A Comprehensive Framework for Saturation Theorem Proving

Sophie Tourret

March 19, 2025

Abstract

This Isabelle/HOL formalization is the companion of the technical report "A comprehensive framework for saturation theorem proving", itself companion of the eponym IJCAR 2020 paper, written by Uwe Waldmann, Sophie Tourret, Simon Robillard and Jasmin Blanchette. It verifies a framework for formal refutational completeness proofs of abstract provers that implement saturation calculi, such as ordered resolution or superposition, and allows to model entire prover architectures in such a way that the static refutational completeness of a calculus immediately implies the dynamic refutational completeness of a prover implementing the calculus using a variant of the given clause loop.

The technical report "A comprehensive framework for saturation theorem proving" is available at http://matryoshka.gforge.inria.fr/pubs/satur_report.pdf. The names of the Isabelle lemmas and theorems corresponding to the results in the report are indicated in the margin of the report.

Contents

1 Calculi Based on a Redundancy Criterion

This section introduces the most basic notions upon which the framework is built: consequence relations and inference systems. It also defines the notion of a family of consequence relations and of redundancy criteria. This corresponds to sections 2.1 and 2.2 of the report.

theory *Calculus*
imports
Ordered-Resolution-Prover.Lazy-List-Liminf
Ordered-Resolution-Prover.Lazy-List-Chain
begin

1.1 Consequence Relations

locale *consequence-relation* =
fixes
Bot :: *'f set* **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ‹⊨› *50*)
assumes
bot-not-empty: $Bot \neq \{\}$ **and**
bot-entails-all: $B \in Bot \Longrightarrow \{B\} \models N1$ **and**
subset-entailed: $N2 \subseteq N1 \Longrightarrow N1 \models N2$ **and**
all-formulas-entailed: $(\forall\, C \in N2.\ N1 \models \{C\}) \Longrightarrow N1 \models N2$ **and**
entails-trans[*trans*]: $N1 \models N2 \Longrightarrow N2 \models N3 \Longrightarrow N1 \models N3$
begin

lemma *entail-set-all-formulas*: $N1 \models N2 \longleftrightarrow (\forall\, C \in N2.\ N1 \models \{C\})$
by (*meson all-formulas-entailed empty-subsetI insert-subset subset-entailed entails-trans*)

lemma *entail-union*: $N \models N1 \wedge N \models N2 \longleftrightarrow N \models N1 \cup N2$
using *entail-set-all-formulas*[*of N N1*] *entail-set-all-formulas*[*of N N2*]
entail-set-all-formulas[*of N N1* ∪ *N2*] **by** *blast*

lemma *entail-unions*: $(\forall\, i \in I.\ N \models Ni\ i) \longleftrightarrow N \models \bigcup (Ni\ `\ I)$
using *entail-set-all-formulas*[*of N* $\bigcup$ (*Ni* ‘ *I*)] *entail-set-all-formulas*[*of N*]
Complete-Lattices.UN-ball-bex-simps(*2*)[*of Ni I* λ*C*. *N* ⊨ {*C*}, *symmetric*]
by *meson*

lemma *entail-all-bot*: $(\exists\, B \in Bot.\ N \models \{B\}) \Longrightarrow \forall\, B' \in Bot.\ N \models \{B'\}$
using *bot-entails-all entails-trans* **by** *blast*

lemma *entails-trans-strong*: $N1 \models N2 \Longrightarrow N1 \cup N2 \models N3 \Longrightarrow N1 \models N3$
by (*meson entail-union entails-trans order-refl subset-entailed*)

end

1.2 Families of Consequence Relations

locale *consequence-relation-family* =
fixes

Bot :: *'f set* **and**
Q :: *'q set* **and**
entails-q :: *'q* ⇒ *'f set* ⇒ *'f set* ⇒ *bool*
assumes
Q-nonempty: *Q* ≠ {} **and**
q-cons-rel: ∀ *q* ∈ *Q*. *consequence-relation Bot* (*entails-q q*)
begin

lemma *bot-not-empty*: *Bot* ≠ {}
using *Q-nonempty consequence-relation.bot-not-empty consequence-relation-family.q-cons-rel consequence-relation-family-axioms* **by** *blast*

definition *entails* :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ‹|=*Q*› *50*) **where**
N1 |=*Q N2* ⟷ (∀ *q* ∈ *Q*. *entails-q q N1 N2*)

lemma *intersect-cons-rel-family*: *consequence-relation Bot entails*
unfolding *consequence-relation-def entails-def*
by (*intro conjI bot-not-empty*) (*metis consequence-relation-def q-cons-rel*)+

end

1.3 Inference Systems

datatype *'f inference* =
Infer (*prems-of*: *'f list*) (*concl-of*: *'f*)

locale *inference-system* =
fixes
Inf :: ‹*'f inference set*›
begin

definition *Inf-from* :: *'f set* ⇒ *'f inference set* **where**
Inf-from N = {ι ∈ *Inf*. *set* (*prems-of* ι) ⊆ *N*}

definition *Inf-between* :: *'f set* ⇒ *'f set* ⇒ *'f inference set* **where**
Inf-between N M = *Inf-from* (*N* ∪ *M*) − *Inf-from* (*N* − *M*)

lemma *Inf-if-Inf-from*: ι ∈ *Inf-from N* ⟹ ι ∈ *Inf*
unfolding *Inf-from-def* **by** *simp*

lemma *Inf-if-Inf-between*: ι ∈ *Inf-between N M* ⟹ ι ∈ *Inf*
unfolding *Inf-between-def Inf-from-def* **by** *simp*

lemma *Inf-between-alt*:
Inf-between N M = {ι ∈ *Inf*. ι ∈ *Inf-from* (*N* ∪ *M*) ∧ *set* (*prems-of* ι) ∩ *M* ≠ {}}
unfolding *Inf-from-def Inf-between-def* **by** *auto*

lemma *Inf-from-mono*: *N* ⊆ *N'* ⟹ *Inf-from N* ⊆ *Inf-from N'*
unfolding *Inf-from-def* **by** *auto*

lemma *Inf-between-mono*: *N* ⊆ *N'* ⟹ *M* ⊆ *M'* ⟹ *Inf-between N M* ⊆ *Inf-between N' M'*
unfolding *Inf-between-alt* **using** *Inf-from-mono*[*of N* ∪ *M N'* ∪ *M'*] **by** *auto*

end

1.4 Families of Inference Systems

locale *inference-system-family* =
  fixes
    Q :: $'q$ *set* **and**
    Inf-q :: $'q \Rightarrow$ $'f$ *inference set*
  assumes
    Q-nonempty: $Q \neq \{\}$
begin

definition *Inf-from-q* :: $'q \Rightarrow$ $'f$ *set* $\Rightarrow$ $'f$ *inference set* **where**
  Inf-from-q q = *inference-system.Inf-from* (*Inf-q q*)

definition *Inf-between-q* :: $'q \Rightarrow$ $'f$ *set* $\Rightarrow$ $'f$ *set* $\Rightarrow$ $'f$ *inference set* **where**
  Inf-between-q q = *inference-system.Inf-between* (*Inf-q q*)

lemma *Inf-between-q-alt*:
  Inf-between-q q N M = $\{\iota \in$ *Inf-q q*. $\iota \in$ *Inf-from-q q* $(N \cup M) \wedge$ *set* (*prems-of* $\iota) \cap M \neq \{\}\}$
  unfolding *Inf-from-q-def Inf-between-q-def inference-system.Inf-between-alt* **by** *auto*

end

1.5 Calculi Based on a Single Redundancy Criterion

locale *calculus* = *inference-system Inf* + *consequence-relation Bot entails*
  for
    Bot :: $'f$ *set* **and**
    Inf :: $\langle'f$ *inference set*$\rangle$ **and**
    entails :: $'f$ *set* $\Rightarrow$ $'f$ *set* $\Rightarrow$ *bool* (**infix** $\langle\models\rangle$ *50*)
  + **fixes**
    Red-I :: $'f$ *set* $\Rightarrow$ $'f$ *inference set* **and**
    Red-F :: $'f$ *set* $\Rightarrow$ $'f$ *set*
  assumes
    Red-I-to-Inf: *Red-I N* $\subseteq$ *Inf* **and**
    Red-F-Bot: $B \in$ *Bot* $\Longrightarrow N \models \{B\} \Longrightarrow N -$ *Red-F N* $\models \{B\}$ **and**
    Red-F-of-subset: $N \subseteq N' \Longrightarrow$ *Red-F N* $\subseteq$ *Red-F* N' **and**
    Red-I-of-subset: $N \subseteq N' \Longrightarrow$ *Red-I N* $\subseteq$ *Red-I* N' **and**
    Red-F-of-Red-F-subset: $N' \subseteq$ *Red-F N* $\Longrightarrow$ *Red-F N* $\subseteq$ *Red-F* $(N - N')$ **and**
    Red-I-of-Red-F-subset: $N' \subseteq$ *Red-F N* $\Longrightarrow$ *Red-I N* $\subseteq$ *Red-I* $(N - N')$ **and**
    Red-I-of-Inf-to-N: $\iota \in$ *Inf* $\Longrightarrow$ *concl-of* $\iota \in N \Longrightarrow \iota \in$ *Red-I N*
begin

lemma *Red-I-of-Inf-to-N-subset*: $\{\iota \in$ *Inf*. *concl-of* $\iota \in N\} \subseteq$ *Red-I N*
  using *Red-I-of-Inf-to-N* **by** *blast*

