)
**thus** *?thesis*
**using** *superpositionI*
**by** *force*
**qed**
**next**
**fix** *P1 P2 C*
**assume** *superposition P1 P2 C*
**thus** *superposition′ P1 P2 C*
**proof** (*cases P1 P2 C rule*: *superposition.cases*)
**case** (*superpositionI* $L_1\ P_1{}'\ L_2\ P_2{}'\ \mathcal{P}\ s\ t\ s'\ t'$)
**thus** *?thesis*
**using** *superposition′I*
**by** (*metis literals-distinct*(*2*))
**qed**
**qed**

**inductive** *pos-superposition* ::
*′f gatom clause ⇒ ′f gatom clause ⇒ ′f gatom clause ⇒ bool*
**where**
*pos-superpositionI*:
$P_1 = add\text{-}mset\ L_1\ P_1{}' \Longrightarrow$
$P_2 = add\text{-}mset\ L_2\ P_2{}' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$L_1 = s\langle t\rangle_G \approx s' \Longrightarrow$
$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$select\ P_1 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\ L_1\ P_1 \Longrightarrow$
$select\ P_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\ L_2\ P_2 \Longrightarrow$
$C = add\text{-}mset\ (s\langle t'\rangle_G \approx s')\ (P_1{}' + P_2{}') \Longrightarrow$
$pos\text{-}superposition\ P_2\ P_1\ C$

**lemma** *superposition-if-pos-superposition*:
**assumes** *step*: $pos\text{-}superposition\ P_2\ P_1\ C$
**shows** $superposition\ P_2\ P_1\ C$

**using** *step*
**proof** (*cases* $P_2$ $P_1$ $C$ *rule*: *pos-superposition.cases*)
**case** (*pos-superpositionI* $L_1$ $P_1{}'$ $L_2$ $P_2{}'$ $s$ $t$ $s'$ $t'$)
**thus** *?thesis*
**using** *superpositionI*
**by** (*metis insert-iff*)
**qed**

**inductive** *neg-superposition* ::
$'f$ *gatom clause* $\Rightarrow$ $'f$ *gatom clause* $\Rightarrow$ $'f$ *gatom clause* $\Rightarrow$ *bool*
**where**
*neg-superpositionI*:
$P_1 = \textit{add-mset}\ L_1\ P_1{}' \Longrightarrow$
$P_2 = \textit{add-mset}\ L_2\ P_2{}' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$L_1 = s\langle t\rangle_G\ !\!\approx s' \Longrightarrow$
$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$\textit{select}\ P_1 = \{\#\} \wedge \textit{is-maximal}\ L_1\ P_1 \vee \textit{is-maximal}\ L_1\ (\textit{select}\ P_1) \Longrightarrow$
$\textit{select}\ P_2 = \{\#\} \Longrightarrow$
$\textit{is-strictly-maximal}\ L_2\ P_2 \Longrightarrow$
$C = \textit{add-mset}\ (\textit{Neg}\ (\textit{Upair}\ s\langle t'\rangle_G\ s'))\ (P_1{}' + P_2{}') \Longrightarrow$
*neg-superposition* $P_2$ $P_1$ $C$

**lemma** *superposition-if-neg-superposition*:
**assumes** *neg-superposition* $P_2$ $P_1$ $C$
**shows** *superposition* $P_2$ $P_1$ $C$
**using** *assms*
**proof** (*cases* $P_2$ $P_1$ $C$ *rule*: *neg-superposition.cases*)
**case** (*neg-superpositionI* $L_1$ $P_1{}'$ $L_2$ $P_2{}'$ $s$ $t$ $s'$ $t'$)
**then show** *?thesis*
**using** *superpositionI*
**by** (*metis insert-iff*)
**qed**

**lemma** *superposition-iff-pos-or-neg*:
*superposition* $P_2$ $P_1$ $C$ $\longleftrightarrow$
*pos-superposition* $P_2$ $P_1$ $C$ $\vee$ *neg-superposition* $P_2$ $P_1$ $C$
**proof** (*rule iffI*)
**assume** *superposition* $P_2$ $P_1$ $C$
**thus** *pos-superposition* $P_2$ $P_1$ $C$ $\vee$ *neg-superposition* $P_2$ $P_1$ $C$
**proof** (*cases* $P_2$ $P_1$ $C$ *rule*: *superposition.cases*)
**case** (*superpositionI* $L_1$ $P_1{}'$ $L_2$ $P_2{}'$ $\mathcal{P}$ $s$ $t$ $s'$ $t'$)
**then show** *?thesis*
**using** *pos-superpositionI*[*of* $P_1$ $L_1$ $P_1{}'$ $P_2$ $L_2$ $P_2{}'$ $s$ $t$ $s'$ $t'$]
**using** *neg-superpositionI*[*of* $P_1$ $L_1$ $P_1{}'$ $P_2$ $L_2$ $P_2{}'$ $s$ $t$ $s'$ $t'$]
**by** *metis*
**qed**

**next**
  **assume** *pos-superposition* $P_2$ $P_1$ $C$ ∨ *neg-superposition* $P_2$ $P_1$ $C$
  **thus** *superposition* $P_2$ $P_1$ $C$
    **using**
      *superposition-if-neg-superposition*
      *superposition-if-pos-superposition*
    **by** *metis*
**qed**

## 1.2 Ground Layer

**definition** *G-Inf* :: *′f gatom clause inference set* **where**
  *G-Inf* =
    {*Infer* [$P_2$, $P_1$] $C$ | $P_2$ $P_1$ $C$. *superposition* $P_2$ $P_1$ $C$} ∪
    {*Infer* [$P$] $C$ | $P$ $C$. *eq-resolution* $P$ $C$} ∪
    {*Infer* [$P$] $C$ | $P$ $C$. *eq-factoring* $P$ $C$}

**abbreviation** *G-Bot* :: *′f gatom clause set* **where**
  *G-Bot* ≡ {{#}}

**definition** *G-entails* :: *′f gatom clause set* ⇒ *′f gatom clause set* ⇒ *bool* **where**
  *G-entails* $N_1$ $N_2$ ⟷ (∀ (*I* :: *′f gterm rel*). *refl* $I$ ⟶ *trans* $I$ ⟶ *sym* $I$ ⟶
    *compatible-with-gctxt* $I$ ⟶ *upair* ‘ $I$ ⊫*s* $N_1$ ⟶ *upair* ‘ $I$ ⊫*s* $N_2$)

**lemma** *superposition-smaller-conclusion*:
  **assumes**
    *step*: *superposition P1 P2 C*
  **shows** $C \prec_c$ *P2*
  **using** *step*
**proof** (*cases P1 P2 C rule*: *superposition.cases*)
  **case** (*superpositionI* $L_1$ $P_1'$ $L_2$ $P_2'$ $\mathcal{P}$ $s$ $t$ $s'$ $t'$)

   **have** $P_1'$ + *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ $s'$)) $P_2'$ $\prec_c$ $P_1'$ + {#$\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$)#}
    **unfolding** *less*$_c$*-def*
  **proof** (*intro one-step-implies-multp ballI*)
    **fix** *K* **assume** $K \in\#$ *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ $s'$)) $P_2'$

    **moreover have** $\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ $s'$) $\prec_l$ $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$)
    **proof** −
      **have** $s\langle t'\rangle_G \prec_t s\langle t\rangle_G$
        **using** ‹$t' \prec_t t$›
        **by** *simp*

      **hence** *multp* ($\prec_t$) {#$s\langle t'\rangle_G$, $s'$#} {#$s\langle t\rangle_G$, $s'$#}
        **by** (*simp add*: *add-mset-commute multp-cancel-add-mset*)

      **have** *?thesis* **if** $\mathcal{P}$ = *Pos*
        **unfolding** *that less*$_l$*-def*

```
    using ‹multp (≺_t) {#s⟨t'⟩_G, s'#} {#s⟨t⟩_G, s'#}›
    by simp

  moreover have ?thesis if 𝒫 = Neg
    unfolding that less_l-def
    using ‹multp (≺_t) {#s⟨t'⟩_G, s'#} {#s⟨t⟩_G, s'#}› multp-double-doubleI
    by force

  ultimately show ?thesis
    using ‹𝒫 ∈ {Pos, Neg}›
    by auto
qed

moreover have ∀ K ∈# P_2'. K ≺_l 𝒫 (Upair s⟨t⟩_G s')
proof −
  have is-strictly-maximal L_2 P1
    using superpositionI
    by argo

  hence ∀ K ∈# P_2'. ¬ Pos (Upair t t') ≺_l K ∧ Pos (Upair t t') ≠ K
    unfolding
      is-strictly-maximal-def
      ‹P1 = add-mset L_2 P_2'› ‹L_2 = t ≈ t'›
    by simp

  hence ∀ K ∈# P_2'. K ≺_l Pos (Upair t t')
    by auto

  have thesis-if-Neg: Pos (Upair t t') ≺_l 𝒫 (Upair s⟨t⟩_G s')
    if 𝒫 = Neg
  proof −
    have t ⪯_t s⟨t⟩_G
      using term.order.less-eq-subterm-property .

    hence multp (≺_t) {#t, t'#} {#s⟨t⟩_G, s', s⟨t⟩_G, s'#}
      unfolding reflclp-iff
    proof (elim disjE)
      assume t ≺_t s⟨t⟩_G

      moreover hence t' ≺_t s⟨t⟩_G
        using superpositionI(8)
        by order

      ultimately show ?thesis
        by (auto intro: one-step-implies-multp[of - - - {#}, simplified])
    next
      assume t = s⟨t⟩_G
      thus ?thesis
        using ‹t' ≺_t t›
```

**by** *simp*
**qed**
**thus** $Pos\ (Upair\ t\ t') \prec_l \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s')$
**using** ‹$\mathcal{P} = Neg$›
**by** (*simp add*: $less_l$*-def*)
**qed**

**have** *thesis-if-Pos*: $Pos\ (Upair\ t\ t') \preceq_l \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s')$
**if** $\mathcal{P} = Pos$ **and** *is-maximal* $L_1$ *P2*
**proof** (*cases s*)
**case** *Hole*
**show** *?thesis*
**proof** (*cases* $t' \preceq_t s'$)
**case** *True*

**hence** $(multp\ (\prec_t))^{==}\ \{\#t,\ t'\#\}\ \{\#s\langle t\rangle_G,\ s'\#\}$
**unfolding** *Hole*
**by** (*simp add*: *multp-cancel-add-mset*)

**thus** *?thesis*
**unfolding** *Hole* ‹$\mathcal{P} = Pos$›
**by** (*auto simp*: $less_l$*-def*)
**next**
**case** *False*
**hence** $s' \prec_t t'$
**by** *order*

**hence** $multp\ (\prec_t)\ \{\#s\langle t\rangle_G,\ s'\#\}\ \{\#t,\ t'\#\}$
**by** (*simp add*: *Hole multp-cancel-add-mset*)

**hence** $\mathcal{P}\ (Upair\ s\langle t\rangle_G\ s') \prec_l Pos\ (Upair\ t\ t')$
**using** ‹$\mathcal{P} = Pos$›
**by** (*simp add*: $less_l$*-def*)

**moreover have** $\forall\, K \in\#\ P_1{}'.\ K \preceq_l \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s')$
**using** *that*
**unfolding** *superpositionI is-maximal-def*
**by** *auto*

**ultimately have** $\forall\, K \in\#\ P_1{}'.\ K \preceq_l Pos\ (Upair\ t\ t')$
**by** *auto*

**hence** $P2 \prec_c P1$
**using**
‹$\mathcal{P}\ (Upair\ s\langle t\rangle_G\ s') \prec_l Pos\ (Upair\ t\ t')$›
*one-step-implies-multp*[*of P1 P2* $(\prec_l)$ $\{\#\}$, *simplified*]
*literal.order.multp-if-maximal-less-that-maximal*
*superpositionI*
*that*

```
      unfolding lessc-def
      by blast

    hence False
      using ‹P1 ≺c P2› by order

    thus ?thesis ..
  qed
next
  case (More f ts1 ctxt ts2)
  hence t ≺t s⟨t⟩G
    using term.order.subterm-property[of s t]
    by simp

  moreover hence t′ ≺t s⟨t⟩G
    using ‹t′ ≺t t›
    by order

  ultimately have multp (≺t) {#t, t′#} {#s⟨t⟩G, s′#}
    using one-step-implies-multp[of {#s⟨t⟩G, s′#} {#t, t′#} (≺t) {#}]
    by simp

  hence Pos (Upair t t′) ≺l 𝒫 (Upair s⟨t⟩G s′)
    using ‹𝒫 = Pos›
    by (simp add: lessl-def)

  thus ?thesis
    by order
qed

have 𝒫 = Pos ∨ 𝒫 = Neg
  using ‹𝒫 ∈ {Pos, Neg}›
  by simp

thus ?thesis
proof (elim disjE; intro ballI)
  fix K assume 𝒫 = Pos K ∈# P2′
  have K ≺l t ≈ t′
    using ‹∀ K∈#P2′. K ≺l t ≈ t′› ‹K ∈# P2′›
    by metis

  also have t ≈ t′ ⪯l 𝒫 (Upair s⟨t⟩G s′)
  proof (rule thesis-if-Pos[OF ‹𝒫 = Pos›])

    have is-strictly-maximal L1 P2
      using ‹𝒫 = Pos› superpositionI
      by simp

    thus is-maximal L1 P2
```

**by** *blast*
**qed**

**finally show** $K \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$) **.**
**next**
**fix** $K$ **assume** $\mathcal{P} = Neg\ K \in\# P_2{}'$

**have** $K \prec_l t \approx t'$
**using** ‹$\forall K{\in}\#P_2{}'.\ K \prec_l t \approx t'$› ‹$K \in\# P_2{}'$›
**by** *metis*

**also have** $t \approx t' \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$)
**using** *thesis-if-Neg*[*OF* ‹$\mathcal{P} = Neg$›] **.**

**finally show** $K \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$) **.**
**qed**
**qed**

**ultimately show** $\exists j \in\# \{\#\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$)$\#\}.\ K \prec_l j$
**by** *auto*
**qed** *simp*

**moreover have** $C$ = *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ $s'$)) ($P_1{}' + P_2{}'$)
**unfolding** *superpositionI* **..**

**moreover have** *P2* = $P_1{}' + \{\#\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ $s'$)$\#\}$
**unfolding** *superpositionI* **by** *simp*

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

**lemma** *ground-eq-resolution-smaller-conclusion*:
**assumes** *step*: *eq-resolution P C*
**shows** $C \prec_c P$
**using** *step*
**proof** (*cases P C rule*: *eq-resolution.cases*)
**case** (*eq-resolutionI L t*)
**then show** *?thesis*
**unfolding** $less_c$*-def*
**by** (*metis add.left-neutral add-mset-add-single empty-not-add-mset multi-member-split one-step-implies-multp union-commute*)
**qed**

**lemma** *ground-eq-factoring-smaller-conclusion*:
**assumes** *step*: *eq-factoring P C*
**shows** $C \prec_c P$
**using** *step*
**proof** (*cases P C rule*: *eq-factoring.cases*)

**case** (*eq-factoringI* $L_1$ $L_2$ $P'$ $t$ $t'$ $t''$)
**have** *is-maximal* $L_1$ $P$
**using** *eq-factoringI*
**by** *simp*

**hence** $\forall K \in\# $ *add-mset* (*Pos* (*Upair* $t$ $t''$)) $P'$. $\neg$ *Pos* (*Upair* $t$ $t'$) $\prec_l$ $K$
**unfolding** *eq-factoringI is-maximal-def*
**by** *auto*

**hence** $\neg$ *Pos* (*Upair* $t$ $t'$) $\prec_l$ *Pos* (*Upair* $t$ $t''$)
**by** *simp*

**hence** *Pos* (*Upair* $t$ $t''$) $\preceq_l$ *Pos* (*Upair* $t$ $t'$)
**by** *order*

**hence** $t'' \preceq_t t'$
**unfolding** *reflclp-iff*
**by** (*auto simp*: *less*$_l$*-def multp-cancel-add-mset*)

**have** $C$ = *add-mset* (*Neg* (*Upair* $t'$ $t''$)) (*add-mset* (*Pos* (*Upair* $t$ $t''$)) $P'$)
**using** *eq-factoringI*
**by** *argo*

**moreover have** *add-mset* (*Neg* (*Upair* $t'$ $t''$)) (*add-mset* (*Pos* (*Upair* $t$ $t''$)) $P'$)
$\prec_c$ $P$
**unfolding** *eq-factoringI less*$_c$*-def*
**proof** (*intro one-step-implies-multp*[*of* {#-#} {#-#}, *simplified*])
**have** $t'' \prec_t t$
**using** ‹$t' \prec_t t$› ‹$t'' \preceq_t t'$›
**by** *order*

**hence** *multp* ($\prec_t$) {#$t'$, $t''$, $t'$, $t''$#} {#$t$, $t'$#}
**using** *one-step-implies-multp*[*of* - - - {#}, *simplified*]
**by** (*metis* ‹$t' \prec_t t$› *diff-empty id-remove-1-mset-iff-notin insert-iff set-mset-add-mset-insert*)

**thus** *Neg* (*Upair* $t'$ $t''$) $\prec_l$ *Pos* (*Upair* $t$ $t'$)
**by** (*simp add*: *less*$_l$*-def*)
**qed**

**ultimately show** *?thesis*
**by** *argo*
**qed**

**sublocale** *consequence-relation* **where** *Bot* = *G-Bot* **and** *entails* = *G-entails*
**proof** *unfold-locales*
**show** *G-Bot* $\neq$ {}
**by** *simp*
**next**

**show** $\bigwedge B\ N.\ B \in \textit{G-Bot} \Longrightarrow \textit{G-entails}\ \{B\}\ N$
**by** (*simp add*: *G-entails-def*)
**next**
**show** $\bigwedge N2\ N1.\ N2 \subseteq N1 \Longrightarrow \textit{G-entails}\ N1\ N2$
**by** (*auto simp*: *G-entails-def elim*!: *true-clss-mono*[*rotated*])
**next**
**fix** *N1 N2* **assume** *ball-G-entails*: $\forall C \in N2.\ \textit{G-entails}\ N1\ \{C\}$
**show** *G-entails N1 N2*
**unfolding** *G-entails-def*
**proof** (*intro allI impI*)
**fix** *I* :: *'f gterm rel*
**assume** *refl I* **and** *trans I* **and** *sym I* **and** *compatible-with-gctxt I* **and**
$(\lambda(x,\ y).\ \textit{Upair}\ x\ y)$ ' *I* $\models$*s N1*

**hence** $\forall C \in N2.\ (\lambda(x,\ y).\ \textit{Upair}\ x\ y)$ ' *I* $\models$*s* $\{C\}$
**using** *ball-G-entails*
**by** (*simp add*: *G-entails-def*)

**then show** $(\lambda(x,\ y).\ \textit{Upair}\ x\ y)$ ' *I* $\models$*s N2*
**by** (*simp add*: *true-clss-def*)
**qed**
**next**
**show** $\bigwedge N1\ N2\ N3.\ \textit{G-entails}\ N1\ N2 \Longrightarrow \textit{G-entails}\ N2\ N3 \Longrightarrow \textit{G-entails}\ N1\ N3$
**using** *G-entails-def*
**by** *simp*
**qed**

## 1.3 Redundancy Criterion

**sublocale** *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** $\bigwedge\iota.\ \iota \in \textit{G-Inf} \Longrightarrow \textit{prems-of}\ \iota \neq []$
**by** (*auto simp*: *G-Inf-def*)
**next**
**fix** $\iota$
**have** *concl-of* $\iota \prec_c$ *main-prem-of* $\iota$
**if** $\iota$*-def*: $\iota = \textit{Infer}\ [P_2,\ P_1]\ C$ **and**
*infer*: *superposition* $P_2\ P_1\ C$

**for** $P_2$ $P_1$ $C$
**unfolding** *ι-def*
**using** *infer*
**using** *superposition-smaller-conclusion*
**by** *simp*

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

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

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

**lemma** *redundant-infer-singleton*:
**assumes** $\exists D \in N$. *G-entails* (*insert* $D$ (*set* (*side-prems-of* $\iota$))) {*concl-of* $\iota$} $\land$ $D$ $\prec_c$ *main-prem-of* $\iota$
**shows** *redundant-infer* $N$ $\iota$
**proof**−
**obtain** $D$ **where** $D$:
$D \in N$
*G-entails* (*insert* $D$ (*set* (*side-prems-of* $\iota$))) {*concl-of* $\iota$}
$D \prec_c$ *main-prem-of* $\iota$
**using** *assms*
**by** *blast*

**show** *?thesis*
**unfolding** *redundant-infer-def*
**by** (*rule exI*[*of* - {$D$}]) (*auto simp*: $D$)
**qed**

**end**

**end**

```
theory Abstract-Rewriting-Extra
  imports
    First-Order-Clause.Transitive-Closure-Extra
    Abstract−Rewriting.Abstract-Rewriting
begin

lemma mem-join-union-iff-mem-join-lhs:
  assumes
    ⋀z. (x, z) ∈ A* ⟹ z ∉ Domain B and
    ⋀z. (y, z) ∈ A* ⟹ z ∉ Domain B
  shows (x, y) ∈ (A ∪ B)↓ ⟷ (x, y) ∈ A↓
proof (rule iffI)
  assume (x, y) ∈ (A ∪ B)↓
  then obtain z where
    (x, z) ∈ (A ∪ B)* and (y, z) ∈ (A ∪ B)*
    by auto

  show (x, y) ∈ A↓
  proof (rule joinI)
    from assms(1) show (x, z) ∈ A*
      using ‹(x, z) ∈ (A ∪ B)*› mem-rtrancl-union-iff-mem-rtrancl-lhs[of x A B z]
by simp
  next
    from assms(2) show (y, z) ∈ A*
      using ‹(y, z) ∈ (A ∪ B)*› mem-rtrancl-union-iff-mem-rtrancl-lhs[of y A B z]
by simp
  qed
next
  show (x, y) ∈ A↓ ⟹ (x, y) ∈ (A ∪ B)↓
    by (metis UnCI join-mono subset-Un-eq sup.left-idem)
qed

lemma mem-join-union-iff-mem-join-rhs:
  assumes
    ⋀z. (x, z) ∈ B* ⟹ z ∉ Domain A and
    ⋀z. (y, z) ∈ B* ⟹ z ∉ Domain A
  shows (x, y) ∈ (A ∪ B)↓ ⟷ (x, y) ∈ B↓
  using mem-join-union-iff-mem-join-lhs
  by (metis assms(1) assms(2) sup-commute)

lemma refl-join: refl (r↓)
  by (simp add: joinI-right reflI)

lemma trans-join:
  assumes strongly-norm: SN r and confluent: WCR r
  shows trans (r↓)
proof −
  from confluent strongly-norm have CR r
    using Newman by metis
```

**hence** $r^{\leftrightarrow *} = r^{\downarrow}$
**using** *CR-imp-conversionIff-join* **by** *metis*
**thus** *?thesis*
**using** *conversion-trans* **by** *metis*
**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-Superposition-Completeness*
**imports**
*Ground-Superposition*

*First-Order-Clause.HOL-Extra*
*Abstract-Rewriting-Extra*
*Relation-Extra*
**begin**

## 1.4 Model Construction

**context** *ground-superposition-calculus* **begin**

**function** *epsilon* :: - $\Rightarrow$ *'f gatom clause* $\Rightarrow$ *'f gterm rel* **where**
*epsilon N C* = $\{(s, t)|\ s\ t\ C'.$
$C \in N\ \wedge$
$C =$ *add-mset* (*Pos* (*Upair s t*)) $C'\ \wedge$
*select* $C = \{\#\}\ \wedge$
*is-strictly-maximal* (*Pos* (*Upair s t*)) $C\ \wedge$
$t \prec_t s\ \wedge$

(*let* $R_C$ = ($\bigcup D \in \{D \in N.\ D \prec_c C\}$. *epsilon* $\{E \in N.\ E \preceq_c D\}$ $D$) *in*
¬ *upair* ‘ (*rewrite-inside-gctxt* $R_C$)$^\downarrow$ ⊫ $C$ ∧
¬ *upair* ‘ (*rewrite-inside-gctxt* (*insert* ($s$, $t$) $R_C$))$^\downarrow$ ⊫ $C'$ ∧
$s \in$ *NF* (*rewrite-inside-gctxt* $R_C$))}
**by** *auto*

**termination** *epsilon*
**proof** (*relation* $\{((x1, x2), (y1, y2)).\ x2 \prec_c y2\}$)
**define** $f$ :: $'c \times 'f$ *gterm uprod literal multiset* ⇒ $'f$ *gterm uprod literal multiset*
**where**
$f = (\lambda(x1, x2).\ x2)$
**have** *wfp* $(\lambda(x1, x2)\ (y1, y2).\ x2 \prec_c y2)$
**proof** (*rule wfp-if-convertible-to-wfp*)
**show** $\bigwedge x\ y.$ (*case* $x$ *of* $(x1, x2) \Rightarrow \lambda(y1, y2).\ x2 \prec_c y2)\ y \Longrightarrow$ (*snd* $x$) $\prec_c$ (*snd* $y$)
**by** *auto*
**next**
**show** *wfp* $(\prec_c)$
**by** *auto*
**qed**
**thus** *wf* $\{((x1, x2), (y1, y2)).\ x2 \prec_c y2\}$
**by** (*simp add*: *wfp-def*)
**next**
**show** $\bigwedge N\ C\ x\ xa\ xb\ xc\ xd.\ xd \in \{D \in N.\ D \prec_c C\} \Longrightarrow ((\{E \in N.\ E \preceq_c xd\}, xd), N, C) \in \{((x1, x2), y1, y2).\ x2 \prec_c y2\}$
**by** *simp*
**qed**

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

**lemma** *epsilon-filter-le-conv*: *epsilon* $\{D \in N.\ D \preceq_c C\}$ $C$ = *epsilon* $N$ $C$
**proof** (*intro subset-antisym subrelI*)
**fix** $x$ $y$
**assume** $(x, y) \in$ *epsilon* $\{D \in N.\ D \preceq_c C\}$ $C$
**then obtain** $C'$ **where**
$C \in N$ **and**
$C$ = *add-mset* $(x \approx y)$ $C'$ **and**
*select* $C = \{\#\}$ **and**
*is-strictly-maximal* $(x \approx y)$ $C$ **and**
$y \prec_t x$ **and**
(*let* $R_C = \bigcup x{\in}\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\}$. *epsilon* $\{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}$ $x$ *in*
¬ *upair* ‘ (*rewrite-inside-gctxt* $R_C$)$^\downarrow$ ⊫ $C$ ∧
¬ *upair* ‘ (*rewrite-inside-gctxt* (*insert* $(x, y)$ $R_C$))$^\downarrow$ ⊫ $C'$ ∧
$x \in$ *NF* (*rewrite-inside-gctxt* $R_C$))
**unfolding** *epsilon.simps*[*of - C*] *mem-Collect-eq*
**by** *auto*

**moreover have** $(\bigcup x{\in}\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\}$. *epsilon* $\{E \in$

$N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x) = (\bigcup D{\in}\{D \in N.\ D \prec_c C\}.\ epsilon\ \{E \in N.\ E \preceq_c D\}\ D)$

**proof** (*rule SUP-cong*)

**show** $\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\} = \{D \in N.\ D \prec_c C\}$

**by** *metis*

**next**

**show** $\bigwedge x.\ x \in \{D \in N.\ D \prec_c C\} \Longrightarrow epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x = epsilon\ \{E \in N.\ E \preceq_c x\}\ x$

**by** (*metis* (*no-types*, *lifting*) *clause.order.dual-order.strict-trans2 mem-Collect-eq*)

**qed**

**ultimately show** $(x, y) \in epsilon\ N\ C$

**unfolding** *epsilon.simps*[*of* - *C*] **by** *simp*

**next**

**fix** *x y*

**assume** $(x, y) \in epsilon\ N\ C$

**then obtain** $C'$ **where**

$C \in N$ **and**

$C = add\text{-}mset\ (x \approx y)\ C'$ **and**

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

$is\text{-}strictly\text{-}maximal\ (x \approx y)\ C$ **and**

$y \prec_t x$ **and**

$(let\ R_C = \bigcup x{\in}\{D \in N.\ D \prec_c C\}.\ epsilon\ \{E \in N.\ E \preceq_c x\}\ x\ in$

$\neg\ upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ R_C)^{\downarrow} \models C \land$

$\neg\ upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ (insert\ (x, y)\ R_C))^{\downarrow} \models C' \land$

$x \in NF\ (rewrite\text{-}inside\text{-}gctxt\ R_C))$

**unfolding** *epsilon.simps*[*of* - *C*] *mem-Collect-eq*

**by** *auto*

**moreover have** $(\bigcup x{\in}\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\}.\ epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x) = (\bigcup D{\in}\{D \in N.\ D \prec_c C\}.\ epsilon\ \{E \in N.\ E \preceq_c D\}\ D)$

**proof** (*rule SUP-cong*)

**show** $\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\} = \{D \in N.\ D \prec_c C\}$

**by** *metis*

**next**

**show** $\bigwedge x.\ x \in \{D \in N.\ D \prec_c C\} \Longrightarrow epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x = epsilon\ \{E \in N.\ E \preceq_c x\}\ x$

**by** (*metis* (*mono-tags*, *lifting*) *clause.order.dual-order.strict-trans2 mem-Collect-eq*)

**qed**

**ultimately show** $(x, y) \in epsilon\ \{D \in N.\ (\prec_c)^{==}\ D\ C\}\ C$

**unfolding** *epsilon.simps*[*of* - *C*] **by** *simp*

**qed**

**end**

**lemma** (**in** *ground-superposition-calculus*) *epsilon-eq-empty-or-singleton*:

*epsilon N C* = {} ∨ (∃ *s t*. *epsilon N C* = {(*s*, *t*)})
**proof** −
**have** $\exists_{\leq 1}$ (*x*, *y*). ∃ *C′*.
*C* = *add-mset* (*Pos* (*Upair x y*)) *C′* ∧ *is-strictly-maximal* (*Pos* (*Upair x y*)) *C* ∧ *y* $\prec_t$ *x*
**by** (*rule Uniq-prodI*)
(*metis Upair-inject add-mset-remove-trivial insert-iff is-strictly-maximal-def literal.inject*(*1*) *literal.order.nle-le set-mset-add-mset-insert term.order.dual-order.asym*)

**hence** *Uniq-epsilon*: $\exists_{\leq 1}$ (*x*, *y*). ∃ *C′*.
*C* ∈ *N* ∧
*C* = *add-mset* (*Pos* (*Upair x y*)) *C′* ∧ *select C* = {#} ∧
*is-strictly-maximal* (*Pos* (*Upair x y*)) *C* ∧ *y* $\prec_t$ *x* ∧
(*let* $R_C$ = ⋃ *D* ∈ {*D* ∈ *N*. *D* $\prec_c$ *C*}. *epsilon* {*E* ∈ *N*. *E* $\preceq_c$ *D*} *D in*
¬ *upair* ‘ (*rewrite-inside-gctxt* $R_C$)$^\downarrow$ ⊫ *C* ∧
¬ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*x*, *y*) $R_C$))$^\downarrow$ ⊫ *C′* ∧
*x* ∈ *NF* (*rewrite-inside-gctxt* $R_C$))
**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-prod*[*OF Uniq-epsilon*]
**by** (*smt* (*verit*, *best*) *Collect-cong Collect-empty-eq Uniq-def Uniq-epsilon case-prod-conv insertCI mem-Collect-eq*)
**qed**

**lemma** (**in** *ground-superposition-calculus*) *card-epsilon-le-one*:
*card* (*epsilon N C*) ≤ *1*
**using** *epsilon-eq-empty-or-singleton*[*of N C*]
**by** *auto*

**definition** (**in** *ground-superposition-calculus*) *rewrite-sys* **where**
*rewrite-sys N C* ≡ (⋃ *D* ∈ {*D* ∈ *N*. *D* $\prec_c$ *C*}. *epsilon* {*E* ∈ *N*. *E* $\preceq_c$ *D*} *D*)

**definition** (**in** *ground-superposition-calculus*) *rewrite-sys′* **where**
*rewrite-sys′ N* ≡ (⋃ *C* ∈ *N*. *epsilon N C*)

**lemma** (**in** *ground-superposition-calculus*) *rewrite-sys-alt*: *rewrite-sys′* {*D* ∈ *N*. *D* $\prec_c$ *C*} = *rewrite-sys N C*
**unfolding** *rewrite-sys′-def rewrite-sys-def*
**proof** (*rule SUP-cong*)
**show** {*D* ∈ *N*. *D* $\prec_c$ *C*} = {*D* ∈ *N*. *D* $\prec_c$ *C*} **..**
**next**
**show** ⋀*x*. *x* ∈ {*D* ∈ *N*. *D* $\prec_c$ *C*} ⟹ *epsilon* {*D* ∈ *N*. *D* $\prec_c$ *C*} *x* = *epsilon* {*E* ∈ *N*. $(\prec_c)^{==}$ *E x*} *x*
**using** *epsilon-filter-le-conv*
**by** (*smt* (*verit*, *best*) *Collect-cong clause.order.le-less-trans mem-Collect-eq*)

**qed**

**lemma** (**in** *ground-superposition-calculus*) *mem-epsilonE*:
  **assumes** *rule-in*: *rule* $\in$ *epsilon N C*
  **obtains** *l r C'* **where**
    *C* $\in$ *N* **and**
    *rule* = (*l*, *r*) **and**
    *C* = *add-mset* (*Pos* (*Upair l r*)) *C'* **and**
    *select C* = {#} **and**
    *is-strictly-maximal* (*Pos* (*Upair l r*)) *C* **and**
    $r \prec_t l$ **and**
    $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\models$ *C* **and**
    $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$ $\models$ *C'* **and**
    *l* $\in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
  **using** *rule-in*
  **unfolding** *epsilon.simps*[*of N C*] *mem-Collect-eq Let-def rewrite-sys-def*
  **by** (*metis* (*no-types*, *lifting*))

**lemma** (**in** *ground-superposition-calculus*) *mem-epsilon-iff*:
  (*l*, *r*) $\in$ *epsilon N C* $\longleftrightarrow$
    ($\exists$ *C'*. *C* $\in$ *N* $\wedge$ *C* = *add-mset* (*Pos* (*Upair l r*)) *C'* $\wedge$ *select C* = {#} $\wedge$
      *is-strictly-maximal* (*Pos* (*Upair l r*)) *C* $\wedge$ $r \prec_t l$ $\wedge$
      $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys'* {*D* $\in$ *N*. $D \prec_c C$}))$^{\downarrow}$ $\models$ *C* $\wedge$
     $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys'* {*D* $\in$ *N*. $D \prec_c C$})))$^{\downarrow}$
$\models$ *C'* $\wedge$
      *l* $\in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys'* {*D* $\in$ *N*. $D \prec_c C$})))
  (**is** *?LHS* $\longleftrightarrow$ *?RHS*)
**proof** (*rule iffI*)
  **assume** *?LHS*
  **thus** *?RHS*
    **using** *rewrite-sys-alt*
    **by** (*auto elim*: *mem-epsilonE*)
**next**
  **assume** *?RHS*
  **thus** *?LHS*
    **unfolding** *epsilon.simps*[*of N C*] *mem-Collect-eq*
    **unfolding** *rewrite-sys-alt rewrite-sys-def* **by** *auto*
**qed**

**lemma** (**in** *ground-superposition-calculus*) *rhs-lt-lhs-if-mem-rewrite-sys*:
  **assumes** (*t1*, *t2*) $\in$ *rewrite-sys N C*
  **shows** $t2 \prec_t t1$
  **using** *assms*
  **unfolding** *rewrite-sys-def*
  **by** (*smt* (*verit*, *best*) *UN-iff mem-epsilonE prod.inject*)