lemma *red-concl-to-red-inf*:
  assumes
    i-in: $\iota \in$ *Inf* **and**
    concl: *concl-of* $\iota \in$ *Red-F N*
  shows $\iota \in$ *Red-I N*
proof −
  have $\iota \in$ *Red-I* (*Red-F N*) **by** (*simp add*: *Red-I-of-Inf-to-N i-in concl*)
  then have *i-in-Red*: $\iota \in$ *Red-I* $(N \cup$ *Red-F N*) **by** (*simp add*: *Red-I-of-Inf-to-N concl i-in*)
  have *red-n-subs*: *Red-F N* $\subseteq$ *Red-F* $(N \cup$ *Red-F N*) **by** (*simp add*: *Red-F-of-subset*)
  then have $\iota \in$ *Red-I* $((N \cup$ *Red-F N*$) -$ (*Red-F N* $- N))$ **using** *Red-I-of-Red-F-subset i-in-Red*

```
    by (meson Diff-subset subsetCE subset-trans)
  then show ?thesis by (metis Diff-cancel Diff-subset Un-Diff Un-Diff-cancel contra-subsetD
    calculus.Red-I-of-subset calculus-axioms sup-bot.right-neutral)
qed

definition saturated :: 'f set ⇒ bool where
  saturated N ⟷ Inf-from N ⊆ Red-I N

definition reduc-saturated :: 'f set ⇒ bool where
  reduc-saturated N ⟷ Inf-from (N − Red-F N) ⊆ Red-I N

lemma Red-I-without-red-F:
  Red-I (N − Red-F N) = Red-I N
  using Red-I-of-subset [of N − Red-F N N]
    and Red-I-of-Red-F-subset [of Red-F N N] by blast

lemma saturated-without-red-F:
  assumes saturated: saturated N
  shows saturated (N − Red-F N)
proof −
  have Inf-from (N − Red-F N) ⊆ Inf-from N unfolding Inf-from-def by auto
  also have Inf-from N ⊆ Red-I N using saturated unfolding saturated-def by auto
  also have Red-I N ⊆ Red-I (N − Red-F N) using Red-I-of-Red-F-subset by auto
  finally have Inf-from (N − Red-F N) ⊆ Red-I (N − Red-F N) by auto
  then show ?thesis unfolding saturated-def by auto
qed

definition fair :: 'f set llist ⇒ bool where
  fair Ns ⟷ Inf-from (Liminf-llist Ns) ⊆ Sup-llist (lmap Red-I Ns)

inductive derive :: 'f set ⇒ 'f set ⇒ bool (infix ‹▷› 50) where
  derive: M − N ⊆ Red-F N ⟹ M ▷ N

lemma gt-Max-notin: ‹finite A ⟹ A ≠ {} ⟹ x > Max A ⟹ x ∉ A› by auto

lemma equiv-Sup-Liminf:
  assumes
    in-Sup: C ∈ Sup-llist Ns and
    not-in-Liminf: C ∉ Liminf-llist Ns
  shows
    ∃ i ∈ {i. enat (Suc i) < llength Ns}. C ∈ lnth Ns i − lnth Ns (Suc i)
proof −
  obtain i where C-D-i: C ∈ Sup-upto-llist Ns (enat i) and enat i < llength Ns
    using elem-Sup-llist-imp-Sup-upto-llist in-Sup by fast
  then obtain j where j: j ≥ i ∧ enat j < llength Ns ∧ C ∉ lnth Ns j
    using not-in-Liminf unfolding Sup-upto-llist-def chain-def Liminf-llist-def by auto
  obtain k where k: C ∈ lnth Ns k enat k < llength Ns k ≤ i using C-D-i
    unfolding Sup-upto-llist-def by auto
  let ?S = {i. i < j ∧ i ≥ k ∧ C ∈ lnth Ns i}
  define l where l = Max ?S
  have ‹k ∈ {i. i < j ∧ k ≤ i ∧ C ∈ lnth Ns i}› using k j by (auto simp: order.order-iff-strict)
  then have nempty: {i. i < j ∧ k ≤ i ∧ C ∈ lnth Ns i} ≠ {} by auto
  then have l-prop: l < j ∧ l ≥ k ∧ C ∈ lnth Ns l using Max-in[of ?S, OF - nempty]
    unfolding l-def by auto
  then have C ∈ lnth Ns l − lnth Ns (Suc l) using j gt-Max-notin[OF - nempty, of Suc l]
```

```
    unfolding l-def[symmetric] by (auto intro: Suc-lessI)
  then show ?thesis
  proof (rule bexI[of - l])
    show l ∈ {i. enat (Suc i) < llength Ns}
      using l-prop j
      by (clarify, metis Suc-leI dual-order.order-iff-strict enat-ord-simps(2) less-trans)
  qed
qed


lemma Red-in-Sup:
  assumes deriv: chain (▷) Ns
  shows Sup-llist Ns − Liminf-llist Ns ⊆ Red-F (Sup-llist Ns)
proof
  fix C
  assume C-in-subset: C ∈ Sup-llist Ns − Liminf-llist Ns
  {
    fix C i
    assume
      in-ith-elem: C ∈ lnth Ns i − lnth Ns (Suc i) and
      i: enat (Suc i) < llength Ns
    have lnth Ns i ▷ lnth Ns (Suc i) using i deriv in-ith-elem chain-lnth-rel by auto
    then have C ∈ Red-F (lnth Ns (Suc i)) using in-ith-elem derive.cases by blast
    then have C ∈ Red-F (Sup-llist Ns) using Red-F-of-subset
      by (meson contra-subsetD i lnth-subset-Sup-llist)
  }
  then show C ∈ Red-F (Sup-llist Ns) using equiv-Sup-Liminf[of C] C-in-subset by fast
qed


lemma Red-I-subset-Liminf:
  assumes deriv: ‹chain (▷) Ns› and
    i: ‹enat i < llength Ns›
  shows ‹Red-I (lnth Ns i) ⊆ Red-I (Liminf-llist Ns)›
proof −
  have Sup-in-diff: ‹Red-I (Sup-llist Ns) ⊆ Red-I (Sup-llist Ns − (Sup-llist Ns − Liminf-llist Ns))›
    using Red-I-of-Red-F-subset[OF Red-in-Sup] deriv by auto
  also have ‹Sup-llist Ns − (Sup-llist Ns − Liminf-llist Ns) = Liminf-llist Ns›
    by (simp add: Liminf-llist-subset-Sup-llist double-diff)
  then have Red-I-Sup-in-Liminf: ‹Red-I (Sup-llist Ns) ⊆ Red-I (Liminf-llist Ns)›
    using Sup-in-diff by auto
  have ‹lnth Ns i ⊆ Sup-llist Ns› unfolding Sup-llist-def using i by blast
  then have Red-I (lnth Ns i) ⊆ Red-I (Sup-llist Ns) using Red-I-of-subset
    unfolding Sup-llist-def by auto
  then show ?thesis using Red-I-Sup-in-Liminf by auto
qed


lemma Red-F-subset-Liminf:
  assumes deriv: ‹chain (▷) Ns› and
    i: ‹enat i < llength Ns›
  shows ‹Red-F (lnth Ns i) ⊆ Red-F (Liminf-llist Ns)›
proof −
  have Sup-in-diff: ‹Red-F (Sup-llist Ns) ⊆ Red-F (Sup-llist Ns − (Sup-llist Ns − Liminf-llist Ns))›
    using Red-F-of-Red-F-subset[OF Red-in-Sup] deriv by auto
```

also have ‹*Sup-llist Ns* − (*Sup-llist Ns* − *Liminf-llist Ns*) = *Liminf-llist Ns*›
by (*simp add*: *Liminf-llist-subset-Sup-llist double-diff*)
then have *Red-F-Sup-in-Liminf*: ‹*Red-F* (*Sup-llist Ns*) ⊆ *Red-F* (*Liminf-llist Ns*)›
using *Sup-in-diff* **by** *auto*
have ‹*lnth Ns i* ⊆ *Sup-llist Ns*› **unfolding** *Sup-llist-def* **using** *i* **by** *blast*
then have *Red-F* (*lnth Ns i*) ⊆ *Red-F* (*Sup-llist Ns*) **using** *Red-F-of-subset*
unfolding *Sup-llist-def* **by** *auto*
then show *?thesis* **using** *Red-F-Sup-in-Liminf* **by** *auto*
qed

lemma *i-in-Liminf-or-Red-F*:
assumes
deriv: ‹*chain* (▷) *Ns*› **and**
i: ‹*enat i* < *llength Ns*›
shows ‹*lnth Ns i* ⊆ *Red-F* (*Liminf-llist Ns*) ∪ *Liminf-llist Ns*›
proof (*rule*,*rule*)
fix *C*
assume *C*: ‹*C* ∈ *lnth Ns i*›
and *C-not-Liminf*: ‹*C* ∉ *Liminf-llist Ns*›
have ‹*C* ∈ *Sup-llist Ns*› **unfolding** *Sup-llist-def* **using** *C i* **by** *auto*
then obtain *j* **where** *j*: ‹*C* ∈ *lnth Ns j* − *lnth Ns* (*Suc j*)› ‹*enat* (*Suc j*) < *llength Ns*›
using *equiv-Sup-Liminf*[*of C Ns*] *C-not-Liminf* **by** *auto*
then have ‹*C* ∈ *Red-F* (*lnth Ns* (*Suc j*))›
using *deriv* **by** (*meson chain-lnth-rel contra-subsetD derive.cases*)
then show ‹*C* ∈ *Red-F* (*Liminf-llist Ns*)› **using** *Red-F-subset-Liminf*[*of Ns Suc j*] *deriv j*(*2*) **by** *blast*
qed

lemma *fair-implies-Liminf-saturated*:
assumes
deriv: ‹*chain* (▷) *Ns*› **and**
fair: ‹*fair Ns*›
shows ‹*saturated* (*Liminf-llist Ns*)›
unfolding *saturated-def*
proof
fix ι
assume ι: ‹ι ∈ *Inf-from* (*Liminf-llist Ns*)›
have ‹ι ∈ *Sup-llist* (*lmap Red-I Ns*)› **using** *fair* ι **unfolding** *fair-def* **by** *auto*
then obtain *i* **where** *i*: ‹*enat i* < *llength Ns*› ‹ι ∈ *Red-I* (*lnth Ns i*)›
unfolding *Sup-llist-def* **by** *auto*
then show ‹ι ∈ *Red-I* (*Liminf-llist Ns*)›
using *deriv i-in-Liminf-or-Red-F*[*of Ns i*] *Red-I-subset-Liminf* **by** *blast*
qed

end

locale *statically-complete-calculus* = *calculus* +
assumes *statically-complete*: *B* ∈ *Bot* ⟹ *saturated N* ⟹ *N* ⊨ {*B*} ⟹ ∃ *B*′∈*Bot*. *B*′ ∈ *N*
begin

lemma *dynamically-complete-Liminf*:
fixes *B Ns*
assumes
bot-elem: ‹*B* ∈ *Bot*› **and**

```
    deriv: ‹chain (▷) Ns› and
    fair: ‹fair Ns› and
    unsat: ‹lhd Ns ⊨ {B}›
  shows ‹∃ B'∈Bot. B' ∈ Liminf-llist Ns›
proof −
  note lhd-is = lhd-conv-lnth[OF chain-not-lnull[OF deriv]]
  have non-empty: ‹¬ lnull Ns› using chain-not-lnull[OF deriv] .
  have subs: ‹lhd Ns ⊆ Sup-llist Ns›
    using lhd-subset-Sup-llist[of Ns] non-empty by (simp add: lhd-conv-lnth)
  have ‹Sup-llist Ns ⊨ {B}›
    using unsat subset-entailed[OF subs] entails-trans[of Sup-llist Ns lhd Ns] by auto
  then have Sup-no-Red: ‹Sup-llist Ns − Red-F (Sup-llist Ns) ⊨ {B}›
    using bot-elem Red-F-Bot by auto
  have Sup-no-Red-in-Liminf: ‹Sup-llist Ns − Red-F (Sup-llist Ns) ⊆ Liminf-llist Ns›
    using deriv Red-in-Sup by auto
  have Liminf-entails-Bot: ‹Liminf-llist Ns ⊨ {B}›
    using Sup-no-Red subset-entailed[OF Sup-no-Red-in-Liminf] entails-trans by blast
  have ‹saturated (Liminf-llist Ns)›
    using deriv fair fair-implies-Liminf-saturated unfolding saturated-def by auto
  then show ?thesis
    using bot-elem statically-complete Liminf-entails-Bot by auto
qed

end

locale dynamically-complete-calculus = calculus +
  assumes
    dynamically-complete: B ∈ Bot ⟹ chain (▷) Ns ⟹ fair Ns ⟹ lhd Ns ⊨ {B} ⟹
      ∃ i ∈ {i. enat i < llength Ns}. ∃ B'∈Bot. B' ∈ lnth Ns i
begin


sublocale statically-complete-calculus
proof
  fix B N
  assume
    bot-elem: ‹B ∈ Bot› and
    saturated-N: saturated N and
    refut-N: N ⊨ {B}
  define Ns where Ns = LCons N LNil
  have[simp]: ‹¬ lnull Ns› by (auto simp: Ns-def)
  have deriv-Ns: ‹chain (▷) Ns› by (simp add: chain.chain-singleton Ns-def)
  have liminf-is-N: Liminf-llist Ns = N by (simp add: Ns-def Liminf-llist-LCons)
  have head-Ns: N = lhd Ns by (simp add: Ns-def)
  have Sup-llist (lmap Red-I Ns) = Red-I N by (simp add: Ns-def)
  then have fair-Ns: fair Ns using saturated-N by (simp add: fair-def saturated-def liminf-is-N)
  obtain i B' where B'-is-bot: ‹B' ∈ Bot› and B'-in: B' ∈ lnth Ns i and ‹i < llength Ns›
    using dynamically-complete[of B Ns] bot-elem fair-Ns head-Ns saturated-N deriv-Ns refut-N
    by auto
  then have i = 0
    by (auto simp: Ns-def enat-0-iff)
  show ‹∃ B'∈Bot. B' ∈ N›
    using B'-is-bot B'-in unfolding ‹i = 0› head-Ns[symmetric] Ns-def by auto
qed
```

end

sublocale *statically-complete-calculus* $\subseteq$ *dynamically-complete-calculus*
proof
  fix *B Ns*
  assume
    ‹*B* $\in$ *Bot*› **and**
    ‹*chain* ($\rhd$) *Ns*› **and**
    ‹*fair Ns*› **and**
    ‹*lhd Ns* $\models$ {*B*}›
  then have ‹$\exists$ *B*′$\in$*Bot*. *B*′ $\in$ *Liminf-llist Ns*›
    by (*rule dynamically-complete-Liminf*)
  then show ‹$\exists$ *i*$\in${*i*. *enat i* < *llength Ns*}. $\exists$ *B*′$\in$*Bot*. *B*′ $\in$ *lnth Ns i*›
    unfolding *Liminf-llist-def* **by** *auto*
qed

end

2 Calculi Based on the Intersection of Redundancy Criteria

In this section, section 2.3 of the report is covered, on calculi equipped with a family of redundancy criteria.

theory *Intersection-Calculus*
  imports
    Calculus
    Ordered-Resolution-Prover.Lazy-List-Liminf
    Ordered-Resolution-Prover.Lazy-List-Chain
begin

2.1 Calculi with a Family of Redundancy Criteria

locale *intersection-calculus* =
  inference-system Inf + *consequence-relation-family Bot Q entails-q*
  for
    Bot :: *'f set* **and**
    Inf :: ‹*'f inference set*› **and**
    Q :: *'q set* **and**
    entails-q :: *'q* $\Rightarrow$ *'f set* $\Rightarrow$ *'f set* $\Rightarrow$ *bool*
  + **fixes**
    Red-I-q :: *'q* $\Rightarrow$ *'f set* $\Rightarrow$ *'f inference set* **and**
    Red-F-q :: *'q* $\Rightarrow$ *'f set* $\Rightarrow$ *'f set*
  assumes
    Q-nonempty: *Q* $\neq$ {} **and**
    all-red-crit: $\forall$ *q* $\in$ *Q*. *calculus Bot Inf* (*entails-q q*) (*Red-I-q q*) (*Red-F-q q*)
begin

definition *Red-I* :: *'f set* $\Rightarrow$ *'f inference set* **where**
  Red-I N = ($\bigcap$ *q* $\in$ *Q*. *Red-I-q q N*)

definition *Red-F* :: *'f set* $\Rightarrow$ *'f set* **where**
  Red-F N = ($\bigcap$ *q* $\in$ *Q*. *Red-F-q q N*)

sublocale *calculus Bot Inf entails Red-I Red-F*
unfolding *calculus-def calculus-axioms-def*
proof (*intro conjI*)
show *consequence-relation Bot entails*
using *intersect-cons-rel-family* **.**
next
show $\forall N.$ *Red-I N* $\subseteq$ *Inf*
unfolding *Red-I-def*
proof
fix *N*
show $(\bigcap q \in Q.$ *Red-I-q q N*$) \subseteq$ *Inf*
proof (*intro Inter-subset*)
fix *Red-Is*
assume *one-red-inf*: *Red-Is* $\in (\lambda q.$ *Red-I-q q N*$)$ ‘ *Q*
show *Red-Is* $\subseteq$ *Inf*
using *one-red-inf all-red-crit calculus.Red-I-to-Inf* **by** *blast*
next
show $(\lambda q.$ *Red-I-q q N*$)$ ‘ *Q* $\neq \{\}$
using *Q-nonempty* **by** *blast*
qed
qed
next
show $\forall B\ N.\ B \in Bot \longrightarrow N \models Q\ \{B\} \longrightarrow N -$ *Red-F N* $\models Q\ \{B\}$
proof (*intro allI impI*)
fix *B N*
assume
B-in: $B \in Bot$ **and**
N-unsat: $N \models Q\ \{B\}$
show $N -$ *Red-F N* $\models Q\ \{B\}$ **unfolding** *entails-def Red-F-def*
proof
fix *qi*
assume *qi-in*: $qi \in Q$
define *entails-qi* (**infix** ‹$\models qi$› *50*) **where** *entails-qi* = *entails-q qi*
have *cons-rel-qi*: *consequence-relation Bot entails-qi*
unfolding *entails-qi-def* **using** *qi-in all-red-crit calculus.axioms*(*1*) **by** *blast*
define *Red-F-qi* **where** *Red-F-qi* = *Red-F-q qi*
have *red-qi-in*: *Red-F N* $\subseteq$ *Red-F-qi N*
unfolding *Red-F-def Red-F-qi-def* **using** *qi-in image-iff* **by** *blast*
then have $N -$ *Red-F-qi N* $\subseteq N -$ *Red-F N* **by** *blast*
then have *entails-1*: $N -$ *Red-F N* $\models qi\ N -$ *Red-F-qi N*
using *qi-in all-red-crit*
unfolding *calculus-def consequence-relation-def entails-qi-def* **by** *metis*
have *N-unsat-qi*: $N \models qi\ \{B\}$ **using** *qi-in N-unsat* **unfolding** *entails-qi-def entails-def*
by *simp*
then have *N-unsat-qi*: $N -$ *Red-F-qi N* $\models qi\ \{B\}$
using *qi-in all-red-crit Red-F-qi-def calculus.Red-F-Bot*[*OF - B-in*] *entails-qi-def*
by *fastforce*
show $N - (\bigcap q \in Q.$ *Red-F-q q N*$) \models qi\ \{B\}$
using *consequence-relation.entails-trans*[*OF cons-rel-qi entails-1 N-unsat-qi*]
unfolding *Red-F-def* **.**
qed
qed
next
show $\forall$ *N1 N2*. *N1* $\subseteq$ *N2* $\longrightarrow$ *Red-F N1* $\subseteq$ *Red-F N2*
proof (*intro allI impI*)

```
    fix N1 :: 'f set
    and N2 :: 'f set
    assume
      N1-in-N2: N1 ⊆ N2
    show Red-F N1 ⊆ Red-F N2
    proof
      fix C
      assume C ∈ Red-F N1
      then have ∀ qi ∈ Q. C ∈ Red-F-q qi N1 unfolding Red-F-def by blast
      then have ∀ qi ∈ Q. C ∈ Red-F-q qi N2
        using N1-in-N2 all-red-crit calculus.axioms(2) calculus.Red-F-of-subset by blast
      then show C ∈ Red-F N2 unfolding Red-F-def by blast
    qed
  qed
next
  show ∀ N1 N2. N1 ⊆ N2 ⟶ Red-I N1 ⊆ Red-I N2
  proof (intro allI impI)
    fix N1 :: 'f set
    and N2 :: 'f set
    assume
      N1-in-N2: N1 ⊆ N2
    show Red-I N1 ⊆ Red-I N2
    proof
      fix ι
      assume ι ∈ Red-I N1
      then have ∀ qi ∈ Q. ι ∈ Red-I-q qi N1 unfolding Red-I-def by blast
      then have ∀ qi ∈ Q. ι ∈ Red-I-q qi N2
        using N1-in-N2 all-red-crit calculus.axioms(2) calculus.Red-I-of-subset by blast
      then show ι ∈ Red-I N2 unfolding Red-I-def by blast
    qed
  qed
next
  show ∀ N2 N1. N2 ⊆ Red-F N1 ⟶ Red-F N1 ⊆ Red-F (N1 − N2)
  proof (intro allI impI)
    fix N2 N1
    assume N2-in-Red-N1: N2 ⊆ Red-F N1
    show Red-F N1 ⊆ Red-F (N1 − N2)
    proof
      fix C
      assume C ∈ Red-F N1
      then have ∀ qi ∈ Q. C ∈ Red-F-q qi N1 unfolding Red-F-def by blast
       moreover have ∀ qi ∈ Q. N2 ⊆ Red-F-q qi N1 using N2-in-Red-N1 unfolding Red-F-def by
blast
      ultimately have ∀ qi ∈ Q. C ∈ Red-F-q qi (N1 − N2)
        using all-red-crit calculus.axioms(2) calculus.Red-F-of-Red-F-subset
        by blast
      then show C ∈ Red-F (N1 − N2) unfolding Red-F-def by blast
    qed
  qed
next
  show ∀ N2 N1. N2 ⊆ Red-F N1 ⟶ Red-I N1 ⊆ Red-I (N1 − N2)
  proof (intro allI impI)
    fix N2 N1
    assume N2-in-Red-N1: N2 ⊆ Red-F N1
    show Red-I N1 ⊆ Red-I (N1 − N2)
```

```
    proof
      fix ι
      assume ι ∈ Red-I N1
      then have ∀ qi ∈ Q. ι ∈ Red-I-q qi N1 unfolding Red-I-def by blast
       moreover have ∀ qi ∈ Q. N2 ⊆ Red-F-q qi N1 using N2-in-Red-N1 unfolding Red-F-def by
blast
      ultimately have ∀ qi ∈ Q. ι ∈ Red-I-q qi (N1 − N2)
        using all-red-crit calculus.axioms(2) calculus.Red-I-of-Red-F-subset by blast
      then show ι ∈ Red-I (N1 − N2) unfolding Red-I-def by blast
    qed
  qed
next
  show ∀ ι N. ι ∈ Inf ⟶ concl-of ι ∈ N ⟶ ι ∈ Red-I N
  proof (intro allI impI)
    fix ι N
    assume
      i-in: ι ∈ Inf and
      concl-in: concl-of ι ∈ N
    then have ∀ qi ∈ Q. ι ∈ Red-I-q qi N
      using all-red-crit calculus.axioms(2) calculus.Red-I-of-Inf-to-N by blast
    then show ι ∈ Red-I N unfolding Red-I-def by blast
  qed
qed


lemma sat-int-to-sat-q: calculus.saturated Inf Red-I N ⟷
  (∀ qi ∈ Q. calculus.saturated Inf (Red-I-q qi) N) for N
proof
  fix N
  assume inter-sat: calculus.saturated Inf Red-I N
  show ∀ qi ∈ Q. calculus.saturated Inf (Red-I-q qi) N
  proof
    fix qi
    assume qi-in: qi ∈ Q
    then interpret one: calculus Bot Inf entails-q qi Red-I-q qi Red-F-q qi
      by (metis all-red-crit)
    show one.saturated N
      using qi-in inter-sat
      unfolding one.saturated-def saturated-def Red-I-def by blast
  qed
next
  fix N
  assume all-sat: ∀ qi ∈ Q. calculus.saturated Inf (Red-I-q qi) N
  show saturated N
    unfolding saturated-def Red-I-def
  proof
    fix ι
    assume ι-in: ι ∈ Inf-from N
    have ∀ Red-I-qi ∈ Red-I-q ‘ Q. ι ∈ Red-I-qi N
    proof
      fix Red-I-qi
      assume red-inf-in: Red-I-qi ∈ Red-I-q ‘ Q
      then obtain qi where
        qi-in: qi ∈ Q and
        red-inf-qi-def: Red-I-qi = Red-I-q qi by blast
```

```
    then interpret one: calculus Bot Inf entails-q qi Red-I-q qi Red-F-q qi
      by (metis all-red-crit)
    have one.saturated N using qi-in all-sat red-inf-qi-def by blast
    then show ι ∈ Red-I-qi N unfolding one.saturated-def using ι-in red-inf-qi-def by blast
   qed
   then show ι ∈ (⋂ q ∈ Q. Red-I-q q N) by blast
  qed
qed


lemma stat-ref-comp-from-bot-in-sat:
  (∀ N. calculus.saturated Inf Red-I N ∧ (∀ B ∈ Bot. B ∉ N) ⟶
     (∃ B ∈ Bot. ∃ qi ∈ Q. ¬ entails-q qi N {B})) ⟹
   statically-complete-calculus Bot Inf entails Red-I Red-F
proof (rule ccontr)
  assume
    N-saturated: ∀ N. (calculus.saturated Inf Red-I N ∧ (∀ B ∈ Bot. B ∉ N)) ⟶
      (∃ B ∈ Bot. ∃ qi ∈ Q. ¬ entails-q qi N {B}) and
    no-stat-ref-comp: ¬ statically-complete-calculus Bot Inf (⊨Q) Red-I Red-F
  obtain N1 B1 where B1-in:
    B1 ∈ Bot and N1-saturated: calculus.saturated Inf Red-I N1 and
    N1-unsat: N1 ⊨Q {B1} and no-B-in-N1: ∀ B ∈ Bot. B ∉ N1
    using no-stat-ref-comp by (metis calculus-axioms statically-complete-calculus.intro
       statically-complete-calculus-axioms.intro)
  obtain B2 qi where
    qi-in: qi ∈ Q and
    no-qi: ¬ entails-q qi N1 {B2}
    using N-saturated N1-saturated no-B-in-N1 by auto
  have N1 ⊨Q {B2} using N1-unsat B1-in intersect-cons-rel-family
    unfolding consequence-relation-def by metis
  then have entails-q qi N1 {B2} unfolding entails-def using qi-in by blast
  then show False using no-qi by simp
qed

end

end
```