**lemma** (**in** *ground-superposition-calculus*) *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys*:
  **assumes** *rule-in*: (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
  **shows** $t2 \prec_t t1$

```
proof –
  from rule-in obtain ctxt t1' t2' where
    (t1, t2) = (ctxt⟨t1'⟩_G, ctxt⟨t2'⟩_G) ∧ (t1', t2') ∈ rewrite-sys N C
    unfolding rewrite-inside-gctxt-def mem-Collect-eq
    by auto
  thus ?thesis
  using rhs-lt-lhs-if-mem-rewrite-sys[of t1' t2']
  by (metis Pair-inject term.order.context-compatibility)
qed

lemma (in ground-superposition-calculus) rhs-lesseq-trm-lhs-if-mem-rtrancl-rewrite-inside-gctxt-rewrite-sys:
  assumes rule-in: (t1, t2) ∈ (rewrite-inside-gctxt (rewrite-sys N C))*
  shows t2 ⪯_t t1
  using rule-in
proof (induction t2 rule: rtrancl-induct)
  case base
  show ?case
    by order
next
  case (step t2 t3)
  from step.hyps have t3 ≺_t t2
    using rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys by metis
  with step.IH show ?case
    by order
qed

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

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

lemma (in ground-superposition-calculus) rewrite-sys-subset-if-less-cls:
  C ≺_c D ⟹ rewrite-sys N C ⊆ rewrite-sys N D
  unfolding rewrite-sys-def
  unfolding epsilon-filter-le-conv
 by (smt (verit, del-insts) SUP-mono clause.order.dual-order.strict-trans mem-Collect-eq
subset-eq)

lemma (in ground-superposition-calculus) mem-rewrite-sys-if-less-cls:
  assumes D ∈ N and D ≺_c C and (u, v) ∈ epsilon N D
  shows (u, v) ∈ rewrite-sys N C
  unfolding rewrite-sys-def UN-iff
proof (intro bexI)
  show D ∈ {D ∈ N. D ≺_c C}
    using ‹D ∈ N› ‹D ≺_c C› by simp
next
  show (u, v) ∈ epsilon {E ∈ N. E ⪯_c D} D
    using ‹(u, v) ∈ epsilon N D› epsilon-filter-le-conv by simp
```

**qed**

**lemma** (**in** *ground-superposition-calculus*) *less-trm-iff-less-cls-if-lhs-epsilon*:
  **assumes** $E_C$: *epsilon N C* = {(*s*, *t*)} **and** $E_D$: *epsilon N D* = {(*u*, *v*)}
  **shows** $u \prec_t s \longleftrightarrow D \prec_c C$
**proof** −
  **from** $E_C$ **have** (*s*, *t*) ∈ *epsilon N C*
    **by** *simp*
  **then obtain** $C'$ **where**
    *C* ∈ *N* **and**
    *C-def*: *C* = *add-mset* (*Pos* (*Upair s t*)) $C'$ **and**
    *is-strictly-maximal* (*Pos* (*Upair s t*)) *C* **and**
    $t \prec_t s$ **and**
    *s-irreducible*: *s* ∈ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
    **by** (*auto elim*!: *mem-epsilonE*)
  **hence** ∀ *L* ∈# $C'$. $L \prec_l$ *Pos* (*Upair s t*)
    **unfolding** *is-strictly-maximal-def*
    **by** *auto*

  **from** $E_D$ **obtain** $D'$ **where**
    *D* ∈ *N* **and**
    *D-def*: *D* = *add-mset* (*Pos* (*Upair u v*)) $D'$ **and**
    *is-strictly-maximal* (*Pos* (*Upair u v*)) *D* **and**
    $v \prec_t u$
    **by** (*auto simp*:  *elim*: *epsilon.elims dest*: *singleton-eq-CollectD*)
  **hence** ∀ *L* ∈# $D'$. $L \prec_l$ *Pos* (*Upair u v*)
    **by** (*auto simp*: *is-strictly-maximal-def*)

  **show** *?thesis*
  **proof** (*rule iffI*)
    **assume** $u \prec_t s$

    **moreover hence** $v \prec_t s$
      **using** ‹$v \prec_t u$›
      **by** *order*

    **ultimately have** *multp* ($\prec_t$) {#*u*, *v*#} {#*s*, *t*#}
      **using** *one-step-implies-multp*[*of* {#*s*, *t*#} {#*u*, *v*#} - {#}]
      **by** *simp*

    **hence** *Pos* (*Upair u v*) $\prec_l$ *Pos* (*Upair s t*)
      **by** (*simp add*: $less_l$*-def*)

    **moreover hence** ∀ *L* ∈# $D'$. $L \prec_l$ *Pos* (*Upair s t*)
      **using** ‹∀ *L* ∈# $D'$. $L \prec_l$ *Pos* (*Upair u v*)›
      **by** (*meson literal.order.transp-on-less transpD*)

    **ultimately show** $D \prec_c C$
      **using** *one-step-implies-multp*[*of C D* - {#}] $less_c$*-def*

**by** (*simp add*: *D-def C-def*)
**next**
**assume** $D \prec_c C$

**have** $(u, v) \in$ *rewrite-sys N C*
**using** $E_D$ ‹$D \in N$› ‹$D \prec_c C$› *mem-rewrite-sys-if-less-cls*
**by** *auto*

**hence** $(u, v) \in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
**by** *blast*

**hence** $s \neq u$
**using** *s-irreducible*
**by** *auto*

**moreover have** $\neg (s \prec_t u)$
**proof** (*rule notI*)
**assume** $s \prec_t u$

**moreover hence** $t \prec_t u$
**using** ‹$t \prec_t s$›
**by** *order*

**ultimately have** *multp* $(\prec_t)$ {#s, t#} {#u, v#}
**using** *one-step-implies-multp*[*of* {#u, v#} {#s, t#} - {#}]
**by** *simp*

**hence** *Pos* (*Upair s t*) $\prec_l$ *Pos* (*Upair u v*)
**by** (*simp add*: $less_l$*-def*)

**moreover hence** $\forall L \in\# C'. L \prec_l$ *Pos* (*Upair u v*)
**using** ‹$\forall L \in\# C'. L \prec_l$ *Pos* (*Upair s t*)›
**by** (*meson literal.order.transp-on-less transpD*)

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

**thus** *False*
**using** ‹$D \prec_c C$›
**by** *order*
**qed**
**ultimately show** $u \prec_t s$
**by** *order*
**qed**
**qed**

**lemma** (**in** *ground-superposition-calculus*) *termination-rewrite-sys*: *wf* ((*rewrite-sys N C*)$^{-1}$)

```
proof (rule wf-if-convertible-to-wf)
  show wf {(x, y). x ≺t y}
    using term.order.wfp
    by (simp add: wfp-def)
next
  fix t s
  assume (t, s) ∈ (rewrite-sys N C)⁻¹
  hence (s, t) ∈ rewrite-sys N C
    by simp
  then obtain D where D ≺c C and (s, t) ∈ epsilon N D
    unfolding rewrite-sys-def using epsilon-filter-le-conv by blast
  hence t ≺t s
    by (auto elim: mem-epsilonE)
  thus (t, s) ∈ {(x, y). x ≺t y}
    by (simp add: )
qed

lemma (in ground-superposition-calculus) termination-Union-rewrite-sys:
  wf ((⋃D ∈ N. rewrite-sys N D)⁻¹)
proof (rule wf-if-convertible-to-wf)
  show wf {(x, y). x ≺t y}
    using term.order.wfp
    by (simp add: wfp-def)
next
  fix t s
  assume (t, s) ∈ (⋃D ∈ N. rewrite-sys N D)⁻¹

  hence (s, t) ∈ (⋃D ∈ N. rewrite-sys N D)
    by simp

  then obtain C where C ∈ N (s, t) ∈ rewrite-sys N C
    by auto

  then obtain D where D ≺c C and (s, t) ∈ epsilon N D
    unfolding rewrite-sys-def using epsilon-filter-le-conv
    by blast

  hence t ≺t s
    by (auto elim: mem-epsilonE)

  thus (t, s) ∈ {(x, y). x ≺t y}
    by simp
qed

lemma (in ground-superposition-calculus) no-crit-pairs:
  {(t1, t2) ∈ ground-critical-pairs (⋃ (epsilon N2 ‘ N)). t1 ≠ t2} = {}
proof (rule ccontr)
  assume {(t1, t2).
    (t1, t2) ∈ ground-critical-pairs (⋃ (epsilon N2 ‘ N)) ∧ t1 ≠ t2} ≠ {}
```

**then obtain** *ctxt l r1 r2* **where**
($ctxt\langle r2\rangle_G$, *r1*) ∈ *ground-critical-pairs* (⋃ (*epsilon N2* ‘ *N*)) **and**
$ctxt\langle r2\rangle_G \neq r1$ **and**
*rule1-in*: ($ctxt\langle l\rangle_G$, *r1*) ∈ ⋃ (*epsilon N2* ‘ *N*) **and**
*rule2-in*: (*l*, *r2*) ∈ ⋃ (*epsilon N2* ‘ *N*)
**unfolding** *ground-critical-pairs-def mem-Collect-eq* **by** *blast*

**from** *rule1-in rule2-in* **obtain** *C1 C2* **where**
*C1* ∈ *N* **and** *rule1-in′*: ($ctxt\langle l\rangle_G$, *r1*) ∈ *epsilon N2 C1* **and**
*C2* ∈ *N* **and** *rule2-in′*: (*l*, *r2*) ∈ *epsilon N2 C2*
**by** *auto*

**from** *rule1-in′* **obtain** *C1′* **where**
*C1-def*: *C1* = *add-mset* (*Pos* (*Upair* $ctxt\langle l\rangle_G$ *r1*)) *C1′* **and**
*C1-max*: *is-strictly-maximal* (*Pos* (*Upair* $ctxt\langle l\rangle_G$ *r1*)) *C1* **and**
$r1 \prec_t ctxt\langle l\rangle_G$ **and**
*l1-irreducible*: $ctxt\langle l\rangle_G$ ∈ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N2 C1*))
**by** (*auto elim*: *mem-epsilonE*)

**from** *rule2-in′* **obtain** *C2′* **where**
*C2-def*: *C2* = *add-mset* (*Pos* (*Upair l r2*)) *C2′* **and**
*C2-max*: *is-strictly-maximal* (*Pos* (*Upair l r2*)) *C2* **and**
$r2 \prec_t l$
**by** (*auto elim*: *mem-epsilonE*)

**have** *epsilon N2 C1* = {($ctxt\langle l\rangle_G$, *r1*)}
**using** *rule1-in′ epsilon-eq-empty-or-singleton* **by** *fastforce*

**have** *epsilon N2 C2* = {(*l*, *r2*)}
**using** *rule2-in′ epsilon-eq-empty-or-singleton* **by** *fastforce*

**show** *False*
**proof** (*cases ctxt* = □)
**case** *True*
**hence** ¬ ($ctxt\langle l\rangle_G \prec_t l$) **and** ¬ ($l \prec_t ctxt\langle l\rangle_G$)
**by** (*simp-all add*: *irreflpD*)

**hence** ¬ ($C1 \prec_c C2$) **and** ¬ ($C2 \prec_c C1$)
**using** ‹*epsilon N2 C1* = {($ctxt\langle l\rangle_G$, *r1*)}› ‹*epsilon N2 C2* = {(*l*, *r2*)}›
*less-trm-iff-less-cls-if-lhs-epsilon*
**by** *simp-all*

**hence** *C1* = *C2*
**by** *order*

**hence** *r1* = *r2*
**using** ‹*epsilon N2 C1* = {($ctxt\langle l\rangle_G$, *r1*)}› ‹*epsilon N2 C2* = {(*l*, *r2*)}›
**by** *simp*

**moreover have** $r1 \neq r2$
  **using** ‹$ctxt\langle r2\rangle_G \neq r1$›
  **unfolding** ‹$ctxt = \square$›
  **by** *simp*

**ultimately show** *?thesis*
  **by** *contradiction*
**next**
  **case** *False*
  **hence** $l \prec_t ctxt\langle l\rangle_G$
    **by** (*metis term.order.subterm-property*)

  **hence** $C2 \prec_c C1$
    **using** ‹$epsilon\ N2\ C1 = \{(ctxt\langle l\rangle_G, r1)\}$› ‹$epsilon\ N2\ C2 = \{(l, r2)\}$›
      *less-trm-iff-less-cls-if-lhs-epsilon*
    **by** *simp*

  **have** $(l, r2) \in rewrite\text{-}sys\ N2\ C1$
  **by** (*metis* ‹$C2 \prec_c C1$› ‹$epsilon\ N2\ C2 = \{(l, r2)\}$› *mem-epsilonE mem-rewrite-sys-if-less-cls*
      *singletonI*)

  **hence** $(ctxt\langle l\rangle_G, ctxt\langle r2\rangle_G) \in rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N2\ C1)$
    **by** *auto*

  **thus** *False*
    **using** *l1-irreducible*
    **by** *auto*
  **qed**
**qed**

**lemma** (**in** *ground-superposition-calculus*) *WCR-Union-rewrite-sys*:
  $WCR\ (rewrite\text{-}inside\text{-}gctxt\ (\bigcup D \in N.\ epsilon\ N2\ D))$
  **unfolding** *ground-critical-pair-theorem*
**proof** (*intro subsetI ballI*)
  **fix** *tuple*
  **assume** *tuple-in*: $tuple \in ground\text{-}critical\text{-}pairs\ (\bigcup\ (epsilon\ N2\ `\ N))$

  **then obtain** *t1 t2* **where** *tuple-def*: $tuple = (t1, t2)$
    **by** *fastforce*

  **moreover have** $(t1, t2) \in (rewrite\text{-}inside\text{-}gctxt\ (\bigcup\ (epsilon\ N2\ `\ N)))^{\downarrow}$ **if** $t1 =$
$t2$
    **using** *that* **by** *auto*

  **moreover have** *False* **if** $t1 \neq t2$
    **using** *that tuple-def tuple-in no-crit-pairs* **by** *simp*

  **ultimately show** $tuple \in (rewrite\text{-}inside\text{-}gctxt\ (\bigcup\ (epsilon\ N2\ `\ N)))^{\downarrow}$
    **by** (*cases* $t1 = t2$) *simp-all*

**qed**

**lemma** (**in** *ground-superposition-calculus*)
**assumes**
$D \preceq_c C$ **and**
$E_C$-*eq*: *epsilon N C* = {(*s*, *t*)} **and**
*L-in*: $L \in\# D$ **and**
*topmost-trms-of-L*: *mset-uprod* (*atm-of L*) = {#*u*, *v*#}
**shows**
*lesseq-trm-if-pos*: *is-pos L* $\Longrightarrow u \preceq_t s$ **and**
*less-trm-if-neg*: *is-neg L* $\Longrightarrow u \prec_t s$
**proof** −
**from** $E_C$-*eq* **have** (*s*, *t*) ∈ *epsilon N C*
**by** *simp*
**then obtain** $C'$ **where**
*C-def*: *C* = *add-mset* (*Pos* (*Upair s t*)) $C'$ **and**
*C-max-lit*: *is-strictly-maximal* (*Pos* (*Upair s t*)) *C* **and**
$t \prec_t s$
**by** (*auto elim*: *mem-epsilonE*)

**have** *Pos* (*Upair s t*) $\prec_l$ *L* **if** *is-pos L* **and** $\neg\ u \preceq_t s$
**proof** −
**from** *that*(*2*) **have** $s \prec_t u$
**by** *order*

**hence** *multp* ($\prec_t$) {#*s*, *t*#} {#*u*, *v*#}
**using** ‹$t \prec_t s$›
**by** (*smt* (*verit*, *del-insts*) *add.right-neutral empty-iff insert-iff one-step-implies-multp set-mset-add-mset-insert set-mset-empty transpD term.order.transp union-mset-add-mset-right*)

**with** *that*(*1*) **show** *Pos* (*Upair s t*) $\prec_l$ *L*
**using** *topmost-trms-of-L*
**by** (*cases L*) (*simp-all add*: $less_l$-*def*)
**qed**

**moreover have** *Pos* (*Upair s t*) $\prec_l$ *L* **if** *is-neg L* **and** $\neg\ u \prec_t s$
**proof** −
**from** *that*(*2*) **have** $s \preceq_t u$
**by** *order*

**hence** *multp* ($\prec_t$) {#*s*, *t*#} {#*u*, *v*, *u*, *v*#}
**using** ‹$t \prec_t s$›
**by** (*smt* (*z3*) *add-mset-add-single add-mset-remove-trivial add-mset-remove-trivial-iff empty-not-add-mset insert-DiffM insert-noteq-member one-step-implies-multp reflclp-iff transp-def term.order.transp union-mset-add-mset-left union-mset-add-mset-right*)

**with** *that*(*1*) **show** *Pos* (*Upair s t*) $\prec_l$ *L*

**using** *topmost-trms-of-L*
**by** (*cases L*) (*simp-all add*: *less*$_l$*-def*)
**qed**

**moreover have** *False* **if** *Pos* (*Upair s t*) $\prec_l$ *L*
**proof** −
**have** *C* $\prec_c$ *D*
**unfolding** *less*$_c$*-def*
**proof** (*rule multp-if-maximal-of-lhs-is-less*)
**show** *Pos* (*Upair s t*) ∈# *C*
**by** (*simp add*: *C-def*)
**next**
**show** *L* ∈# *D*
**using** *L-in*
**by** *simp*
**next**
**show** *is-maximal* (*Pos* (*Upair s t*)) *C*
**using** *is-maximal-if-is-strictly-maximal*[*OF C-max-lit*]**.**
**next**
**show** *Pos* (*Upair s t*) $\prec_l$ *L*
**using** *that* **.**
**qed** *simp-all*

**with** ‹*D* $\preceq_c$ *C*› **show** *False*
**by** *order*
**qed**

**ultimately show** *is-pos L* ⟹ *u* $\preceq_t$ *s* **and** *is-neg L* ⟹ *u* $\prec_t$ *s*
**by** *argo+*
**qed**

**lemma** (**in** *ground-order*) *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*:
**fixes** *t t1 t2 r*
**assumes**
*rule-in*: (*t1* , *t2*) ∈ *rewrite-inside-gctxt r* **and**
*ball-lt-lhs*: ⋀*t1 t2*. (*t1* , *t2*) ∈ *r* ⟹ *t* $\prec_t$ *t1*
**shows** *t* $\prec_t$ *t1*
**proof** −
**from** *rule-in* **obtain** *ctxt t1*′ *t2*′ **where**
*rule-in*′: (*t1*′, *t2*′) ∈ *r* **and**
*l-def*: *t1* = *ctxt*⟨*t1*′⟩$_G$ **and**
*r-def*: *t2* = *ctxt*⟨*t2*′⟩$_G$
**unfolding** *rewrite-inside-gctxt-def* **by** *fast*

**show** *?thesis*
**using** *ball-lt-lhs*[*OF rule-in*′] *term.order.less-eq-subterm-property*[*of t1*′ *ctxt*]
*l-def*
**by** *order*
**qed**

**lemma** (**in** *ground-superposition-calculus*) *split-Union-epsilon*:
  **assumes** *D-in*: $D \in N$
  **shows** $(\bigcup C \in N.\ epsilon\ N\ C) =$
    $rewrite\text{-}sys\ N\ D \cup epsilon\ N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ epsilon\ N\ C)$
**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** $(\bigcup C \in N.\ epsilon\ N\ C) =$
    $rewrite\text{-}sys\ N\ D \cup epsilon\ N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ epsilon\ N\ C)$
    **using** *epsilon-filter-le-conv rewrite-sys-def*
    **by** *simp*
**qed**

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

**lemma** (**in** *ground-superposition-calculus*) *split-rewrite-sys*:
  **assumes** $C \in N$ **and** *D-in*: $D \in N$ **and** $D \prec_c C$
  **shows** $rewrite\text{-}sys\ N\ C = rewrite\text{-}sys\ N\ D \cup (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C'$
$\prec_c C\}.\ epsilon\ N\ C')$
**proof** −
  **have** $\{D \in N.\ D \prec_c C\} =$
    $\{y \in \{D \in N.\ D \prec_c C\}.\ y \prec_c D\} \cup \{D\} \cup \{y \in \{D \in N.\ D \prec_c C\}.\ D \prec_c$
$y\}$
  **proof** (*rule partition-set-around-element*)
    **show** *totalp-on* $\{D \in N.\ D \prec_c C\}$ $(\prec_c)$
      **using** *clause.order.totalp-on-less* **.**
  **next**
    **from** *D-in* ‹$D \prec_c C$› **show** $D \in \{D \in N.\ D \prec_c C\}$
      **by** *simp*
  **qed**

**also have** $\ldots = \{x \in N.\ x \prec_c C \wedge x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
  **by** *auto*

**also have** $\ldots = \{x \in N.\ x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
  **using** ‹$D \prec_c C$› *clause.order.less-trans*
  **by** *blast*

**finally have** *Collect-N-lt-C*: $\{x \in N.\ x \prec_c C\} = \{x \in N.\ x \prec_c D\} \cup \{x \in N.\ D \preceq_c x \wedge x \prec_c C\}$
  **by** *auto*

**have** *rewrite-sys* $N\ C = (\bigcup C' \in \{D \in N.\ D \prec_c C\}.$ *epsilon* $N\ C')$
  **using** *epsilon-filter-le-conv*
  **by** (*simp add*: *rewrite-sys-def*)

**also have** $\ldots = (\bigcup C' \in \{x \in N.\ x \prec_c D\}.$ *epsilon* $N\ C') \cup (\bigcup C' \in \{x \in N.\ D \preceq_c x \wedge x \prec_c C\}.$ *epsilon* $N\ C')$
  **unfolding** *Collect-N-lt-C*
  **by** *simp*

**finally show** *rewrite-sys* $N\ C =$ *rewrite-sys* $N\ D \cup \bigcup$ (*epsilon* $N$ ‘ $\{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}$)
  **using** *epsilon-filter-le-conv*
  **unfolding** *rewrite-sys-def*
  **by** *simp*
**qed**

**lemma** (**in** *ground-order*) *mem-join-union-iff-mem-join-lhs'*:
  **assumes**
    *ball-R₁-rhs-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_1 \Longrightarrow t2 \prec_t t1$ **and**
    *ball-R₂-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_2 \Longrightarrow s \prec_t t1 \wedge t \prec_t t1$
  **shows** $(s, t) \in (R_1 \cup R_2)^\downarrow \longleftrightarrow (s, t) \in R_1^\downarrow$
**proof** −
  **have** *ball-R₁-rhs-lt-lhs'*: $(t1, t2) \in R_1^* \Longrightarrow t2 \preceq_t t1$ **for** *t1 t2*
  **proof** (*induction t2 rule*: *rtrancl-induct*)
    **case** *base*
    **show** *?case*
      **by** *order*
  **next**
    **case** (*step y z*)
    **thus** *?case*
      **using** *ball-R₁-rhs-lt-lhs*
      **by** (*metis reflclp-iff transpD term.order.transp*)
  **qed**

  **show** *?thesis*
  **proof** (*rule mem-join-union-iff-mem-join-lhs*)
    **fix** *u* **assume** $(s, u) \in R_1^*$

**hence** $u \preceq_t s$
**using** *ball-*$R_1$*-rhs-lt-lhs*$'$ **by** *metis*

**show** $u \notin$ *Domain* $R_2$
**proof** (*rule notI*)
**assume** $u \in$ *Domain* $R_2$
**then obtain** $u'$ **where** $(u, u') \in R_2$
**by** *auto*

**hence** $s \prec_t u$
**using** *ball-*$R_2$*-lt-lhs*
**by** *simp*

**with** ‹$u \preceq_t s$› **show** *False*
**by** *order*
**qed**
**next**
**fix** $u$ **assume** $(t, u) \in R_1{}^*$
**hence** $u \preceq_t t$
**using** *ball-*$R_1$*-rhs-lt-lhs*$'$
**by** *simp*

**show** $u \notin$ *Domain* $R_2$
**proof** (*rule notI*)
**assume** $u \in$ *Domain* $R_2$
**then obtain** $u'$ **where** $(u, u') \in R_2$
**by** *auto*

**hence** $t \prec_t u$
**using** *ball-*$R_2$*-lt-lhs*
**by** *simp*

**with** ‹$u \preceq_t t$› **show** *False*
**by** *order*
**qed**
**qed**
**qed**

**lemma** (**in** *ground-order*) *mem-join-union-iff-mem-join-rhs*$'$:
**assumes**
*ball-*$R_1$*-rhs-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_2 \Longrightarrow t2 \prec_t t1$ **and**
*ball-*$R_2$*-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_1 \Longrightarrow s \prec_t t1 \wedge t \prec_t t1$
**shows** $(s, t) \in (R_1 \cup R_2)^{\downarrow} \longleftrightarrow (s, t) \in R_2{}^{\downarrow}$
**using** *assms mem-join-union-iff-mem-join-lhs*$'$
**by** (*metis* (*no-types*, *opaque-lifting*) *sup-commute*)

**lemma** (**in** *ground-order*) *mem-join-union-iff-mem-join-lhs*$''$:
**assumes**
*Range-*$R_1$*-lt-Domain-*$R_2$: $\bigwedge t1\ t2.\ t1 \in$ *Range* $R_1 \Longrightarrow t2 \in$ *Domain* $R_2 \Longrightarrow t1$

$\prec_t$ *t2* **and**
*s-lt-Domain-*$R_2$: $\bigwedge t2.\ t2 \in Domain\ R_2 \Longrightarrow s \prec_t t2$ **and**
*t-lt-Domain-*$R_2$: $\bigwedge t2.\ t2 \in Domain\ R_2 \Longrightarrow t \prec_t t2$
**shows** $(s, t) \in (R_1 \cup R_2)^\downarrow \longleftrightarrow (s, t) \in R_1{}^\downarrow$
**proof** (*rule mem-join-union-iff-mem-join-lhs*)
**fix** *u* **assume** $(s, u) \in R_1{}^*$
**hence** $u = s \lor u \in Range\ R_1$
**by** (*meson Range.intros rtrancl.cases*)

**thus** $u \notin Domain\ R_2$
**using** *Range-*$R_1$*-lt-Domain-*$R_2$ *s-lt-Domain-*$R_2$
**by** (*metis irreflpD term.order.irreflp-on-less*)
**next**
**fix** *u* **assume** $(t, u) \in R_1{}^*$
**hence** $u = t \lor u \in Range\ R_1$
**by** (*meson Range.intros rtrancl.cases*)

**thus** $u \notin Domain\ R_2$
**using** *Range-*$R_1$*-lt-Domain-*$R_2$ *t-lt-Domain-*$R_2$
**by** (*metis irreflpD term.order.irreflp-on-less*)
**qed**

**lemma** (**in** *ground-superposition-calculus*) *lift-entailment-to-Union*:
**fixes** *N D*
**defines** $R_D \equiv rewrite\text{-}sys\ N\ D$
**assumes**
*D-in*: $D \in N$ **and**
$R_D$*-entails-D*: *upair* ‘ $(rewrite\text{-}inside\text{-}gctxt\ R_D)^\downarrow \Vdash D$
**shows**
*upair* ‘ $(rewrite\text{-}inside\text{-}gctxt\ (\bigcup D \in N.\ epsilon\ N\ D))^\downarrow \Vdash D$ **and**
$\bigwedge C.\ C \in N \Longrightarrow D \prec_c C \Longrightarrow$ *upair* ‘ $(rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ C))^\downarrow$
$\Vdash D$
**proof** −
**from** $R_D$*-entails-D* **obtain** *L s t* **where**
*L-in*: $L \in\# D$ **and**
*L-eq-disj-L-eq*: $L = Pos\ (Upair\ s\ t) \land (s, t) \in (rewrite\text{-}inside\text{-}gctxt\ R_D)^\downarrow \lor$
$L = Neg\ (Upair\ s\ t) \land (s, t) \notin (rewrite\text{-}inside\text{-}gctxt\ R_D)^\downarrow$
**unfolding** *true-cls-def true-lit-iff*
**by** (*metis* (*no-types, opaque-lifting*) *image-iff prod.case surj-pair uprod-exhaust*)

**from** *L-eq-disj-L-eq* **show**
*upair* ‘ $(rewrite\text{-}inside\text{-}gctxt\ (\bigcup D \in N.\ epsilon\ N\ D))^\downarrow \Vdash D$ **and**
$\bigwedge C.\ C \in N \Longrightarrow D \prec_c C \Longrightarrow$ *upair* ‘ $(rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ C))^\downarrow$
$\Vdash D$
**unfolding** *atomize-all atomize-conj atomize-imp*
**proof** (*elim disjE conjE*)
**assume** *L-def*: $L = Pos\ (Upair\ s\ t)$ **and** $(s, t) \in (rewrite\text{-}inside\text{-}gctxt\ R_D)^\downarrow$
**have** $R_D \subseteq (\bigcup D \in N.\ epsilon\ N\ D)$ **and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow R_D \subseteq rewrite\text{-}sys\ N\ C$

**unfolding** $R_D$*-def rewrite-sys-def*
**using** *D-in clause.order.transp-on-less*[*THEN transpD*]
**using** *epsilon-filter-le-conv*
**by** (*auto intro*: *Collect-mono*)

**hence** *rewrite-inside-gctxt* $R_D \subseteq$ *rewrite-inside-gctxt* ($\bigcup D \in N.$ *epsilon N D*)
**and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow$ *rewrite-inside-gctxt* $R_D \subseteq$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
**by** (*auto intro*!: *rewrite-inside-gctxt-mono*)

**hence** $(s,\ t) \in$ (*rewrite-inside-gctxt* ($\bigcup D \in N.$ *epsilon N D*))$^\downarrow$ **and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow (s,\ t) \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
**by** (*auto intro*!: *join-mono intro*: *set-mp*[*OF* - ‹$(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D$)$^\downarrow$›])

**thus** *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ *N*)))$^\downarrow \models D \wedge$
($\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow \models D$)
**unfolding** *true-cls-def true-lit-iff*
**using** *L-in L-def* **by** *blast*
**next**
**have** (*t1*, *t2*) $\in R_D \Longrightarrow$ *t2* $\prec_t$ *t1* **for** *t1 t2*
**by** (*auto simp*: $R_D$*-def rewrite-sys-def elim*: *mem-epsilonE*)

**hence** *ball-*$R_D$*-rhs-lt-lhs*: (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* $R_D \Longrightarrow$ *t2* $\prec_t$ *t1* **for** *t1 t2*
**by** (*smt* (*verit*, *ccfv-SIG*) *Pair-inject term.order.context-compatibility mem-Collect-eq rewrite-inside-gctxt-def*)

**assume** *L-def*: $L =$ *Neg* (*Upair s t*) **and** $(s,\ t) \notin$ (*rewrite-inside-gctxt* $R_D$)$^\downarrow$

**have** $(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D \cup$ *rewrite-inside-gctxt* ($\bigcup C \in \{C \in N.\ D \preceq_c C\}.$ *epsilon N C*))$^\downarrow \longleftrightarrow$
$(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D$)$^\downarrow$
**proof** (*rule mem-join-union-iff-mem-join-lhs′*)
**show** $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* $R_D \Longrightarrow$ *t2* $\prec_t$ *t1*
**using** *ball-*$R_D$*-rhs-lt-lhs* **by** *simp*
**next**
**have** *ball-Rinf-minus-lt-lhs*: $s \prec_t$ *fst rule* $\wedge\ t \prec_t$ *fst rule*
**if** *rule-in*: *rule* $\in$ ($\bigcup C \in \{C \in N.\ D \preceq_c C\}.$ *epsilon N C*)
**for** *rule*
**proof** −
**from** *rule-in* **obtain** $C$ **where**
$C \in N$ **and** $D \preceq_c C$ **and** *rule* $\in$ *epsilon N C*
**by** *auto*

**have** *epsilon-C-eq*: *epsilon N C* $= \{$(*fst rule*, *snd rule*)$\}$
**using** ‹*rule* $\in$ *epsilon N C*› *epsilon-eq-empty-or-singleton* **by** *force*

```
        show ?thesis
          using less-trm-if-neg[OF ‹D ⪯c C› epsilon-C-eq L-in]
          by (simp add: L-def)
      qed

    show ⋀t1 t2. (t1, t2) ∈ rewrite-inside-gctxt (⋃ (epsilon N ‘ {C ∈ N. (≺c)==
D C})) ⟹
        s ≺t t1 ∧ t ≺t t1
        using less-trm-const-lhs-if-mem-rewrite-inside-gctxt
        using ball-Rinf-minus-lt-lhs
        by force
    qed

    moreover have
      (s, t) ∈ (rewrite-inside-gctxt RD ∪ rewrite-inside-gctxt (⋃C′ ∈ {C′ ∈ N. D
⪯c C′ ∧ C′ ≺c C}. epsilon N C′))↓ ⟷
       (s, t) ∈ (rewrite-inside-gctxt RD)↓
      if C ∈ N and D ≺c C
      for C
    proof (rule mem-join-union-iff-mem-join-lhs′)
      show ⋀t1 t2. (t1, t2) ∈ rewrite-inside-gctxt RD ⟹ t2 ≺t t1
        using ball-RD-rhs-lt-lhs by simp
    next
      have ball-lt-lhs: s ≺t t1 ∧ t ≺t t1
        if C ∈ N and D ≺c C and
          rule-in: (t1, t2) ∈ (⋃C′ ∈ {C′ ∈ N. D ⪯c C′ ∧ C′ ≺c C}. epsilon N C′)
        for C t1 t2
      proof −
        from rule-in obtain C′ where
          C′ ∈ N and D ⪯c C′ and C′ ≺c C and (t1, t2) ∈ epsilon N C′
          by (auto simp: rewrite-sys-def)

        have epsilon-C′-eq: epsilon N C′ = {(t1, t2)}
          using ‹(t1, t2) ∈ epsilon N C′› epsilon-eq-empty-or-singleton by force

        show ?thesis
          using less-trm-if-neg[OF ‹D ⪯c C′› epsilon-C′-eq L-in]
          by (simp add: L-def)
      qed

    show ⋀t1 t2. (t1, t2) ∈ rewrite-inside-gctxt (⋃ (epsilon N ‘ {C′ ∈ N. (≺c)==
D C′ ∧ C′ ≺c C})) ⟹
        s ≺t t1 ∧ t ≺t t1
        using less-trm-const-lhs-if-mem-rewrite-inside-gctxt
        using ball-lt-lhs[OF that(1,2)]
        by (metis (no-types, lifting))
    qed
```

**ultimately have** $(s,\ t) \notin (\textit{rewrite-inside-gctxt}\ R_D \cup \textit{rewrite-inside-gctxt}\ (\bigcup C \in \{C \in N.\ D \preceq_c C\}.\ \textit{epsilon}\ N\ C))^{\downarrow}$ **and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow$
$(s,\ t) \notin (\textit{rewrite-inside-gctxt}\ R_D \cup \textit{rewrite-inside-gctxt}\ (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}.\ \textit{epsilon}\ N\ C'))^{\downarrow}$
**using** ‹$(s,\ t) \notin (\textit{rewrite-inside-gctxt}\ R_D)^{\downarrow}$› **by** *simp-all*

**hence** $(s,\ t) \notin (\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{\downarrow}$ **and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow (s,\ t) \notin (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$
**using** *split-Union-epsilon′*[*OF D-in*, *folded* $R_D$*-def*]
**using** *split-rewrite-sys*[*OF - D-in*, *folded* $R_D$*-def*]
**by** (*simp-all add*: *rewrite-inside-gctxt-union*)

**hence** $(\textit{Upair}\ s\ t) \notin \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{\downarrow}$ **and**
$\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow (\textit{Upair}\ s\ t) \notin \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$
**unfolding** *atomize-conj*
**by** (*meson sym-join true-lit-simps*(*2*) *true-lit-uprod-iff-true-lit-prod*(*2*))

**thus** *upair* ‘ $(\textit{rewrite-inside-gctxt}\ (\bigcup\ (\textit{epsilon}\ N$ ‘ $N)))^{\downarrow} \models D \wedge$
$(\forall C.\ C \in N \longrightarrow D \prec_c C \longrightarrow \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow} \models D)$
**unfolding** *true-cls-def true-lit-iff*
**using** *L-in L-def* **by** *metis*
**qed**
**qed**

**lemma** (**in** *ground-superposition-calculus*)
**assumes** *productive*: $\textit{epsilon}\ N\ C = \{(l,\ r)\}$
**shows**
*true-cls-if-productive-epsilon*:
*upair* ‘ $(\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{\downarrow} \models C$
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ D))^{\downarrow} \models C$ **and**
*false-cls-if-productive-epsilon*:
$\neg\ \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{\downarrow} \models C - \{\#\textit{Pos}\ (\textit{Upair}\ l\ r)\#\}$
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow \neg\ \textit{upair}$ ‘ $(\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ D))^{\downarrow} \models C - \{\#\textit{Pos}\ (\textit{Upair}\ l\ r)\#\}$
**proof** −
**from** *productive* **have** $(l,\ r) \in \textit{epsilon}\ N\ C$
**by** *simp*