3 Variations on a Theme

In this section, section 2.4 of the report is covered, demonstrating that various notions of redundancy are equivalent.

```
theory Calculus-Variations
  imports Calculus
begin

locale reduced-calculus = calculus Bot Inf entails Red-I Red-F
  for
    Bot :: 'f set and
    Inf :: ‹'f inference set› and
    entails :: 'f set ⇒ 'f set ⇒ bool (infix ‹⊨› 50) and
    Red-I :: 'f set ⇒ 'f inference set and
    Red-F :: 'f set ⇒ 'f set
 + assumes
```

inf-in-red-inf: *Inf-between UNIV* (*Red-F N*) ⊆ *Red-I N*
begin

lemma *sat-eq-reduc-sat*: *saturated N* ⟷ *reduc-saturated N*
proof
fix *N*
assume *saturated N*
then show *reduc-saturated N*
using *Red-I-without-red-F saturated-without-red-F*
unfolding *saturated-def reduc-saturated-def*
by *blast*
next
fix *N*
assume *red-sat-n*: *reduc-saturated N*
show *saturated N* **unfolding** *saturated-def*
using *red-sat-n inf-in-red-inf* **unfolding** *reduc-saturated-def Inf-from-def Inf-between-def*
by *blast*
qed

end

locale *reducedly-statically-complete-calculus* = *calculus* +
assumes *reducedly-statically-complete*:
B ∈ *Bot* ⟹ *reduc-saturated N* ⟹ *N* ⊨ {*B*} ⟹ ∃ *B′*∈*Bot*. *B′* ∈ *N*

locale *reducedly-statically-complete-reduced-calculus* = *reduced-calculus* +
assumes *reducedly-statically-complete*:
B ∈ *Bot* ⟹ *reduc-saturated N* ⟹ *N* ⊨ {*B*} ⟹ ∃ *B′*∈*Bot*. *B′* ∈ *N*
begin

sublocale *reducedly-statically-complete-calculus*
by (*simp add*: *calculus-axioms reducedly-statically-complete*
reducedly-statically-complete-calculus-axioms.intro
reducedly-statically-complete-calculus-def)

sublocale *statically-complete-calculus*
proof
fix *B N*
assume
bot-elem: ‹*B* ∈ *Bot*› **and**
saturated-N: *saturated N* **and**
refut-N: *N* ⊨ {*B*}
have *reduc-saturated-N*: *reduc-saturated N* **using** *saturated-N sat-eq-reduc-sat* **by** *blast*
show ∃ *B′*∈*Bot*. *B′* ∈ *N* **using** *reducedly-statically-complete*[*OF bot-elem reduc-saturated-N refut-N*] **.**
qed

end

context *reduced-calculus*
begin

lemma *stat-ref-comp-imp-red-stat-ref-comp*:

```
  statically-complete-calculus Bot Inf entails Red-I Red-F ⟹
   reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F
proof
  fix B N
  assume
    stat-ref-comp: statically-complete-calculus Bot Inf (⊨) Red-I Red-F and
    bot-elem: ‹B ∈ Bot› and
    saturated-N: reduc-saturated N and
    refut-N: N ⊨ {B}
  have reduc-saturated-N: saturated N using saturated-N sat-eq-reduc-sat by blast
  show ∃ B'∈Bot. B' ∈ N
    using statically-complete-calculus.statically-complete[OF stat-ref-comp
      bot-elem reduc-saturated-N refut-N] .
qed

end

context calculus
begin

definition Red-Red-I :: 'f set ⇒ 'f inference set where
  Red-Red-I N = Red-I N ∪ Inf-between UNIV (Red-F N)

lemma reduced-calc-is-calc: calculus Bot Inf entails Red-Red-I Red-F
proof
  fix N
  show Red-Red-I N ⊆ Inf
    unfolding Red-Red-I-def Inf-between-def Inf-from-def using Red-I-to-Inf by auto
next
  fix B N
  assume
    b-in: B ∈ Bot and
    n-entails: N ⊨ {B}
  show N − Red-F N ⊨ {B}
    by (simp add: Red-F-Bot b-in n-entails)
next
  fix N N' :: 'f set
  assume N ⊆ N'
  then show Red-F N ⊆ Red-F N' by (simp add: Red-F-of-subset)
next
  fix N N' :: 'f set
  assume n-in: N ⊆ N'
  then have Inf-from (UNIV − (Red-F N')) ⊆ Inf-from (UNIV − (Red-F N))
    using Red-F-of-subset[OF n-in] unfolding Inf-from-def by auto
  then have Inf-between UNIV (Red-F N) ⊆ Inf-between UNIV (Red-F N')
    unfolding Inf-between-def by auto
  then show Red-Red-I N ⊆ Red-Red-I N'
    unfolding Red-Red-I-def using Red-I-of-subset[OF n-in] by blast
next
  fix N N' :: 'f set
  assume N' ⊆ Red-F N
  then show Red-F N ⊆ Red-F (N − N') by (simp add: Red-F-of-Red-F-subset)
next
  fix N N' :: 'f set
  assume np-subs: N' ⊆ Red-F N
```

```
  have Red-F N ⊆ Red-F (N − N′) by (simp add: Red-F-of-Red-F-subset np-subs)
  then have Inf-from (UNIV − (Red-F (N − N′))) ⊆ Inf-from (UNIV − (Red-F N))
    by (metis Diff-subset Red-F-of-subset eq-iff)
  then have Inf-between UNIV (Red-F N) ⊆ Inf-between UNIV (Red-F (N − N′))
    unfolding Inf-between-def by auto
  then show Red-Red-I N ⊆ Red-Red-I (N − N′)
    unfolding Red-Red-I-def using Red-I-of-Red-F-subset[OF np-subs] by blast
next
  fix ι N
  assume ι ∈ Inf
    concl-of ι ∈ N
  then show ι ∈ Red-Red-I N
    by (simp add: Red-I-of-Inf-to-N Red-Red-I-def)
qed

lemma inf-subs-reduced-red-inf: Inf-between UNIV (Red-F N) ⊆ Red-Red-I N
  unfolding Red-Red-I-def by simp
```