**then obtain** $C'$ **where**
*C-in*: $C \in N$ **and**
*C-def*: $C = \textit{add-mset}\ (\textit{Pos}\ (\textit{Upair}\ l\ r))\ C'$ **and**
$\textit{select}\ C = \{\#\}$ **and**
*is-strictly-maximal* $(\textit{Pos}\ (\textit{Upair}\ l\ r))\ C$ **and**

$r \prec_t l$ **and**
$e$: ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ $C$ **and**
$f$: ¬ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$ ⊫ $C'$ **and**
$l \in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
**by** (*rule mem-epsilonE*) *blast*

**have** (*l*, *r*) ∈ (*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*))$^{\downarrow}$
**using** *C-in* ‹(*l*, *r*) ∈ *epsilon N C*› *mem-rewrite-inside-gctxt-if-mem-rewrite-rules*
**by** *blast*

**thus** *upair* ‘ (*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*))$^{\downarrow}$ ⊫ *C*
**using** *C-def*
**by** *blast*

**have** *rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*) =
*rewrite-inside-gctxt* (*rewrite-sys N C* ∪ *epsilon N C* ∪ (⋃ *D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*))
**using** *split-Union-epsilon*[*OF C-in*]
**by** *simp*

**also have** ... =
*rewrite-inside-gctxt* (*rewrite-sys N C* ∪ *epsilon N C*) ∪
*rewrite-inside-gctxt* (⋃ *D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*)
**by** (*simp add*: *rewrite-inside-gctxt-union*)

**finally have** *rewrite-inside-gctxt-Union-epsilon-eq*:
*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*) =
*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)) ∪
*rewrite-inside-gctxt* (⋃ *D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*)
**unfolding** *productive*
**by** *simp*

**have** *mem-join-union-iff-mem-lhs*:(*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)) ∪
*rewrite-inside-gctxt* (⋃ *D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*))$^{\downarrow}$ ⟷
(*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$
**if** *t1* $\preceq_t$ *l* **and** *t2* $\preceq_t$ *l*
**for** *t1 t2*
**proof** (*rule mem-join-union-iff-mem-join-lhs'*)
**fix** *s1 s2* **assume** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*))

**moreover have** *s2* $\prec_t$ *s1* **if** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* {(*l*, *r*)}
**proof** (*rule rhs-lt-lhs-if-rule-in-rewrite-inside-gctxt*[*OF that*])
**show** ⋀*s1 s2*. (*s1*, *s2*) ∈ {(*l*, *r*)} ⟹ *s2* $\prec_t$ *s1*
**using** ‹*r* $\prec_t$ *l*›
**by** *simp*
**qed** *simp-all*

**moreover have** *s2* $\prec_t$ *s1* **if** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* (*rewrite-sys N C*)
**proof** (*rule rhs-lt-lhs-if-rule-in-rewrite-inside-gctxt*[*OF that*])
**show** ⋀*s1 s2*. (*s1*, *s2*) ∈ *rewrite-sys N C* ⟹ *s2* $\prec_t$ *s1*
**by** (*simp add*: *rhs-lt-lhs-if-mem-rewrite-sys*)
**qed** *simp*

**ultimately show** *s2* $\prec_t$ *s1*
**using** *rewrite-inside-gctxt-union*[*of* {(*l*, *r*)}, *simplified*] **by** *blast*
**next**
**have** *ball-lt-lhs*: *t1* $\prec_t$ *s1* ∧ *t2* $\prec_t$ *s1*
**if** *rule-in*: (*s1*, *s2*) ∈ (⋃*D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*)
**for** *s1 s2*
**proof** −
**from** *rule-in* **obtain** *D* **where**
*D* ∈ *N* **and** *C* $\prec_c$ *D* **and** (*s1*, *s2*) ∈ *epsilon N D*
**by** (*auto simp*: *rewrite-sys-def*)

**have** $E_D$*-eq*: *epsilon N D* = {(*s1*, *s2*)}
**using** ‹(*s1*, *s2*) ∈ *epsilon N D*› *epsilon-eq-empty-or-singleton* **by** *force*

**have** *l* $\prec_t$ *s1*
**using** ‹*C* $\prec_c$ *D*›
**using** *less-trm-iff-less-cls-if-lhs-epsilon*[*OF* $E_D$*-eq productive*]
**by** *metis*

**with** ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*› **show** *?thesis*
**by** (*metis reflclp-iff transpD term.order.transp*)
**qed**
**thus** ⋀*l r*. (*l*, *r*) ∈ *rewrite-inside-gctxt* (⋃ (*epsilon N* ‘ {*D* ∈ *N*. *C* $\prec_c$ *D*}))
⟹ *t1* $\prec_t$ *l* ∧ *t2* $\prec_t$ *l*
**using** *rewrite-inside-gctxt-Union-epsilon-eq*
**using** *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*
**by** *presburger*
**qed**

**have** *neg-concl1*: ¬ *upair* ‘ (*rewrite-inside-gctxt* (⋃*D* ∈ *N*. *epsilon N D*))$^{\downarrow}$ ⊫ *C′*
**unfolding** *true-cls-def Set.bex-simps*
**proof** (*intro ballI*)
**fix** *L* **assume** *L-in*: *L* ∈# *C′*
**hence** *L* ∈# *C*
**by** (*simp add*: *C-def*)

**obtain** *t1 t2* **where**
*atm-L-eq*: *atm-of L* = *Upair t1 t2*
**by** (*metis uprod-exhaust*)

**hence** *trms-of-L*: *mset-uprod* (*atm-of L*) = {#*t1*, *t2*#}
**by** *simp*

**hence** *t1* $\preceq_t$ *l* **and** *t2* $\preceq_t$ *l*
**unfolding** *atomize-conj*
**using** *less-trm-if-neg*[*OF reflclp-refl productive* ‹*L* ∈# *C*›]
**using** *lesseq-trm-if-pos*[*OF reflclp-refl productive* ‹*L* ∈# *C*›]
**by** (*metis* (*no-types*, *opaque-lifting*) *add-mset-commute sup2CI*)

**have** (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (⋃*D* ∈ *N*. *epsilon N D*))$^{\downarrow}$ **if** *L-def*: *L* = *Pos* (*Upair t1 t2*)
**proof** −
**from** *that* **have** (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$
**using** *f* ‹*L* ∈# *C'*›
**by** *blast*

**thus** *?thesis*
**using** *rewrite-inside-gctxt-Union-epsilon-eq mem-join-union-iff-mem-lhs*[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
**by** *simp*
**qed**

**moreover have** (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (⋃*D* ∈ *N*. *epsilon N D*))$^{\downarrow}$
**if** *L-def*: *L* = *Neg* (*Upair t1 t2*)
**proof** −
**from** *that* **have** (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$
**using** *f* ‹*L* ∈# *C'*›
**by** (*meson true-lit-uprod-iff-true-lit-prod*(*2*) *sym-join true-cls-def true-lit-simps*(*2*))

**thus** *?thesis*
**using** *rewrite-inside-gctxt-Union-epsilon-eq*
*mem-join-union-iff-mem-lhs*[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
**by** *simp*
**qed**

**ultimately show** ¬ *upair* ‘ (*rewrite-inside-gctxt* (⋃ (*epsilon N* ‘ *N*)))$^{\downarrow}$ ⊫*l L*
**using** *atm-L-eq true-lit-uprod-iff-true-lit-prod*[*OF sym-join*] *true-lit-simps*
**by** (*smt* (*verit*, *ccfv-SIG*) *literal.exhaust-sel*)
**qed**

**then show** ¬ *upair* ‘ (*rewrite-inside-gctxt* (⋃*D* ∈ *N*. *epsilon N D*))$^{\downarrow}$ ⊫ *C* − {#*Pos* (*Upair l r*)#}
**by** (*simp add*: *C-def*)

**fix** *D*
**assume** *D* ∈ *N* **and** *C* $\prec_c$ *D*

**have** (*l*, *r*) ∈ *rewrite-sys N D*
**using** *C-in* ‹(*l*, *r*) ∈ *epsilon N C*› ‹*C* $\prec_c$ *D*› *mem-rewrite-sys-if-less-cls*
**by** *metis*

**hence** $(l, r) \in (rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ D))^{\downarrow}$
**by** *auto*

**thus** $upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ D))^{\downarrow} \models C$
**using** *C-def*
**by** *blast*

**from** $\langle D \in N \rangle$ **have** $rewrite\text{-}sys\ N\ D \subseteq (\bigcup D \in N.\ epsilon\ N\ D)$
**by** (*simp add*: *split-Union-epsilon′*)

**hence** $rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ D) \subseteq rewrite\text{-}inside\text{-}gctxt\ (\bigcup D \in N.\ epsilon\ N\ D)$
**using** *rewrite-inside-gctxt-mono*
**by** *metis*

**hence** $(rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ D))^{\downarrow} \subseteq (rewrite\text{-}inside\text{-}gctxt\ (\bigcup D \in N.\ epsilon\ N\ D))^{\downarrow}$
**using** *join-mono*
**by** *metis*

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

**obtain** *t1 t2* **where**
*atm-L-eq*: $atm\text{-}of\ L = Upair\ t1\ t2$
**by** (*metis uprod-exhaust*)

**hence** *trms-of-L*: $mset\text{-}uprod\ (atm\text{-}of\ L) = \{\#t1,\ t2\#\}$
**by** *simp*

**hence** $t1 \preceq_t l$ **and** $t2 \preceq_t l$
**unfolding** *atomize-conj*
**using** *less-trm-if-neg*[*OF reflclp-refl productive* $\langle L \in\#\ C \rangle$]
**using** *lesseq-trm-if-pos*[*OF reflclp-refl productive* $\langle L \in\#\ C \rangle$]
**by** (*metis* (*no-types*, *opaque-lifting*) *add-mset-commute sup2CI*)

**have** $(t1,\ t2) \notin (rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ D))^{\downarrow}$ **if** *L-def*: $L = Pos\ (Upair\ t1\ t2)$
**proof** −
**from** *that* **have** $(t1,\ t2) \notin (rewrite\text{-}inside\text{-}gctxt\ (insert\ (l, r)\ (rewrite\text{-}sys\ N\ C)))^{\downarrow}$
**using** *f* $\langle L \in\#\ C' \rangle$ **by** *blast*
**thus** *?thesis*
**using** *rewrite-inside-gctxt-Union-epsilon-eq*

**using** *mem-join-union-iff-mem-lhs*[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
**using** ‹(*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ ⊆ (*rewrite-inside-gctxt* (⋃ (*epsilon N* ‘ *N*)))$^\downarrow$› **by** *auto*
**qed**

**moreover have** (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ **if** *L-def*: *L* = *Neg* (*Upair t1 t2*)
**using** *e*
**proof** (*rule contrapos-np*)
**assume** (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$

**hence** (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
**using** *rewrite-sys-subset-if-less-cls*[*OF* ‹*C* $\prec_c$ *D*›]
**by** (*meson join-mono rewrite-inside-gctxt-mono subsetD*)

**thus** *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$ ⊫ *C*
**using** *neg-literal-notin-imp-true-cls*[*of Upair t1 t2 C upair* ‘ -$^\downarrow$]
**unfolding** *uprod-mem-image-iff-prod-mem*[*OF sym-join*]
**using** *L-def L-in C-def*
**by** *simp*
**qed**

**ultimately show** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ ⊫*l L*
**using** *atm-L-eq true-lit-uprod-iff-true-lit-prod*[*OF sym-join*] *true-lit-simps*
**by** (*smt* (*verit*, *ccfv-SIG*) *literal.exhaust-sel*)
**qed**
**thus** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ ⊫ *C* − {#*Pos* (*Upair l r*)#}
**by** (*simp add*: *C-def*)
**qed**

**lemma** *from-neq-double-rtrancl-to-eqE*:
**assumes** $x \neq y$ **and** $(x, z) \in r^*$ **and** $(y, z) \in r^*$
**obtains**
*w* **where** $(x, w) \in r$ **and** $(w, z) \in r^*$ |
*w* **where** $(y, w) \in r$ **and** $(w, z) \in r^*$
**using** *assms*
**by** (*metis converse-rtranclE*)

**lemma** *ex-step-if-joinable*:
**assumes** *asymp R* $(x, z) \in r^*$ **and** $(y, z) \in r^*$
**shows**
$R^{==}\ z\ y \Longrightarrow R\ y\ x \Longrightarrow \exists w.\ (x, w) \in r \wedge (w, z) \in r^*$
$R^{==}\ z\ x \Longrightarrow R\ x\ y \Longrightarrow \exists w.\ (y, w) \in r \wedge (w, z) \in r^*$
**using** *assms*
**by** (*metis asympD converse-rtranclE reflclp-iff*)+

**lemma** (**in** *ground-superposition-calculus*) *trans-join-rewrite-inside-gctxt-rewrite-sys*:
*trans* ((*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$)

**proof** (*rule trans-join*)
  **have** *wf* ((*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{-1}$)
  **proof** (*rule wf-converse-rewrite-inside-gctxt*)
    **fix** *s t*
    **assume** (*s*, *t*) ∈ *rewrite-sys N C*

    **then obtain** *D* **where** (*s*, *t*) ∈ *epsilon N D*
      **unfolding** *rewrite-sys-def*
      **using** *epsilon-filter-le-conv*
      **by** *auto*

    **thus** $t \prec_t s$
      **by** (*auto elim*: *mem-epsilonE*)
  **qed** *auto*
  **thus** *SN* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
    **by** (*simp only*: *SN-iff-wf*)
**next**
  **show** *WCR* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
    **unfolding** *rewrite-sys-def epsilon-filter-le-conv*
    **using** *WCR-Union-rewrite-sys*
    **by** (*metis* (*mono-tags*, *lifting*))
**qed**

**lemma** (**in** *ground-order*) *true-cls-insert-and-not-true-clsE*:
  **assumes**
    *upair* ‘ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ ⊫ *C* **and**
    ¬ *upair* ‘ (*rewrite-inside-gctxt R*)$^{\downarrow}$ ⊫ *C*
  **obtains** $t\ t'$ **where**
    *Pos* (*Upair* $t\ t'$) ∈# *C* **and**
    $t \prec_t t'$ **and**
    ($t$, $t'$) ∈ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ **and**
    ($t$, $t'$) ∉ (*rewrite-inside-gctxt R*)$^{\downarrow}$
**proof** −
  **assume** *hyp*: $\bigwedge t\ t'$. *Pos* (*Upair* $t\ t'$) ∈# *C* ⟹ $t \prec_t t'$ ⟹ ($t$, $t'$) ∈ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ ⟹
    ($t$, $t'$) ∉ (*rewrite-inside-gctxt R*)$^{\downarrow}$ ⟹ *thesis*

  **from** *assms* **obtain** *L* **where**
    *L* ∈# *C* **and**
    *entails-L*: *upair* ‘ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ ⊫*l L* **and**
    *doesnt-entail-L*: ¬ *upair* ‘ (*rewrite-inside-gctxt R*)$^{\downarrow}$ ⊫*l L*
    **by** (*meson true-cls-def*)

  **have** *totalp-on* (*set-uprod* (*atm-of L*)) ($\prec_t$)
    **by** *simp*

  **then obtain** $t\ t'$ **where** *atm-of L* = *Upair* $t\ t'$ **and** $t \preceq_t t'$
    **using** *ex-ordered-Upair* **by** *metis*

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

    hence L-def: L = Pos (Upair t t′)
      using ‹atm-of L = Upair t t′›
      by simp

    moreover have (t, t′) ∈ (rewrite-inside-gctxt (insert r R))↓
      using entails-L
      unfolding L-def
      unfolding true-lit-uprod-iff-true-lit-prod[OF sym-join]
      by (simp add: true-lit-def)

    moreover have (t, t′) ∉ (rewrite-inside-gctxt R)↓
      using doesnt-entail-L
      unfolding L-def
      unfolding true-lit-uprod-iff-true-lit-prod[OF sym-join]
      by (simp add: true-lit-def)

    ultimately show ?thesis
      using hyp ‹L ∈# C› ‹t ≼t t′› by auto
  next
    case (Neg A)
    hence L-def: L = Neg (Upair t t′)
      using ‹atm-of L = Upair t t′› by simp

    have (t, t′) ∉ (rewrite-inside-gctxt (insert r R))↓
      using entails-L
      unfolding L-def
      unfolding true-lit-uprod-iff-true-lit-prod[OF sym-join]
      by (simp add: true-lit-def)

    moreover have (t, t′) ∈ (rewrite-inside-gctxt R)↓
      using doesnt-entail-L
      unfolding L-def
      unfolding true-lit-uprod-iff-true-lit-prod[OF sym-join]
      by (simp add: true-lit-def)

    moreover have (rewrite-inside-gctxt R)↓ ⊆ (rewrite-inside-gctxt (insert r R))↓
      using join-mono rewrite-inside-gctxt-mono
      by (metis subset-insertI)

    ultimately have False
      by auto

    thus ?thesis ..
  qed
qed
```

**lemma** (**in** *ground-superposition-calculus*) *model-preconstruction*:
  **fixes**
    $N$ :: *'f gatom clause set* **and**
    $C$ :: *'f gatom clause*
  **defines**
    $entails \equiv \lambda E\ C.\ upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ E)^{\downarrow} \models C$
  **assumes** *saturated* $N$ **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
  **shows**
    $epsilon\ N\ C = \{\} \longleftrightarrow entails\ (rewrite\text{-}sys\ N\ C)\ C$
    $\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow entails\ (rewrite\text{-}sys\ N\ D)\ C$
  **unfolding** *atomize-all atomize-conj atomize-imp*
  **using** *clause.order.wfp C-in*
**proof** (*induction C rule*: *wfp-induct-rule*)
  **case** (*less C*)
  **note** *IH* = *less.IH*

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

  **define** *I* **where**
    $I = (rewrite\text{-}inside\text{-}gctxt\ (rewrite\text{-}sys\ N\ C))^{\downarrow}$

  **have** *refl I*
    **by** (*simp only*: *I-def refl-join*)

  **have** *trans I*
    **unfolding** *I-def*
    **using** *trans-join-rewrite-inside-gctxt-rewrite-sys* **.**

  **have** *sym I*
    **by** (*simp only*: *I-def sym-join*)

  **have** *compatible-with-gctxt I*
   **by** (*simp only*: *I-def compatible-with-gctxt-join compatible-with-gctxt-rewrite-inside-gctxt*)

  **note** *I-interp* = ‹*refl I*› ‹*trans I*› ‹*sym I*› ‹*compatible-with-gctxt I*›

  **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** $epsilon\ N\ C = \{\}$

    **have** *cond-conv*: $(\exists L.\ L \in\#\ select\ C \vee (select\ C = \{\#\} \wedge is\text{-}maximal\ L\ C \wedge$
$is\text{-}neg\ L)) \longleftrightarrow$
      $(\exists A.\ Neg\ A \in\#\ C \wedge (Neg\ A \in\#\ select\ C \vee select\ C = \{\#\} \wedge is\text{-}maximal$

(*Neg A*) *C*))
**by** (*metis* (*no-types*, *opaque-lifting*) *is-pos-def literal.order.is-maximal-in-mset-iff literal.disc*(*2*) *literal.exhaust mset-subset-eqD select-negative-literals select-subset*)

**show** *entails* (*rewrite-sys N C*) *C*
**proof** (*cases* ∃ *L*. *is-maximal L* (*select C*) ∨ (*select C* = {#} ∧ *is-maximal L C* ∧ *is-neg L*))
**case** *ex-neg-lit-sel-or-max*: *True*

**hence** ∃ *A*. *Neg A* ∈# *C* ∧ (*is-maximal* (*Neg A*) (*select C*) ∨ *select C* = {#} ∧ *is-maximal* (*Neg A*) *C*)
**by** (*metis is-pos-def literal.exhaust literal.order.is-maximal-in-mset-iff mset-subset-eqD select-negative-literals select-subset*)

**then obtain** *s s′* **where**
*Neg* (*Upair s s′*) ∈# *C* **and**
*sel-or-max*: *select C* = {#} ∧ *is-maximal* (*Neg* (*Upair s s′*)) *C* ∨ *is-maximal* (*Neg* (*Upair s s′*)) (*select C*)
**by** (*metis uprod-exhaust*)

**then obtain** *C′* **where**
*C-def*: *C* = *add-mset* (*Neg* (*Upair s s′*)) *C′*
**by** (*metis mset-add*)

**show** *?thesis*
**proof** (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫*l Pos* (*Upair s s′*))
**case** *True*
**hence** (*s*, *s′*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
**by** (*meson sym-join true-lit-simps*(*1*) *true-lit-uprod-iff-true-lit-prod*(*1*))

**have** $s = s' \lor s \prec_t s' \lor s' \prec_t s$
**by** *auto*

**thus** *?thesis*
**proof** (*rule disjE*)
**assume** *s* = *s′*
**define** *ι* :: ′*f gatom clause inference* **where**
*ι* = *Infer* [*C*] *C′*

**have** *eq-resolution C C′*
**proof** (*rule eq-resolutionI*)
**show** *C* = *add-mset* (*Neg* (*Upair s s′*)) *C′*
**by** (*simp only*: *C-def*)
**next**
**show** *Neg* (*Upair s s′*) = *Neg* (*Upair s s*)
**by** (*simp only*: ‹*s* = *s′*›)
**next**

**show** *select C* = {#} ∧ *is-maximal* (*s* !≈ *s*′) *C* ∨ *is-maximal* (*s* !≈ *s*′) (*select C*)
**using** *sel-or-max* **.**
**qed** *simp*

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

**moreover have** ⋀*t*. *t* ∈ *set* (*prems-of ι*) ⟹ *t* ∈ *N*
**using** ‹*C* ∈ *N*›
**by** (*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-C*′: *G-entails DD* {*C*′} **and**
*ball-DD-lt-C*: ∀ *D* ∈ *DD*. $D \prec_c C$
**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*) *C*′
**using** *I-interp DD-entails-C*′
**unfolding** *entails-def G-entails-def*
**by** (*simp add*: *I-def true-clss-def*)

**then show** *entails* (*rewrite-sys N C*) *C*
**using** *C-def entails-def*
**by** *simp*
**next**
**from** ‹(*s*, *s*′) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$› **obtain** *u* **where**
*s-u*: (*s*, *u*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^*$ **and**
*s*′*-u*: (*s*′, *u*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^*$
**by** *auto*

**moreover hence** $u \preceq_t s$ **and** $u \preceq_t s'$
**using** *rhs-lesseq-trm-lhs-if-mem-rtrancl-rewrite-inside-gctxt-rewrite-sys*
**by** *simp-all*

**moreover assume** $s \prec_t s' \lor s' \prec_t s$

**ultimately obtain** $u_0$ **where**
$s' \prec_t s \Longrightarrow (s, u_0) : \textit{rewrite-inside-gctxt } (\textit{rewrite-sys } N\ C)$
$s \prec_t s' \Longrightarrow (s', u_0) : \textit{rewrite-inside-gctxt } (\textit{rewrite-sys } N\ C)$ **and**
$(u_0, u) : (\textit{rewrite-inside-gctxt } (\textit{rewrite-sys } N\ C))^*$
**using** *ex-step-if-joinable*[*OF - s-u s′-u*]
**by** (*metis asympD term.order.asymp*)

**then obtain** *ctxt t t′* **where**
*s-eq-if*: $s' \prec_t s \Longrightarrow s = \textit{ctxt}\langle t \rangle_G$ **and**
*s′-eq-if*: $s \prec_t s' \Longrightarrow s' = \textit{ctxt}\langle t \rangle_G$ **and**
$u_0 = \textit{ctxt}\langle t' \rangle_G$ **and**
$(t, t') \in \textit{rewrite-sys } N\ C$
**by** (*smt* (*verit*) *Pair-inject* ‹$s \prec_t s' \lor s' \prec_t s$› *asympD term.order.asymp*
*mem-Collect-eq*
*rewrite-inside-gctxt-def*)

**then obtain** $D$ **where**
$(t, t') \in \textit{epsilon } N\ D$ **and** $D \in N$ **and** $D \prec_c C$
**unfolding** *rewrite-sys-def epsilon-filter-le-conv*
**by** *auto*

**then obtain** $D'$ **where**
*D-def*: $D = \textit{add-mset } (\textit{Pos } (\textit{Upair } t\ t'))\ D'$ **and**
*sel-D*: $\textit{select } D = \{\#\}$ **and**
*max-t-t′*: $\textit{is-strictly-maximal } (\textit{Pos } (\textit{Upair } t\ t'))\ D$ **and**
$t' \prec_t t$
**by** (*elim mem-epsilonE*) *fast*

**have** *superI*: $\textit{neg-superposition } D\ C\ (\textit{add-mset } (\textit{Neg } (\textit{Upair } s_1\langle t' \rangle_G\ s_1'))$
$(C' + D'))$
**if** $\{s, s'\} = \{s_1\langle t \rangle_G, s_1'\}$ **and** $s_1' \prec_t s_1\langle t \rangle_G$
**for** $s_1\ s_1'$
**proof** (*rule neg-superpositionI*)
**show** $C = \textit{add-mset } (\textit{Neg } (\textit{Upair } s\ s'))\ C'$
**by** (*simp only*: *C-def*)
**next**
**show** $D = \textit{add-mset } (\textit{Pos } (\textit{Upair } t\ t'))\ D'$
**by** (*simp only*: *D-def*)
**next**
**show** $D \prec_c C$
**using** ‹$D \prec_c C$› **.**
**next**
**show** $\textit{select } C = \{\#\} \land \textit{is-maximal } (\textit{Neg } (\textit{Upair } s\ s'))\ C \lor \textit{is-maximal}$
$(s\ !\!\approx s')\ (\textit{select } C)$
**using** *sel-or-max* **.**
**next**

**show** *select D* = {#}
  **using** *sel-D* .
**next**
  **show** *is-strictly-maximal* (*Pos* (*Upair t t*′)) *D*
  **using** *max-t-t*′ .
**next**
  **show** $t' \prec_t t$
  **using** ‹$t' \prec_t t$› .
**next**
  **from** *that*(*1*) **show** *Neg* (*Upair s s*′) = *Neg* (*Upair* $s_1\langle t\rangle_G$ $s_1{}'$)
  **by** *fastforce*
**next**
  **from** *that*(*2*) **show** $s_1{}' \prec_t s_1\langle t\rangle_G$ .
**qed** *simp-all*

**have** *neg-superposition D C* (*add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ *s*′)) (*C*′ + *D*′))
  **if** ‹$s' \prec_t s$›
**proof** (*rule superI*)
  **from** *that* **show** {*s*, *s*′} = {$ctxt\langle t\rangle_G$, *s*′}
  **using** *s-eq-if*
  **by** *simp*
**next**
  **from** *that* **show** $s' \prec_t ctxt\langle t\rangle_G$
  **using** *s-eq-if*
  **by** *simp*
**qed**

**moreover have** *neg-superposition D C* (*add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ *s*)) (*C*′ + *D*′))
  **if** ‹$s \prec_t s'$›
**proof** (*rule superI*)
  **from** *that* **show** {*s*, *s*′} = {$ctxt\langle t\rangle_G$, *s*}
  **using** *s*′*-eq-if*
  **by** *auto*
**next**
  **from** *that* **show** $s \prec_t ctxt\langle t\rangle_G$
  **using** *s*′*-eq-if*
  **by** *simp*
**qed**

**ultimately obtain** *CD* **where**
  *super*: *neg-superposition D C CD* **and**
  *CD-eq1*: $s' \prec_t s \Longrightarrow$ *CD* = *add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ *s*′)) (*C*′ + *D*′) **and**
  *CD-eq2*: $s \prec_t s' \Longrightarrow$ *CD* = *add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ *s*)) (*C*′ + *D*′)
  **using** ‹$s \prec_t s' \vee s' \prec_t s$› *s*′*-eq-if s-eq-if*
  **by** *metis*

**define** $\iota$ :: *'f gatom clause inference* **where**
  $\iota$ = *Infer* [*D*, *C*] *CD*

**have** $\iota \in$ *G-Inf*
  **using** *superposition-if-neg-superposition*[*OF super*]
  **by** (*auto simp only*: *ι-def G-Inf-def*)

**moreover have** $\bigwedge t.\ t \in set\ (prems\text{-}of\ \iota) \Longrightarrow t \in N$
  **using** ‹*C* ∈ *N*› ‹*D* ∈ *N*›
  **by** (*auto simp add*: *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* ⊆ *N* **and**
  *finite DD* **and**
  *DD-entails-CD*: *G-entails* (*insert D DD*) {*CD*} **and**
  *ball-DD-lt-C*: ∀ *D*∈*DD*. $D \prec_c C$
  **unfolding** *Red-I-def redundant-infer-def mem-Collect-eq*
  **by** (*auto simp*: *ι-def*)

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

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

**moreover have** ¬ *entails* (*rewrite-sys N C*) *D'*
  **unfolding** *entails-def*
  **using** *false-cls-if-productive-epsilon*(*2*)[*OF* - ‹*C* ∈ *N*› ‹$D \prec_c C$›]
 **by** (*metis D-def* ‹(*t*, *t'*) ∈ *epsilon N D*› *add-mset-remove-trivial empty-iff*
    *epsilon-eq-empty-or-singleton singletonD*)

**moreover have** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫*l*
  (*Neg* (*Upair ctxt*⟨*t'*⟩$_G$ *s'*))
  **if** $s' \prec_t s$
 **using** ‹($u_0$, *u*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^*$› ‹$u_0$ = *ctxt*⟨*t'*⟩$_G$›
*s'*-*u*
  **by** *blast*

**moreover have** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫*l*
(*Neg* (*Upair* $ctxt\langle t'\rangle_G$ *s*))
**if** $s \prec_t s'$
**using** ‹$(u_0, u) \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^*$› ‹$u_0 = ctxt\langle t'\rangle_G$›
*s-u*
**by** *blast*

**ultimately show** *entails* (*rewrite-sys N C*) *C*
**unfolding** *entails-def C-def*
**using** ‹$s \prec_t s' \lor s' \prec_t s$› *CD-eq1 CD-eq2*
**by** *fast*
**qed**
**next**
**case** *False*
**thus** *?thesis*
**using** ‹*Neg* (*Upair s s'*) ∈# *C*›
**by** (*auto simp add*: *entails-def true-cls-def*)
**qed**
**next**
**case** *False*
**hence** *select C* = {#}
**using** *literal.order.ex-maximal-in-mset* **by** *blast*

**from** *False* **obtain** *A* **where** *Pos-A-in*: *Pos A* ∈# *C* **and** *max-Pos-A*:
*is-maximal* (*Pos A*) *C*
**using** ‹*select C* = {#}› *literal.order.ex-maximal-in-mset*[*OF* ‹*C* ≠ {#}›]
**by** (*metis is-pos-def literal.order.is-maximal-in-mset-iff*)

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

**have** *totalp-on* (*set-uprod A*) ($\prec_t$)
**by** *simp*

**then obtain** *s s'* **where** *A-def*: *A* = *Upair s s'* **and** $s' \preceq_t s$
**using** *ex-ordered-Upair*[*of A* ($\prec_t$)] **by** *fastforce*

**show** *?thesis*
**proof** (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ *C'* ∨ *s* = *s'*)
**case** *True*
**then show** *?thesis*
**using** ‹*epsilon N C* = {}›
**using** *A-def C-def entails-def*
**by** *blast*
**next**
**case** *False*

**from** *False* **have** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ *C'*
**by** *simp*

**from** *False* **have** $s' \prec_t s$
**using** ‹$s' \preceq_t s$› *term.order.asymp*[*THEN asympD*]
**by** *auto*

**then show** *?thesis*
**proof** (*cases is-strictly-maximal* (*Pos A*) *C*)
**case** *strictly-maximal*: *True*
**show** *?thesis*
**proof** (*cases* $s \in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*)))
**case** *s-irreducible*: *True*

**hence** *e-or-f-doesnt-hold*: *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\Vdash$ $C \vee$
*upair* ‘ (*rewrite-inside-gctxt* (*insert* (*s*, *s'*) (*rewrite-sys N C*)))$^{\downarrow}$ $\Vdash C'$
**using** ‹*epsilon N C* = {}›[*unfolded epsilon.simps*[*of N C*]]
**using** ‹$C \in N$› *C-def* ‹*select C* = {#}› *strictly-maximal* ‹$s' \prec_t s$›
**unfolding** *A-def rewrite-sys-def*
**by** (*smt* (*verit*, *best*) *Collect-empty-eq*)

**show** *?thesis*
**proof** (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\Vdash C$)
**case** *e-doesnt-hold*: *True*
**thus** *?thesis*
**by** (*simp add*: *entails-def*)
**next**
**case** *e-holds*: *False*
**hence** *R-C-doesnt-entail-C'*: $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\Vdash C'$
**unfolding** *C-def*
**by** *simp*

**show** *?thesis*
**proof** (*cases upair* ‘ (*rewrite-inside-gctxt* (*insert* (*s*, *s'*) (*rewrite-sys N C*)))$^{\downarrow}$ $\Vdash C'$)
**case** *f-doesnt-hold*: *True*
**then obtain** $C''$ *t* $t'$ **where** *C'-def*: $C'$ = *add-mset* (*Pos* (*Upair t t'*)) $C''$ **and**
$t' \prec_t t$ **and**
$(t, t') \in$ (*rewrite-inside-gctxt* (*insert* (*s*, *s'*) (*rewrite-sys N C*)))$^{\downarrow}$ **and**
$(t, t') \notin$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
**using** *f-doesnt-hold R-C-doesnt-entail-C'*
**using** *true-cls-insert-and-not-true-clsE*
**by** (*metis insert-DiffM join-sym Upair-sym*)

**have** *Pos* (*Upair t t'*) $\prec_l$ *Pos* (*Upair s s'*)
**using** *strictly-maximal literal.order.not-less-iff-gr-or-eq*
**unfolding** *literal.order.is-strictly-maximal-in-mset-iff A-def C'-def C-def*

**by** *auto*

**have** $\neg\ (t \prec_t s)$
**proof** (*rule notI*)
**assume** $t \prec_t s$

**have** $(t,\ t') \in$ (*rewrite-inside-gctxt* (*insert* $(s,\ s')$ (*rewrite-sys N C*)))$^{\downarrow} \longleftrightarrow$
$(t,\ t') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
**unfolding** *rewrite-inside-gctxt-union*[*of* $\{(s,\ s')\}$ *rewrite-sys N C*, *simplified*]
**proof** (*rule mem-join-union-iff-mem-join-rhs'*)
**show** $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* $\{(s,\ s')\} \Longrightarrow t \prec_t$ *t1* $\wedge\ t' \prec_t$ *t1*
**using** ‹$t \prec_t s$› ‹$t' \prec_t t$›
**by** (*smt* (*verit*, *ccfv-threshold*) *fst-conv singletonD less-trm-const-lhs-if-mem-rewrite-inside-gctxt transpD term.order.transp*)
**next**
**show** $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* (*rewrite-sys N C*) $\Longrightarrow$ *t2* $\prec_t$ *t1*
**using** *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys* **by** *force*
**qed**
**thus** *False*
**using** ‹$(t,\ t') \in$ (*rewrite-inside-gctxt* (*insert* $(s,\ s')$ (*rewrite-sys N C*)))$^{\downarrow}$›
**using** ‹$(t,\ t') \notin$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$›
**by** *metis*
**qed**

**moreover have** $\neg\ (s \prec_t t)$
**proof** (*rule notI*)
**assume** $s \prec_t t$
**hence** *multp* $(\prec_t)$ {#*s*, *s'*#} {#*t*, *t'*#}
**using** ‹$s' \prec_t s$› ‹$t' \prec_t t$›
**using** *one-step-implies-multp*[*of* - - - {#}, *simplified*]
**by** (*metis* (*mono-tags*, *opaque-lifting*) *empty-not-add-mset insert-iff set-mset-add-mset-insert set-mset-empty singletonD transpD term.order.transp*)

**hence** *Pos* (*Upair s s'*) $\prec_l$ *Pos* (*Upair t t'*)
**by** (*simp add*: *less*$_l$*-def*)

**thus** *False*
**using** ‹$t \approx t' \prec_l s \approx s'$›
**by** *order*
**qed**

**ultimately have** $t = s$
**by** *order*
**hence** $t' \prec_t s'$
**using** ‹$t' \prec_t t$› ‹$s' \prec_t s$›
**using** ‹$Pos\ (Upair\ t\ t') \prec_l Pos\ (Upair\ s\ s')$›
**unfolding** $less_l$*-def*
**by** (*simp add*: *multp-cancel-add-mset term.order.transp*)

**obtain** $t''$ **where**
$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*)) **and**
$(t'', t') \in$ (*rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*)))$^\downarrow$
**using** ‹$(t, t') \in$ (*rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*)))$^\downarrow$›[*THEN joinD*]
**using** *ex-step-if-joinable*[*OF term.order.asymp - - -* ‹$t' \prec_t t$›]
**by** (*smt* (*verit*, *ccfv-threshold*) ‹$t = s$› *converse-rtranclE insertCI joinI-right*
*join-sym r-into-rtrancl mem-rewrite-inside-gctxt-if-mem-rewrite-rules rtrancl-join-join*)

**have** $t'' \prec_t t$
**proof** (*rule predicate-holds-of-mem-rewrite-inside-gctxt*[*of - - -* $\lambda x\ y.\ y \prec_t x$])
**show** $(t, t'') \in$ *rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*))
**using** ‹$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*))› **.**
**next**
**show** $\bigwedge t1\ t2.\ (t1, t2) \in$ *insert* $(s, s')$ (*rewrite-sys N C*) $\Longrightarrow t2 \prec_t t1$
**by** (*metis* ‹$s' \prec_t s$› *insert-iff old.prod.inject rhs-lt-lhs-if-mem-rewrite-sys*)
**next**
**show** $\bigwedge t1\ t2\ ctxt\ \sigma.\ (t1, t2) \in$ *insert* $(s, s')$ (*rewrite-sys N C*) $\Longrightarrow$
$t2 \prec_t t1 \Longrightarrow ctxt\langle t2\rangle_G \prec_t ctxt\langle t1\rangle_G$
**by** (*simp only*: *term.order.context-compatibility*)
**qed**

**have** $(t, t'') \in$ *rewrite-inside-gctxt* $\{(s, s')\}$
**using** ‹$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*))›
**using** ‹$t = s$› *s-irreducible mem-rewrite-step-union-NF*
**using** *rewrite-inside-gctxt-insert*
**by** *blast*

**hence** $\exists\, ctxt.\ s = ctxt\langle s\rangle_G \wedge t'' = ctxt\langle s'\rangle_G$
**by** (*simp add*: ‹$t = s$› *rewrite-inside-gctxt-def*)

**hence** $t'' = s'$
**by** (*metis gctxt-ident-iff-eq-GHole*)

**moreover have** $(t'', t') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$

**proof** (*rule mem-join-union-iff-mem-join-rhs*′[*THEN iffD1*])
**show** $(t'', t') \in$ (*rewrite-inside-gctxt* $\{(s, s')\} \cup$
*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
**using** ‹$(t'', t') \in$ (*rewrite-inside-gctxt* (*insert* $(s, s')$ (*rewrite-sys N C*)))$^\downarrow$›
**using** *rewrite-inside-gctxt-union*[*of* {-}, *simplified*]
**by** *metis*
**next**
**show** $\bigwedge t1\ t2.\ (t1, t2) \in$ *rewrite-inside-gctxt* (*rewrite-sys N C*) $\Longrightarrow t2 \prec_t t1$
**using** *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys* **.**
**next**
**show** $\bigwedge t1\ t2.\ (t1, t2) \in$ *rewrite-inside-gctxt* $\{(s, s')\} \Longrightarrow t'' \prec_t t1 \wedge t' \prec_t t1$
**using** ‹$t' \prec_t t$› ‹$t'' \prec_t t$›
**unfolding** ‹$t = s$›
**using** *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*
**by** *fastforce*
**qed**

**ultimately have** $(s', t') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
**by** *simp*

**let** *?concl* = *add-mset* (*Neg* (*Upair* $s'$ $t'$)) (*add-mset* (*Pos* (*Upair* $t$ $t'$)) $C''$)

**define** $\iota$ :: ′*f gatom clause inference* **where**
$\iota$ = *Infer* [*C*] *?concl*

**have** *eq-fact*: *eq-factoring C ?concl*
**proof** (*rule eq-factoringI*)
**show** *C* = *add-mset* (*Pos* (*Upair s s*′)) (*add-mset* (*Pos* (*Upair t t*′)) $C''$)
**by** (*simp add*: *C-def C*′*-def A-def*)
**next**
**show** *select C* = {#}
**using** ‹*select C* = {#}› **.**
**next**
**show** *is-maximal* (*Pos* (*Upair s s*′)) *C*
**by** (*metis A-def max-Pos-A*)
**next**
**show** $s' \prec_t s$
**using** ‹$s' \prec_t s$› **.**
**next**
**show** *Pos* (*Upair t t*′) = *Pos* (*Upair s t*′)
**unfolding** ‹$t = s$› **..**
**next**
**show** *add-mset* (*Neg* (*Upair s*′ *t*′)) (*add-mset* (*Pos* (*Upair t t*′)) $C''$) =