The following is a lemma and not a sublocale as was previously used in similar cases. Here, a sublocale cannot be used because it would create an infinitely descending chain of sublocales.

```
lemma reduc-calc: reduced-calculus Bot Inf entails Red-Red-I Red-F
  using inf-subs-reduced-red-inf reduced-calc-is-calc
  by (simp add: reduced-calculus.intro reduced-calculus-axioms-def)

interpretation reduc-calc: reduced-calculus Bot Inf entails Red-Red-I Red-F
  by (fact reduc-calc)


lemma sat-imp-red-calc-sat: saturated N ⟹ reduc-calc.saturated N
  unfolding saturated-def reduc-calc.saturated-def Red-Red-I-def by blast


lemma red-sat-eq-red-calc-sat: reduc-saturated N ⟷ reduc-calc.saturated N
proof
  assume red-sat-n: reduc-saturated N
  show reduc-calc.saturated N
    unfolding reduc-calc.saturated-def
  proof
    fix ι
    assume i-in: ι ∈ Inf-from N
    show ι ∈ Red-Red-I N
      using i-in red-sat-n
      unfolding reduc-saturated-def Inf-between-def Inf-from-def Red-Red-I-def by blast
  qed
next
  assume red-sat-n: reduc-calc.saturated N
  show reduc-saturated N
    unfolding reduc-saturated-def
  proof
    fix ι
    assume i-in: ι ∈ Inf-from (N − Red-F N)
    show ι ∈ Red-I N
      using i-in red-sat-n
      unfolding Inf-from-def reduc-calc.saturated-def Red-Red-I-def Inf-between-def by blast
  qed
```

qed

lemma *red-sat-eq-sat*: *reduc-saturated N* $\longleftrightarrow$ *saturated* (*N* − *Red-F N*)
unfolding *reduc-saturated-def saturated-def* **by** (*simp add*: *Red-I-without-red-F*)

theorem *stat-is-stat-red*: *statically-complete-calculus Bot Inf entails Red-I Red-F* $\longleftrightarrow$
statically-complete-calculus Bot Inf entails Red-Red-I Red-F
proof
assume
stat-ref1: *statically-complete-calculus Bot Inf entails Red-I Red-F*
show *statically-complete-calculus Bot Inf entails Red-Red-I Red-F*
using *reduc-calc.calculus-axioms*
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
proof
show ∀ *B N*. *B* ∈ *Bot* ⟶ *reduc-calc.saturated N* ⟶ *N* ⊨ {*B*} ⟶ (∃ *B′*∈*Bot*. *B′* ∈ *N*)
proof (*clarify*)
fix *B N*
assume
b-in: *B* ∈ *Bot* **and**
n-sat: *reduc-calc.saturated N* **and**
n-imp-b: *N* ⊨ {*B*}
have *saturated* (*N* − *Red-F N*) **using** *red-sat-eq-red-calc-sat*[*of N*] *red-sat-eq-sat*[*of N*] *n-sat* **by** *blast*
moreover have (*N* − *Red-F N*) ⊨ {*B*} **using** *n-imp-b b-in* **by** (*simp add*: *reduc-calc.Red-F-Bot*)
ultimately show ∃ *B′*∈*Bot*. *B′*∈ *N*
using *stat-ref1* **by** (*meson DiffD1 b-in statically-complete-calculus.statically-complete*)
qed
qed
next
assume
stat-ref3: *statically-complete-calculus Bot Inf entails Red-Red-I Red-F*
show *statically-complete-calculus Bot Inf entails Red-I Red-F*
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
using *calculus-axioms*
proof
show ∀ *B N*. *B* ∈ *Bot* ⟶ *saturated N* ⟶ *N* ⊨ {*B*} ⟶ (∃ *B′*∈*Bot*. *B′* ∈ *N*)
proof *clarify*
fix *B N*
assume
b-in: *B* ∈ *Bot* **and**
n-sat: *saturated N* **and**
n-imp-b: *N* ⊨ {*B*}
then show ∃ *B′*∈ *Bot*. *B′* ∈ *N*
using *stat-ref3 sat-imp-red-calc-sat*[*OF n-sat*]
by (*meson statically-complete-calculus.statically-complete*)
qed
qed
qed

theorem *red-stat-red-is-stat-red*:
reducedly-statically-complete-calculus Bot Inf entails Red-Red-I Red-F $\longleftrightarrow$
statically-complete-calculus Bot Inf entails Red-Red-I Red-F

using *reduc-calc.stat-ref-comp-imp-red-stat-ref-comp*
by (*metis reduc-calc.sat-eq-reduc-sat reducedly-statically-complete-calculus.axioms*(*2*)
reducedly-statically-complete-calculus-axioms-def reduced-calc-is-calc
statically-complete-calculus.intro statically-complete-calculus-axioms.intro)

theorem *red-stat-is-stat-red*:
reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F $\longleftrightarrow$
statically-complete-calculus Bot Inf entails Red-Red-I Red-F
using *reduc-calc.calculus-axioms calculus-axioms red-sat-eq-red-calc-sat*
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
reducedly-statically-complete-calculus-def reducedly-statically-complete-calculus-axioms-def
by *blast*

lemma *sup-red-f-in-red-liminf*:
chain derive Ns $\Longrightarrow$ *Sup-llist* (*lmap Red-F Ns*) $\subseteq$ *Red-F* (*Liminf-llist Ns*)
proof
fix *N*
assume
deriv: *chain derive Ns* **and**
n-in-sup: *N* $\in$ *Sup-llist* (*lmap Red-F Ns*)
obtain *i0* **where** *i-smaller*: *enat i0* $<$ *llength Ns* **and** *n-in*: *N* $\in$ *Red-F* (*lnth Ns i0*)
using *n-in-sup* **by** (*metis Sup-llist-imp-exists-index llength-lmap lnth-lmap*)
have *Red-F* (*lnth Ns i0*) $\subseteq$ *Red-F* (*Liminf-llist Ns*)
using *i-smaller* **by** (*simp add*: *deriv Red-F-subset-Liminf*)
then show *N* $\in$ *Red-F* (*Liminf-llist Ns*)
using *n-in* **by** *fast*
qed

lemma *sup-red-inf-in-red-liminf*:
chain derive Ns $\Longrightarrow$ *Sup-llist* (*lmap Red-I Ns*) $\subseteq$ *Red-I* (*Liminf-llist Ns*)
proof
fix ι
assume
deriv: *chain derive Ns* **and**
i-in-sup: ι $\in$ *Sup-llist* (*lmap Red-I Ns*)
obtain *i0* **where** *i-smaller*: *enat i0* $<$ *llength Ns* **and** *n-in*: ι $\in$ *Red-I* (*lnth Ns i0*)
using *i-in-sup* **unfolding** *Sup-llist-def* **by** *auto*
have *Red-I* (*lnth Ns i0*) $\subseteq$ *Red-I* (*Liminf-llist Ns*)
using *i-smaller* **by** (*simp add*: *deriv Red-I-subset-Liminf*)
then show ι $\in$ *Red-I* (*Liminf-llist Ns*)
using *n-in* **by** *fast*
qed

definition *reduc-fair* :: *'f set llist* $\Rightarrow$ *bool* **where**
reduc-fair Ns $\longleftrightarrow$
Inf-from (*Liminf-llist Ns* $-$ *Sup-llist* (*lmap Red-F Ns*)) $\subseteq$ *Sup-llist* (*lmap Red-I Ns*)

lemma *reduc-fair-imp-Liminf-reduc-sat*:
chain derive Ns $\Longrightarrow$ *reduc-fair Ns* $\Longrightarrow$ *reduc-saturated* (*Liminf-llist Ns*)
unfolding *reduc-saturated-def*
proof $-$
fix *Ns*
assume

deriv: *chain derive Ns* **and**
red-fair: *reduc-fair Ns*
have *Inf-from* (*Liminf-llist Ns* − *Red-F* (*Liminf-llist Ns*))
⊆ *Inf-from* (*Liminf-llist Ns* − *Sup-llist* (*lmap Red-F Ns*))
using *sup-red-f-in-red-liminf*[*OF deriv*] **unfolding** *Inf-from-def* **by** *blast*
then have *Inf-from* (*Liminf-llist Ns* − *Red-F* (*Liminf-llist Ns*)) ⊆ *Sup-llist* (*lmap Red-I Ns*)
using *red-fair* **unfolding** *reduc-fair-def* **by** *simp*
then show *Inf-from* (*Liminf-llist Ns* − *Red-F* (*Liminf-llist Ns*)) ⊆ *Red-I* (*Liminf-llist Ns*)
using *sup-red-inf-in-red-liminf*[*OF deriv*] **by** *fast*
qed

end

locale *reducedly-dynamically-complete-calculus* = *calculus* +
assumes
reducedly-dynamically-complete: *B* ∈ *Bot* ⟹ *chain derive Ns* ⟹ *reduc-fair Ns* ⟹
lhd Ns ⊨ {*B*} ⟹ ∃ *i* ∈ {*i*. *enat i* < *llength Ns*}. ∃ *B′*∈*Bot*. *B′* ∈ *lnth Ns i*
begin

sublocale *reducedly-statically-complete-calculus*
proof
fix *B N*
assume
bot-elem: ‹*B* ∈ *Bot*› **and**
saturated-N: *reduc-saturated N* **and**
refut-N: *N* ⊨ {*B*}
define *Ns* **where** *Ns* = *LCons N LNil*
have[*simp*]: ‹¬ *lnull Ns*› **by** (*auto simp*: *Ns-def*)
have *deriv-D*: ‹*chain* (▷) *Ns*› **by** (*simp add*: *chain.chain-singleton Ns-def*)
have *liminf-is-N*: *Liminf-llist Ns* = *N* **by** (*simp add*: *Ns-def Liminf-llist-LCons*)
have *head-D*: *N* = *lhd Ns* **by** (*simp add*: *Ns-def*)
have *Sup-llist* (*lmap Red-F Ns*) = *Red-F N* **by** (*simp add*: *Ns-def*)
moreover have *Sup-llist* (*lmap Red-I Ns*) = *Red-I N* **by** (*simp add*: *Ns-def*)
ultimately have *fair-D*: *reduc-fair Ns*
using *saturated-N liminf-is-N* **unfolding** *reduc-fair-def reduc-saturated-def*
by (*simp add*: *reduc-fair-def reduc-saturated-def liminf-is-N*)
obtain *i B′* **where** *B′-is-bot*: ‹*B′* ∈ *Bot*› **and** *B′-in*: *B′* ∈ *lnth Ns i* **and** ‹*i* < *llength Ns*›
using *reducedly-dynamically-complete*[*of B Ns*] *bot-elem fair-D head-D saturated-N deriv-D refut-N*
by *auto*
then have *i* = *0*
by (*auto simp*: *Ns-def enat-0-iff*)
show ‹∃ *B′*∈*Bot*. *B′* ∈ *N*›
using *B′-is-bot B′-in* **unfolding** ‹*i* = *0*› *head-D*[*symmetric*] *Ns-def* **by** *auto*
qed

end

sublocale *reducedly-statically-complete-calculus* ⊆ *reducedly-dynamically-complete-calculus*
proof
fix *B Ns*
assume
bot-elem: ‹*B* ∈ *Bot*› **and**
deriv: ‹*chain* (▷) *Ns*› **and**
fair: ‹*reduc-fair Ns*› **and**
unsat: ‹*lhd Ns* ⊨ {*B*}›

```
    have non-empty: ‹¬ lnull Ns› using chain-not-lnull[OF deriv] .
    have subs: ‹lhd Ns ⊆ Sup-llist Ns›
      using lhd-subset-Sup-llist[of Ns] non-empty by (simp add: lhd-conv-lnth)
    have ‹Sup-llist Ns ⊨ {B}›
      using unsat subset-entailed[OF subs] entails-trans[of Sup-llist Ns lhd Ns] by auto
    then have Sup-no-Red: ‹Sup-llist Ns − Red-F (Sup-llist Ns) ⊨ {B}›
      using bot-elem Red-F-Bot by auto
    have Sup-no-Red-in-Liminf: ‹Sup-llist Ns − Red-F (Sup-llist Ns) ⊆ Liminf-llist Ns›
      using deriv Red-in-Sup by auto
    have Liminf-entails-Bot: ‹Liminf-llist Ns ⊨ {B}›
      using Sup-no-Red subset-entailed[OF Sup-no-Red-in-Liminf] entails-trans by blast
    have ‹reduc-saturated (Liminf-llist Ns)›
    using deriv fair reduc-fair-imp-Liminf-reduc-sat unfolding reduc-saturated-def
      by auto
   then have ‹∃ B'∈Bot. B' ∈ Liminf-llist Ns›
     using bot-elem reducedly-statically-complete Liminf-entails-Bot
     by auto
   then show ‹∃ i∈{i. enat i < llength Ns}. ∃ B'∈Bot. B' ∈ lnth Ns i›
     unfolding Liminf-llist-def by auto
qed

context calculus
begin

lemma dyn-equiv-stat: dynamically-complete-calculus Bot Inf entails Red-I Red-F =
  statically-complete-calculus Bot Inf entails Red-I Red-F
proof
  assume dynamically-complete-calculus Bot Inf entails Red-I Red-F
  then interpret dynamically-complete-calculus Bot Inf entails Red-I Red-F
    by simp
  show statically-complete-calculus Bot Inf entails Red-I Red-F
    by (simp add: statically-complete-calculus-axioms)
next
  assume statically-complete-calculus Bot Inf entails Red-I Red-F
  then interpret statically-complete-calculus Bot Inf entails Red-I Red-F
    by simp
  show dynamically-complete-calculus Bot Inf entails Red-I Red-F
    by (simp add: dynamically-complete-calculus-axioms)
qed

lemma red-dyn-equiv-red-stat:
  reducedly-dynamically-complete-calculus Bot Inf entails Red-I Red-F =
   reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F
proof
  assume reducedly-dynamically-complete-calculus Bot Inf entails Red-I Red-F
  then interpret reducedly-dynamically-complete-calculus Bot Inf entails Red-I Red-F
    by simp
  show reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F
    by (simp add: reducedly-statically-complete-calculus-axioms)
next
  assume reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F
  then interpret reducedly-statically-complete-calculus Bot Inf entails Red-I Red-F
    by simp
  show reducedly-dynamically-complete-calculus Bot Inf entails Red-I Red-F
    by (simp add: reducedly-dynamically-complete-calculus-axioms)
```

qed

interpretation *reduc-calc*: *reduced-calculus Bot Inf entails Red-Red-I Red-F*
by (*fact reduc-calc*)

theorem *dyn-ref-eq-dyn-ref-red*:
dynamically-complete-calculus Bot Inf entails Red-I Red-F $\longleftrightarrow$
dynamically-complete-calculus Bot Inf entails Red-Red-I Red-F
using *dyn-equiv-stat stat-is-stat-red reduc-calc.dyn-equiv-stat* **by** *meson*

theorem *red-dyn-ref-red-eq-dyn-ref-red*:
reducedly-dynamically-complete-calculus Bot Inf entails Red-Red-I Red-F $\longleftrightarrow$
dynamically-complete-calculus Bot Inf entails Red-Red-I Red-F
using *red-dyn-equiv-red-stat dyn-equiv-stat red-stat-red-is-stat-red*
by (*simp add*: *reduc-calc.dyn-equiv-stat reduc-calc.red-dyn-equiv-red-stat*)

theorem *red-dyn-ref-eq-dyn-ref-red*:
reducedly-dynamically-complete-calculus Bot Inf entails Red-I Red-F $\longleftrightarrow$
dynamically-complete-calculus Bot Inf entails Red-Red-I Red-F
using *red-dyn-equiv-red-stat dyn-equiv-stat red-stat-is-stat-red*
reduc-calc.dyn-equiv-stat reduc-calc.red-dyn-equiv-red-stat
by *blast*

end

end

4 Lifting to Non-ground Calculi

The section 3.1 to 3.3 of the report are covered by the current section. Various forms of lifting are proven correct. These allow to obtain the dynamic refutational completeness of a non-ground calculus from the static refutational completeness of its ground counterpart.

theory *Lifting-to-Non-Ground-Calculi*
imports
Intersection-Calculus
Calculus-Variations
begin

4.1 Standard Lifting

locale *standard-lifting* = *inference-system Inf-F* +
ground: *calculus Bot-G Inf-G entails-G Red-I-G Red-F-G*
for
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
Inf-G :: ‹*'g inference set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** ‹$\models G$› *50*) **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**
Red-F-G :: ‹*'g set* ⇒ *'g set*›
+ **fixes**
Bot-F :: ‹*'f set*› **and**

```
    𝒢-F :: ‹'f ⇒ 'g set› and
    𝒢-I :: ‹'f inference ⇒ 'g inference set option›
  assumes
    Bot-F-not-empty: Bot-F ≠ {} and
    Bot-map-not-empty: ‹B ∈ Bot-F ⟹ 𝒢-F B ≠ {}› and
    Bot-map: ‹B ∈ Bot-F ⟹ 𝒢-F B ⊆ Bot-G› and
    Bot-cond: ‹𝒢-F C ∩ Bot-G ≠ {} ⟶ C ∈ Bot-F› and
    inf-map: ‹ι ∈ Inf-F ⟹ 𝒢-I ι ≠ None ⟹ the (𝒢-I ι) ⊆ Red-I-G (𝒢-F (concl-of ι))›
begin

abbreviation 𝒢-Fset :: ‹'f set ⇒ 'g set› where
  ‹𝒢-Fset N ≡ ⋃ (𝒢-F ‘ N)›

lemma 𝒢-subset: ‹N1 ⊆ N2 ⟹ 𝒢-Fset N1 ⊆ 𝒢-Fset N2› by auto

abbreviation entails-𝒢 (infix ‹⊨𝒢› 50) where
  ‹N1 ⊨𝒢 N2 ≡ 𝒢-Fset N1 ⊨G 𝒢-Fset N2›

lemma subs-Bot-G-entails:
  assumes
    not-empty: ‹sB ≠ {}› and
    in-bot: ‹sB ⊆ Bot-G›
  shows ‹sB ⊨G N›
proof −
  have ‹∃ B. B ∈ sB› using not-empty by auto
  then obtain B where B-in: ‹B ∈ sB› by auto
  then have r-trans: ‹{B} ⊨G N› using ground.bot-entails-all in-bot by auto
  have l-trans: ‹sB ⊨G {B}› using B-in ground.subset-entailed by auto
  then show ?thesis using r-trans ground.entails-trans[of sB {B}] by auto
qed


sublocale consequence-relation Bot-F entails-𝒢
proof
  show Bot-F ≠ {} using Bot-F-not-empty .
next
  show ‹B∈Bot-F ⟹ {B} ⊨𝒢 N› for B N
  proof −
    assume ‹B ∈ Bot-F›
    then show ‹{B} ⊨𝒢 N›
      using Bot-map ground.bot-entails-all[of - 𝒢-Fset N] subs-Bot-G-entails Bot-map-not-empty
      by auto
  qed
next
  fix N1 N2 :: ‹'f set›
  assume
    ‹N2 ⊆ N1›
  then show ‹N1 ⊨𝒢 N2› using 𝒢-subset ground.subset-entailed by auto
next
  fix N1 N2
  assume
    N1-entails-C: ‹∀ C ∈ N2. N1 ⊨𝒢 {C}›
  show ‹N1 ⊨𝒢 N2› using ground.all-formulas-entailed N1-entails-C
    by (simp add: ground.entail-unions)
next
```

fix *N1 N2 N3*
assume
‹N1 ⊨𝒢 N2› **and** *‹N2 ⊨𝒢 N3›*
then show *‹N1 ⊨𝒢 N3›* **using** *ground.entails-trans* **by** *blast*
qed

definition *Red-I-𝒢* :: *'f set ⇒ 'f inference set* **where**
‹Red-I-𝒢 N = {ι ∈ Inf-F. (𝒢-I ι ≠ None ∧ the (𝒢-I ι) ⊆ Red-I-G (𝒢-Fset N))
∨ (𝒢-I ι = None ∧ 𝒢-F (concl-of ι) ⊆ 𝒢-Fset N ∪ Red-F-G (𝒢-Fset N))}›

definition *Red-F-𝒢* :: *'f set ⇒ 'f set* **where**
‹Red-F-𝒢 N = {C. ∀ D ∈ 𝒢-F C. D ∈ Red-F-G (𝒢-Fset N)}›
end

4.2 Strong Standard Lifting

locale *strong-standard-lifting = inference-system Inf-F +*
ground: calculus Bot-G Inf-G entails-G Red-I-G Red-F-G
for
Inf-F :: ‹'f inference set› **and**
Bot-G :: ‹'g set› **and**
Inf-G :: ‹'g inference set› **and**
entails-G :: ‹'g set ⇒ 'g set ⇒ bool› (**infix** *‹⊨G› 50*) **and**
Red-I-G :: ‹'g set ⇒ 'g inference set› **and**
Red-F-G :: ‹'g set ⇒ 'g set›
+ fixes
Bot-F :: ‹'f set› **and**
𝒢-F :: ‹'f ⇒ 'g set› **and**
𝒢-I :: ‹'f inference ⇒ 'g inference set option›
assumes
Bot-F-not-empty: Bot-F ≠ {} **and**
Bot-map-not-empty: ‹B ∈ Bot-F ⟹ 𝒢-F B ≠ {}› **and**
Bot-map: ‹B ∈ Bot-F ⟹ 𝒢-F B ⊆ Bot-G› **and**
Bot-cond: ‹𝒢-F C ∩ Bot-G ≠ {} ⟶ C ∈ Bot-F› **and**
strong-inf-map: ‹ι ∈ Inf-F ⟹ 𝒢-I ι ≠ None ⟹ concl-of ‘ (the (𝒢-I ι)) ⊆ (𝒢-F (concl-of ι))› **and**
inf-map-in-Inf: ‹ι ∈ Inf-F ⟹ 𝒢-I ι ≠ None ⟹ the (𝒢-I ι) ⊆ Inf-G›
begin