```
      add-mset (Neg (Upair s' t')) (add-mset (Pos (Upair s t')) C'')
      by (auto simp add: ‹t = s›)
  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-C': G-entails DD {?concl} and
    ball-DD-lt-C: ∀D∈DD. D ≺c C
    unfolding Red-I-def redundant-infer-def
    by (auto simp: ι-def)

  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

  hence entails (rewrite-sys N C) ?concl
    unfolding entails-def I-def[symmetric]
    using DD-entails-C'[unfolded G-entails-def]
    using I-interp
    by (simp add: true-clss-def)

  thus entails (rewrite-sys N C) C
    unfolding entails-def I-def[symmetric]
    unfolding C-def C'-def A-def
    using I-def ‹(s', t') ∈ (rewrite-inside-gctxt (rewrite-sys N C))↓›
    by blast
next
  case f-holds: False
  hence False
    using e-or-f-doesnt-hold e-holds
    by metis

  thus ?thesis ..
```

```
    qed
  qed
next
  case s-reducible: False

  hence ∃ss. (s, ss) ∈ rewrite-inside-gctxt (rewrite-sys N C)
    unfolding NF-def
    by auto

  then obtain ctxt t t′ D where
    D ∈ N and
    D ≺c C and
    (t, t′) ∈ epsilon N D and
    s = ctxt⟨t⟩G
    using epsilon-filter-le-conv
    by (auto simp: rewrite-inside-gctxt-def rewrite-sys-def)

  obtain D′ where
    D-def: D = add-mset (Pos (Upair t t′)) D′ and
    select D = {#} and
    max-t-t′: is-strictly-maximal (t ≈ t′) D and
    t′ ≺t t
    using ‹(t, t′) ∈ epsilon N D›
    by (elim mem-epsilonE) simp

  let ?concl = add-mset (Pos (Upair ctxt⟨t′⟩G s′)) (C′ + D′)

  define ι :: ′f gatom clause inference where
    ι = Infer [D, C] ?concl

  have super: pos-superposition D C ?concl
  proof (rule pos-superpositionI)
    show C = add-mset (Pos (Upair s s′)) C′
      by (simp only: C-def A-def)
  next
    show D = add-mset (Pos (Upair t t′)) D′
      by (simp only: D-def)
  next
    show D ≺c C
      using ‹D ≺c C› .
  next
    show select D = {#}
      using ‹select D = {#}› .
  next
    show select C = {#}
      using ‹select C = {#}› .
  next
    show is-strictly-maximal (s ≈ s′) C
      using A-def strictly-maximal
```

**by** *simp*
**next**
**show** *is-strictly-maximal* $(t \approx t')$ *D*
**using** *max-t-t′* **.**
**next**
**show** $t' \prec_t t$
**using** ‹$t' \prec_t t$› **.**
**next**
**show** *Pos* (*Upair s s′*) = *Pos* (*Upair* $ctxt\langle t\rangle_G$ $s'$)
**by** (*simp only*: ‹$s = ctxt\langle t\rangle_G$›)
**next**
**show** $s' \prec_t ctxt\langle t\rangle_G$
**using** ‹$s' \prec_t s$›
**unfolding** ‹$s = ctxt\langle t\rangle_G$› **.**
**qed** *simp-all*

**hence** $\iota \in$ *G-Inf*
**using** *superposition-if-pos-superposition*
**by** (*auto simp*: *ι-def G-Inf-def*)

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

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

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

**then obtain** *DD* **where**
*DD-subset*: $DD \subseteq N$ **and**
*finite DD* **and**
*DD-entails-concl*: *G-entails* (*insert D DD*) {*?concl*} **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*, *rule-format*, *of - C*]
**using** ‹$C \in N$› ‹$D \in N$› ‹$D \prec_c C$› *DD-subset ball-DD-lt-C*
**by** (*metis in-mono insert-iff*)

**ultimately have** *entails* (*rewrite-sys N C*) *?concl*
**using** *I-interp DD-entails-concl*
**unfolding** *entails-def G-entails-def*
**by** (*simp add*: *I-def true-clss-def*)

**moreover have** ¬ *entails* (*rewrite-sys N C*) $D'$
**unfolding** *entails-def*
**using** *false-cls-if-productive-epsilon*(*2*)[*OF* - ‹$C \in N$› ‹$D \prec_c C$›]
**by** (*metis D-def* ‹($t$, $t'$) ∈ *epsilon N D*› *add-mset-remove-trivial empty-iff epsilon-eq-empty-or-singleton singletonD*)

**ultimately have** *entails* (*rewrite-sys N C*) {#*Pos* (*Upair* $ctxt\langle t'\rangle_G$ $s'$)#}
**unfolding** *entails-def*
**using** ‹¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ $C'$›
**by** *fastforce*

**hence** ($ctxt\langle t'\rangle_G$, $s'$) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
**by** (*simp add*: *entails-def true-cls-def uprod-mem-image-iff-prod-mem*[*OF sym-join*])

**moreover have** ($ctxt\langle t\rangle_G$, $ctxt\langle t'\rangle_G$) ∈ *rewrite-inside-gctxt* (*rewrite-sys N C*)
**using** ‹($t$, $t'$) ∈ *epsilon N D*› ‹$D \in N$› ‹$D \prec_c C$› *rewrite-sys-def epsilon-filter-le-conv*
**by** (*auto simp*: *rewrite-inside-gctxt-def*)

**ultimately have** ($ctxt\langle t\rangle_G$, $s'$) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
**using** *r-into-rtrancl rtrancl-join-join*
**by** *metis*

**hence** *entails* (*rewrite-sys N C*) {#*Pos* (*Upair* $ctxt\langle t\rangle_G$ $s'$)#}
**unfolding** *entails-def true-cls-def*
**by** *auto*

**thus** *?thesis*
**using** *A-def C-def* ‹$s = ctxt\langle t\rangle_G$› *entails-def*
**by** *fastforce*
**qed**
**next**
**case** *False*
**hence** $2 \le$ *count* $C$ (*Pos* $A$)
**using** *max-Pos-A*
**by** (*meson is-strictly-maximal-def literal.order.count-ge-2-if-maximal-in-mset-and-not-greatest-in-mset literal.order.is-greatest-in-mset-iff literal.order.leD*)

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

**define** $\iota$ :: *'f gatom clause inference* **where**
$\iota$ = *Infer* [$C$] (*add-mset* (*Pos* (*Upair* $s$ $s'$)) (*add-mset* (*Neg* (*Upair* $s'$ $s'$))

$C'$))

**let** *?concl* = *add-mset* (*Pos* (*Upair s s'*)) (*add-mset* (*Neg* (*Upair s' s'*)) *C'*)

```
have eq-fact: eq-factoring C ?concl
proof (rule eq-factoringI)
  show C = add-mset (Pos A) (add-mset (Pos A) C')
    by (simp add: C-def)
next
  show Pos A = Pos (Upair s s')
    by (simp add: A-def)
next
  show Pos A = Pos (Upair s s')
    by (simp add: A-def)
next
  show select C = {#}
    using ‹select C = {#}› .
next
  show is-maximal (Pos A) C
    using max-Pos-A .
next
  show s' ≺t s
    using ‹s' ≺t s› .
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 {?concl} and
  ball-DD-lt-C: ∀D∈DD. D ≺c C
  unfolding Red-I-def redundant-infer-def mem-Collect-eq
  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) ?concl
          using I-interp DD-entails-concl
          unfolding entails-def G-entails-def
          by (simp add: I-def true-clss-def)

        then show ?thesis
          by (simp add: entails-def A-def C-def joinI-right pair-imageI)
      qed
    qed
  qed
qed

moreover have iib: entails (rewrite-sys N D) C if D ∈ N and C ≺c D for D
  using epsilon-eq-empty-or-singleton[of N C, folded ]
proof (elim disjE exE)
  assume epsilon N C = {}
  hence entails (rewrite-sys N C) C
    unfolding i by simp
  thus ?thesis
    using lift-entailment-to-Union(2)[OF ‹C ∈ N› - that]
    by (simp only: entails-def)
next
  fix l r assume epsilon N C = {(l, r)}
  thus ?thesis
    using true-cls-if-productive-epsilon(2)[OF ‹epsilon N C = {(l, r)}› that]
    by (simp only: entails-def)
qed

ultimately show ?case
  by metis
qed

lemma (in ground-superposition-calculus) model-construction:
  fixes
    N :: 'f gatom clause set and
    C :: 'f gatom clause
  defines
    entails ≡ λE C. upair ‘ (rewrite-inside-gctxt E)↓ ⊫ C
  assumes saturated N and {#} ∉ N and C-in: C ∈ N
  shows entails (⋃D ∈ N. epsilon N D) 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
    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** *l r* **assume** *epsilon N C* = {(*l*, *r*)}
**thus** *?thesis*
**using** *true-cls-if-productive-epsilon*(*1*)[*OF* ‹*epsilon N C* = {(*l*, *r*)}›]
**by** (*simp only*: *entails-def*)
**qed**

## 1.5 Static Refutational Completeness

**lemma** (**in** *ground-superposition-calculus*) *statically-complete*:
**fixes** *N* :: *'f gatom clause set*
**assumes** *saturated N* **and** *G-entails N* {{#}}
**shows** {#} ∈ *N*
**using** ‹*G-entails N* {{#}}›
**proof** (*rule contrapos-pp*)
**assume** {#} ∉ *N*

**define** *I* :: *'f gterm rel* **where**
$I = (\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{\downarrow}$

**show** ¬ *G-entails N G-Bot*
**unfolding** *G-entails-def not-all not-imp*
**proof** (*intro exI conjI*)
**show** *refl I*
**by** (*simp only*: *I-def refl-join*)
**next**
**show** *trans I*
**unfolding** *I-def*
**proof** (*rule trans-join*)
**have** $\textit{wf}\ ((\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))^{-1})$
**proof** (*rule wf-converse-rewrite-inside-gctxt*)
**fix** *s t*
**assume** $(s,\ t) \in (\bigcup D \in N.\ \textit{epsilon}\ N\ D)$
**then obtain** *C* **where** *C* ∈ *N* (*s*, *t*) ∈ *epsilon N C*
**by** *auto*

**thus** $t \prec_t s$
**by** (*auto elim*: *mem-epsilonE*)
**qed** *auto*
**thus** $\textit{SN}\ (\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))$
**unfolding** *SN-iff-wf* **.**
**next**
**show** $\textit{WCR}\ (\textit{rewrite-inside-gctxt}\ (\bigcup D \in N.\ \textit{epsilon}\ N\ D))$
**using** *WCR-Union-rewrite-sys* **.**

```
    qed
  next
    show sym I
      by (simp only: I-def sym-join)
  next
    show compatible-with-gctxt I
      unfolding I-def
     by (simp only: I-def compatible-with-gctxt-join compatible-with-gctxt-rewrite-inside-gctxt)
  next
    show upair ` I ⊫s N
      unfolding I-def
      using model-construction[OF ‹saturated N› ‹{#} ∉ N›]
      by (simp add: true-clss-def)
  next
    show ¬ upair ` I ⊫s G-Bot
      by simp
  qed
qed

sublocale ground-superposition-calculus ⊆ 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
theory Ground-Superposition-Soundness
  imports Ground-Superposition
begin

lemma (in ground-superposition-calculus) soundness-ground-superposition:
  assumes
    step: superposition P1 P2 C
  shows G-entails {P1, P2} {C}
  using step
proof (cases P1 P2 C rule: superposition.cases)
  case (superpositionI L₁ P₁' L₂ P₂' 𝒫 s t s' t')

  show ?thesis
    unfolding G-entails-def true-clss-singleton
    unfolding true-clss-insert
  proof (intro allI impI, elim conjE)
    fix I :: 'f gterm rel
    let ?I' = (λ(t₁, t). Upair t₁ t) ` I
    assume refl I and trans I and sym I and compatible-with-gctxt I and
      ?I' ⊫ P1 and ?I' ⊫ P2
```

**then obtain** *K1 K2* :: *'f gatom literal* **where**
*K1* ∈# *P1* **and** *?I'* ⊨*l K1* **and** *K2* ∈# *P2* **and** *?I'* ⊨*l K2*
**by** (*auto simp*: *true-cls-def*)

**show** *?I'* ⊨ *C*
**proof** (*cases K2* = 𝒫 (*Upair s*⟨*t*⟩$_G$ *s'*))
**case** *K1-def*: *True*
**hence** *?I'* ⊨*l* 𝒫 (*Upair s*⟨*t*⟩$_G$ *s'*)
**using** ‹*?I'* ⊨*l K2*› **by** *simp*

**show** *?thesis*
**proof** (*cases K1* = *Pos* (*Upair t t'*))
**case** *K2-def*: *True*
**hence** (*t*, *t'*) ∈ *I*
**using** ‹*?I'* ⊨*l K1*› *true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*

**have** *?thesis* **if** 𝒫 = *Pos*
**proof** −
**from** *that* **have** (*s*⟨*t*⟩$_G$, *s'*) ∈ *I*
**using** ‹*?I'* ⊨*l K2*› *K1-def true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*
**hence** (*s*⟨*t'*⟩$_G$, *s'*) ∈ *I*
**using** ‹(*t*, *t'*) ∈ *I*›
**using** ‹*compatible-with-gctxt I*› ‹*refl I*› ‹*sym I*› ‹*trans I*›
**by** (*meson compatible-with-gctxtD refl-onD1 symD trans-onD*)
**hence** *?I'* ⊨*l Pos* (*Upair s*⟨*t'*⟩$_G$ *s'*)
**by** *blast*
**thus** *?thesis*
**unfolding** *superpositionI that*
**by** *simp*
**qed**

**moreover have** *?thesis* **if** 𝒫 = *Neg*
**proof** −
**from** *that* **have** (*s*⟨*t*⟩$_G$, *s'*) ∉ *I*
**using** ‹*?I'* ⊨*l K2*› *K1-def true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*
**hence** (*s*⟨*t'*⟩$_G$, *s'*) ∉ *I*
**using** ‹(*t*, *t'*) ∈ *I*›
**using** ‹*compatible-with-gctxt I*› ‹*trans I*›
**by** (*metis compatible-with-gctxtD transD*)
**hence** *?I'* ⊨*l Neg* (*Upair s*⟨*t'*⟩$_G$ *s'*)
**by** (*meson* ‹*sym I*› *true-lit-simps*(*2*) *true-lit-uprod-iff-true-lit-prod*(*2*))
**thus** *?thesis*
**unfolding** *superpositionI that* **by** *simp*
**qed**

**ultimately show** *?thesis*
**using** ‹𝒫 ∈ {*Pos*, *Neg*}› **by** *auto*

```
      next
        case False
        hence K1 ∈# P2'
          using ‹K1 ∈# P1›
          unfolding superpositionI by simp
        hence ?I' ⊫ P2'
          using ‹?I' ⊫l K1› by blast
        thus ?thesis
          unfolding superpositionI by simp
      qed
    next
      case False
      hence K2 ∈# P1'
        using ‹K2 ∈# P2›
        unfolding superpositionI by simp
      hence ?I' ⊫ P1'
        using ‹?I' ⊫l K2› by blast
      thus ?thesis
        unfolding superpositionI by simp
    qed
  qed
qed

lemma (in ground-superposition-calculus) soundness-ground-eq-resolution:
  assumes step: eq-resolution P C
  shows G-entails {P} {C}
  using step
proof (cases P C rule: eq-resolution.cases)
  case (eq-resolutionI L D' t)
  show ?thesis
    unfolding G-entails-def true-clss-singleton
  proof (intro allI impI)
    fix I :: 'f gterm rel
    assume refl I and (λ(t1, t2). Upair t1 t2) ` I ⊫ P
    then obtain K where K ∈# P and (λ(t1, t2). Upair t1 t2) ` I ⊫l K
      by (auto simp: true-cls-def)
    hence K ≠ L
      by (metis ‹refl I› eq-resolutionI(2) pair-imageI reflD true-lit-simps(2))
    hence K ∈# C
      using ‹K ∈# P› ‹P = add-mset L D'› ‹C = D'› by simp
    thus (λ(t1, t2). Upair t1 t2) ` I ⊫ C
      using ‹(λ(t1, t2). Upair t1 t2) ` I ⊫l K› by blast
  qed
qed

lemma (in ground-superposition-calculus) soundness-ground-eq-factoring:
  assumes step: eq-factoring P C
  shows G-entails {P} {C}
  using step
```

```
proof (cases P C rule: eq-factoring.cases)
  case (eq-factoringI L₁ L₂ P' t t' t'')
  show ?thesis
    unfolding G-entails-def true-clss-singleton
  proof (intro allI impI)
    fix I :: 'f gterm rel
    let ?I' = (λ(t₁, t). Upair t₁ t) ` I
    assume trans I and sym I and ?I' ⊨ P
    then obtain K :: 'f gatom literal where
      K ∈# P and ?I' ⊨l K
      by (auto simp: true-cls-def)

    show ?I' ⊨ C
    proof (cases K = L₁ ∨ K = L₂)
      case True
      hence I ⊨l Pos (t, t') ∨ I ⊨l Pos (t, t'')
        unfolding eq-factoringI
        using ‹?I' ⊨l K› true-lit-uprod-iff-true-lit-prod[OF ‹sym I›] by metis
      hence I ⊨l Pos (t, t'') ∨ I ⊨l Neg (t', t'')
      proof (elim disjE)
        assume I ⊨l Pos (t, t')
        then show ?thesis
          unfolding true-lit-simps
          by (metis ‹trans I› transD)
      next
        assume I ⊨l Pos (t, t'')
        then show ?thesis
          by simp
      qed
      hence ?I' ⊨l Pos (Upair t t'') ∨ ?I' ⊨l Neg (Upair t' t'')
        unfolding true-lit-uprod-iff-true-lit-prod[OF ‹sym I›] .
      thus ?thesis
        unfolding eq-factoringI
        by (metis true-cls-add-mset)
    next
      case False
      hence K ∈# P'
        using ‹K ∈# P›
        unfolding eq-factoringI
        by auto
      hence K ∈# C
        by (simp add: eq-factoringI(1,2,7))
      thus ?thesis
        using ‹(λ(t₁, t). Upair t₁ t) ` I ⊨l K› by blast
    qed
  qed
qed

sublocale ground-superposition-calculus ⊆ sound-inference-system where
```

*Inf* = *G-Inf* **and**
*Bot* = *G-Bot* **and**
*entails* = *G-entails*
**proof** *unfold-locales*
**show** $\bigwedge\iota.\ \iota \in$ *G-Inf* $\Longrightarrow$ *G-entails* (*set* (*prems-of* $\iota$)) {*concl-of* $\iota$}
**unfolding** *G-Inf-def*
**using** *soundness-ground-superposition*
**using** *soundness-ground-eq-resolution*
**using** *soundness-ground-eq-factoring*
**by** (*auto simp*: *G-entails-def*)
**qed**

**end**
**theory** *Ground-Superposition-Welltypedness-Preservation*
**imports**
*Ground-Superposition*
*First-Order-Clause.Ground-Typing*
**begin**

**context** *ground-superposition-calculus*
**begin**

**sublocale** *ground-typing* **where** $\mathcal{F} = \mathcal{F}$ :: ('*f*, '*ty*) *fun-types*
**by** *unfold-locales*

**context**
**fixes** $\mathcal{F}$ :: ('*f*, '*ty*) *fun-types*
**begin**

**lemma** *ground-superposition-preserves-typing*:
**assumes**
*superposition D E C*
*clause.is-welltyped D*
*clause.is-welltyped E*
**shows** *clause.is-welltyped C*
**using** *assms*
**by** (*cases rule*: *superposition.cases*) (*auto 4 3*)

**lemma** *ground-eq-resolution-preserves-typing*:
**assumes** *eq-resolution D C clause.is-welltyped D*
**shows** *clause.is-welltyped C*
**using** *assms*
**by** (*cases rule*: *eq-resolution.cases*) *auto*

**lemma** *ground-eq-factoring-preserves-typing*:
**assumes** *eq-factoring D C*
**shows** *clause.is-welltyped D* $\longleftrightarrow$ *clause.is-welltyped C*
**using** *assms*
**by** (*cases rule*: *eq-factoring.cases*) *auto*

**end**

**end**

**end**
**theory** *Superposition*
**imports**
*First-Order-Clause.Nonground-Order*
*First-Order-Clause.Nonground-Selection-Function*
*First-Order-Clause.Nonground-Typing*
*First-Order-Clause.Typed-Tiebreakers*
*First-Order-Clause.Welltyped-IMGU*

*Ground-Superposition*
**begin**

# 2 Nonground Layer

**locale** *superposition-calculus* =
*nonground-inhabited-typing* $\mathcal{F}$ +
*nonground-equality-order* $less_t$ +
*nonground-selection-function select* +
*typed-tiebreakers tiebreakers* +
*ground-critical-pair-theorem* $TYPE('f)$
**for**
*select* :: ('f, 'v :: *infinite*) *select* **and**
$less_t$ :: ('f, 'v) *term* ⇒ ('f, 'v) *term* ⇒ *bool* **and**
$\mathcal{F}$ :: ('f, 'ty) *fun-types* **and**
*tiebreakers* :: ('f, 'v) *tiebreakers* +
**assumes**
*types-ordLeq-variables*: |*UNIV* :: 'ty *set*| ≤o |*UNIV* :: 'v *set*|
**begin**

**interpretation** *term-order-notation*.

**inductive** *eq-resolution* :: ('f, 'v, 'ty) *typed-clause* ⇒ ('f, 'v, 'ty) *typed-clause* ⇒ *bool* **where**
*eq-resolutionI*:
$D$ = *add-mset* $l$ $D'$ ⟹
$l = t$ !≈ $t'$ ⟹
*welltyped-imgu-on* (*clause.vars* $D$) $\mathcal{V}$ $t$ $t'$ $\mu$ ⟹
*select* $D$ = {#} ∧ *is-maximal* ($l$ ·*l* $\mu$) ($D \cdot \mu$) ∨ *is-maximal* ($l$ ·*l* $\mu$) (*select* $D \cdot \mu$) ⟹
$C = D' \cdot \mu$ ⟹
*eq-resolution* ($D$, $\mathcal{V}$) ($C$, $\mathcal{V}$)

**inductive** *eq-factoring* :: ('f, 'v, 'ty) *typed-clause* ⇒ ('f, 'v, 'ty) *typed-clause* ⇒ *bool* **where**

$eq\text{-}factoringI$:
$D = add\text{-}mset\ l_1\ (add\text{-}mset\ l_2\ D') \Longrightarrow$
$l_1 = t_1 \approx t_1{}' \Longrightarrow$
$l_2 = t_2 \approx t_2{}' \Longrightarrow$
$select\ D = \{\#\} \Longrightarrow$
$is\text{-}maximal\ (l_1 \cdot l\ \mu)\ (D \cdot \mu) \Longrightarrow$
$\neg\ (t_1 \cdot t\ \mu \preceq_t t_1{}' \cdot t\ \mu) \Longrightarrow$
$welltyped\text{-}imgu\text{-}on\ (clause.vars\ D)\ \mathcal{V}\ t_1\ t_2\ \mu \Longrightarrow$
$C = add\text{-}mset\ (t_1 \approx t_2{}')\ (add\text{-}mset\ (t_1{}'\ !\!\approx t_2{}')\ D') \cdot \mu \Longrightarrow$
$eq\text{-}factoring\ (D, \mathcal{V})\ (C, \mathcal{V})$

**inductive** *superposition* ::
$('f, 'v, 'ty)\ typed\text{-}clause \Rightarrow ('f, 'v, 'ty)\ typed\text{-}clause \Rightarrow ('f, 'v, 'ty)\ typed\text{-}clause \Rightarrow bool$
**where**
$superpositionI$:
$infinite\text{-}variables\text{-}per\text{-}type\ \mathcal{V}_1 \Longrightarrow$
$infinite\text{-}variables\text{-}per\text{-}type\ \mathcal{V}_2 \Longrightarrow$
$term\text{-}subst.is\text{-}renaming\ \varrho_1 \Longrightarrow$
$term\text{-}subst.is\text{-}renaming\ \varrho_2 \Longrightarrow$
$clause.vars\ (E \cdot \varrho_1) \cap clause.vars\ (D \cdot \varrho_2) = \{\} \Longrightarrow$
$E = add\text{-}mset\ l_1\ E' \Longrightarrow$
$D = add\text{-}mset\ l_2\ D' \Longrightarrow$
$\mathcal{P} \in \{Pos, Neg\} \Longrightarrow$
$l_1 = \mathcal{P}\ (Upair\ c_1\langle t_1\rangle\ t_1{}') \Longrightarrow$
$l_2 = t_2 \approx t_2{}' \Longrightarrow$
$\neg\ is\text{-}Var\ t_1 \Longrightarrow$
$welltyped\text{-}imgu\text{-}on\ (clause.vars\ (E \cdot \varrho_1) \cup clause.vars\ (D \cdot \varrho_2))\ \mathcal{V}_3\ (t_1 \cdot t\ \varrho_1)\ (t_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in clause.vars\ E.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (term.rename\ \varrho_1\ x) \Longrightarrow$
$\forall x \in clause.vars\ D.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (term.rename\ \varrho_2\ x) \Longrightarrow$
$term.subst.is\text{-}welltyped\text{-}on\ (clause.vars\ E)\ \mathcal{V}_1\ \varrho_1 \Longrightarrow$
$term.subst.is\text{-}welltyped\text{-}on\ (clause.vars\ D)\ \mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge \tau\ \tau'.\ typed\ \mathcal{V}_2\ t_2\ \tau \Longrightarrow typed\ \mathcal{V}_2\ t_2{}'\ \tau' \Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$(\mathcal{P} = Pos \Longrightarrow select\ E = \{\#\}) \Longrightarrow$
$(\mathcal{P} = Pos \Longrightarrow is\text{-}strictly\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)) \Longrightarrow$
$(\mathcal{P} = Neg \Longrightarrow select\ E = \{\#\} \Longrightarrow is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)) \Longrightarrow$
$(\mathcal{P} = Neg \Longrightarrow select\ E \neq \{\#\} \Longrightarrow is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ ((select\ E) \cdot \varrho_1 \odot \mu)) \Longrightarrow$
$select\ D = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\ (l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$\neg\ (c_1\langle t_1\rangle \cdot t\ \varrho_1 \odot \mu \preceq_t t_1{}' \cdot t\ \varrho_1 \odot \mu) \Longrightarrow$
$\neg\ (t_2 \cdot t\ \varrho_2 \odot \mu \preceq_t t_2{}' \cdot t\ \varrho_2 \odot \mu) \Longrightarrow$
$C = add\text{-}mset\ (\mathcal{P}\ (Upair\ (c_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle\ (t_1{}' \cdot t\ \varrho_1)))\ (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$
$superposition\ (D, \mathcal{V}_2)\ (E, \mathcal{V}_1)\ (C, \mathcal{V}_3)$

**abbreviation** *eq-factoring-inferences* **where**

*eq-factoring-inferences* ≡ { *Infer* [*D*] *C* | *D C*. *eq-factoring D C* }

**abbreviation** *eq-resolution-inferences* **where**
*eq-resolution-inferences* ≡ { *Infer* [*D*] *C* | *D C*. *eq-resolution D C* }

**abbreviation** *superposition-inferences* **where**
*superposition-inferences* ≡ { *Infer* [*D*, *E*] *C* | *D E C*. *superposition D E C*}

**definition** *inferences* :: ('*f*, '*v*, '*ty*) *typed-clause inference set* **where**
*inferences* ≡ *superposition-inferences* ∪ *eq-resolution-inferences* ∪ *eq-factoring-inferences*

**abbreviation** $bottom_F$ :: ('*f*, '*v*, '*ty*) *typed-clause set* ($\bot_F$) **where**
$bottom_F \equiv \{(\{\#\}, \mathcal{V}) \mid \mathcal{V}.\ \textit{infinite-variables-per-type}\ \mathcal{V}\ \}$

### 2.0.1 Alternative Specification of the Superposition Rule

**inductive** *superposition'* ::
('*f*, '*v*, '*ty*) *typed-clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ *bool*
**where**
*superposition'I*:
$\textit{infinite-variables-per-type}\ \mathcal{V}_1 \Longrightarrow$
$\textit{infinite-variables-per-type}\ \mathcal{V}_2 \Longrightarrow$
$\textit{term-subst.is-renaming}\ \varrho_1 \Longrightarrow$
$\textit{term-subst.is-renaming}\ \varrho_2 \Longrightarrow$
$\textit{clause.vars}\ (E \cdot \varrho_1) \cap \textit{clause.vars}\ (D \cdot \varrho_2) = \{\} \Longrightarrow$
$E = \textit{add-mset}\ l_1\ E' \Longrightarrow$
$D = \textit{add-mset}\ l_2\ D' \Longrightarrow$
$\mathcal{P} \in \{Pos,\ Neg\} \Longrightarrow$
$l_1 = \mathcal{P}\ (\textit{Upair}\ c_1\langle t_1\rangle\ t_1{}') \Longrightarrow$
$l_2 = t_2 \approx t_2{}' \Longrightarrow$
$\neg\ \textit{is-Var}\ t_1 \Longrightarrow$
$\textit{welltyped-imgu-on}\ (\textit{clause.vars}\ (E \cdot \varrho_1) \cup \textit{clause.vars}\ (D \cdot \varrho_2))\ \mathcal{V}_3\ (t_1 \cdot t\ \varrho_1)\ (t_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in \textit{clause.vars}\ E.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (\textit{term.rename}\ \varrho_1\ x) \Longrightarrow$
$\forall x \in \textit{clause.vars}\ D.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (\textit{term.rename}\ \varrho_2\ x) \Longrightarrow$
$\textit{term.subst.is-welltyped-on}\ (\textit{clause.vars}\ E)\ \mathcal{V}_1\ \varrho_1 \Longrightarrow$
$\textit{term.subst.is-welltyped-on}\ (\textit{clause.vars}\ D)\ \mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge \tau\ \tau'.\ \textit{typed}\ \mathcal{V}_2\ t_2\ \tau \Longrightarrow \textit{typed}\ \mathcal{V}_2\ t_2{}'\ \tau' \Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$(\mathcal{P} = Pos \wedge \textit{select}\ E = \{\#\} \wedge \textit{is-strictly-maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu) \vee$
$\mathcal{P} = Neg \wedge (\textit{select}\ E = \{\#\} \wedge \textit{is-maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu) \vee$
$\textit{is-maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ ((\textit{select}\ E) \cdot \varrho_1 \odot \mu))) \Longrightarrow$
$\textit{select}\ D = \{\#\} \Longrightarrow$
$\textit{is-strictly-maximal}\ (l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$\neg\ (c_1\langle t_1\rangle \cdot t\ \varrho_1 \odot \mu \preceq_t t_1{}' \cdot t\ \varrho_1 \odot \mu) \Longrightarrow$
$\neg\ (t_2 \cdot t\ \varrho_2 \odot \mu \preceq_t t_2{}' \cdot t\ \varrho_2 \odot \mu) \Longrightarrow$
$C = \textit{add-mset}\ (\mathcal{P}\ (\textit{Upair}\ (c_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle\ (t_1{}' \cdot t\ \varrho_1)))\ (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$

*superposition'* $(D, \mathcal{V}_2)$ $(E, \mathcal{V}_1)$ $(C, \mathcal{V}_3)$

**lemma** *superposition-eq-superposition'*: *superposition* = *superposition'*
**proof** (*intro ext iffI*)
**fix** $D\ E\ C$
**assume** *superposition* $D\ E\ C$
**then show** *superposition'* $D\ E\ C$

**proof** (*cases* $D\ E\ C$ *rule*: *superposition.cases*)
**case** (*superpositionI* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ \mathcal{P}\ c_1\ t_1\ t_1'\ t_2\ t_2'\ \mathcal{V}_3\ \mu\ C$)