sublocale *standard-lifting Inf-F Bot-G Inf-G (⊨G) Red-I-G Red-F-G Bot-F 𝒢-F 𝒢-I*
proof
show *Bot-F ≠ {}* **using** *Bot-F-not-empty* **.**
next
fix *B*
assume *b-in: B ∈ Bot-F*
show *𝒢-F B ≠ {}* **using** *Bot-map-not-empty[OF b-in]* **.**
next
fix *B*
assume *b-in: B ∈ Bot-F*
show *𝒢-F B ⊆ Bot-G* **using** *Bot-map[OF b-in]* **.**
next
show *⋀C. 𝒢-F C ∩ Bot-G ≠ {} ⟶ C ∈ Bot-F* **using** *Bot-cond* **.**
next
fix *ι*
assume *i-in: ι ∈ Inf-F* **and**
some-g: 𝒢-I ι ≠ None
show *the (𝒢-I ι) ⊆ Red-I-G (𝒢-F (concl-of ι))*

proof
fix ιG
assume *ig-in1*: $\iota G \in$ *the* $(\mathcal{G}\text{-}I\ \iota)$
then have *ig-in2*: $\iota G \in$ *Inf-G* **using** *inf-map-in-Inf*[*OF i-in some-g*] **by** *blast*
show $\iota G \in$ *Red-I-G* $(\mathcal{G}\text{-}F\ (\textit{concl-of}\ \iota))$
using *strong-inf-map*[*OF i-in some-g*] *ground.Red-I-of-Inf-to-N*[*OF ig-in2*]
ig-in1 **by** *blast*
qed
qed

end

4.3 Lifting with a Family of Tiebreaker Orderings

locale *tiebreaker-lifting* =
empty-ord?: *standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G Red-F-G Bot-F* $\mathcal{G}\text{-}F$ $\mathcal{G}\text{-}I$
for
Bot-F :: ‹$'f$ *set*› **and**
Inf-F :: ‹$'f$ *inference set*› **and**
Bot-G :: ‹$'g$ *set*› **and**
entails-G :: ‹$'g$ *set* ⇒ $'g$ *set* ⇒ *bool*› (**infix** ‹⊨*G*› *50*) **and**
Inf-G :: ‹$'g$ *inference set*› **and**
Red-I-G :: ‹$'g$ *set* ⇒ $'g$ *inference set*› **and**
Red-F-G :: ‹$'g$ *set* ⇒ $'g$ *set*› **and**
$\mathcal{G}\text{-}F$ :: $'f$ ⇒ $'g$ *set* **and**
$\mathcal{G}\text{-}I$ :: $'f$ *inference* ⇒ $'g$ *inference set option*
+ **fixes**
Prec-F-g :: ‹$'g$ ⇒ $'f$ ⇒ $'f$ ⇒ *bool*›
assumes
all-wf: *wfp* (*Prec-F-g g*) *transp* (*Prec-F-g g*)
begin

definition *Red-F-*$\mathcal{G}$ :: $'f$ *set* ⇒ $'f$ *set* **where**
‹*Red-F-*$\mathcal{G}$ *N* = {*C*. ∀ *D* ∈ $\mathcal{G}$*-F C*. *D* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*) ∨ (∃ *E* ∈ *N*. *Prec-F-g D E C* ∧ *D* ∈ $\mathcal{G}$*-F E*)}›

lemma *Prec-trans*:
assumes
‹*Prec-F-g D A B*› **and**
‹*Prec-F-g D B C*›
shows
‹*Prec-F-g D A C*›
using *assms all-wf*
unfolding *transp-on-def*
by *blast*

lemma *prop-nested-in-set*: *D* ∈ *P C* ⟹ *C* ∈ {*C*. ∀ *D* ∈ *P C*. *A D* ∨ *B C D*} ⟹ *A D* ∨ *B C D*
by *blast*

lemma *Red-F-*$\mathcal{G}$*-equiv-def*:
‹*Red-F-*$\mathcal{G}$ *N* = {*C*. ∀ *Di* ∈ $\mathcal{G}$*-F C*. *Di* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*) ∨
(∃ *E* ∈ (*N* − *Red-F-*$\mathcal{G}$ *N*). *Prec-F-g Di E C* ∧ *Di* ∈ $\mathcal{G}$*-F E*)}›
proof (*rule*; *clarsimp*)
fix *C D*
assume

C-in: ‹$C \in$ *Red-F-*$\mathcal{G}$ N› **and**
D-in: ‹$D \in \mathcal{G}$-F C› **and**
not-sec-case: ‹$\forall E \in N - $ *Red-F-*$\mathcal{G}$ N. *Prec-F-g* D E $C \longrightarrow D \notin \mathcal{G}$-$F$ E›
have *C-in-unfolded*: $C \in \{C. \forall Di \in \mathcal{G}$-$F$ C. $Di \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\vee$
($\exists E{\in}N$. *Prec-F-g* Di E $C \wedge Di \in \mathcal{G}$-$F$ E)$\}$
using *C-in* **unfolding** *Red-F-*$\mathcal{G}$*-def* **.**
have *neg-not-sec-case*: ‹$\neg$ ($\exists E{\in}N - $ *Red-F-*$\mathcal{G}$ N. *Prec-F-g* D E $C \wedge D \in \mathcal{G}$-$F$ E)›
using *not-sec-case* **by** *clarsimp*
have *unfol-C-D*: ‹$D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\vee$ ($\exists E{\in}N$. *Prec-F-g* D E $C \wedge D \in \mathcal{G}$-$F$ E)›
using *prop-nested-in-set*[*of* D $\mathcal{G}$-F C λx. $x \in$ *Red-F-G* ($\bigcup$ ($\mathcal{G}$-F ‘ N))
λx y. $\exists E \in N$. *Prec-F-g* y E $x \wedge y \in \mathcal{G}$-$F$ E, *OF D-in C-in-unfolded*] **by** *blast*
show ‹$D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N)›
proof (*rule ccontr*)
assume *contrad*: ‹$D \notin$ *Red-F-G* ($\mathcal{G}$-*Fset* N)›
have *non-empty*: ‹$\exists E{\in}N$. *Prec-F-g* D E $C \wedge D \in \mathcal{G}$-$F$ E› **using** *contrad unfol-C-D* **by** *auto*
define B **where** ‹$B = \{E \in N$. *Prec-F-g* D E $C \wedge D \in \mathcal{G}$-$F$ $E\}$›
then have *B-non-empty*: ‹$B \neq \{\}$› **using** *non-empty* **by** *auto*

obtain $F :: 'f$ **where** F: $F \in B \, \forall y$. *Prec-F-g* D y $F \longrightarrow y \notin B$
using *all-wf*[*of* D]
by (*metis B-non-empty wfp-iff-ex-minimal*)

then have *D-in-F*: ‹$D \in \mathcal{G}$-F F›
unfolding *B-def* **using** *non-empty*
by *blast*

have *F-prec*: ‹*Prec-F-g* D F C› **using** F **unfolding** *B-def* **by** *auto*
have *F-not-in*: ‹$F \notin$ *Red-F-*$\mathcal{G}$ N›
proof
assume *F-in*: ‹$F \in$ *Red-F-*$\mathcal{G}$ N›
have *unfol-F-D*: ‹$D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\vee$ ($\exists G{\in}N$. *Prec-F-g* D G $F \wedge D \in \mathcal{G}$-$F$ G)›
using *F-in D-in-F* **unfolding** *Red-F-*$\mathcal{G}$*-def* **by** *auto*
then have ‹$\exists G{\in}N$. *Prec-F-g* D G $F \wedge D \in \mathcal{G}$-$F$ G› **using** *contrad D-in* **unfolding** *Red-F-*$\mathcal{G}$*-def*
by *auto*
then obtain G **where** *G-in*: ‹$G \in N$› **and** *G-prec*: ‹*Prec-F-g* D G F› **and** *G-map*: ‹$D \in \mathcal{G}$-F G›
by *auto*
have ‹*Prec-F-g* D G C› **using** *G-prec F-prec Prec-trans* **by** *blast*
then have ‹$G \in B$› **unfolding** *B-def* **using** *G-in G-map* **by** *auto*
then show ‹*False*› **using** F *G-prec* **by** *auto*
qed
have ‹$F \in N$› **using** F *B-def* **by** *blast*
then have ‹$F \in N -$ *Red-F-*$\mathcal{G}$ N› **using** *F-not-in* **by** *auto*
then show ‹*False*›
using *D-in-F neg-not-sec-case F-prec* **by** *blast*
qed
next
fix C
assume *only-if*: ‹$\forall D{\in}\mathcal{G}$-$F$ C. $D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\vee$ ($\exists E{\in}N -$ *Red-F-*$\mathcal{G}$ N. *Prec-F-g* D E $C \wedge$
$D \in \mathcal{G}$-F E)›
show ‹$C \in$ *Red-F-*$\mathcal{G}$ N› **unfolding** *Red-F-*$\mathcal{G}$*-def* **using** *only-if* **by** *auto*
qed

lemma *not-red-map-in-map-not-red*: ‹$\mathcal{G}$-*Fset* $N -$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\subseteq \mathcal{G}$-*Fset* ($N -$ *Red-F-*$\mathcal{G}$ N)›
proof

fix D
assume *D-hyp*: ‹$D \in \mathcal{G}$-*Fset* N − *Red-F-G* ($\mathcal{G}$-*Fset* N)›

have *D-in*: ‹$D \in \mathcal{G}$-*Fset* N› **using** *D-hyp* **by** *blast*
have *D-not-in*: ‹$D \notin$ *Red-F-G* ($\mathcal{G}$-*Fset* N)› **using** *D-hyp* **by** *blast*
have *exist-C*: ‹$\exists\, C.\ C \in N \wedge D \in \mathcal{G}$-*F* C› **using** *D-in* **by** *auto*
define B **where** ‹$B = \{C \in N.\ D \in \mathcal{G}$-*F* $C\}$›
obtain C **where** C: $C \in B\ \forall y.$ *Prec-F-g* $D\ y\ C \longrightarrow y \notin B$
by (*metis* (*no-types*, *lifting*) *B-def all-wf*(*1*) *exist-C mem-Collect-eq wfp-eq-minimal*)
have *C-in-N*: ‹$C \in N$›
using *B-def* C **by** *auto*
have *D-in-C*: ‹$D \in \mathcal{G}$-*F* C›
using *B-def* C **by** *auto*
have *C-not-in*: ‹$C \notin$ *Red-F-*$\mathcal{G}$ N›
proof
assume *C-in*: ‹$C \in$ *Red-F-*$\mathcal{G}$ N›
have ‹$D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N) $\vee$ ($\exists\, E{\in}N.$ *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$-*F* E)›
using *C-in D-in-C* **unfolding** *Red-F-*$\mathcal{G}$*-def* **by** *auto*
then show ‹*False*›
proof
assume ‹$D \in$ *Red-F-G* ($\mathcal{G}$-*Fset* N)›
then show ‹*False*› **using** *D-not-in* **by** *simp*
next
assume ‹$\exists\, E{\in}N.$ *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$-*F* E›
then show ‹*False*›
using C *B-def* **by** *auto*
qed
qed
show ‹$D \in \mathcal{G}$-*Fset* (N − *Red-F-*$\mathcal{G}$ N)› **using** *D-in-C C-not-in C-in-N* **by** *blast*
qed