**show** *?thesis*
**proof** (*unfold superpositionI*(*1−3*), *rule superposition'I*[*of* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2$]; (*rule superpositionI*)*?*)

**show** $\mathcal{P} = Pos \land select\ E = \{\#\} \land is\text{-}strictly\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu) \lor$
$\mathcal{P} = Neg \land (select\ E = \{\#\} \land is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu) \lor$
$is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (select\ E \cdot \varrho_1 \odot \mu))$
**using** *superpositionI*(*11*,*22−25*)
**by** *fastforce*
**qed**
**qed**
**next**
**fix** $D\ E\ C$
**assume** *superposition'* $D\ E\ C$
**then show** *superposition* $D\ E\ C$
**proof** (*cases* $D\ E\ C$ *rule*: *superposition'.cases*)
**case** (*superposition'I* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ \mathcal{P}\ c_1\ t_1\ t_1'\ t_2\ t_2'\ \mathcal{V}_3\ \mu\ C$)

**show** *?thesis*
**proof** (*unfold superposition'I*(*1−3*), *rule superpositionI*[*of* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2$]; (*rule superposition'I*)*?*)

**show**
$\mathcal{P} = Pos \Longrightarrow select\ E = \{\#\}$
$\mathcal{P} = Pos \Longrightarrow is\text{-}strictly\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)$
$\mathcal{P} = Neg \Longrightarrow select\ E = \{\#\} \Longrightarrow is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)$
$\mathcal{P} = Neg \Longrightarrow select\ E \neq \{\#\} \Longrightarrow is\text{-}maximal\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (select\ E \cdot \varrho_1 \odot \mu)$
**using** *superposition'I*(*22*) *is-maximal-not-empty*
**by** *auto*
**qed**
**qed**
**qed**

**inductive** *pos-superposition* ::
('*f*, '*v*, '*ty*) *typed-clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ *bool*

**where**
*pos-superpositionI*:
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_1 \Longrightarrow$
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_2 \Longrightarrow$
$\mathit{term\text{-}subst.is\text{-}renaming}\ \varrho_1 \Longrightarrow$
$\mathit{term\text{-}subst.is\text{-}renaming}\ \varrho_2 \Longrightarrow$
$\mathit{clause.vars}\ (E \cdot \varrho_1) \cap \mathit{clause.vars}\ (D \cdot \varrho_2) = \{\} \Longrightarrow$
$E = \mathit{add\text{-}mset}\ l_1\ E' \Longrightarrow$
$D = \mathit{add\text{-}mset}\ l_2\ D' \Longrightarrow$
$l_1 = c_1\langle t_1\rangle \approx t_1' \Longrightarrow$
$l_2 = t_2 \approx t_2' \Longrightarrow$
$\neg\ \mathit{is\text{-}Var}\ t_1 \Longrightarrow$
$\mathit{welltyped\text{-}imgu\text{-}on}\ (\mathit{clause.vars}\ (E \cdot \varrho_1) \cup \mathit{clause.vars}\ (D \cdot \varrho_2))\ \mathcal{V}_3\ (t_1 \cdot t\ \varrho_1)\ (t_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in \mathit{clause.vars}\ E.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (\mathit{term.rename}\ \varrho_1\ x) \Longrightarrow$
$\forall x \in \mathit{clause.vars}\ D.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (\mathit{term.rename}\ \varrho_2\ x) \Longrightarrow$
$\mathit{term.subst.is\text{-}welltyped\text{-}on}\ (\mathit{clause.vars}\ E)\ \mathcal{V}_1\ \varrho_1 \Longrightarrow$
$\mathit{term.subst.is\text{-}welltyped\text{-}on}\ (\mathit{clause.vars}\ D)\ \mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge \tau\ \tau'.\ \mathit{typed}\ \mathcal{V}_2\ t_2\ \tau \Longrightarrow \mathit{typed}\ \mathcal{V}_2\ t_2'\ \tau' \Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$\mathit{select}\ E = \{\#\} \Longrightarrow$
$\mathit{is\text{-}strictly\text{-}maximal}\ (l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu) \Longrightarrow$
$\mathit{select}\ D = \{\#\} \Longrightarrow$
$\mathit{is\text{-}strictly\text{-}maximal}\ (l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$\neg\ (c_1\langle t_1\rangle \cdot t\ \varrho_1 \odot \mu \preceq_t t_1' \cdot t\ \varrho_1 \odot \mu) \Longrightarrow$
$\neg\ (t_2 \cdot t\ \varrho_2 \odot \mu \preceq_t t_2' \cdot t\ \varrho_2 \odot \mu) \Longrightarrow$
$C = \mathit{add\text{-}mset}\ ((c_1 \cdot t_c\ \varrho_1)\langle t_2' \cdot t\ \varrho_2\rangle \approx (t_1' \cdot t\ \varrho_1))\ (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$
$\mathit{pos\text{-}superposition}\ (D, \mathcal{V}_2)\ (E, \mathcal{V}_1)\ (C, \mathcal{V}_3)$

**lemma** *superposition-if-pos-superposition*:
**assumes** *pos-superposition D E C*
**shows** *superposition D E C*
**using** *assms*
**proof** (*cases rule*: *pos-superposition.cases*)
**case** $(\mathit{pos\text{-}superpositionI}\ \mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ c_1\ t_1\ t_1'\ t_2\ t_2'\ \mu\ \mathcal{V}_3\ \ C)$
**then show** *?thesis*
**using** $\mathit{superpositionI}[\mathit{of}\ \mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ \mathit{Pos}\ c_1\ t_1\ t_1'\ t_2\ t_2'\ \mu\ \mathcal{V}_3\ C]$
**by** *fast*
**qed**

**inductive** *neg-superposition* ::
$('f, 'v, 'ty)\ \mathit{typed\text{-}clause} \Rightarrow ('f, 'v, 'ty)\ \mathit{typed\text{-}clause} \Rightarrow ('f, 'v, 'ty)\ \mathit{typed\text{-}clause} \Rightarrow \mathit{bool}$
**where**
*neg-superpositionI*:
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_1 \Longrightarrow$
$\mathit{infinite\text{-}variables\text{-}per\text{-}type}\ \mathcal{V}_2 \Longrightarrow$
$\mathit{term\text{-}subst.is\text{-}renaming}\ \varrho_1 \Longrightarrow$

*term-subst.is-renaming* $\varrho_2 \Longrightarrow$
*clause.vars* $(E \cdot \varrho_1) \cap$ *clause.vars* $(D \cdot \varrho_2) = \{\} \Longrightarrow$
$E =$ *add-mset* $l_1\ E' \Longrightarrow$
$D =$ *add-mset* $l_2\ D' \Longrightarrow$
$l_1 = c_1\langle t_1\rangle$ !$\approx t_1{}' \Longrightarrow$
$l_2 = t_2 \approx t_2{}' \Longrightarrow$
$\neg$ *is-Var* $t_1 \Longrightarrow$
*welltyped-imgu-on* (*clause.vars* $(E \cdot \varrho_1) \cup$ *clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3\ (t_1 \cdot t\ \varrho_1)\ (t_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in$ *clause.vars* $E$. $\mathcal{V}_1\ x = \mathcal{V}_3$ (*term.rename* $\varrho_1\ x$) $\Longrightarrow$
$\forall x \in$ *clause.vars* $D$. $\mathcal{V}_2\ x = \mathcal{V}_3$ (*term.rename* $\varrho_2\ x$) $\Longrightarrow$
*term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1\ \varrho_1 \Longrightarrow$
*term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge\tau\ \tau'.$ *typed* $\mathcal{V}_2\ t_2\ \tau \Longrightarrow$ *typed* $\mathcal{V}_2\ t_2{}'\ \tau' \Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (E \cdot \varrho_1 \odot \mu \preceq_c D \cdot \varrho_2 \odot \mu) \Longrightarrow$
(*select* $E = \{\#\} \Longrightarrow$ *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \mu)\ (E \cdot \varrho_1 \odot \mu)) \Longrightarrow$
(*select* $E \neq \{\#\} \Longrightarrow$ *is-maximal* $(l_1 \cdot l\ \varrho_1 \odot \mu)$ ((*select* $E) \cdot \varrho_1 \odot \mu)) \Longrightarrow$
*select* $D = \{\#\} \Longrightarrow$
*is-strictly-maximal* $(l_2 \cdot l\ \varrho_2 \odot \mu)\ (D \cdot \varrho_2 \odot \mu) \Longrightarrow$
$\neg\ (c_1\langle t_1\rangle \cdot t\ \varrho_1 \odot \mu \preceq_t t_1{}' \cdot t\ \varrho_1 \odot \mu) \Longrightarrow$
$\neg\ (t_2 \cdot t\ \varrho_2 \odot \mu \preceq_t t_2{}' \cdot t\ \varrho_2 \odot \mu) \Longrightarrow$
$C =$ *add-mset* $((c_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle$ !$\approx (t_1{}' \cdot t\ \varrho_1))\ (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu \Longrightarrow$
*neg-superposition* $(D, \mathcal{V}_2)\ (E, \mathcal{V}_1)\ (C, \mathcal{V}_3)$

**lemma** *superposition-if-neg-superposition*:
**assumes** *neg-superposition E D C*
**shows** *superposition E D C*
**using** *assms*
**proof** (*cases E D C rule*: *neg-superposition.cases*)
**case** (*neg-superpositionI* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ c_1\ t_1\ t_1{}'\ t_2\ t_2{}'\ \mu\ \mathcal{V}_3\ C$)
**then show** *?thesis*
**using**
*superpositionI*[*of* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'$ *Neg* $c_1\ t_1\ t_1{}'\ t_2\ t_2{}'\ \mu\ \mathcal{V}_3\ C$]
*literals-distinct*(*2*)
**by** *blast*
**qed**

**lemma** *superposition-iff-pos-or-neg*:
*superposition D E C* $\longleftrightarrow$ *pos-superposition D E C* $\vee$ *neg-superposition D E C*
**proof** (*rule iffI*)
**assume** *superposition D E C*
**thus** *pos-superposition D E C* $\vee$ *neg-superposition D E C*
**proof** (*cases D E C rule*: *superposition.cases*)
**case** (*superpositionI* $\mathcal{V}_1\ \mathcal{V}_2\ \varrho_1\ \varrho_2\ E\ D\ l_1\ E'\ l_2\ D'\ \mathcal{P}\ c_1\ t_1\ t_1{}'\ t_2\ t_2{}'\ \mathcal{V}_3\ \mu\ C$)

**show** *?thesis*
**proof**(*cases* $\mathcal{P}$ = *Pos*)
**case** *True*
**then have** *pos-superposition* $(D, \mathcal{V}_2)\ (E, \mathcal{V}_1)\ (C, \mathcal{V}_3)$

```
      using
        superpositionI
        pos-superpositionI[of V1 V2 ϱ1 ϱ2 E D l1 E' l2 D' c1 t1 t1' t2 t2' V3 μ C]
      by argo

    then show ?thesis
      unfolding superpositionI(1−3)
      by simp

  next
    case False

    then have P = Neg
      using superpositionI(11)
      by blast

    then have neg-superposition (D, V2) (E, V1) (C, V3)
      using
        superpositionI
        neg-superpositionI[of V1 V2 ϱ1 ϱ2 E D l1 E' l2 D' c1 t1 t1' t2 t2' V3 μ C]
      by argo

    then show ?thesis
      unfolding superpositionI(1−3)
      by simp
  qed
 qed
next
 assume pos-superposition D E C ∨ neg-superposition D E C
 thus superposition D E C
   using superposition-if-neg-superposition superposition-if-pos-superposition
   by metis
qed

end

end
theory Grounded-Superposition
 imports
   Superposition
   Ground-Superposition

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

locale grounded-superposition-calculus =
 superposition-calculus where select = select and F = F +
```

*grounded-selection-function* **where** *select* = *select* **and** $\mathcal{F}$ = $\mathcal{F}$
**for**
*select* :: ('*f*, '*v* :: *infinite*) *select* **and**
$\mathcal{F}$ :: ('*f*, '*ty*) *fun-types*
**begin**

**sublocale** *nonground-inference***.**

**sublocale** *ground*: *ground-superposition-calculus* **where**
$less_t = (\prec_{tG})$ **and** *select* = $select_G$
**rewrites**
*multiset-extension.multiset-extension* $(\prec_{tG})$ *mset-lit* = $(\prec_{lG})$ **and**
*multiset-extension.multiset-extension* $(\prec_{lG})$ $(\lambda x.\ x)$ = $(\prec_{cG})$ **and**
$\bigwedge l\ C$. *ground.is-maximal l C* $\longleftrightarrow$ *is-maximal* (*literal.from-ground l*) (*clause.from-ground C*) **and**
$\bigwedge l\ C$. *ground.is-strictly-maximal l C* $\longleftrightarrow$
*is-strictly-maximal* (*literal.from-ground l*) (*clause.from-ground C*)
**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* ((*D*, $\mathcal{V}'$), (*C*, $\mathcal{V}$)) $\Rightarrow$
*inference.is-ground* (*Infer* [*D*] *C* $\cdot\iota\ \gamma$) $\wedge$
$\iota_G$ = *inference.to-ground* (*Infer* [*D*] *C* $\cdot\iota\ \gamma$) $\wedge$
*clause.is-welltyped* $\mathcal{V}$ *D* $\wedge$
*term.subst.is-welltyped-on* (*clause.vars C*) $\mathcal{V}\ \gamma \wedge$
*clause.is-welltyped* $\mathcal{V}$ *C* $\wedge$
$\mathcal{V} = \mathcal{V}' \wedge$
*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* ((*D*, $\mathcal{V}_2$), (*E*, $\mathcal{V}_1$), (*C*, $\mathcal{V}_3$)) $\Rightarrow$
*term-subst.is-renaming* $\varrho_1$
$\wedge$ *term-subst.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* $\cdot\iota\ \gamma$)
$\wedge\ \iota_G$ = *inference.to-ground* (*Infer* $[D \cdot \varrho_2,\ E \cdot \varrho_1]$ *C* $\cdot\iota\ \gamma$)
$\wedge$ *clause.is-welltyped* $\mathcal{V}_1$ *E*
$\wedge$ *clause.is-welltyped* $\mathcal{V}_2$ *D*
$\wedge$ *term.subst.is-welltyped-on* (*clause.vars C*) $\mathcal{V}_3\ \gamma$
$\wedge$ *clause.is-welltyped* $\mathcal{V}_3$ *C*
$\wedge$ *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* $\iota\ \iota_G\ \gamma \equiv$
(*case* $\iota$ *of*

*Infer* [$D$] $C \Rightarrow$ *is-inference-ground-instance-one-premise* $D$ $C$ $\iota_G$ $\gamma$
| *Infer* [$D$, $E$] $C \Rightarrow \exists\, \varrho_1\ \varrho_2$. *is-inference-ground-instance-two-premises* $D$ $E$ $C$ $\iota_G$ $\gamma$ $\varrho_1$ $\varrho_2$
| - $\Rightarrow$ *False*)
$\wedge$ $\iota_G \in$ *ground.G-Inf*

**definition** *inference-ground-instances* **where**
*inference-ground-instances* $\iota$ = { $\iota_G$ | $\iota_G$ $\gamma$. *is-inference-ground-instance* $\iota$ $\iota_G$ $\gamma$ }

**lemma** *is-inference-ground-instance*:
*is-inference-ground-instance* $\iota$ $\iota_G$ $\gamma \Longrightarrow \iota_G \in$ *inference-ground-instances* $\iota$
**unfolding** *inference-ground-instances-def*
**by** *blast*

**lemma** *is-inference-ground-instance-one-premise*:
**assumes** *is-inference-ground-instance-one-premise* $D$ $C$ $\iota_G$ $\gamma$ $\iota_G \in$ *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* $D$ $E$ $C$ $\iota_G$ $\gamma$ $\varrho_1$ $\varrho_2$ $\iota_G \in$ *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-welltyped-ground-instances*:
**assumes** $\iota_G \in$ *inference-ground-instances* $\iota$
**shows** *concl-of* $\iota_G \in$ *clause.welltyped-ground-instances* (*concl-of* $\iota$)
**proof**−
**obtain** *premises* $C$ $\mathcal{V}$ **where**
$\iota$: $\iota$ = *Infer premises* ($C$, $\mathcal{V}$)
**using** *Calculus.inference.exhaust*
**by** (*metis prod.collapse*)

**show** *?thesis*
**using** *assms*
**unfolding** $\iota$ *inference-ground-instances-def clause.welltyped-ground-instances-def*
**by** (*cases premises rule*: *list-4-cases*) *auto*
**qed**

**lemma** *ground-inference-red-in-welltyped-ground-instances-of-concl*:
**assumes** $\iota_G \in$ *inference-ground-instances* $\iota$
**shows** $\iota_G \in$ *ground.Red-I* (*clause.welltyped-ground-instances* (*concl-of* $\iota$))
**proof**−
**from** *assms* **have** $\iota_G \in$ *ground.G-Inf*

**unfolding** *inference-ground-instances-def*
**by** *blast*

**moreover have** *concl-of* $\iota_G$ ∈ *clause.welltyped-ground-instances* (*concl-of* $\iota$)
**using** *assms ground-inference_concl-in-welltyped-ground-instances*
**by** *auto*

**ultimately show** $\iota_G$ ∈ *ground.Red-I* (*clause.welltyped-ground-instances* (*concl-of* $\iota$))
**using** *ground.Red-I-of-Inf-to-N*
**by** *blast*
**qed**

**thm** *option.sel*

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

**show** $\bot_F \neq \{\}$
**using** *exists-infinite-variables-per-type*[*OF types-ordLeq-variables*]
**by** *blast*
**next**
**fix** *bottom*
**assume** *bottom* ∈ $\bot_F$

**then show** *clause.welltyped-ground-instances bottom* ≠ {}
**unfolding** *clause.welltyped-ground-instances-def*
**by** *auto*
**next**
**fix** *bottom*
**assume** *bottom* ∈ $\bot_F$

**then show** *clause.welltyped-ground-instances bottom* ⊆ *ground.G-Bot*
**unfolding** *clause.welltyped-ground-instances-def*
**by** *auto*
**next**
**fix** *C* :: (*'f*, *'v*, *'ty*) *typed-clause*

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

**moreover then have** *fst* $C = \{\#\}$
**unfolding** *clause.welltyped-ground-instances-def*
**by** *simp*

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

**ultimately show** $C \in \bot_F$
**unfolding** *clause.welltyped-ground-instances-def*
**by** *blast*
**next**
**fix** $\iota$ :: ('*f*, '*v*, '*ty*) *typed-clause inference*

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

**end**

**context** *superposition-calculus*
**begin**

**sublocale**
*lifting-intersection*
*inferences*
$\{\{\#\}\}$
$select_{Gs}$
*ground-superposition-calculus.G-Inf* ($\prec_{tG}$)
$\lambda$-. *ground-superposition-calculus.G-entails*
*ground-superposition-calculus.GRed-I* ($\prec_{tG}$)
$\lambda$-. *ground-superposition-calculus.GRed-F* ($\prec_{tG}$)
$\bot_F$
$\lambda$-. *clause.welltyped-ground-instances*
$\lambda select_G$. *Some* $\circ$ (*grounded-superposition-calculus.inference-ground-instances*
($\prec_t$) $select_G$ $\mathcal{F}$)
*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-superposition-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*
    *clause.welltyped-ground-instances*
    (*Some* ∘ *inference-ground-instances*)
    *typed-tiebreakers*
  **by** *unfold-locales*
**qed**

**end**

**end**
**theory** *Superposition-Welltypedness-Preservation*
  **imports** *Superposition*
**begin**

**context** *superposition-calculus*
**begin**

**lemma** *eq-resolution-preserves-typing*:
  **assumes** *eq-resolution* ($D$, $\mathcal{V}$) ($C$, $\mathcal{V}$)
  **shows** *clause.is-welltyped* $\mathcal{V}$ $D \longleftrightarrow$ *clause.is-welltyped* $\mathcal{V}$ $C$
  **using** *assms*
  **by**(*cases* ($D$, $\mathcal{V}$) ($C$, $\mathcal{V}$) *rule*: *eq-resolution.cases*) *auto*

**lemma** *eq-factoring-preserves-typing*:
  **assumes** *eq-factoring* ($D$, $\mathcal{V}$) ($C$, $\mathcal{V}$)
  **shows** *clause.is-welltyped* $\mathcal{V}$ $D \longleftrightarrow$ *clause.is-welltyped* $\mathcal{V}$ $C$
  **using** *assms*
  **by** (*cases* ($D$, $\mathcal{V}$) ($C$, $\mathcal{V}$) *rule*: *eq-factoring.cases*) *fastforce*

**lemma** *superposition-preserves-typing-C*:
  **assumes**
    *superposition*: *superposition* ($D$, $\mathcal{V}_2$) ($E$, $\mathcal{V}_1$) ($C$, $\mathcal{V}_3$) **and**
    *D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_2$ $D$ **and**
    *E-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_1$ $E$
  **shows** *clause.is-welltyped* $\mathcal{V}_3$ $C$

```
  using superposition
proof (cases (D, 𝒱₂) (E, 𝒱₁) (C, 𝒱₃) rule: superposition.cases)
  case (superpositionI ϱ₁ ϱ₂ l₁ E' l₂ D' 𝒫 c₁ t₁ t₁' t₂ t₂' μ)

  then have welltyped-μ:
    term.subst.is-welltyped-on (clause.vars (E · ϱ₁) ∪ clause.vars (D · ϱ₂)) 𝒱₃ μ
    by meson

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

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

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

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

  have imgu: t₁ ·t ϱ₁ ⊙ μ = t₂ ·t ϱ₂ ⊙ μ
    using superpositionI(12) term.is-imgu-unifies-pair
    by auto

  from literal-cases[OF superpositionI(8)] Eμ-is-welltyped Dμ-is-welltyped imgu
  show ?thesis
    unfolding superpositionI
    by cases auto
qed

lemma superposition-preserves-typing-D:
  assumes
    superposition: superposition (D, 𝒱₂) (E, 𝒱₁) (C, 𝒱₃) and
    C-is-welltyped: clause.is-welltyped 𝒱₃ C
  shows clause.is-welltyped 𝒱₂ D
  using superposition
proof (cases (D, 𝒱₂) (E, 𝒱₁) (C, 𝒱₃) rule: superposition.cases)
  case (superpositionI ϱ₁ ϱ₂ l₁ E' l₂ D' 𝒫 c₁ t₁ t₁' t₂ t₂' μ)

  have μ-is-welltyped:
    term.subst.is-welltyped-on (clause.vars (E · ϱ₁) ∪ clause.vars (D · ϱ₂)) 𝒱₃ μ
    using superpositionI(12)
    by blast
```

**show** *?thesis*
**proof**−

**have** *clause.is-welltyped* $\mathcal{V}_2$ $D'$
**proof**−
**have** *clause.is-welltyped* $\mathcal{V}_3$ $(D' \cdot \varrho_2)$
**using** *C-is-welltyped* $\mu$*-is-welltyped*
**unfolding** *superpositionI*
**by** *auto*

**moreover have** $\forall\, x \in$*clause.vars* $D'$. $\mathcal{V}_2\ x = \mathcal{V}_3$ (*clause.rename* $\varrho_2$ $x$)
**using** *superpositionI(14)*
**unfolding** *superpositionI*
**by** *simp*

**ultimately show** *?thesis*
**using** *clause.is-welltyped.typed-renaming*[*OF superpositionI(4)*]
**unfolding** *superpositionI*
**by** *blast*
**qed**

**moreover have** *literal.is-welltyped* $\mathcal{V}_2$ $l_2$
**proof**−

**have** $\mathcal{V}_2$-$\mathcal{V}_3$: $\forall\, x \in$ *literal.vars* $l_2$. $\mathcal{V}_2\ x = \mathcal{V}_3$ (*clause.rename* $\varrho_2$ $x$)
**using** *superpositionI(14)*
**unfolding** *superpositionI*
**by** *auto*

**have** *literal.is-welltyped* $\mathcal{V}_3$ $(l_2 \cdot l\ \varrho_2)$
**proof**−
**obtain** $\tau$ **where** $\tau$: *welltyped* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)\ \tau$
**using** *superpositionI(12)*
**by** *force*

**moreover obtain** $\tau'$ **where** $\tau'$: *welltyped* $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2)\ \tau'$
**proof**−
**have** $\mu$*-is-welltyped*: *term.subst.is-welltyped-on* (*term.vars* $((c_1 \cdot t_c\ \varrho_1)\langle t_2{}'$
$\cdot t\ \varrho_2\rangle))$ $\mathcal{V}_3$ $\mu$
**using** $\mu$*-is-welltyped superpositionI(8)*
**unfolding** *superpositionI*
**by** *auto*

**have** *term.is-welltyped* $\mathcal{V}_3$ $((c_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle \cdot t\ \mu)$
**using** *C-is-welltyped superpositionI(8)*
**unfolding** *superpositionI*
**by** *auto*

```
        then show ?thesis
          unfolding term.welltyped.explicit-subst-stability[OF μ-is-welltyped]
          using that term.welltyped.subterm
          by meson
      qed

      moreover have τ = τ'
      proof−
        have welltyped 𝒱₂ t₂ τ welltyped 𝒱₂ t₂' τ'
          using
            τ τ'
            superpositionI(12, 14)
            term.welltyped.explicit-typed-renaming[OF superpositionI(4)]
          unfolding superpositionI
          by(auto simp: Set.ball-Un)

        then show ?thesis
          using superpositionI(17)
          by (simp add: term.typed-if-welltyped)
      qed

      ultimately show ?thesis
        unfolding superpositionI
        by auto
    qed

    then show ?thesis
      using literal.is-welltyped.typed-renaming[OF superpositionI(4) 𝒱₂-𝒱₃]
      unfolding superpositionI
      by simp
  qed

  ultimately show ?thesis
    unfolding superpositionI
    by simp
 qed
qed

lemma superposition-preserves-typing-E:
 assumes
  superposition: superposition (D, 𝒱₂) (E, 𝒱₁) (C, 𝒱₃) and
  C-is-welltyped: clause.is-welltyped 𝒱₃ C
 shows clause.is-welltyped 𝒱₁ E
 using superposition
proof (cases (D, 𝒱₂) (E, 𝒱₁) (C, 𝒱₃) rule: superposition.cases)
 case (superpositionI ϱ₁ ϱ₂ l₁ E' l₂ D' 𝒫 c₁ t₁ t₁' t₂ t₂' μ)

 have [simp]: ⋀a σ. 𝒫 a ·l σ = 𝒫 (a ·a σ)
  using superpositionI(8)
```

**by** *auto*

**have** [*simp*]: $\bigwedge\mathcal{V}$ *a. literal.is-welltyped* $\mathcal{V}$ ($\mathcal{P}$ *a*) $\longleftrightarrow$ *atom.is-welltyped* $\mathcal{V}$ *a*
**using** *superpositionI*(*8*)
**by** (*auto simp*: *literal-is-welltyped-iff-atm-of*)

**have** [*simp*]: $\bigwedge$*a. literal.vars* ($\mathcal{P}$ *a*) = *atom.vars a*
**using** *superpositionI*(*8*)
**by** *auto*

**have** $\mu$*-is-welltyped*:
*term.subst.is-welltyped-on* (*clause.vars* ($E \cdot \varrho_1$) ∪ *clause.vars* ($D \cdot \varrho_2$)) $\mathcal{V}_3$ $\mu$
**using** *superpositionI*(*12*)
**by** *blast*

**show** *?thesis*
**proof**−
**have** *clause.is-welltyped* $\mathcal{V}_1$ $E'$
**proof**−
**have** *clause.is-welltyped* $\mathcal{V}_3$ ($E' \cdot \varrho_1$)
**using** *C-is-welltyped* $\mu$*-is-welltyped*
**unfolding** *superpositionI*
**by** *auto*

**moreover have** $\forall$ *x*∈*clause.vars* $E'$. $\mathcal{V}_1$ *x* = $\mathcal{V}_3$ (*clause.rename* $\varrho_1$ *x*)
**using** *superpositionI*(*13*)
**unfolding** *superpositionI*
**by** *simp*

**ultimately show** *?thesis*
**using** *clause.is-welltyped.typed-renaming*[*OF superpositionI*(*3*)]
**unfolding** *superpositionI*
**by** *blast*
**qed**

**moreover have** *literal.is-welltyped* $\mathcal{V}_1$ $l_1$
**proof**−

**have** $\mathcal{V}_1$-$\mathcal{V}_3$: $\forall$ *x* ∈ *literal.vars* $l_1$. $\mathcal{V}_1$ *x* = $\mathcal{V}_3$ (*clause.rename* $\varrho_1$ *x*)
**using** *superpositionI*(*13*)
**unfolding** *superpositionI*
**by** *auto*

**have** *literal.is-welltyped* $\mathcal{V}_3$ ($\mathcal{P}$ (*Upair* ($c_1 \cdot t_c\ \varrho_1$)$\langle t_1 \cdot t\ \varrho_1\rangle$ ($t_1{}' \cdot t\ \varrho_1$)))
**proof**−

**have** $\mu$*-is-welltyped*:
*term.subst.is-welltyped-on*
(*clause.vars* (*add-mset* ($\mathcal{P}$ (*Upair* ($c_1 \cdot t_c\ \varrho_1$)$\langle t_2{}' \cdot t\ \varrho_2\rangle$ ($t_1{}' \cdot t\ \varrho_1$))) ($E' \cdot$

$\varrho_1 + D' \cdot \varrho_2)))$
$\mathcal{V}_3\ \mu$
**using** *μ-is-welltyped*
**unfolding** *superpositionI*
**by** *auto*

**have** *atom.is-welltyped* $\mathcal{V}_3$ (*Upair* $(t_2{}' \cdot t\ \varrho_2)\ (t_1\ \cdot t\ \varrho_1)$)
**using**
*superpositionI(12)*
*superposition-preserves-typing-D*[*OF superposition C-is-welltyped*]
*clause.is-welltyped.typed-renaming*[*OF superpositionI(4) superpositionI(14)*]
**unfolding** *superpositionI*
**by** *auto*

**moreover have** *literal.is-welltyped* $\mathcal{V}_3$ ($\mathcal{P}$ (*Upair* $(c_1\ \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle\ (t_1{}'\ \cdot t\ \varrho_1)))$
**using** *C-is-welltyped*
**unfolding** *superpositionI clause.is-welltyped.subst-stability*[*OF μ-is-welltyped*]
**by** *simp*

**ultimately show** *?thesis*
**by** *auto*
**qed**

**then show** *?thesis*
**using** *literal.is-welltyped.typed-renaming*[*OF superpositionI(3)* $\mathcal{V}_1$-$\mathcal{V}_3$]
**unfolding** *superpositionI*
**by** *simp*
**qed**

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

**lemma** *superposition-preserves-typing*:
**assumes** *superposition* $(D, \mathcal{V}_2)\ (E, \mathcal{V}_1)\ (C, \mathcal{V}_3)$
**shows** *clause.is-welltyped* $\mathcal{V}_2\ D \wedge$ *clause.is-welltyped* $\mathcal{V}_1\ E \longleftrightarrow$ *clause.is-welltyped* $\mathcal{V}_3\ C$
**using**
*superposition-preserves-typing-C*
*superposition-preserves-typing-D*
*superposition-preserves-typing-E*
*assms*
**by** *blast*

**end**

**end**
**theory** *Superposition-Completeness*
  **imports**
    *Grounded-Superposition*
    *Ground-Superposition-Completeness*
    *Superposition-Welltypedness-Preservation*

    *First-Order-Clause.Nonground-Entailment*
**begin**

# 3 Completeness

**context** *grounded-superposition-calculus*
**begin**

## 3.1 Liftings

**lemma** *eq-resolution-lifting*:
  **fixes**
    $D_G$ $C_G$ :: *'f ground-atom clause* **and**
    $D$ $C$ :: (*'f*, *'v*) *atom clause* **and**
    $\gamma$ :: (*'f*, *'v*) *subst*
  **defines**
    [*simp*]: $D_G \equiv$ *clause.to-ground* $(D \cdot \gamma)$ **and**
    [*simp*]: $C_G \equiv$ *clause.to-ground* $(C \cdot \gamma)$
  **assumes**
    *ground-eq-resolution*: *ground.eq-resolution* $D_G$ $C_G$ **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**
    *D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}$ $D$ **and**
    $\gamma$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}$ $\gamma$ **and**
    $\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$
  **obtains** $C'$
  **where**
    *eq-resolution* $(D, \mathcal{V})$ $(C', \mathcal{V})$
    *Infer* $[D_G]$ $C_G \in$ *inference-ground-instances* (*Infer* $[(D, \mathcal{V})]$ $(C', \mathcal{V})$)
    $C' \cdot \gamma = C \cdot \gamma$
  **using** *ground-eq-resolution*
**proof**(*cases* $D_G$ $C_G$ *rule*: *ground.eq-resolution.cases*)
  **case** *ground-eq-resolutionI*: (*eq-resolutionI* $l_G$ $D_G{}'$ $t_G$)

  **let** *?select*$_G$*-empty* = *select*$_G$ $D_G$ = {#}
  **let** *?select*$_G$*-not-empty* = *select*$_G$ $D_G \neq$ {#}

  **obtain** $l$ **where**
    $l$-$\gamma$: $l \cdot l\ \gamma$ = *term.from-ground* $t_G$ !≈ *term.from-ground* $t_G$ **and**
    *l-in-D*: $l \in\#\ D$ **and**

*l-selected*: *?select*$_G$*-not-empty* ⟹ *is-maximal l* (*select D*) **and**
*l-γ-selected*: *?select*$_G$*-not-empty* ⟹ *is-maximal* (*l* ·*l* γ) (*select D* · γ) **and**
*l-is-maximal*: *?select*$_G$*-empty* ⟹ *is-maximal l D* **and**
*l-γ-is-maximal*: *?select*$_G$*-empty* ⟹ *is-maximal* (*l* ·*l* γ) (*D* · γ)
**proof**−
**obtain** *max-l* **where**
*is-maximal max-l D* **and**
*is-max-in-D-γ*: *is-maximal* (*max-l* ·*l* γ) (*D* · γ)
**proof**−
**have** *D* ≠ {#}
**using** *ground-eq-resolutionI*(*1*)
**by** *auto*

**then show** *?thesis*
**using** *that D-grounding obtain-maximal-literal*
**by** *blast*
**qed**

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

**moreover have** *max-l* ·*l* γ = *term.from-ground* $t_G$ !≈ *term.from-ground* $t_G$ **if** *?select*$_G$*-empty*
**proof**(*rule unique-maximal-in-ground-clause*[*OF D-grounding is-max-in-D-γ*])
**have** *ground-is-maximal* $l_G$ $D_G$
**using** *ground-eq-resolutionI*(*3*) *that*
**unfolding** *is-maximal-def*
**by** *simp*

**then show** *is-maximal* (*term.from-ground* $t_G$ !≈ *term.from-ground* $t_G$) (*D* · γ)
**using** *D-grounding*
**unfolding** *ground-eq-resolutionI*(*2*)
**by** *simp*
**qed**

**moreover obtain** *selected-l* **where**
*selected-l* ·*l* γ = *term.from-ground* $t_G$ !≈ *term.from-ground* $t_G$ **and**
*is-maximal selected-l* (*select D*)
*is-maximal* (*selected-l* ·*l* γ) (*select D* · γ)
**if** *?select*$_G$*-not-empty*
**proof**−
**have** *is-maximal* (*term.from-ground* $t_G$ !≈ *term.from-ground* $t_G$) (*select D* · γ)
**if** *?select*$_G$*-not-empty*
**using** *ground-eq-resolutionI*(*3*) *that select*
**unfolding** *ground-eq-resolutionI*(*2*)
**by** *simp*

**then show** *?thesis*
**using**
*that*
*obtain-maximal-literal*[*OF - select-ground-subst*[*OF D-grounding*]]
*unique-maximal-in-ground-clause*[*OF select-ground-subst*[*OF D-grounding*]]
**by** (*metis is-maximal-not-empty clause.magma-subst-empty*)
**qed**

**moreover then have** *selected-l* ∈# *D* **if** *?select*$_G$*-not-empty*
**by** (*meson that maximal-in-clause mset-subset-eqD select-subset*)

**ultimately show** *?thesis*
**using** *that*
**by** *blast*
**qed**

**obtain** $C'$ **where** *D*: $D$ = *add-mset* $l$ $C'$
**using** *multi-member-split*[*OF l-in-D*]
**by** *blast*

**obtain** $t$ $t'$ **where** *l*: $l = t$ !≈ $t'$
**using** *l*-$\gamma$ *obtain-from-neg-literal-subst*
**by** *meson*

**obtain** $\mu$ $\sigma$ **where** $\gamma$: $\gamma = \mu \odot \sigma$ **and** *imgu*: *welltyped-imgu-on* (*clause.vars* $D$) $\mathcal{V}$ $t$ $t'$ $\mu$
**proof**−
**have** *unified*: $t \cdot t\ \gamma = t' \cdot t\ \gamma$
**using** *l*-$\gamma$
**unfolding** *l*
**by** *simp*

**moreover obtain** $\tau$ **where** *welltyped*: *welltyped* $\mathcal{V}$ $t$ $\tau$ *welltyped* $\mathcal{V}$ $t'$ $\tau$
**using** *D-is-welltyped*
**unfolding** *D l*
**by** *auto*

**show** *?thesis*
**using** *obtain-welltyped-imgu-on*[*OF unified welltyped*] *that*
**by** *metis*
**qed**

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

**show** *eq-resolution*: *eq-resolution* ($D$, $\mathcal{V}$) ($C' \cdot \mu$, $\mathcal{V}$)
**proof** (*rule eq-resolutionI*, *rule D*, *rule l*, *rule imgu*)
**show** *select* $D$ = {#} ∧ *is-maximal* ($l \cdot l\ \mu$) ($D \cdot \mu$) ∨ *is-maximal* ($l \cdot l\ \mu$)

$((\textit{select } D) \cdot \mu)$

      **proof**(*cases ?select*$_G$*-empty*)
        **case** *True*

        **moreover have** *is-maximal* $(l \cdot l\ \mu)$ $(D \cdot \mu)$
        **proof**−
          **have** $l \cdot l\ \mu \in\#\ D \cdot \mu$
            **using** *l-in-D*
            **by** *blast*

          **then show** *?thesis*
                **using** *l-γ-is-maximal*[*OF True*] *is-maximal-if-grounding-is-maximal*
*D-grounding*
            **unfolding** $\gamma$
            **by** *simp*
        **qed**

        **ultimately show** *?thesis*
          **using** *select*
          **by** *simp*
      **next**
        **case** *False*

        **have** $l \cdot l\ \mu \in\#\ \textit{select } D \cdot \mu$
          **using** *l-selected*[*OF False*] *maximal-in-clause*
          **by** *blast*

        **then have** *is-maximal* $(l \cdot l\ \mu)$ $(\textit{select } D \cdot \mu)$
          **using**
            *select-ground-subst*[*OF D-grounding*]
            *l-γ-selected*[*OF False*]
            *is-maximal-if-grounding-is-maximal*
          **unfolding** $\gamma$
          **by** *auto*

        **then show** *?thesis*
          **using** *select*
          **by** *blast*
      **qed**
    **qed** (*rule refl*)

    **show** *C′-μ-γ*: $C' \cdot \mu \cdot \gamma = C \cdot \gamma$
    **proof**−
      **have** *term.is-idem* $\mu$
        **using** *imgu*
        **unfolding** *term-subst.is-imgu-iff-is-idem-and-is-mgu*
        **by** *blast*

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

**unfolding** $\gamma$ *term-subst.is-idem-def subst-compose-assoc*[*symmetric*]
**by** *argo*

**have** $D \cdot \gamma = \textit{add-mset}\ (l \cdot l\ \gamma)\ (C \cdot \gamma)$
**proof**−
**have** $\textit{clause.to-ground}\ (D \cdot \gamma) = \textit{clause.to-ground}\ (\textit{add-mset}\ (l \cdot l\ \gamma)\ (C \cdot \gamma))$
**using** *ground-eq-resolutionI*(*1*)
**unfolding** *ground-eq-resolutionI*(*2*) *l*-$\gamma$ *ground-eq-resolutionI*(*4*)[*symmetric*]
**by** *simp*

**moreover have** $\textit{clause.is-ground}\ (\textit{add-mset}\ (l \cdot l\ \gamma)\ (C \cdot \gamma))$
**using** *C-grounding clause.to-set-is-ground-subst*[*OF l-in-D D-grounding*]
**by** *simp*

**ultimately show** *?thesis*
**using** *clause.to-ground-eq*[*OF D-grounding*]
**by** *blast*
**qed**

**then have** $C' \cdot \gamma = C \cdot \gamma$
**unfolding** *D*
**by** *simp*

**then show** *?thesis*
**unfolding** *clause.subst-comp-subst*[*symmetric*] $\mu$-$\gamma$**.**
**qed**

**show** *Infer* $[D_G]$ $C_G \in$ *inference-ground-instances* (*Infer* $[(D, \mathcal{V})]$ $(C' \cdot \mu, \mathcal{V})$)
**proof** (*rule is-inference-ground-instance-one-premise*)

**show** *is-inference-ground-instance-one-premise* $(D, \mathcal{V})$ $(C' \cdot \mu, \mathcal{V})$ (*Infer* $[D_G]$ $C_G$) $\gamma$
**proof**(*unfold split*, *intro conjI*; (*rule D-is-welltyped refl* $\mathcal{V}$)*?*)
**show** *inference.is-ground* (*Infer* [*D*] $(C' \cdot \mu) \cdot\iota\ \gamma$)
**using** *D-grounding C-grounding* $C'$-$\mu$-$\gamma$
**by** *auto*
**next**
**show** *Infer* $[D_G]$ $C_G =$ *inference.to-ground* (*Infer* [*D*] $(C' \cdot \mu) \cdot\iota\ \gamma$)
**using** $C'$-$\mu$-$\gamma$
**by** *simp*
**next**
**have** *clause.vars* $(C' \cdot \mu) \subseteq$ *clause.vars D*
**using** *clause.variables-in-base-imgu imgu*
**unfolding** *D l*
**by** *auto*

**then show** *term.subst.is-welltyped-on* (*clause.vars* $(C' \cdot \mu)$) $\mathcal{V}$ $\gamma$
**using** *D-is-welltyped* $\gamma$-*is-welltyped*

**by** *blast*
**next**
**show** *clause.is-welltyped* $\mathcal{V}$ ($C' \cdot \mu$)
**using** *D-is-welltyped eq-resolution eq-resolution-preserves-typing*
**by** *blast*
**qed**

**show** *Infer* [$D_G$] $C_G \in$ *ground.G-Inf*
**unfolding** *ground.G-Inf-def*
**using** *ground-eq-resolution*
**by** *blast*
**qed**
**qed**
**qed**

**lemma** *eq-factoring-lifting*:
**fixes**
$D_G$ $C_G$ :: *'f ground-atom clause* **and**
*D C* :: (*'f*, *'v*) *atom clause* **and**
$\gamma$ :: (*'f*, *'v*) *subst*
**defines**
[*simp*]: $D_G \equiv$ *clause.to-ground* ($D \cdot \gamma$) **and**
[*simp*]: $C_G \equiv$ *clause.to-ground* ($C \cdot \gamma$)
**assumes**
*ground-eq-factoring*: *ground.eq-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**
*D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}$ *D* **and**
$\gamma$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars D*) $\mathcal{V}$ $\gamma$ **and**
$\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$
**obtains** $C'$
**where**
*eq-factoring* ($D$, $\mathcal{V}$) ($C'$, $\mathcal{V}$)
*Infer* [$D_G$] $C_G \in$ *inference-ground-instances* (*Infer* [($D$, $\mathcal{V}$)] ($C'$, $\mathcal{V}$))
$C' \cdot \gamma = C \cdot \gamma$
**using** *ground-eq-factoring*
**proof**(*cases* $D_G$ $C_G$ *rule*: *ground.eq-factoring.cases*)
**case** *ground-eq-factoringI*: (*eq-factoringI* $l_{G1}$ $l_{G2}$ $D_G'$ $t_{G1}$ $t_{G2}$ $t_{G3}$)

**have** $D \neq \{\#\}$
**using** *ground-eq-factoringI*(*1*)
**by** *auto*

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

**obtain** $t_1$ $t_1{}'$ **where**
$l_1$: $l_1 = t_1 \approx t_1{}'$ **and**
$l_1$-$\gamma$: $l_1 \cdot l\ \gamma =$ *term.from-ground* $t_{G1} \approx$ *term.from-ground* $t_{G2}$ **and**
$t_1$-$\gamma$: $t_1 \cdot t\ \gamma =$ *term.from-ground* $t_{G1}$ **and**
$t_1{}'$-$\gamma$: $t_1{}' \cdot t\ \gamma =$ *term.from-ground* $t_{G2}$
**proof**−
**have** *is-maximal* (*literal.from-ground* $l_{G1}$) ($D \cdot \gamma$)
**using** *ground-eq-factoringI*(*5*) *D-grounding*
**by** *simp*

**then have** $l_1 \cdot l\ \gamma =$ *term.from-ground* $t_{G1} \approx$ *term.from-ground* $t_{G2}$
**unfolding** *ground-eq-factoringI*(*2*)
**using** *unique-maximal-in-ground-clause*[*OF D-grounding* $l_1$-$\gamma$-*is-maximal*]
**by** *simp*

**then show** *?thesis*
**using** *that*
**unfolding** *ground-eq-factoringI*(*2*)
**by** (*metis obtain-from-pos-literal-subst*)
**qed**

**obtain** $l_2$ $D'$ **where**
$l_2$-$\gamma$: $l_2 \cdot l\ \gamma =$ *term.from-ground* $t_{G1} \approx$ *term.from-ground* $t_{G3}$ **and**
$D$: $D =$ *add-mset* $l_1$ (*add-mset* $l_2$ $D'$)
**proof**−
**obtain** $D''$ **where** $D$: $D =$ *add-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-ground* (*add-mset* $l_{G1}$ (*add-mset* $l_{G2}$ $D_G{}'$))
**using** *ground-eq-factoringI*(*1*) $D_G$-*def*
**by** (*metis D-grounding clause.to-ground-inverse*)

**ultimately have** $D'' \cdot \gamma =$ *add-mset* (*literal.from-ground* $l_{G2}$) (*clause.from-ground* $D_G{}'$)
**using** $l_1$-$\gamma$
**by** (*simp add*: *ground-eq-factoringI*(*2*))

**then obtain** $l_2$ **where** $l_2 \cdot l\ \gamma =$ *term.from-ground* $t_{G1} \approx$ *term.from-ground* $t_{G3}$ $l_2 \in\#\ D''$
**unfolding** *clause.subst-def ground-eq-factoringI*
**using** *msed-map-invR*
**by** *force*

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

**qed**

**then obtain** $t_2$ $t_2'$ **where**
$l_2$: $l_2 = t_2 \approx t_2'$ **and**
$t_2$-$\gamma$: $t_2 \cdot t\ \gamma$ = *term.from-ground* $t_{G1}$ **and**
$t_2'$-$\gamma$: $t_2' \cdot t\ \gamma$ = *term.from-ground* $t_{G3}$
**unfolding** *ground-eq-factoringI*(*3*)
**using** *obtain-from-pos-literal-subst*
**by** *metis*

**have** $D'$-$\gamma$: $D' \cdot \gamma$ = *clause.from-ground* $D_G'$
**using** *D D-grounding ground-eq-factoringI*(*1*,*2*,*3*) $l_1$-$\gamma$ $l_2$-$\gamma$
**by** *force*

**obtain** $\mu$ $\sigma$ **where** $\gamma$: $\gamma = \mu \odot \sigma$ **and** *imgu*: *welltyped-imgu-on* (*clause.vars D*) $\mathcal{V}$ $t_1$ $t_2$ $\mu$
**proof**−
**have** *unified*: $t_1 \cdot t\ \gamma = t_2 \cdot t\ \gamma$
**unfolding** $t_1$-$\gamma$ $t_2$-$\gamma$ **..**

**then obtain** $\tau$ **where** *welltyped* $\mathcal{V}$ ($t_1 \cdot t\ \gamma$) $\tau$ *welltyped* $\mathcal{V}$ ($t_2 \cdot t\ \gamma$) $\tau$
**using** *D-is-welltyped* $\gamma$*-is-welltyped*
**unfolding** *D* $l_1$ $l_2$
**by** *auto*

**then have** *welltyped*: *welltyped* $\mathcal{V}$ $t_1$ $\tau$ *welltyped* $\mathcal{V}$ $t_2$ $\tau$
**using** $\gamma$*-is-welltyped*
**unfolding** *D* $l_1$ $l_2$
**by** *simp-all*

**then show** *?thesis*
**using** *obtain-welltyped-imgu-on*[*OF unified welltyped*] *that*
**by** *metis*
**qed**

**let** *?C''* = *add-mset* ($t_1 \approx t_2'$) (*add-mset* ($t_1'$ !≈ $t_2'$) $D'$)
**let** *?C'* = *?C''* $\cdot$ $\mu$

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

**show** *eq-factoring*: *eq-factoring* (*D*, $\mathcal{V}$) (*?C'*, $\mathcal{V}$)
**proof** (*rule eq-factoringI*; (*rule D* $l_1$ $l_2$ *imgu refl*)*?*)
**show** *select D* = {#}
**using** *ground-eq-factoringI*(*4*) *select*
**by** *simp*
**next**
**have** $l_1 \cdot l\ \mu$ ∈# $D \cdot \mu$
**using** $l_1$*-is-maximal clause.subst-in-to-set-subst maximal-in-clause*

**by** *blast*

**then show** *is-maximal* $(l_1 \cdot l\ \mu)$ $(D \cdot \mu)$
**using** *is-maximal-if-grounding-is-maximal D-grounding* $l_1$-$\gamma$-*is-maximal*
**unfolding** $\gamma$
**by** *auto*
**next**
**have** *groundings*: *term.is-ground* $(t_1{}' \cdot t\ \mu \cdot t\ \sigma)$ *term.is-ground* $(t_1 \cdot t\ \mu \cdot t\ \sigma)$
**using** $t_1{}'$-$\gamma$ $t_1$-$\gamma$
**unfolding** $\gamma$
**by** *simp-all*

**have** $t_1{}' \cdot t\ \gamma \prec_t t_1 \cdot t\ \gamma$
**using** *ground-eq-factoringI*(*6*)
**unfolding** $t_1{}'$-$\gamma$ $t_1$-$\gamma$ *term.order.less*$_G$-*def* **.**

**then show** $\neg\ t_1 \cdot t\ \mu \preceq_t t_1{}' \cdot t\ \mu$
**unfolding** $\gamma$
**using** *term.order.ground-less-not-less-eq*[*OF groundings*]
**by** *simp*
**qed**

**show** $C'$-$\gamma$: *?C'* $\cdot\ \gamma = C \cdot \gamma$
**proof**−
**have** *term.is-idem* $\mu$
**using** *imgu*
**unfolding** *term-subst.is-imgu-iff-is-idem-and-is-mgu*
**by** *blast*

**then have** $\mu$-$\gamma$: $\mu \odot \gamma = \gamma$
**unfolding** $\gamma$ *term-subst.is-idem-def subst-compose-assoc*[*symmetric*]
**by** *argo*

**have** $C \cdot \gamma =$ *clause.from-ground* (*add-mset* $(t_{G2}$ !≈ $t_{G3})$ (*add-mset* $(t_{G1} \approx$ $t_{G3})$ $D_G{}'$))
**using** *ground-eq-factoringI*(*7*) *clause.to-ground-eq*[*OF C-grounding clause.ground-is-ground*]
**unfolding** $C_G$-*def*
**by** (*metis clause.from-ground-inverse*)

**also have** ... = *?C''* $\cdot\ \gamma$
**using** $t_1$-$\gamma$ $t_1{}'$-$\gamma$ $t_2{}'$-$\gamma$ $D'$-$\gamma$
**by** *simp*

**also have** ... = *?C'* $\cdot\ \gamma$
**unfolding** *clause.subst-comp-subst*[*symmetric*] $\mu$-$\gamma$ **..**

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

**show** *Infer* $[D_G]$ $C_G$ ∈ *inference-ground-instances* (*Infer* [(*D*, $\mathcal{V}$)] (*?C′*, $\mathcal{V}$))
**proof** (*rule is-inference-ground-instance-one-premise*)

**show** *is-inference-ground-instance-one-premise* (*D*, $\mathcal{V}$) (*?C′*, $\mathcal{V}$) (*Infer* $[D_G]$ $C_G$) $\gamma$
**proof**(*unfold split*, *intro conjI*; (*rule D-is-welltyped refl* $\mathcal{V}$)*?*)
**show** *inference.is-ground* (*Infer* [*D*] *?C′* ·ι $\gamma$)
**using** *C-grounding D-grounding C′-*$\gamma$
**by** *auto*
**next**
**show** *Infer* $[D_G]$ $C_G$ = *inference.to-ground* (*Infer* [*D*] *?C′* ·ι $\gamma$)
**using** *C′-*$\gamma$
**by** *simp*
**next**
**have** *imgu*: *term.is-imgu* $\mu$ $\{\{t_1, t_2\}\}$
**using** *imgu*
**by** *blast*

**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** *term.subst.is-welltyped-on* (*clause.vars ?C′*) $\mathcal{V}$ $\gamma$
**using** *D-is-welltyped* $\gamma$*-is-welltyped*
**by** *blast*
**next**
**show** *clause.is-welltyped* $\mathcal{V}$ *?C′*
**using** *D-is-welltyped eq-factoring eq-factoring-preserves-typing*
**by** *blast*
**qed**

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

**lemma** *superposition-lifting*:
**fixes**
$E_G$ $D_G$ $C_G$ :: *′f ground-atom clause* **and**
*E D C* :: (*′f*, *′v*) *atom clause* **and**
$\gamma$ $\varrho_1$ $\varrho_2$ :: (*′f*, *′v*) *subst* **and**
$\mathcal{V}_1$ $\mathcal{V}_2$ :: (*′v*, *′ty*) *var-types*
**defines**
[*simp*]: $E_G$ ≡ *clause.to-ground* ($E \cdot \varrho_1 \odot \gamma$) **and**
[*simp*]: $D_G$ ≡ *clause.to-ground* ($D \cdot \varrho_2 \odot \gamma$) **and**

[*simp*]: $C_G \equiv$ *clause.to-ground* $(C \cdot \gamma)$ **and**
[*simp*]: $N_G \equiv$ *clause.welltyped-ground-instances* $(E, \mathcal{V}_1) \cup$
*clause.welltyped-ground-instances* $(D, \mathcal{V}_2)$ **and**
[*simp*]: $\iota_G \equiv$ *Infer* $[D_G, E_G]$ $C_G$
**assumes**
*ground-superposition*: *ground.superposition* $D_G$ $E_G$ $C_G$ **and**
$\varrho_1$: *term-subst.is-renaming* $\varrho_1$ **and**
$\varrho_2$: *term-subst.is-renaming* $\varrho_2$ **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**
*E-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_1$ $E$ **and**
*D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_2$ $D$ **and**
$\varrho_1$-$\gamma$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ $(\varrho_1 \odot \gamma)$ **and**
$\varrho_2$-$\gamma$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ $(\varrho_2 \odot \gamma)$ **and**
$\varrho_1$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ $\varrho_1$ **and**
$\varrho_2$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ $\varrho_2$ **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**
*not-redundant*: $\iota_G \notin$ *ground.Red-I* $N_G$
**obtains** $C'$ $\mathcal{V}_3$
**where**
*superposition* $(D, \mathcal{V}_2)$ $(E, \mathcal{V}_1)$ $(C', \mathcal{V}_3)$
$\iota_G \in$ *inference-ground-instances* (*Infer* $[(D, \mathcal{V}_2), (E, \mathcal{V}_1)]$ $(C', \mathcal{V}_3)$)
$C' \cdot \gamma = C \cdot \gamma$
**using** *ground-superposition*
**proof**(*cases* $D_G$ $E_G$ $C_G$ *rule*: *ground.superposition.cases*)
**case** *ground-superpositionI*: (*superpositionI* $l_{G1}$ $E_G'$ $l_{G2}$ $D_G'$ $\mathcal{P}_G$ $c_G$ $t_{G1}$ $t_{G2}$ $t_{G3}$)

**have** $E$-$\gamma$: $E \cdot \varrho_1 \odot \gamma =$ *clause.from-ground* (*add-mset* $l_{G1}$ $E_G'$)
**using** *ground-superpositionI*(*1*)
**unfolding** $E_G$-*def*
**by** (*metis E-grounding clause.to-ground-inverse*)

**have** $D$-$\gamma$: $D \cdot \varrho_2 \odot \gamma =$ *clause.from-ground* (*add-mset* $l_{G2}$ $D_G'$)
**using** *ground-superpositionI*(*2*) $D_G$-*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**
$l_1$-$\gamma$: $l_1 \cdot l\ \varrho_1 \odot \gamma =$ *literal.from-ground* $l_{G1}$ **and**
$l_1$-*is-strictly-maximal*: $\mathcal{P}_G = Pos \Longrightarrow$ *is-strictly-maximal* $l_1$ $E$ **and**
$l_1$-*is-maximal*: $\mathcal{P}_G = Neg \Longrightarrow$ ?*select*$_G$-*empty* $\Longrightarrow$ *is-maximal* $l_1$ $E$ **and**
$l_1$-*selected*: $\mathcal{P}_G = Neg \Longrightarrow$ ?*select*$_G$-*not-empty* $\Longrightarrow$ *is-maximal* $l_1$ (*select* $E$) **and**

$l_1$-*in-E*: $l_1 \in\# E$
**proof** −

**have** *E-not-empty*: $E \neq \{\#\}$
**using** *ground-superpositionI*(*1*)
**by** *auto*

**have** *is-strictly-maximal* (*literal.from-ground* $l_{G1}$) ($E \cdot \varrho_1 \odot \gamma$) **if** $\mathcal{P}_G = Pos$
**using** *ground-superpositionI*(*9*) *that E-grounding*
**by** *simp*

**then obtain** *positive*-$l_1$ **where**
*is-strictly-maximal positive*-$l_1$ $E$
*positive*-$l_1$ $\cdot l\ \varrho_1 \odot \gamma =$ *literal.from-ground* $l_{G1}$
**if** $\mathcal{P}_G = Pos$
**using** *obtain-strictly-maximal-literal*[*OF E-grounding*]
**by** *metis*

**moreover then have** *positive*-$l_1 \in\# E$ **if** $\mathcal{P}_G = Pos$
**using** *that strictly-maximal-in-clause*
**by** *blast*

**moreover then have** *is-maximal* (*literal.from-ground* $l_{G1}$) ($E \cdot \varrho_1 \odot \gamma$) **if** *?select*$_G$*-empty*
**using** *that ground-superpositionI*(*9*) *is-maximal-not-empty E-grounding*
**by** *auto*

**then obtain** *negative-maximal*-$l_1$ **where**
*is-maximal negative-maximal*-$l_1$ $E$
*negative-maximal*-$l_1$ $\cdot l\ \varrho_1 \odot \gamma =$ *literal.from-ground* $l_{G1}$
**if** $\mathcal{P}_G = Neg$ *?select*$_G$*-empty*
**using**
*obtain-maximal-literal*[*OF E-not-empty E-grounding*[*folded clause.subst-comp-subst*]]
*unique-maximal-in-ground-clause*[*OF E-grounding*[*folded clause.subst-comp-subst*]]
**by** *metis*

**moreover then have** *negative-maximal*-$l_1 \in\# E$ **if** $\mathcal{P}_G = Neg$ *?select*$_G$*-empty*
**using** *that maximal-in-clause*
**by** *blast*

**moreover have** *ground-is-maximal* $l_{G1}$ (*select*$_G$ $E_G$) **if** $\mathcal{P}_G = Neg$ *?select*$_G$*-not-empty*
**using** *ground-superpositionI*(*9*) *that*
**by** *simp*

**then obtain** *negative-selected*-$l_1$ **where**
*is-maximal negative-selected*-$l_1$ (*select E*)
*negative-selected*-$l_1$ $\cdot l\ \varrho_1 \odot \gamma =$ *literal.from-ground* $l_{G1}$
**if** $\mathcal{P}_G = Neg$ *?select*$_G$*-not-empty*
**using**

*select-from-E*
*unique-maximal-in-ground-clause*
*obtain-maximal-literal*
**unfolding** $E_G$*-def*
**by** (*metis* (*no-types*, *lifting*) *clause.ground-is-ground clause.from-ground-empty′ clause.magma-subst-empty*)

**moreover then have** *negative-selected-*$l_1$ $\in\#$ $E$ **if** $\mathcal{P}_G$ = *Neg ?select*$_G$*-not-empty*
**using** *that*
**by** (*meson maximal-in-clause mset-subset-eqD select-subset*)

**ultimately show** *?thesis*
**using** *that ground-superpositionI*(*9*)
**by** (*metis literals-distinct*(*1*))
**qed**

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

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

**let** *?*$\mathcal{P}$ = *if* $\mathcal{P}_G$ = *Pos then Pos else Neg*

**have** [*simp*]: $\mathcal{P}_G \neq$ *Pos* $\longleftrightarrow$ $\mathcal{P}_G$ = *Neg* $\mathcal{P}_G \neq$ *Neg* $\longleftrightarrow$ $\mathcal{P}_G$ = *Pos*
**using** *ground-superpositionI*(*4*)
**by** *auto*

**have** [*simp*]: $\bigwedge a\ \sigma$. *?*$\mathcal{P}$ $a \cdot l\ \sigma$ = *?*$\mathcal{P}$ ($a \cdot a\ \sigma$)
**by** *auto*

**have** [*simp*]: $\bigwedge \mathcal{V}$ $a$. *literal.is-welltyped* $\mathcal{V}$ (*?*$\mathcal{P}$ $a$) $\longleftrightarrow$ *atom.is-welltyped* $\mathcal{V}$ $a$
**by** (*auto simp*: *literal-is-welltyped-iff-atm-of*)

**have** [*simp*]: $\bigwedge a$. *literal.vars* (*?*$\mathcal{P}$ $a$) = *atom.vars* $a$
**by** *auto*

**have** $l_1$-$\gamma$:
$l_1 \cdot l\ \varrho_1 \odot \gamma$ = *?*$\mathcal{P}$ (*Upair* (*context.from-ground* $c_G$)$\langle$*term.from-ground* $t_{G1}\rangle$ (*term.from-ground* $t_{G2}$))
**unfolding** *ground-superpositionI* $l_1$-$\gamma$
**by** *simp*

**obtain** $c_1$ $t_1$ $t_1{}'$ **where**
$l_1$: $l_1$ = *?*$\mathcal{P}$ (*Upair* $c_1\langle t_1\rangle$ $t_1{}'$) **and**
$t_1{}'$-$\gamma$: $t_1{}' \cdot t\ \varrho_1 \odot \gamma$ = *term.from-ground* $t_{G2}$ **and**
$t_1$-$\gamma$: $t_1 \cdot t\ \varrho_1 \odot \gamma$ = *term.from-ground* $t_{G1}$ **and**
$c_1$-$\gamma$: $c_1 \cdot t_c\ \varrho_1 \odot \gamma$ = *context.from-ground* $c_G$ **and**

```
  t₁-is-Fun: is-Fun t₁
proof -

  obtain c₁-t₁ t₁' where
    l₁: l₁ = ?𝒫 (Upair c₁-t₁ t₁') and
    t₁'-γ: t₁' ·t ϱ₁ ⊙ γ = term.from-ground t_G2 and
    c₁-t₁-γ: c₁-t₁ ·t ϱ₁ ⊙ γ = (context.from-ground c_G)⟨term.from-ground t_G1⟩
    using l₁-γ
    by (smt (verit) obtain-from-literal-subst)

  let ?inference-into-Fun =
    ∃ c₁ t₁.
      c₁-t₁ = c₁⟨t₁⟩ ∧
      t₁ ·t ϱ₁ ⊙ γ = term.from-ground t_G1 ∧
      c₁ ·t_c ϱ₁ ⊙ γ = context.from-ground c_G ∧
      is-Fun t₁

  have ¬ ?inference-into-Fun ⟹ ground.redundant-infer N_G ι_G
  proof -
    assume ¬ ?inference-into-Fun

    with c₁-t₁-γ
    obtain t₁ c₁ c_G' where
      c₁-t₁: c₁-t₁ = c₁⟨t₁⟩ and
      t₁-is-Var: is-Var t₁ and
      c_G: c_G = context.to-ground (c₁ ·t_c ϱ₁ ⊙ γ) ∘_c c_G'
    proof(induction c₁-t₁ arbitrary: c_G thesis)
      case (Var x)

      show ?case
      proof(rule Var.prems)
        show Var x = □⟨Var x⟩
          by simp

        show is-Var (Var x)
          by simp

        show c_G = context.to-ground (□ ·t_c ϱ₁ ⊙ γ) ∘_c c_G
          by (simp add: context.to-ground-def)
      qed
    next
      case (Fun f ts)

      have c_G ≠ □
        using Fun.prems(2,3)
        unfolding context.from-ground-def
        by (metis actxt.simps(8) intp-actxt.simps(1) is-FunI)

      then obtain ts_G1 c_G' ts_G2 where
```

$c_G$: $c_G$ = *More f* $ts_{G1}$ $c_G{}'$ $ts_{G2}$
**using** *Fun.prems*
**by** (*cases* $c_G$) (*auto simp*: *context.from-ground-def*)

**have**
*map* ($\lambda t.\ t \cdot t\ \varrho_1 \odot \gamma$) *ts* =
*map term.from-ground* $ts_{G1}$ @ (*context.from-ground* $c_G{}'$)⟨*term.from-ground* $t_{G1}$⟩ #
*map term.from-ground* $ts_{G2}$
**using** *Fun*(*3*)
**unfolding** $c_G$ *context.from-ground-def*
**by** *simp*

**moreover then obtain** $ts_1$ *t* $ts_2$ **where**
*ts*: *ts* = $ts_1$ @ *t* # $ts_2$ **and**
$ts_1$-$\gamma$: *map* ($\lambda$*term. term* $\cdot t\ \varrho_1 \odot \gamma$) $ts_1$ = *map term.from-ground* $ts_{G1}$ **and**
$ts_2$-$\gamma$: *map* ($\lambda$*term. term* $\cdot t\ \varrho_1 \odot \gamma$) $ts_2$ = *map term.from-ground* $ts_{G2}$
**by** (*smt append-eq-map-conv map-eq-Cons-D*)

**ultimately have** *t*-$\gamma$: $t \cdot t\ \varrho_1 \odot \gamma$ = (*context.from-ground* $c_G{}'$)⟨*term.from-ground* $t_{G1}$⟩
**by** *simp*

**obtain** $t_1$ $c_1$ $c_G{}''$ **where**
$t = c_1\langle t_1\rangle$ **and**
*is-Var* $t_1$ **and**
$c_G{}'$ = *context.to-ground* ($c_1 \cdot t_c\ \varrho_1 \odot \gamma$) $\circ_c$ $c_G{}''$
**proof**−
**have** $t \in$ *set ts*
**by** (*simp add*: *ts*)

**moreover have**
$\nexists c_1\ t_1.\ t = c_1\langle t_1\rangle \wedge$
$t_1 \cdot t\ \varrho_1 \odot \gamma$ = *term.from-ground* $t_{G1}$ $\wedge$
$c_1 \cdot t_c\ \varrho_1 \odot \gamma$ = *context.from-ground* $c_G{}'$ $\wedge$
*is-Fun* $t_1$
**proof**(*rule notI*, *elim exE conjE*)
**fix** $c_1$ $t_1$
**assume**
$t = c_1\langle t_1\rangle$
$t_1 \cdot t\ \varrho_1 \odot \gamma$ = *term.from-ground* $t_{G1}$
$c_1 \cdot t_c\ \varrho_1 \odot \gamma$ = *context.from-ground* $c_G{}'$
*is-Fun* $t_1$

**moreover then have**
*Fun f ts* = (*More f* $ts_1$ $c_1$ $ts_2$)⟨$t_1$⟩
(*More f* $ts_1$ $c_1$ $ts_2$) $\cdot t_c\ \varrho_1 \odot \gamma$ = *context.from-ground* $c_G$
**unfolding** *context.from-ground-def* $c_G$ *ts*
**using** $ts_1$-$\gamma$ $ts_2$-$\gamma$

**by** *auto*

**ultimately show** *False*
**using** *Fun.prems(3)*
**by** *blast*
**qed**

**ultimately show** *?thesis*
**using** *Fun(1)* $t$-$\gamma$ *that*
**by** *blast*
**qed**

**moreover then have**
*Fun f ts* = (*More f* $ts_1$ $c_1$ $ts_2$)$\langle t_1 \rangle$
$c_G$ = *context.to-ground* (*More f* $ts_1$ $c_1$ $ts_2$ $\cdot t_c$ $\varrho_1 \odot \gamma$) $\circ_c$ $c_G{}''$
**using** $ts_1$-$\gamma$ $ts_2$-$\gamma$
**unfolding** *context.to-ground-def* $c_G$ *ts*
**by** *auto*

**ultimately show** *?case*
**using** *Fun.prems(1)*
**by** *blast*
**qed**

**obtain** $x$ **where** $t_1$-$\varrho_1$: $t_1$ $\cdot t$ $\varrho_1$ = *Var* $x$
**using** $t_1$-*is-Var* *term.id-subst-rename*[*OF* $\varrho_1$]
**unfolding** *is-Var-def*
**by** *auto*

**have** $\iota_G$-*parts*:
*set* (*side-prems-of* $\iota_G$) = $\{D_G\}$
*main-prem-of* $\iota_G$ = $E_G$
*concl-of* $\iota_G$ = $C_G$
**by** *simp-all*

**show** *?thesis*
**proof**(*rule ground.redundant-infer-singleton*, *unfold* $\iota_G$-*parts*, *intro bexI conjI*)

**let** $?t_G$ = (*context.from-ground* $c_G{}'$)$\langle$*term.from-ground* $t_{G3}\rangle$

**define** $\gamma'$ **where**
$\gamma' \equiv \gamma(x := ?t_G)$

**let** $?E_G{}'$ = *clause.to-ground* ($E \cdot \varrho_1 \odot \gamma'$)

**have** $t_1$-$\gamma$: $t_1$ $\cdot t$ $\varrho_1 \odot \gamma$ = (*context.from-ground* $c_G{}'$)$\langle$*term.from-ground* $t_{G1}\rangle$
**proof** −

**have** *context.is-ground* ($c_1$ $\cdot t_c$ $\varrho_1 \odot \gamma$)

**using** $c_1$-$t_1$-$\gamma$
**unfolding** $c_1$-$t_1$ *context.safe-unfolds*
**by** (*metis context.ground-is-ground context.term-with-context-is-ground term.ground-is-ground*)

**then show** *?thesis*
**using** $c_1$-$t_1$-$\gamma$
**unfolding** $c_1$-$t_1$ $c_1$-$t_1$-$\gamma$ $c_G$
**by** *auto*
**qed**

**have** $t_1$-$\gamma'$: $t_1 \cdot t\ \varrho_1 \odot \gamma' = (\textit{context.from-ground}\ c_G{}')\langle\textit{term.from-ground}\ t_{G3}\rangle$
**unfolding** $\gamma'$-*def*
**using** $t_1$-$\varrho_1$
**by** *simp*

**show** $?E_G{}' \in N_G$
**proof**−

**have** $?E_G{}' \in$ *clause.welltyped-ground-instances* $(E, \mathcal{V}_1)$
**proof**(*unfold clause.welltyped-ground-instances-def mem-Collect-eq fst-conv snd-conv*,
*intro exI conjI E-is-welltyped* $\mathcal{V}_1$,
*rule refl*)

**show** *clause.is-ground* $(E \cdot \varrho_1 \odot \gamma')$
**unfolding** $\gamma'$-*def*
**using** *E-grounding*
**by** *simp*

**show** *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ $(\varrho_1 \odot \gamma')$
**proof**(*intro term.welltyped.typed-subst-compose* $\varrho_1$-*is-welltyped*)

**have** *welltyped* $\mathcal{V}_1$ $?t_G$ $(\mathcal{V}_1\ x)$
**proof**−

**have** *welltyped* $\mathcal{V}_1$ $(\textit{context.from-ground}\ c_G{}')\langle\textit{term.from-ground}\ t_{G1}\rangle$ $(\mathcal{V}_1\ x)$
**proof**−

**have** *welltyped* $\mathcal{V}_1$ $(t_1 \cdot t\ \varrho_1)$ $(\mathcal{V}_1\ x)$
**using** $t_1$-$\varrho_1$
**by** *auto*

**then have** *welltyped* $\mathcal{V}_1$ $(t_1 \cdot t\ \varrho_1 \odot \gamma)$ $(\mathcal{V}_1\ x)$
**using** $\varrho_1$-*is-welltyped* $\varrho_1$-$\gamma$-*is-welltyped*
**unfolding** $E$ $c_1$-$t_1$ $l_1$ *subst-compose-def*
**by** *simp*

**moreover have** *context.is-ground* $(c_1 \cdot t_c\ \varrho_1 \odot \gamma)$
**using** $c_1$-$t_1$-$\gamma$
**unfolding** $c_1$-$t_1$ *context.safe-unfolds*
**by** (*metis context.ground-is-ground context.term-with-context-is-ground term.ground-is-ground*)

**then have** $t_1 \cdot t\ \varrho_1 \odot \gamma = (\textit{context.from-ground}\ c_G{}')\langle \textit{term.from-ground}\ t_{G1}\rangle$
**using** $c_1$-$t_1$-$\gamma$
**unfolding** $c_1$-$t_1$ $c_1$-$t_1$-$\gamma$ $c_G$
**by** *auto*

**ultimately show** *?thesis*
**by** *argo*
**qed**

**moreover obtain** $\tau$ **where**
*welltyped* $\mathcal{V}_1$ (*term.from-ground* $t_{G1}$) $\tau$
*welltyped* $\mathcal{V}_1$ (*term.from-ground* $t_{G3}$) $\tau$
**proof**−
**have** *clause.is-welltyped* $\mathcal{V}_2$ (*clause.from-ground* $D_G$)
**using** *D-is-welltyped*
**unfolding**
$D_G$-*def*
*clause.to-ground-inverse*[*OF D-grounding*]
*clause.is-welltyped.subst-stability*[*OF* $\varrho_2$-$\gamma$-*is-welltyped*]**.**

**then obtain** $\tau$ **where**
*welltyped* $\mathcal{V}_2$ (*term.from-ground* $t_{G1}$) $\tau$
*welltyped* $\mathcal{V}_2$ (*term.from-ground* $t_{G3}$) $\tau$
**unfolding** *ground-superpositionI*
**by** *auto*

**then show** *?thesis*
**using** *that term.welltyped.explicit-replace-*$\mathcal{V}$*-iff*[*of* - $\mathcal{V}_2$ $\mathcal{V}_1$]
**by** *simp*
**qed**

**ultimately show** *?thesis*
**by** *auto*
**qed**

**moreover have** *term.subst.is-welltyped-on* $(\bigcup$ (*term.vars* ‘ $\varrho_1$ ‘ *clause.vars E*)) $\mathcal{V}_1$ $\gamma$
**by** (*intro term.welltyped.renaming-subst-compose* $\varrho_1$-$\gamma$-*is-welltyped* $\varrho_1$-*is-welltyped* $\varrho_1$)

**ultimately show**
*term.subst.is-welltyped-on* $(\bigcup$ (*term.vars* ‘ $\varrho_1$ ‘ *clause.vars E*)) $\mathcal{V}_1$ $\gamma'$

```
        unfolding γ'-def
        by simp
    qed
  qed

  then show ?thesis
    by simp
qed

show ground.G-entails {?E_G', D_G} {C_G}
proof(unfold ground.G-entails-def, intro allI impI)
  fix I :: 'f gterm rel
  let ?I = upair ` I

  assume
    refl-I: refl I and
    trans-I: trans I and
    sym-I: sym I and
    compatible-with-gctxt-I: compatible-with-gctxt I and
    premise: ?I ⊫s {?E_G', D_G}

   then interpret clause-entailment I
     by unfold-locales

  have γ-x-is-ground: term.is-ground (γ x)
    using t_1-γ t_1-ϱ_1
    by auto

  show ?I ⊫s {C_G}
  proof(cases ?I ⊫ D_G')
    case True

    then show ?thesis
      unfolding ground-superpositionI
      by auto
  next
    case False

    then have t_G1-t_G3: Upair t_G1 t_G3 ∈ ?I
      using premise sym
      unfolding ground-superpositionI
      by auto

    have ?I ⊫l c_G'⟨t_G1⟩_G ≈ c_G'⟨t_G3⟩_G
      using upair-compatible-with-gctxtI[OF t_G1-t_G3]
      by auto

    then have ?I ⊫l term.to-ground (t_1 ·t ϱ_1 ⊙ γ) ≈ term.to-ground (t_1 ·t
ϱ_1 ⊙ γ')
```

**unfolding** $t_1$-$\gamma$ $t_1$-$\gamma'$
**by** *simp*

**then have** $(\mathit{term.to\text{-}ground}\ (\gamma\ x),\ c_G{}'\langle t_{G3}\rangle_G) \in I$
**unfolding** $\gamma'$-*def*
**using** $t_1$-$\varrho_1$
**by** (*auto simp*: *sym*)

**moreover have** $?I \models ?E_G{}'$
**using** *premise*
**by** *simp*

**ultimately have** $?I \models E_G$
**unfolding** $\gamma'$-*def*
**using**
*clause.symmetric-congruence*$[\mathit{of}\ \text{-}\ \gamma,\ \mathit{OF}\ \text{-}\ \gamma\text{-}x\text{-}is\text{-}ground]$
*E-grounding*
**by** *simp*

**then have** $?I \models \mathit{add\text{-}mset}\ (\mathcal{P}_G\ (\mathit{Upair}\ c_G\langle t_{G3}\rangle_G\ t_{G2}))\ E_G{}'$
**unfolding** *ground-superpositionI*
**using** *symmetric-literal-context-congruence*$[\mathit{OF}\ t_{G1}\text{-}t_{G3}]$
**by** $(\mathit{cases}\ \mathcal{P}_G = \mathit{Pos})$ *simp-all*

**then show** *?thesis*
**unfolding** *ground-superpositionI*
**by** *blast*
**qed**
**qed**

**show** $?E_G{}' \prec_{cG} E_G$
**proof**−

**have** $\gamma\ x = t_1 \cdot t\ \varrho_1 \odot \gamma$
**using** $t_1$-$\varrho_1$
**by** *simp*

**then have** $t_G$-*smaller*: $?t_G \prec_t \gamma\ x$
**using** *ground-superpositionI*(*8*)
**unfolding** $t_1$-$\gamma$ *term.order.less*$_G$-*def*
**by** *simp*

**have** $\mathit{add\text{-}mset}\ (l_1 \cdot l\ \varrho_1 \odot \gamma')\ (E' \cdot \varrho_1 \odot \gamma') \prec_c \mathit{add\text{-}mset}\ (l_1 \cdot l\ \varrho_1 \odot \gamma)$
$(E' \cdot \varrho_1 \odot \gamma)$
**proof**(*rule less*$_c$-*add-mset*)

**have** $x \in \mathit{literal.vars}\ (l_1 \cdot l\ \varrho_1)$
**unfolding** $l_1$ $c_1$-$t_1$ *literal.vars-def atom.vars-def*
**using** $t_1$-$\varrho_1$

```
          by auto

        moreover have literal.is-ground (l₁ ·l ϱ₁ ⊙ γ)
          using E-grounding
          unfolding E
          by simp

        ultimately show l₁ ·l ϱ₁ ⊙ γ' ≺l l₁ ·l ϱ₁ ⊙ γ
          unfolding γ'-def
          using literal.order.subst-update-stability t_G-smaller
          by simp
      next

        have clause.is-ground (E' · ϱ₁ ⊙ γ)
          using E'-γ
          by simp

        then show E' · ϱ₁ ⊙ γ' ≼c E' · ϱ₁ ⊙ γ
          unfolding γ'-def
          using clause.order.subst-update-stability t_G-smaller
          by (cases x ∈ clause.vars (E' · ϱ₁)) simp-all
      qed

      then have E · ϱ₁ ⊙ γ' ≺c E · ϱ₁ ⊙ γ
        unfolding E
        by simp

      moreover have clause.is-ground (E · ϱ₁ ⊙ γ')
        unfolding γ'-def
        using E-grounding
        by simp

      ultimately show ?thesis
        using E-grounding
        unfolding clause.order.less_G-def
        by simp
    qed
  qed
qed

then show ?thesis
  using not-redundant ground-superposition that l₁ t₁'-γ c₁-t₁-γ
  unfolding ground.Red-I-def ground.G-Inf-def
  by auto
qed

obtain l₂ where
  l₂-γ: l₂ ·l ϱ₂ ⊙ γ = literal.from-ground l_G2 and
  l₂-is-strictly-maximal: is-strictly-maximal l₂ D
```

**proof**−
**have** *is-strictly-maximal* (*literal.from-ground* $l_{G2}$) ($D \cdot \varrho_2 \odot \gamma$)
**using** *ground-superpositionI*(*11*) *D-grounding*
**by** *simp*

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

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

**from** $l_2$-$\gamma$ **have** $l_2$-$\gamma$: $l_2 \cdot l\ \varrho_2 \odot \gamma$ = *term.from-ground* $t_{G1} \approx$ *term.from-ground* $t_{G3}$
**unfolding** *ground-superpositionI*
**by** *simp*

**then obtain** $t_2$ $t_2'$ **where**
$l_2$: $l_2 = t_2 \approx t_2'$ **and**
$t_2$-$\gamma$: $t_2 \cdot t\ \varrho_2 \odot \gamma$ = *term.from-ground* $t_{G1}$ **and**
$t_2'$-$\gamma$: $t_2' \cdot t\ \varrho_2 \odot \gamma$ = *term.from-ground* $t_{G3}$
**using** *obtain-from-pos-literal-subst*
**by** *metis*

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

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

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

**have** $\gamma$*-is-welltyped*:
*term.subst.is-welltyped-on* (*clause.vars* ($E \cdot \varrho_1$) $\cup$ *clause.vars* ($D \cdot \varrho_2$)) $\mathcal{V}_3$ $\gamma$
**proof**(*unfold Set.ball-Un*, *intro conjI*)

**show** *term.subst.is-welltyped-on* (*clause.vars* ($E \cdot \varrho_1$)) $\mathcal{V}_3$ $\gamma$
**using** *clause.is-welltyped.renaming-grounding*[*OF* $\varrho_1$ $\varrho_1$-$\gamma$*-is-welltyped* *E-grounding* $\mathcal{V}_1$-$\mathcal{V}_3$]**.**
**next**

**show** *term.subst.is-welltyped-on* (*clause.vars* ($D \cdot \varrho_2$)) $\mathcal{V}_3$ $\gamma$
**using** *clause.is-welltyped.renaming-grounding*[*OF* $\varrho_2$ $\varrho_2$-$\gamma$-*is-welltyped* *D-grounding* $\mathcal{V}_2$-$\mathcal{V}_3$].
**qed**

**obtain** $\mu$ $\sigma$ **where**
$\gamma$: $\gamma = \mu \odot \sigma$ **and**
*imgu*: *welltyped-imgu-on* (*clause.vars* ($E \cdot \varrho_1$) $\cup$ *clause.vars* ($D \cdot \varrho_2$)) $\mathcal{V}_3$ ($t_1 \cdot t$ $\varrho_1$) ($t_2 \cdot t$ $\varrho_2$) $\mu$
**proof**−

**have** *unified*: $t_1 \cdot t\ \varrho_1 \cdot t\ \gamma = t_2 \cdot t\ \varrho_2 \cdot t\ \gamma$
**using** $t_1$-$\gamma$ $t_2$-$\gamma$
**by** *simp*

**obtain** $\tau$ **where** *welltyped*: *welltyped* $\mathcal{V}_3$ ($t_1 \cdot t$ $\varrho_1$) $\tau$ *welltyped* $\mathcal{V}_3$ ($t_2 \cdot t$ $\varrho_2$) $\tau$
**proof**−
**have** *clause.is-welltyped* $\mathcal{V}_2$ ($D \cdot \varrho_2 \odot \gamma$)
**using** $\varrho_2$-$\gamma$-*is-welltyped* *D-is-welltyped*
**by** (*metis clause.is-welltyped.subst-stability*)

**then obtain** $\tau$ **where**
*welltyped* $\mathcal{V}_2$ (*term.from-ground* $t_{G1}$) $\tau$
**unfolding** *D*-$\gamma$ *ground-superpositionI*
**by** *auto*

**then have** *welltyped* $\mathcal{V}_3$ (*term.from-ground* $t_{G1}$) $\tau$
**using** *term.welltyped.is-ground-typed*
**by** (*meson term.ground-is-ground term.welltyped.explicit-is-ground-typed*)

**then have** *welltyped* $\mathcal{V}_3$ ($t_1 \cdot t$ $\varrho_1 \odot \gamma$) $\tau$ *welltyped* $\mathcal{V}_3$ ($t_2 \cdot t$ $\varrho_2 \odot \gamma$) $\tau$
**using** $t_1$-$\gamma$ $t_2$-$\gamma$
**by** *presburger*+

**moreover have**
*term.vars* ($t_1 \cdot t$ $\varrho_1$) $\subseteq$ *clause.vars* ($E \cdot \varrho_1$)
*term.vars* ($t_2 \cdot t$ $\varrho_2$) $\subseteq$ *clause.vars* ($D \cdot \varrho_2$)
**unfolding** *E* $l_1$ *clause.add-subst* *D* $l_2$
**by** *auto*

**ultimately have** *welltyped* $\mathcal{V}_3$ ($t_1 \cdot t$ $\varrho_1$) $\tau$ *welltyped* $\mathcal{V}_3$ ($t_2 \cdot t$ $\varrho_2$) $\tau$
**using** $\gamma$-*is-welltyped*
**by** (*simp-all add*: *subsetD*)

**then show** *?thesis*
**using** *that*
**by** *blast*
**qed**

**show** *?thesis*
**using** *obtain-welltyped-imgu-on*[*OF unified welltyped*] *that*
**by** *metis*
**qed**

**define** $C'$ **where**
$C'$: $C' = add\text{-}mset\ (?\mathcal{P}\ (Upair\ (c_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle\ (t_1{}' \cdot t\ \varrho_1)))\ (E' \cdot \varrho_1 + D' \cdot \varrho_2) \cdot \mu$

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

**show** *superposition*: *superposition* $(D, \mathcal{V}_2)$ $(E, \mathcal{V}_1)$ $(C', \mathcal{V}_3)$
**proof**(*rule superpositionI*;
((*rule* $\varrho_1$ $\varrho_2$ $E$ $D$ $l_1$ $l_2$ $t_1$*-is-Fun imgu rename-apart* $\varrho_1$*-is-welltyped* $\varrho_2$*-is-welltyped* $\mathcal{V}_1$ $\mathcal{V}_2$ $C'$
$\mathcal{V}_1$-$\mathcal{V}_3$ $\mathcal{V}_2$-$\mathcal{V}_3$)+)?)

**show** $?\mathcal{P} \in \{Pos, Neg\}$
**by** *simp*
**next**

**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** $\gamma$
**by** *simp*

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

**show** $D \cdot \varrho_2 \odot \mu \cdot \sigma \prec_c E \cdot \varrho_1 \odot \mu \cdot \sigma$
**using** *ground-superpositionI*(*3*) *D-grounding E-grounding*
**unfolding** $E_G$*-def* $D_G$*-def clause.order.less*$_G$*-def* $\gamma$
**by** *simp*
**qed**
**next**
**assume** $?\mathcal{P} = Pos$

**then show** *select* $E = \{\#\}$
**using** *ground-superpositionI*(*9*) *select-from-E*
**by** *fastforce*

**next**
**assume** *Pos*: $?\mathcal{P} = Pos$

**show** *is-strictly-maximal* ($l_1$ ·*l* $\varrho_1 \odot \mu$) ($E \cdot \varrho_1 \odot \mu$)
**proof**(*rule is-strictly-maximal-if-grounding-is-strictly-maximal*)

**show** $l_1$ ·*l* $\varrho_1 \odot \mu$ ∈# $E \cdot \varrho_1 \odot \mu$
**using** $l_1$*-in-E*
**by** *blast*

**show** *clause.is-ground* ($E \cdot \varrho_1 \odot \mu \cdot \sigma$)
**using** *E-grounding*[*unfolded* $\gamma$]
**by** *simp*

**show** *is-strictly-maximal* ($l_1$ ·*l* $\varrho_1 \odot \mu$ ·*l* $\sigma$) ($E \cdot \varrho_1 \odot \mu \cdot \sigma$)
**using** *Pos* $l_1$-$\gamma$ *E*-$\gamma$ *ground-superpositionI*(*9*)
**unfolding** $\gamma$ *ground-superpositionI*
**by** *fastforce*
**qed**
**next**
**assume** *Neg*: *?*$\mathcal{P}$ = *Neg select E* = {#}

**show** *is-maximal* ($l_1$ ·*l* $\varrho_1 \odot \mu$) ($E \cdot \varrho_1 \odot \mu$)
**proof**(*rule is-maximal-if-grounding-is-maximal*)

**show** $l_1$ ·*l* $\varrho_1 \odot \mu$ ∈# $E \cdot \varrho_1 \odot \mu$
**using** $l_1$*-in-E*
**by** *blast*
**next**

**show** *clause.is-ground* ($E \cdot \varrho_1 \odot \mu \cdot \sigma$)
**using** *E-grounding* $\gamma$
**by** *auto*
**next**

**show** *is-maximal* ($l_1$ ·*l* $\varrho_1 \odot \mu$ ·*l* $\sigma$) ($E \cdot \varrho_1 \odot \mu \cdot \sigma$)
**using** $l_1$-$\gamma$ $\gamma$ *E*-$\gamma$ *ground-superpositionI*(*5*,*9*) *is-maximal-not-empty Neg select-from-E*
**by** *auto*
**qed**
**next**
**assume** *Neg*: *?*$\mathcal{P}$ = *Neg select E* ≠ {#}

**show** *is-maximal* ($l_1$ ·*l* $\varrho_1 \odot \mu$) ((*select E*) $\cdot \varrho_1 \odot \mu$)
**proof**(*rule is-maximal-if-grounding-is-maximal*)

**show** $l_1$ ·*l* $\varrho_1 \odot \mu$ ∈# *select E* $\cdot \varrho_1 \odot \mu$
**using** *ground-superpositionI*(*9*) $l_1$*-selected maximal-in-clause Neg select-from-E*
**by** *force*
**next**

```
      show clause.is-ground (select E · ϱ1 ⊙ μ · σ)
        using select-ground-subst[OF E-grounding]
        unfolding γ
        by simp
    next
      show is-maximal (l1 ·l ϱ1 ⊙ μ ·l σ) (select E · ϱ1 ⊙ μ · σ)
        using γ ground-superpositionI(5,9) l1-γ that select-from-E Neg
        by fastforce
    qed
  next

    show select D = {#}
      using ground-superpositionI(10) select-from-D
      by simp
  next

    show is-strictly-maximal (l2 ·l ϱ2 ⊙ μ) (D · ϱ2 ⊙ μ)
    proof(rule is-strictly-maximal-if-grounding-is-strictly-maximal)

      show l2 ·l ϱ2 ⊙ μ ∈# D · ϱ2 ⊙ μ
        using l2-in-D
        by blast
    next

      show clause.is-ground (D · ϱ2 ⊙ μ · σ)
        using D-grounding
        unfolding γ
        by simp
    next

      show is-strictly-maximal (l2 ·l ϱ2 ⊙ μ ·l σ) (D · ϱ2 ⊙ μ · σ)
        using l2-γ γ D-γ ground-superpositionI(6,11)
        by auto
    qed
  next

    show ¬ c1⟨t1⟩ ·t ϱ1 ⊙ μ ⪯t t1' ·t ϱ1 ⊙ μ
    proof(rule term.order.ground-less-not-less-eq)

      show term.is-ground (t1' ·t ϱ1 ⊙ μ ·t σ)
        using t1'-γ γ
        by simp
    next

      show term.is-ground (c1⟨t1⟩ ·t ϱ1 ⊙ μ ·t σ)
        using t1-γ c1-γ γ
        by simp
    next
```

**show** $t_1{}' \cdot t\ \varrho_1 \odot \mu \cdot t\ \sigma \prec_t c_1\langle t_1\rangle \cdot t\ \varrho_1 \odot \mu \cdot t\ \sigma$
**using** *ground-superpositionI(7)* $c_1$-$\gamma$ $t_1{}'$-$\gamma$ $t_1$-$\gamma$
**unfolding** *term.order.less*$_G$*-def* $\gamma$
**by** *auto*
**qed**
**next**

**show** $\neg\ t_2 \cdot t\ \varrho_2 \odot \mu \preceq_t t_2{}' \cdot t\ \varrho_2 \odot \mu$
**proof**(*rule term.order.ground-less-not-less-eq*)

**show** *term.is-ground* $(t_2{}' \cdot t\ \varrho_2 \odot \mu \cdot t\ \sigma)$
**using** $t_2{}'$-$\gamma$ $\gamma$
**by** *simp*
**next**

**show** *term.is-ground* $(t_2 \cdot t\ \varrho_2 \odot \mu \cdot t\ \sigma)$
**using** $t_2$-$\gamma$ $\gamma$
**by** *simp*
**next**

**show** $t_2{}' \cdot t\ \varrho_2 \odot \mu \cdot t\ \sigma \prec_t t_2 \cdot t\ \varrho_2 \odot \mu \cdot t\ \sigma$
**using** *ground-superpositionI(8)* $t_2$-$\gamma$ $t_2{}'$-$\gamma$
**unfolding** $\gamma$ *term.order.less*$_G$*-def*
**by** *simp*
**qed**
**next**

**have** $\exists\tau.\ welltyped\ \mathcal{V}_2\ t_2\ \tau \wedge welltyped\ \mathcal{V}_2\ t_2{}'\ \tau$
**using** *D-is-welltyped*
**unfolding** $D$ $l_2$
**by** *auto*

**then show** $\bigwedge\tau\ \tau'.\ [\![typed\ \mathcal{V}_2\ t_2\ \tau;\ typed\ \mathcal{V}_2\ t_2{}'\ \tau']\!] \Longrightarrow \tau = \tau'$
**using** *term.typed-if-welltyped*
**by** *blast*
**qed**

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

**have** *term-subst.is-idem* $\mu$
**using** *imgu term.is-imgu-iff-is-idem-and-is-mgu*
**by** *blast*

**then have** $\mu$-$\gamma$: $\mu \odot \gamma = \gamma$
**unfolding** $\gamma$ *term-subst.is-idem-def*
**by** (*metis subst-compose-assoc*)

```
    have C · γ =
          add-mset
                  (?𝒫 (Upair (context.from-ground c_G)⟨term.from-ground t_G3⟩
(term.from-ground t_G2)))
               (clause.from-ground E_G' + clause.from-ground D_G')
      using ground-superpositionI(4, 12) clause.to-ground-inverse[OF C-grounding]
        by auto

      then show ?thesis
        unfolding
          C'
          E'-γ[symmetric]
          D'-γ[symmetric]
          t₁'-γ[symmetric]
          t₂'-γ[symmetric]
          c₁-γ[symmetric]
          clause.subst-comp-subst[symmetric]
          μ-γ
        by simp
    qed

    show ι_G ∈ inference-ground-instances (Infer [(D, 𝒱₂), (E, 𝒱₁)] (C', 𝒱₃))
    proof (rule is-inference-ground-instance-two-premises)

      show is-inference-ground-instance-two-premises (D, 𝒱₂) (E, 𝒱₁) (C', 𝒱₃) ι_G
γ ϱ₁ ϱ₂
      proof(unfold split, intro conjI;
          (rule ϱ₁ ϱ₂ rename-apart D-is-welltyped E-is-welltyped refl 𝒱₁
            𝒱₂ 𝒱₃)?)

        show inference.is-ground (Infer [D · ϱ₂, E · ϱ₁] C' ·ι γ)
          using  D-grounding E-grounding C-grounding C'-γ
          by auto
      next

        show ι_G = inference.to-ground (Infer [D · ϱ₂, E · ϱ₁] C' ·ι γ)
          using C'-γ
          by simp
      next

        show term.subst.is-welltyped-on (clause.vars C') 𝒱₃ γ
        proof(rule term.is-welltyped-on-subset[OF γ-is-welltyped])

          show clause.vars C' ⊆ clause.vars (E · ϱ₁) ∪ clause.vars (D · ϱ₂)
          proof (unfold subset-eq, intro ballI)
            fix x

            have is-imgu: term.is-imgu μ {{t₁ ·t ϱ₁, t₂ ·t ϱ₂}}
              using imgu
```

**by** *blast*

**assume** $x \in$ *clause.vars* $C'$

**then consider**
($t_2'$) $x \in$ *term.vars* ($t_2' \cdot t\ \varrho_2 \odot \mu$) |
($c_1$) $x \in$ *context.vars* ($c_1 \cdot t_c\ \varrho_1 \odot \mu$) |
($t_1'$) $x \in$ *term.vars* ($t_1' \cdot t\ \varrho_1 \odot \mu$) |
($E'$) $x \in$ *clause.vars* ($E' \cdot \varrho_1 \odot \mu$) |
($D'$) $x \in$ *clause.vars* ($D' \cdot \varrho_2 \odot \mu$)
**unfolding** $C'$
**by** *auto*

**then show** $x \in$ *clause.vars* ($E \cdot \varrho_1$) $\cup$ *clause.vars* ($D \cdot \varrho_2$)
**proof** *cases*
**case** $t_2'$

**then show** *?thesis*
**using** *term.variables-in-base-imgu*[*OF is-imgu*]
**unfolding** $E\ l_1\ D\ l_2$
**by** *auto*
**next**
**case** $c_1$

**then show** *?thesis*
**using** *context.variables-in-base-imgu*[*OF is-imgu*]
**unfolding** $E\ l_1\ D\ l_2$
**by** *force*
**next**
**case** $t_1'$

**then show** *?thesis*
**using** *term.variables-in-base-imgu*[*OF is-imgu*]
**unfolding** $E$ *clause.add-subst* $l_1\ D\ l_2$
**by** *auto*
**next**
**case** $E'$

**then show** *?thesis*
**using** *clause.variables-in-base-imgu*[*OF is-imgu*]
**unfolding** $E\ l_1\ D\ l_2$
**by** *auto*
**next**
**case** $D'$

**then show** *?thesis*
**using** *clause.variables-in-base-imgu*[*OF is-imgu*]
**unfolding** $E\ l_1\ D\ l_2$
**by** *auto*

**qed**
**qed**
**qed**
**next**

**show** *clause.is-welltyped* $\mathcal{V}_3$ $C'$
**using** *superposition superposition-preserves-typing E-is-welltyped D-is-welltyped*
**by** *blast*
**qed**

**show** $\iota_G \in$ *ground.G-Inf*
**unfolding** *ground.G-Inf-def*
**using** *ground-superposition*
**by** *simp*
**qed**
**qed**
**qed**

## 3.2 Ground instances

**context**
**fixes** $\iota_G$ $N$
**assumes**
*subst-stability*: *subst-stability-on* $N$ **and**
$\iota_G$*-Inf-from*: $\iota_G \in$ *ground.Inf-from-q* $select_G$ ($\bigcup$(*clause.welltyped-ground-instances* ‘ $N$))
**begin**

**lemma** *single-premise-ground-instance*:
**assumes**
*ground-inference*: $\iota_G \in$ {*Infer* [$D$] $C$ | $D$ $C$. *ground-inference* $D$ $C$} **and**
*lifting*: $\bigwedge D\ \gamma\ C\ \mathcal{V}$ *thesis*. ⟦
*ground-inference* (*clause.to-ground* ($D \cdot \gamma$)) (*clause.to-ground* ($C \cdot \gamma$));
*clause.is-ground* ($D \cdot \gamma$);
*clause.is-ground* ($C \cdot \gamma$);
*clause.from-ground* ($select_G$ (*clause.to-ground* ($D \cdot \gamma$))) = *select* $D \cdot \gamma$;
*clause.is-welltyped* $\mathcal{V}$ $D$; *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}$ $\gamma$;
*infinite-variables-per-type* $\mathcal{V}$;
$\bigwedge C'$. ⟦
*inference* ($D$, $\mathcal{V}$) ($C'$, $\mathcal{V}$);
*Infer* [*clause.to-ground* ($D \cdot \gamma$)] (*clause.to-ground* ($C \cdot \gamma$))
$\in$ *inference-ground-instances* (*Infer* [($D$, $\mathcal{V}$)] ($C'$, $\mathcal{V}$));
$C' \cdot \gamma = C \cdot \gamma$⟧ $\Longrightarrow$ *thesis*⟧
$\Longrightarrow$ *thesis* **and**
*inference-eq*: *inference* = *eq-factoring* $\vee$ *inference* = *eq-resolution*
**obtains** $\iota$ **where**
$\iota \in$ *Inf-from* $N$
$\iota_G \in$ *inference-ground-instances* $\iota$
**proof**−

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

**have** $D_G$*-in-groundings*: $D_G \in \bigcup$(*clause.welltyped-ground-instances* ‘ $N$)
**using** $\iota_G$*-Inf-from*
**unfolding** $\iota_G$ *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**
*D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}$ $D$ **and**
$\gamma$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}$ $\gamma$ **and**
$\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$ **and**
*D-in-N*: ($D$, $\mathcal{V}$)$\in N$ **and**
*select*$_G$ $D_G$ = *clause.to-ground* (*select* $D \cdot \gamma$)
$D \cdot \gamma$ = *clause.from-ground* $D_G$
**using** *subst-stability*[*rule-format*, *OF* $D_G$*-in-groundings*]
**by** *blast*

**then have**
$D_G$: $D_G$ = *clause.to-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-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**
*inference*: *inference* ($D$, $\mathcal{V}$) ($C'$, $\mathcal{V}$) **and**
*inference-ground-instances*: $\iota_G \in$ *inference-ground-instances* (*Infer* [($D$, $\mathcal{V}$)] ($C'$,
$\mathcal{V}$)) **and**
$C'$*-C*: $C' \cdot \gamma = C \cdot \gamma$
**using**
*lifting*[*OF*
*ground-inference*[*unfolded* $D_G$ $C_G$]
*D-grounding*
*C-grounding*
*select*[*unfolded* $D_G$]
*D-is-welltyped*
$\gamma$*-is-welltyped*
$\mathcal{V}$]
**unfolding** $D_G$ $C_G$ $\iota_G$**.**

```
  let ?ι = Infer [(D, 𝒱)] (C′, 𝒱)

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

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

lemma eq-resolution-ground-instance:
  assumes ground-eq-resolution: ι_G ∈ ground.eq-resolution-inferences
  obtains ι where
    ι ∈ Inf-from N
    ι_G ∈ inference-ground-instances ι
  using eq-resolution-lifting single-premise-ground-instance[OF ground-eq-resolution]
  by blast

lemma eq-factoring-ground-instance:
  assumes ground-eq-factoring: ι_G ∈ ground.eq-factoring-inferences
  obtains ι where
    ι ∈ Inf-from N
    ι_G ∈ inference-ground-instances ι
  using eq-factoring-lifting single-premise-ground-instance[OF ground-eq-factoring]
  by blast

lemma superposition-ground-instance:
  assumes
    ground-superposition: ι_G ∈ ground.superposition-inferences and
    not-redundant: ι_G ∉ ground.GRed-I (⋃ (clause.welltyped-ground-instances ‘ N))
  obtains ι where
    ι ∈ Inf-from N
    ι_G ∈ inference-ground-instances ι
proof−
  obtain E_G D_G C_G where
    ι_G : ι_G = Infer [D_G, E_G] C_G and
    ground-superposition: ground.superposition D_G E_G C_G
    using assms(1)
    by blast

  have
    E_G-in-groundings: E_G ∈ ⋃ (clause.welltyped-ground-instances ‘ N) and
    D_G-in-groundings: D_G ∈ ⋃ (clause.welltyped-ground-instances ‘ N)
    using ι_G-Inf-from
    unfolding ι_G ground.Inf-from-q-def ground.Inf-from-def
    by simp-all
```

**obtain** $E$ $\mathcal{V}_1$ $\gamma_1$ **where**
*E-grounding*: *clause.is-ground* $(E \cdot \gamma_1)$ **and**
*E-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_1$ $E$ **and**
$\gamma_1$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ $\gamma_1$ **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
*E-in-N*: $(E, \mathcal{V}_1) \in N$ **and**
$select_G$ $E_G$ = *clause.to-ground* (*select* $E \cdot \gamma_1$)
$E \cdot \gamma_1$ = *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* $E \cdot \gamma_1$
**by** (*simp-all add*: *select-ground-subst*)

**obtain** $D$ $\mathcal{V}_2$ $\gamma_2$ **where**
*D-grounding*: *clause.is-ground* $(D \cdot \gamma_2)$ **and**
*D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_2$ $D$ **and**
$\gamma_2$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ $\gamma_2$ **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**
*D-in-N*: $(D, \mathcal{V}_2) \in N$ **and**
$select_G$ $D_G$ = *clause.to-ground* (*select* $D \cdot \gamma_2$)
$D \cdot \gamma_2$ = *clause.from-ground* $D_G$
**using** *subst-stability*[*rule-format*, *OF* $D_G$*-in-groundings*]
**by** *blast*

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

**obtain** $\varrho_1$ $\varrho_2$ $\gamma$ :: ($'f$, $'v$) *subst* **where**
$\varrho_1$: *term-subst.is-renaming* $\varrho_1$ **and**
$\varrho_2$: *term-subst.is-renaming* $\varrho_2$ **and**
*rename-apart*: *clause.vars* $(E \cdot \varrho_1) \cap$ *clause.vars* $(D \cdot \varrho_2) = \{\}$ **and**
$\varrho_1$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ $\varrho_1$ **and**
$\varrho_2$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ $\varrho_2$ **and**
$\gamma_1$-$\gamma$: $\forall X \subseteq$ *clause.vars* $E$. $\forall x \in X$. $\gamma_1\ x = (\varrho_1 \odot \gamma)\ x$ **and**
$\gamma_2$-$\gamma$: $\forall X \subseteq$ *clause.vars* $D$. $\forall x \in X$. $\gamma_2\ x = (\varrho_2 \odot \gamma)\ x$
**using**
*clause.is-welltyped.obtain-merged-grounding*[*OF* $\gamma_1$*-is-welltyped* $\gamma_2$*-is-welltyped*
*E-grounding*
*D-grounding* $\mathcal{V}_2$ *clause.finite-vars*]**.**

**have** *E-grounding*: *clause.is-ground* $(E \cdot \varrho_1 \odot \gamma)$
**using** *clause.subst-eq* $\gamma_1$-$\gamma$ *E-grounding*
**by** *fastforce*

**have** $E_G$: $E_G$ = *clause.to-ground* ($E \cdot \varrho_1 \odot \gamma$)
  **using** *clause.subst-eq* $\gamma_1$-$\gamma$ $E_G$
  **by** *fastforce*

**have** *D-grounding*: *clause.is-ground* ($D \cdot \varrho_2 \odot \gamma$)
  **using** *clause.subst-eq* $\gamma_2$-$\gamma$ *D-grounding*
  **by** *fastforce*

**have** $D_G$: $D_G$ = *clause.to-ground* ($D \cdot \varrho_2 \odot \gamma$)
  **using** *clause.subst-eq* $\gamma_2$-$\gamma$ $D_G$
  **by** *fastforce*

**have** $\varrho_1$-$\gamma$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ ($\varrho_1 \odot \gamma$)
  **using** $\gamma_1$-*is-welltyped* $\gamma_1$-$\gamma$
  **by** *fastforce*

**have** $\varrho_2$-$\gamma$-*is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ ($\varrho_2 \odot \gamma$)
  **using** $\gamma_2$-*is-welltyped* $\gamma_2$-$\gamma$
  **by** *fastforce*

**have** *select-from-E*:
  *clause.from-ground* ($select_G$ (*clause.to-ground* ($E \cdot \varrho_1 \odot \gamma$))) = *select* $E \cdot \varrho_1 \odot$
$\gamma$
**proof**−
  **have** $E \cdot \gamma_1 = E \cdot \varrho_1 \odot \gamma$
    **using** $\gamma_1$-$\gamma$ *clause.subst-eq*
    **by** *fast*

  **moreover have** *select* $E \cdot \gamma_1$ = *select* $E \cdot \varrho_1 \cdot \gamma$
    **using** *clause.subst-eq* $\gamma_1$-$\gamma$ *select-vars-subset*
    **by** (*metis clause.comp-subst.left.monoid-action-compatibility*)

  **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** $\gamma_2$-$\gamma$ *clause.subst-eq*
    **by** *fast*

  **moreover have** *select* $D \cdot \gamma_2$ = *select* $D \cdot \varrho_2 \cdot \gamma$
    **using** *clause.subst-eq* $\gamma_2$-$\gamma$ *select-vars-subset*
    **by** (*metis clause.comp-subst.left.monoid-action-compatibility*)

**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** *clause.welltyped-ground-instances* $(E, \mathcal{V}_1) \cup$ *clause.welltyped-ground-instances* $(D, \mathcal{V}_2) \subseteq$
$\bigcup$ (*clause.welltyped-ground-instances* ‘ $N$)
**using** *E-in-N D-in-N*
**by** *blast*

**then have** $\iota_G$*-not-redundant*:
$\iota_G \notin$ *ground.GRed-I*
(*clause.welltyped-ground-instances* $(E, \mathcal{V}_1) \cup$ *clause.welltyped-ground-instances* $(D, \mathcal{V}_2)$)
**using** *not-redundant ground.Red-I-of-subset*
**by** *blast*

**obtain** $C'$ $\mathcal{V}_3$ **where**
*superposition*: *superposition* $(D, \mathcal{V}_2)$ $(E, \mathcal{V}_1)$ $(C', \mathcal{V}_3)$ **and**
*inference-groundings*: $\iota_G \in$ *inference-ground-instances* (*Infer* $[(D, \mathcal{V}_2), (E, \mathcal{V}_1)]$ $(C', \mathcal{V}_3)$) **and**
$C'$-$\gamma$-$C$-$\gamma$: $C' \cdot \gamma = C \cdot \gamma$
**using**
*superposition-lifting*[*OF*
*ground-superposition*[*unfolded* $D_G$ $E_G$ $C_G$]
$\varrho_1$ $\varrho_2$
*rename-apart*
*E-grounding D-grounding C-grounding*
*select-from-E select-from-D*
*E-is-welltyped D-is-welltyped*
$\varrho_1$-$\gamma$*-is-welltyped* $\varrho_2$-$\gamma$*-is-welltyped*
$\varrho_1$*-is-welltyped* $\varrho_2$*-is-welltyped*
$\mathcal{V}_1$ $\mathcal{V}_2$
$\iota_G$*-not-redundant*[*unfolded* $\iota_G$ $D_G$ $E_G$ $C_G$]
]
**unfolding** $\iota_G$ $C_G$ $E_G$ $D_G$ **.**

**let** $?\iota$ = *Infer* $[(D, \mathcal{V}_2), (E, \mathcal{V}_1)]$ $(C', \mathcal{V}_3)$

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

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

**lemma** *ground-instances*:
**assumes** *not-redundant*: $\iota_G \notin$ *ground.Red-I* (⋃ (*clause.welltyped-ground-instances* ‘ *N*))
**obtains** *ι* **where**
*ι* ∈ *Inf-from N*
$\iota_G \in$ *inference-ground-instances ι*
**proof**−
**consider**
(*superposition*) $\iota_G \in$ *ground.superposition-inferences* |
(*eq-resolution*) $\iota_G \in$ *ground.eq-resolution-inferences* |
(*eq-factoring*) $\iota_G \in$ *ground.eq-factoring-inferences*
**using** $\iota_G$*-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** *superposition*

**then show** *?thesis*
**using** *that superposition-ground-instance not-redundant*
**by** *blast*
**next**
**case** *eq-resolution*

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

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

**end**

**end**

**context** *superposition-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-superposition-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* $\bot_F$ *inferences entails-*$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
**proof**(*unfold static-empty-ord-inter-equiv-static-inter*,
    *rule stat-ref-comp-to-non-ground-fam-inter*,
    *rule ballI*)
  **fix** $select_G$
  **assume** $select_G \in select_{Gs}$
  **then interpret** *grounded-superposition-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** $\bigwedge N.\ \exists\, select_G \in select_{Gs}.$ *ground-Inf-overapproximated* $select_G\ N$
**using** *overapproximation*
**unfolding** $select_{Gs}$*-def*
**by** (*smt* (*verit*, *best*) *mem-Collect-eq*)
**qed**

**end**

**end**
**theory** *Superposition-Soundness*
**imports**
*First-Order-Clause.Nonground-Entailment*

*Grounded-Superposition*
*Superposition-Welltypedness-Preservation*
**begin**

## 3.3 Soundness

**context** *grounded-superposition-calculus*
**begin**

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

**lemma** *eq-resolution-sound*:
**assumes** *eq-resolution*: *eq-resolution* $D\ C$
**shows** $\{D\} \models_F \{C\}$
**using** *eq-resolution*
**proof** (*cases* $D\ C$ *rule*: *eq-resolution.cases*)
**case** (*eq-resolutionI* $D\ l\ D'\ t\ t'\ \mathcal{V}\ \mu\ C$)

{
**fix** $I$ :: *'f ground-term rel* **and** $\gamma$ :: *('f, 'v) subst*

**let** *?I* = *upair* ' $I$

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

**obtain** $\gamma'$ **where**
$\gamma'$*-is-ground-subst*: *term-subst.is-ground-subst* $\gamma'$ **and**
$\gamma'$*-is-welltyped*: *term.subst.is-welltyped* $\mathcal{V}\ \gamma'$ **and**
$\gamma'$-$\gamma$: $\forall\, x \in$ *clause.vars* $C.\ \gamma\ x = \gamma'\ x$

```
    using clause.is-welltyped.ground-subst-extension[OF C-is-ground γ-is-welltyped].

  let ?D_G = clause.to-ground (D · μ · γ')
  let ?l_G = literal.to-ground (l ·l μ ·l γ')
  let ?D_G' = clause.to-ground (D' · μ · γ')
  let ?t_G = term.to-ground (t ·t μ ·t γ')
  let ?t_G' = term.to-ground (t' ·t μ ·t γ')

  have μ-is-welltyped: term.subst.is-welltyped-on (clause.vars D) 𝒱 μ
    using eq-resolutionI
    by meson

  have ?D_G ∈ clause.welltyped-ground-instances (D, 𝒱)
  proof(unfold clause.welltyped-ground-instances-def mem-Collect-eq fst-conv snd-conv,
      intro exI conjI 𝒱)
    show clause.to-ground (D · μ · γ') = clause.to-ground (D · μ ⊙ γ')
      by simp
  next
    show clause.is-ground (D · μ ⊙ γ')
      using γ'-is-ground-subst clause.is-ground-subst-is-ground
      by auto
  next
    show clause.is-welltyped 𝒱 D
     using C-is-welltyped
     unfolding
        eq-resolution-preserves-typing[OF eq-resolution[unfolded eq-resolutionI(1,
2)]].
  next
    show term.subst.is-welltyped-on (clause.vars D) 𝒱 (μ ⊙ γ')
      using γ'-is-welltyped μ-is-welltyped
      by (simp add: subst-compose-def)
  qed

  then have ?I ⊫ ?D_G
    using entails-ground-instances
    by auto

  then obtain l_G where l_G-in-D: l_G ∈# ?D_G and I-models-l_G: ?I ⊫l l_G
    by (auto simp: true-cls-def)

  have l_G ≠ ?l_G
  proof(rule notI)
    assume l_G = ?l_G

    then have [simp]: l_G = ?t_G !≈ ?t_G'
      unfolding eq-resolutionI
      by simp

    moreover have atm-of l_G ∈ ?I
```

**proof**−
**have** *?t*$_G$ = *?t*$_G$*′*
**using** *eq-resolutionI*(*5*) *term-subst.is-imgu-unifies-pair*
**by** *metis*

**then show** *?thesis*
**using** *reflD*[*OF refl-I*, *of ?t*$_G$]
**by** *auto*
**qed**

**ultimately show** *False*
**using** *I-models-l*$_G$
**by** *auto*
**qed**

**then have** $l_G$ ∈# *clause.to-ground* ($C \cdot \gamma'$)
**using** $l_G$*-in-D*
**unfolding** *eq-resolutionI*
**by** *simp*

**then have** *?I* ⊫ *clause.to-ground* ($C \cdot \gamma$)
**using** *clause.subst-eq*[*OF* $\gamma'$-$\gamma$[*rule-format*]] *I-models-l*$_G$
**by** *auto*
**}**

**then show** *?thesis*
**unfolding**
*true-clss-def*
*eq-resolutionI*(*1*,*2*)
*clause.welltyped-ground-instances-def*
*ground.G-entails-def*
**by** *auto*
**qed**

**lemma** *eq-factoring-sound*:
**assumes** *eq-factoring*: *eq-factoring D C*
**shows** {*D*} ⊫$_F$ {*C*}
**using** *eq-factoring*
**proof** (*cases D C rule*: *eq-factoring.cases*)
**case** (*eq-factoringI D* $l_1$ $l_2$ $D'$ $t_1$ $t_1'$ $t_2$ $t_2'$ $\mu$ $\mathcal{V}$ *C*)

**{**
**fix** *I* :: *′f ground-term rel* **and** $\gamma$ :: (*′f*, *′v*) *subst*

**let** *?I* = *upair* ‘ *I*

**assume**
*trans-I*: *trans I* **and**
*sym-I*: *sym I* **and**

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

**obtain** $\gamma'$ **where**
$\gamma'$*-is-ground-subst*: *term-subst.is-ground-subst* $\gamma'$ **and**
$\gamma'$*-is-welltyped*: *term.subst.is-welltyped* $\mathcal{V}$ $\gamma'$ **and**
$\gamma'$-$\gamma$: $\forall x \in$ *clause.vars* $C$. $\gamma\ x = \gamma'\ x$
**using** *clause.is-welltyped.ground-subst-extension*[*OF C-is-ground* $\gamma$*-is-welltyped*].

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

**have** $\mu$*-is-welltyped*: *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}$ $\mu$
**using** *eq-factoringI*(*9*)
**by** *blast*

**have** $?D_G \in$ *clause.welltyped-ground-instances* $(D, \mathcal{V})$
**proof**(*unfold clause.welltyped-ground-instances-def mem-Collect-eq fst-conv snd-conv*, *intro exI conjI* $\mathcal{V}$)
**show** *clause.to-ground* $(D \cdot \mu \cdot \gamma')$ = *clause.to-ground* $(D \cdot \mu \odot \gamma')$
**by** *simp*
**next**
**show** *clause.is-ground* $(D \cdot \mu \odot \gamma')$
**using** $\gamma'$*-is-ground-subst clause.is-ground-subst-is-ground*
**by** *auto*
**next**
**show** *clause.is-welltyped* $\mathcal{V}$ $D$
**using** *C-is-welltyped*
**unfolding** *eq-factoring-preserves-typing*[*OF eq-factoring*[*unfolded eq-factoringI*(*1*, *2*)]].
**next**
**show** *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}$ $(\mu \odot \gamma')$
**using** $\mu$*-is-welltyped* $\gamma'$*-is-welltyped*
**by** (*simp add*: *subst-compose-def*)
**qed**

**then have** $?I \models ?D_G$
**using** *entails-ground-instances*

**by** *blast*

**then obtain** $l_G$ **where** $l_G$*-in-*$D_G$: $l_G \in\#\ ?D_G$ **and** *I-models-*$l_G$: $?I \models l\ l_G$
**by** (*auto simp*: *true-cls-def*)

**have** [*simp*]: $?t_{G2} = ?t_{G1}$
**using** *eq-factoringI*(*9*) *term-subst.is-imgu-unifies-pair*
**by** *metis*

**have** [*simp*]: $?l_{G1} = ?t_{G1} \approx ?t_{G1}'$
**unfolding** *eq-factoringI*
**by** *simp*

**have** [*simp*]: $?l_{G2} = ?t_{G2} \approx ?t_{G2}'$
**unfolding** *eq-factoringI*
**by** *simp*

**have** [*simp*]: $?C_G = add\text{-}mset\ (?t_{G1} \approx ?t_{G2}')\ (add\text{-}mset\ (?t_{G1}'\ !{\approx}\ ?t_{G2}')\ ?D_G')$
**unfolding** *eq-factoringI*
**by** *simp*

**have** $?I \models clause.to\text{-}ground\ (C \cdot \gamma)$
**proof**(*cases* $l_G = ?l_{G1} \lor l_G = ?l_{G2}$)
**case** *True*

**then have** $?I \models l\ ?t_{G1} \approx ?t_{G1}' \lor ?I \models l\ ?t_{G1} \approx ?t_{G2}'$
**using** *I-models-*$l_G$ *sym-I*
**by**(*auto elim*: *symE*)

**then have** $?I \models l\ ?t_{G1} \approx ?t_{G2}' \lor ?I \models l\ ?t_{G1}'\ !{\approx}\ ?t_{G2}'$
**using** *sym-I trans-I*
**by**(*auto dest*: *transD*)

**then show** *?thesis*
**using** *clause.subst-eq*[*OF* $\gamma'$-$\gamma$[*rule-format*]] *sym-I*
**by** *auto*
**next**
**case** *False*

**then have** $l_G \in\#\ ?D_G'$
**using** $l_G$*-in-*$D_G$
**unfolding** *eq-factoringI*
**by** *simp*

**then have** $l_G \in\#\ clause.to\text{-}ground\ (C \cdot \gamma)$
**using** *clause.subst-eq*[*OF* $\gamma'$-$\gamma$[*rule-format*]]
**by** *simp*

**then show** *?thesis*

```
      using I-models-l_G
      by blast
    qed
  }

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

lemma superposition-sound:
  assumes superposition: superposition D E C
  shows {E, D} ⊫_F {C}
  using superposition
proof (cases D E C rule: superposition.cases)
  case (superpositionI 𝒱₁ 𝒱₂ ϱ₁ ϱ₂ E D l₁ E' l₂ D' 𝒫 c₁ t₁ t₁' t₂ t₂' 𝒱₃ μ C)

  {
    fix I :: 'f gterm rel and γ :: 'v ⇒ ('f, 'v) Term.term

    let ?I = (λ(x, y). Upair x y) ` I

    assume
      refl-I: refl I and
      trans-I: trans I and
      sym-I: sym I and
      compatible-with-ground-context-I: compatible-with-gctxt I and
     E-entails-ground-instances: ∀ E_G ∈ clause.welltyped-ground-instances (E, 𝒱₁).
?I ⊨ E_G and
     D-entails-ground-instances: ∀ D_G ∈ clause.welltyped-ground-instances (D, 𝒱₂).
?I ⊨ D_G and
      C-is-ground: clause.is-ground (C · γ) and
      C-is-welltyped: clause.is-welltyped 𝒱₃ C and
      γ-is-welltyped: term.subst.is-welltyped-on (clause.vars C) 𝒱₃ γ

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

    let ?E_G = clause.to-ground (E · ϱ₁ · μ · γ')
    let ?D_G = clause.to-ground (D · ϱ₂ · μ · γ')

    let ?l_G1 = literal.to-ground (l₁ ·l ϱ₁ ·l μ ·l γ')
```

**let** $?l_{G2}$ = *literal.to-ground* $(l_2 \cdot l\ \varrho_2 \cdot l\ \mu \cdot l\ \gamma')$

**let** $?E_G{}'$ = *clause.to-ground* $(E' \cdot \varrho_1 \cdot \mu \cdot \gamma')$
**let** $?D_G{}'$ = *clause.to-ground* $(D' \cdot \varrho_2 \cdot \mu \cdot \gamma')$

**let** $?c_{G1}$ = *context.to-ground* $(c_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')$
**let** $?t_{G1}$ = *term.to-ground* $(t_1 \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$
**let** $?t_{G1}{}'$ = *term.to-ground* $(t_1{}' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$
**let** $?t_{G2}$ = *term.to-ground* $(t_2 \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma')$
**let** $?t_{G2}{}'$ = *term.to-ground* $(t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma')$

**let** $?\mathcal{P}_G$ = *if* $\mathcal{P}$ = *Pos then Pos else Neg*

**let** $?C_G$ = *clause.to-ground* $(C \cdot \gamma')$

**have** $\mathcal{P}$*-subst* [*simp*]: $\bigwedge a\ \sigma.\ \mathcal{P}\ a \cdot l\ \sigma = \mathcal{P}\ (a \cdot a\ \sigma)$
  **using** *superpositionI*(*11*)
  **by** *auto*

**have** [*simp*]: $\bigwedge \mathcal{V}\ a$. *literal.is-welltyped* $\mathcal{V}$ $(\mathcal{P}\ a) \longleftrightarrow$ *atom.is-welltyped* $\mathcal{V}$ $a$
  **using** *superpositionI*(*11*)
  **by** (*auto simp*: *literal-is-welltyped-iff-atm-of*)

**have** [*simp*]: $\bigwedge a$. *literal.vars* $(\mathcal{P}\ a)$ = *atom.vars* $a$
  **using** *superpositionI*(*11*)
  **by** *auto*

**have** $\mu$-$\gamma'$*-is-ground-subst*:
  *term-subst.is-ground-subst* $(\mu \odot \gamma')$
  **using** *term.is-ground-subst-comp-right*[*OF* $\gamma'$*-is-ground-subst*]**.**

**have** $\mu$*-is-welltyped*:
  *term.subst.is-welltyped-on* (*clause.vars* $(E \cdot \varrho_1) \cup$ *clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3$ $\mu$
  **using** *superpositionI*(*15*)
  **by** *blast*

**have** *D-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_2$ $D$
  **using** *superposition-preserves-typing-D*[*OF*
    *superposition*[*unfolded superpositionI*(*1*−*3*)]
    *C-is-welltyped*]**.**

**have** *E-is-welltyped*: *clause.is-welltyped* $\mathcal{V}_1$ $E$
  **using** *superposition-preserves-typing-E*[*OF*
    *superposition*[*unfolded superpositionI*(*1*−*3*)]
    *C-is-welltyped*]**.**

**have** *is-welltyped*-$\mu$-$\gamma$:
  *term.subst.is-welltyped-on* (*clause.vars* $(E \cdot \varrho_1) \cup$ *clause.vars* $(D \cdot \varrho_2)$) $\mathcal{V}_3$ $(\mu$
$\odot\ \gamma')$

**using** $\gamma'$-*is-welltyped* $\mu$-*is-welltyped*
**by** (*simp add*: *term.welltyped.typed-subst-compose*)

**note** *is-welltyped*-$\varrho$-$\mu$-$\gamma$ = *term.welltyped.renaming-ground-subst*[*OF* - - - $\mu$-$\gamma'$-*is-ground-subst*]

**have** ?$E_G$ $\in$ *clause.welltyped-ground-instances* ($E$, $\mathcal{V}_1$)
**proof**(
*unfold clause.welltyped-ground-instances-def mem-Collect-eq fst-conv snd-conv*,
*intro exI conjI E-is-welltyped superpositionI*)

**show** *clause.to-ground* ($E \cdot \varrho_1 \cdot \mu \cdot \gamma'$) = *clause.to-ground* ($E \cdot \varrho_1 \odot \mu \odot \gamma'$)
**by** *simp*
**next**

**show** *clause.is-ground* ($E \cdot \varrho_1 \odot \mu \odot \gamma'$)
**using** $\gamma'$-*is-ground-subst clause.is-ground-subst-is-ground*
**by** *auto*
**next**

**show** *term.subst.is-welltyped-on* (*clause.vars* $E$) $\mathcal{V}_1$ ($\varrho_1 \odot \mu \odot \gamma'$)
**using**
*is-welltyped*-$\mu$-$\gamma$
*is-welltyped*-$\varrho$-$\mu$-$\gamma$[*OF*
*superpositionI*(*6*) - *superpositionI*(*18*, *16*)[*unfolded clause.vars-subst*]]
**by** (*simp add*: *subst-compose-assoc clause.vars-subst*)
**qed**

**then have** *entails*-$E_G$: ?$I \models$ ?$E_G$
**using** *E-entails-ground-instances*
**by** *blast*

**have** ?$D_G$ $\in$ *clause.welltyped-ground-instances* ($D$, $\mathcal{V}_2$)
**proof**(
*unfold clause.welltyped-ground-instances-def mem-Collect-eq fst-conv snd-conv*,
*intro exI conjI D-is-welltyped superpositionI*)

**show** *clause.to-ground* ($D \cdot \varrho_2 \cdot \mu \cdot \gamma'$) = *clause.to-ground* ($D \cdot \varrho_2 \odot \mu \odot \gamma'$)
**by** *simp*
**next**
**show** *clause.is-ground* ($D \cdot \varrho_2 \odot \mu \odot \gamma'$)
**using** $\gamma'$-*is-ground-subst clause.is-ground-subst-is-ground*
**by** *auto*
**next**

**show** *term.subst.is-welltyped-on* (*clause.vars* $D$) $\mathcal{V}_2$ ($\varrho_2 \odot \mu \odot \gamma'$)
**using**
*is-welltyped*-$\mu$-$\gamma$
*is-welltyped*-$\varrho$-$\mu$-$\gamma$[*OF*
*superpositionI*(*7*) - *superpositionI*(*19*, *17*)[*unfolded clause.vars-subst*]]

```
      by (simp add: subst-compose-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(cases ?I ⊫l literal.to-ground (𝒫 (Upair (c₁ ·t_c ϱ₁)⟨t₂' ·t ϱ₂⟩ (t₁' ·t ϱ₁))
·l μ ·l γ'))
      case True
      then show ?thesis
        unfolding superpositionI
        by simp
    next
      case False

      have imgu: term.is-imgu μ {{t₁ ·t ϱ₁, t₂ ·t ϱ₂}}
        using superpositionI(15)
        by blast

      interpret clause-entailment I
        by unfold-locales (rule trans-I sym-I compatible-with-ground-context-I)+

      note unfolds =
        superpositionI
        context.safe-unfolds
        clause-safe-unfolds
        literal-entails-unfolds
        term.is-imgu-unifies-pair[OF imgu]

      from literal-cases[OF superpositionI(11)]
      have ¬ ?I ⊫l ?l_G1 ∨ ¬ ?I ⊫l ?l_G2
      proof cases
        case Pos: 1

        show ?thesis
          using False symmetric-upair-context-congruence
          unfolding Pos unfolds
          by blast
      next
        case Neg: 2

        show ?thesis
          using False symmetric-upair-context-congruence
          unfolding Neg unfolds
          by blast
      qed
```

**then have** *?I* ⊫ *?E*$_G$′ ∨ *?I* ⊫ *?D*$_G$′
**using** *entails-D*$_G$ *entails-E*$_G$
**unfolding** *superpositionI*
**by** *auto*

**then show** *?thesis*
**unfolding** *superpositionI*
**by** *simp*
**qed**

**then have** *?I* ⊫ *clause.to-ground* (*C* · *γ*)
**by** (*metis γ′-γ clause.subst-eq*)
**}**

**then show** *?thesis*
**unfolding**
*ground.G-entails-def clause.welltyped-ground-instances-def true-clss-def superpositionI*(*1*−*3*)
**by** *auto*
**qed**

**end**

**sublocale** *grounded-superposition-calculus* ⊆ *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**
*eq-factoring-sound*
*eq-resolution-sound*
*superposition-sound*
**unfolding** *inferences-def ground.G-entails-def*
**by** *auto*
**qed**

**sublocale** *superposition-calculus* ⊆ *sound-inference-system inferences* ⊥$_F$ *entails-*$\mathcal{G}$
**proof** *unfold-locales*

**obtain** *select*$_G$ **where** *select*$_G$: *select*$_G$ ∈ *select*$_{Gs}$
**using** *Q-nonempty* **by** *blast*

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

**fix** $\iota$
**assume** $\iota \in$ *inferences*

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

**end**

# 4 Integration of IsaFoR Terms and the Knuth–Bendix Order

This theory implements the abstract interface for atoms and substitutions using the IsaFoR library.

**theory** *IsaFoR-Term-Copy*
**imports**
*First-Order-Terms.Unification*
*HOL−Cardinals.Wellorder-Extension*
*Knuth-Bendix-Order.KBO*
**begin**

This part extends and integrates and the Knuth–Bendix order defined in IsaFoR.

**record** *'f weights* =
*w* :: *'f* × *nat* ⇒ *nat*
*w0* :: *nat*
*pr-strict* :: *'f* × *nat* ⇒ *'f* × *nat* ⇒ *bool*
*least* :: *'f* ⇒ *bool*
*scf* :: *'f* × *nat* ⇒ *nat* ⇒ *nat*

**class** *weighted* =
**fixes** *weights* :: *'a weights*
**assumes** *weights-adm*:
*admissible-kbo*
(*w weights*) (*w0 weights*) (*pr-strict weights*) ((*pr-strict weights*)$^{==}$) (*least weights*) (*scf weights*)
**and** *pr-strict-total*: *fi* = *gj* ∨ *pr-strict weights fi gj* ∨ *pr-strict weights gj fi*
**and** *pr-strict-asymp*: *asymp* (*pr-strict weights*)
**and** *scf-ok*: $i < n \Longrightarrow$ *scf weights* (*f*, *n*) $i \leq 1$

**instantiation** *unit* :: *weighted* **begin**

**definition** *weights-unit* :: *unit weights* **where** *weights-unit* =
(|*w* = *Suc* ∘ *snd*, *w0* = *1*, *pr-strict* = λ(-, *n*) (-, *m*). *n* > *m*, *least* = λ-. *True*, *scf* = λ- -. *1*|)

**instance**
**by** (*intro-classes*, *unfold-locales*) (*auto simp*: *weights-unit-def SN-iff-wf irreflp-def intro*!: *wf-subset*[*OF wf-inv-image*[*OF wf*], *of - snd*])
**end**

**global-interpretation** *KBO*:
*admissible-kbo*
*w* (*weights* :: *'f* :: *weighted weights*) *w0* (*weights* :: *'f* :: *weighted weights*)
*pr-strict weights* ((*pr-strict weights*)$^{==}$) *least weights scf weights*
**defines** *weight* = *KBO.weight*
**and** *kbo* = *KBO.kbo*
**by** (*simp add*: *weights-adm*)

**lemma** *kbo-code*[*code*]: *kbo s t* =
(*let wt* = *weight t*; *ws* = *weight s in*
*if vars-term-ms* (*KBO.SCF t*) ⊆# *vars-term-ms* (*KBO.SCF s*) ∧ *wt* ≤ *ws*
*then*
(*if wt* < *ws then* (*True*, *True*)
*else*
(*case s of*
*Var y* ⇒ (*False*, *case t of Var x* ⇒ *True* | *Fun g ts* ⇒ *ts* = [] ∧ *least weights g*)
| *Fun f ss* ⇒
(*case t of*
*Var x* ⇒ (*True*, *True*)
| *Fun g ts* ⇒
*if pr-strict weights* (*f*, *length ss*) (*g*, *length ts*) *then* (*True*, *True*)
*else if* (*f*, *length ss*) = (*g*, *length ts*) *then lex-ext-unbounded kbo ss ts*
*else* (*False*, *False*))))
*else* (*False*, *False*))
**by** (*subst KBO.kbo.simps*) (*auto simp*: *Let-def split*: *term.splits*)

**definition** *less-kbo s t* = *fst* (*kbo t s*)

**lemma** *less-kbo-gtotal*: *ground s* ⟹ *ground t* ⟹ *s* = *t* ∨ *less-kbo s t* ∨ *less-kbo t s*
**unfolding** *less-kbo-def* **using** *KBO.S-ground-total* **by** (*metis pr-strict-total subset-UNIV*)

**lemma** *less-kbo-subst*:
**fixes** σ :: (*'f* :: *weighted*, *'v*) *subst*
**shows** *less-kbo s t* ⟹ *less-kbo* (*s* · σ) (*t* · σ)
**unfolding** *less-kbo-def* **by** (*rule KBO.S-subst*)

**lemma** *wfP-less-kbo*: *wfP less-kbo*
**proof** −
**have** *SN* {(*x*, *y*). *fst* (*kbo x y*)}
**using** *pr-strict-asymp* **by** (*fastforce simp*: *asympI irreflp-def intro*!: *KBO.S-SN*

```
scf-ok)
  then show ?thesis
    unfolding SN-iff-wf wfp-def by (rule wf-subset) (auto simp: less-kbo-def)
qed

end
theory Superposition-Example
  imports
    Superposition
    IsaFoR-Term-Copy
    VeriComp.Well-founded
begin

sublocale nonground-term-with-context ⊆
  nonground-term-order less-kbo :: ('f :: weighted,'v) term ⇒ ('f,'v) term ⇒ bool
proof unfold-locales
   show transp less-kbo
    using KBO.S-trans
    unfolding transp-def less-kbo-def
    by blast
next
  show asymp less-kbo
    using wfp-imp-asymp wfP-less-kbo
    by blast
next
  show wfp-on (range term.from-ground) less-kbo
    using wfp-on-subset[OF wfP-less-kbo subset-UNIV] .
next
  show totalp-on (range term.from-ground) less-kbo
    using less-kbo-gtotal
   unfolding totalp-on-def Term.ground-vars-term-empty term.is-ground-iff-range-from-ground
    by blast
next
  fix
    c :: ('f, 'v) context and
    t1 t2 :: ('f, 'v) term

  assume less-kbo t1 t2

  then show less-kbo c⟨t1⟩ c⟨t2⟩
    using KBO.S-ctxt less-kbo-def
    by blast
next
  fix
    t1 t2 :: ('f, 'v) term and
    γ :: ('f, 'v) subst

  assume less-kbo t1 t2
```

**then show** *less-kbo* $(t_1 \cdot t\ \gamma)$ $(t_2 \cdot t\ \gamma)$
**using** *less-kbo-subst* **by** *blast*
**next**
**fix**
$t$ :: $('f,\ 'v)$ *term* **and**
$c$ :: $('f,\ 'v)$ *context*
**assume**
*term.is-ground t*
*context.is-ground c*
$c \neq \square$

**then show** *less-kbo t* $c\langle t\rangle$
**by** (*simp add*: *KBO.S-supt less-kbo-def nectxt-imp-supt-ctxt*)
**qed**

**abbreviation** *trivial-tiebreakers* ::
$'f$ *gatom clause* $\Rightarrow$ $('f,'v)$ *atom clause* $\Rightarrow$ $('f,'v)$ *atom clause* $\Rightarrow$ *bool* **where**
*trivial-tiebreakers* - - - $\equiv$ *False*

**lemma** *trivial-tiebreakers*: *wellfounded-strict-order* (*trivial-tiebreakers* $C_G$)
**by** *unfold-locales auto*

**locale** *trivial-superposition-example* =
*ground-critical-pair-theorem* *TYPE*$('f$ :: *weighted*)
**begin**

**sublocale** *nonground-term-with-context*.

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

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

**sublocale** *selection-function trivial-select*
**by** *unfold-locales auto*

**sublocale**
*superposition-calculus*
*trivial-select* :: $('f\ ,\ 'v$ :: *infinite*) *select*
*less-kbo*
*unit-types*
*trivial-tiebreakers*
**by** *unfold-locales* (*auto simp*: *UNIV-unit*)

**end**

```
context nonground-equality-order
begin

abbreviation select-max where
  select-max C ≡
    if ∃ l∈#C. is-maximal l C ∧ is-neg l
    then {#SOME l. is-maximal l C ∧ is-neg l#}
    else {#}

sublocale select-max: selection-function select-max
proof unfold-locales
  fix C

  {
    assume ∃ l∈#C. is-maximal l C ∧ is-neg l

    then have ∃ l. is-maximal l C ∧ is-neg l
      by blast

    then have (SOME l. is-maximal l C ∧ is-neg l) ∈# C
      by(rule someI2-ex) (simp add: maximal-in-clause)
  }

  then show select-max C ⊆# C
    by auto
next
  fix C l

  {
    assume ∃ l∈#C. is-maximal l C ∧ is-neg l

    then have ∃ l. is-maximal l C ∧ is-neg l
      by blast

    then have is-neg (SOME l. is-maximal l C ∧ is-neg l)
      by (rule someI2-ex) simp
  }

  then show l ∈# select-max C ⟹ is-neg l
   by (smt (verit, ccfv-threshold) ex-in-conv set-mset-add-mset-insert set-mset-eq-empty-iff

        singletonD)
qed

end


datatype type = A | B
```

**lemma** *UNIV-type* [*simp*]: (*UNIV* :: *type set*) = {*A*, *B*}
  **using** *type.exhaust* **by** *blast*

**lemma** *UNIV-type-ordLeq-UNIV-nat*: |*UNIV* :: *type set*| ≤*o* |*UNIV* :: *nat set*|
  **by** (*simp add*: *ordLeq3-finite-infinite*)

**definition** *pr-strict* :: (′*f* :: *wellorder* × *nat*) ⇒ - ⇒ *bool* **where**
  *pr-strict* = *lex-prodp* ((<) :: ′*f* ⇒ ′*f* ⇒ *bool*) ((<) :: *nat* ⇒ *nat* ⇒ *bool*)

**lemma** *wfp-pr-strict*: *wfp pr-strict*
  **by** (*simp add*: *lex-prodp-wfP pr-strict-def*)

**lemma** *transp-pr-strict*: *transp pr-strict*
**proof** (*rule transpI*)
  **show** ⋀*x y z*. *pr-strict x y* ⟹ *pr-strict y z* ⟹ *pr-strict x z*
    **unfolding** *pr-strict-def lex-prodp-def*
    **by** *force*
**qed**

**definition** *least* **where**
  *least x* ⟷ (∀ *y*. *x* ≤ *y*)

**definition** *weight* :: ′*f* × *nat* ⇒ *nat* **where**
  *weight p* = *1*

**abbreviation** *weights* **where** *weights* ≡
  ⦇*w* = *weight*, *w0* = *1*, *pr-strict* = *pr-strict*$^{-1-1}$, *least* = *least*, *scf* = λ- -. *1*⦈

**interpretation** *weighted weights*
**proof** (*unfold-locales*, *unfold weights.select-convs weight-def least-def pr-strict-def lex-prodp-def*)

  **show** *SN* {(*fn* :: (′*b* :: *wellorder*) × *nat*, *gm*).
          (λ*x y*. *fst x* < *fst y* ∨ *fst x* = *fst y* ∧ *snd x* < *snd y*)$^{-1-1}$ *fn gm*}
  **proof** (*fold lex-prodp-def pr-strict-def*, *rule wf-imp-SN*)
    **show** *wf* ({(*fn*, *gm*). *pr-strict*$^{-1-1}$ *fn gm*}$^{-1}$)
      **using** *wfp-pr-strict*
      **by** (*simp add*: *wfp-pr-strict converse-def wfp-def*)
  **qed**
**qed** (*auto simp*: *order.order-iff-strict*)

**instantiation** *nat* :: *weighted* **begin**

**definition** *weights-nat* :: *nat weights* **where** *weights-nat* ≡ *weights*

**instance**
  **using** *weights-adm pr-strict-total pr-strict-asymp*
  **by** (*intro-classes*, *unfold weights-nat-def*) *auto*

**end**

**instantiation** *nat* :: *infinite* **begin**

**instance**
  **by** *intro-classes simp*

**end**

**fun** *repeat* :: *nat* ⇒ *'a* ⇒ *'a list* **where**
  *repeat 0 - =* []
| *repeat* (*Suc n*) *x* = *x* # *repeat n x*

**abbreviation** *types* :: *nat* ⇒ *type list* × *type* **where**
  *types n* ≡
    *let type* = *if even n then A else B*
    *in* (*repeat* (*n div 2*) *type*, *type*)

**lemma** *types-inhabited*: ∃*f*. *types f* = ([], *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**

**locale** *superposition-example* =
  *ground-critical-pair-theorem TYPE*(*nat*)
**begin**

**sublocale** *wellfounded-strict-order trivial-tiebreakers* $C_G$
  **using** *trivial-tiebreakers*.

**sublocale** *nonground-term-with-context* .

**sublocale** *nonground-equality-order less-kbo*
  **by** *unfold-locales*

**sublocale**
  *superposition-calculus*
    *select-max* :: (*nat*, *nat*) *select*
    *less-kbo*
    *types*
    *trivial-tiebreakers*

**proof** *unfold-locales*
  **fix** $\tau$
  **show** $\exists f.$ *types* $f = ([], \tau)$
    **using** *types-inhabited* **.**
**next**
  **show** $|UNIV :: type\ set| \leq o\ |UNIV :: nat\ set|$
    **using** *UNIV-type-ordLeq-UNIV-nat* **.**
**qed**

**end**

**end**

# References


[1] M. Desharnais, B. Toth, U. Waldmann, J. Blanchette, and S. Tourret. A Modular Formalization of Superposition in Isabelle/HOL. In Y. Bertot, T. Kutsia, and M. Norrish, editors, *15th International Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.