lemma *Red-F-Bot-F*: ‹$B \in$ *Bot-F* $\Longrightarrow N \models\mathcal{G} \{B\} \Longrightarrow N$ − *Red-F-*$\mathcal{G}$ $N \models\mathcal{G} \{B\}$›
proof −
fix $B\ N$
assume
B-in: ‹$B \in$ *Bot-F*› **and**
N-entails: ‹$N \models\mathcal{G} \{B\}$›
then have *to-bot*: ‹$\mathcal{G}$-*Fset* N − *Red-F-G* ($\mathcal{G}$-*Fset* N) $\models G\ \mathcal{G}$-*F* B›
using *ground.Red-F-Bot Bot-map*
by (*metis SUP-upper ground.entail-set-all-formulas in-mono singletonI*)
have *from-f*: ‹$\mathcal{G}$-*Fset* (N − *Red-F-*$\mathcal{G}$ N) $\models G\ \mathcal{G}$-*Fset* N − *Red-F-G* ($\mathcal{G}$-*Fset* N)›
using *ground.subset-entailed*[*OF not-red-map-in-map-not-red*] **by** *blast*
then have ‹$\mathcal{G}$-*Fset* (N − *Red-F-*$\mathcal{G}$ N) $\models G\ \mathcal{G}$-*F* B› **using** *to-bot ground.entails-trans* **by** *blast*
then show ‹N − *Red-F-*$\mathcal{G}$ $N \models\mathcal{G} \{B\}$› **using** *Bot-map* **by** *simp*
qed

lemma *Red-F-of-subset-F*: ‹$N \subseteq N' \Longrightarrow$ *Red-F-*$\mathcal{G}$ $N \subseteq$ *Red-F-*$\mathcal{G}$ N'›
using *ground.Red-F-of-subset* **unfolding** *Red-F-*$\mathcal{G}$*-def* **by** *clarsimp* (*meson* $\mathcal{G}$*-subset subsetD*)

lemma *Red-I-of-subset-F*: ‹$N \subseteq N' \Longrightarrow$ *Red-I-*$\mathcal{G}$ $N \subseteq$ *Red-I-*$\mathcal{G}$ N'›
using *Collect-mono* $\mathcal{G}$*-subset subset-iff ground.Red-I-of-subset* **unfolding** *Red-I-*$\mathcal{G}$*-def*

by (*smt* (*verit*, *del-insts*) *UnCI UnE ground.Red-F-of-subset subsetD*)

lemma *Red-F-of-Red-F-subset-F*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ $N \Longrightarrow$ *Red-F-*$\mathcal{G}$ $N \subseteq$ *Red-F-*$\mathcal{G}$ $(N - N')$›
proof
fix $N\ N'\ C$
assume
N'*-in-Red-F-N*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ N› **and**
C-in-red-F-N: ‹$C \in$ *Red-F-*$\mathcal{G}$ N›
have *lem8*: ‹$\forall\, D \in \mathcal{G}$*-F* $C.\ D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* N) $\vee$ ($\exists\, E \in (N -$ *Red-F-*$\mathcal{G}$ $N)$. *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$*-F* E)›
using *Red-F-*$\mathcal{G}$*-equiv-def C-in-red-F-N* **by** *blast*
show ‹$C \in$ *Red-F-*$\mathcal{G}$ $(N - N')$› **unfolding** *Red-F-*$\mathcal{G}$*-def*
proof (*rule*,*rule*)
fix D
assume ‹$D \in \mathcal{G}$*-F* C›
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* N) $\vee$ ($\exists\, E \in (N -$ *Red-F-*$\mathcal{G}$ $N)$. *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$*-F* E)›
using *lem8* **by** *auto*
then show ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* $(N - N')$) $\vee$ ($\exists\, E \in N - N'$. *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$*-F* E)›
proof
assume ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* N)›
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* $N -$ *Red-F-G* ($\mathcal{G}$*-Fset* N))›
using *ground.Red-F-of-Red-F-subset*[*of Red-F-G* ($\mathcal{G}$*-Fset* N) $\mathcal{G}$*-Fset* N] **by** *auto*
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* $(N -$ *Red-F-*$\mathcal{G}$ $N)$)›
using *ground.Red-F-of-subset*[*OF not-red-map-in-map-not-red*[*of* N]] **by** *auto*
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* $(N - N')$)›
using N'*-in-Red-F-N* $\mathcal{G}$*-subset*[*of* $N -$ *Red-F-*$\mathcal{G}$ $N\ N - N'$]
by (*meson Diff-mono ground.Red-F-of-subset subset-iff*)
then show *?thesis* **by** *blast*
next
assume ‹$\exists\, E \in N -$ *Red-F-*$\mathcal{G}$ $N.$ *Prec-F-g* $D\ E\ C \wedge D \in \mathcal{G}$*-F* E›
then obtain E **where**
E-in: ‹$E \in N -$ *Red-F-*$\mathcal{G}$ N› **and**
E-prec-C: ‹*Prec-F-g* $D\ E\ C$› **and**
D-in: ‹$D \in \mathcal{G}$*-F* E›
by *auto*
have ‹$E \in N - N'$› **using** *E-in* N'*-in-Red-F-N* **by** *blast*
then show *?thesis* **using** *E-prec-C D-in* **by** *blast*
qed
qed
qed

lemma *Red-I-of-Red-F-subset-F*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ $N \Longrightarrow$ *Red-I-*$\mathcal{G}$ $N \subseteq$ *Red-I-*$\mathcal{G}$ $(N - N')$ ›
proof
fix $N\ N'\ \iota$
assume
N'*-in-Red-F-N*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ N› **and**
i-in-Red-I-N: ‹$\iota \in$ *Red-I-*$\mathcal{G}$ N›
have *i-in*: ‹$\iota \in$ *Inf-F*› **using** *i-in-Red-I-N* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *blast*
{
assume *not-none*: $\mathcal{G}$*-I* $\iota \neq$ *None*
have ‹$\forall\, \iota' \in$ *the* ($\mathcal{G}$*-I* ι). $\iota' \in$ *Red-I-G* ($\mathcal{G}$*-Fset* N)›
using *not-none i-in-Red-I-N* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *auto*
then have ‹$\forall\, \iota' \in$ *the* ($\mathcal{G}$*-I* ι). $\iota' \in$ *Red-I-G* ($\mathcal{G}$*-Fset* $N -$ *Red-F-G* ($\mathcal{G}$*-Fset* N))›

using *not-none ground.Red-I-of-Red-F-subset* **by** *blast*
then have *ip-in-Red-I-G*: ‹$\forall \iota'$ ∈ *the* ($\mathcal{G}$-*I* ι). ι' ∈ *Red-I-G* ($\mathcal{G}$-*Fset* (*N* − *Red-F-*$\mathcal{G}$ *N*))›
using *not-none ground.Red-I-of-subset*[*OF not-red-map-in-map-not-red*[*of N*]] **by** *auto*
then have *not-none-in*: ‹$\forall \iota'$ ∈ *the* ($\mathcal{G}$-*I* ι). ι' ∈ *Red-I-G* ($\mathcal{G}$-*Fset* (*N* − *N'*))›
using *not-none N'-in-Red-F-N*
by (*meson Diff-mono ground.Red-I-of-subset* $\mathcal{G}$*-subset subset-iff subset-refl*)
then have *the* ($\mathcal{G}$-*I* ι) ⊆ *Red-I-G* ($\mathcal{G}$-*Fset* (*N* − *N'*)) **by** *blast*
}
moreover {
assume *none*: $\mathcal{G}$-*I* ι = *None*
have *ground-concl-subs*: $\mathcal{G}$-*F* (*concl-of* ι) ⊆ ($\mathcal{G}$-*Fset N* ∪ *Red-F-G* ($\mathcal{G}$-*Fset N*))
using *none i-in-Red-I-N* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *blast*
then have *d-in-imp12*: *D* ∈ $\mathcal{G}$-*F* (*concl-of* ι) ⟹ *D* ∈ $\mathcal{G}$-*Fset N* − *Red-F-G* ($\mathcal{G}$-*Fset N*) ∨ *D* ∈ *Red-F-G* ($\mathcal{G}$-*Fset N*)
by *blast*
have *d-in-imp1*: *D* ∈ $\mathcal{G}$-*Fset N* − *Red-F-G* ($\mathcal{G}$-*Fset N*) ⟹ *D* ∈ $\mathcal{G}$-*Fset* (*N* − *N'*)
using *not-red-map-in-map-not-red N'-in-Red-F-N* **by** *blast*
have *d-in-imp-d-in*: *D* ∈ *Red-F-G* ($\mathcal{G}$-*Fset N*) ⟹ *D* ∈ *Red-F-G* ($\mathcal{G}$-*Fset N* − *Red-F-G* ($\mathcal{G}$-*Fset N*))
using *ground.Red-F-of-Red-F-subset*[*of Red-F-G* ($\mathcal{G}$-*Fset N*) $\mathcal{G}$-*Fset N*] **by** *blast*
have *g-subs1*: $\mathcal{G}$-*Fset N* − *Red-F-G* ($\mathcal{G}$-*Fset N*) ⊆ $\mathcal{G}$-*Fset* (*N* − *Red-F-*$\mathcal{G}$ *N*)
using *not-red-map-in-map-not-red* **unfolding** *Red-F-*$\mathcal{G}$*-def* **by** *auto*
have *g-subs2*: $\mathcal{G}$-*Fset* (*N* − *Red-F-*$\mathcal{G}$ *N*) ⊆ $\mathcal{G}$-*Fset* (*N* − *N'*)
using *N'-in-Red-F-N* **by** *blast*
have *d-in-imp2*: *D* ∈ *Red-F-G* ($\mathcal{G}$-*Fset N*) ⟹ *D* ∈ *Red-F-G* ($\mathcal{G}$-*Fset* (*N* − *N'*))
using *ground.Red-F-of-subset ground.Red-F-of-subset*[*OF g-subs1*]
ground.Red-F-of-subset[*OF g-subs2*] *d-in-imp-d-in* **by** *blast*
have $\mathcal{G}$-*F* (*concl-of* ι) ⊆ ($\mathcal{G}$-*Fset* (*N* − *N'*) ∪ *Red-F-G* ($\mathcal{G}$-*Fset* (*N* − *N'*)))
using *d-in-imp12 d-in-imp1 d-in-imp2*
by (*smt* (*verit*, *ccfv-SIG*) *Un-Diff-cancel2 calculus.Red-F-of-subset g-subs1 g-subs2*
ground.Red-F-of-Red-F-subset ground.reduced-calc-is-calc ground-concl-subs order-eq-refl
order-trans sup-mono)
}
ultimately show ‹ι ∈ *Red-I-*$\mathcal{G}$ (*N* − *N'*)› **using** *i-in* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *auto*
qed

lemma *Red-I-of-Inf-to-N-F*:
assumes
i-in: ‹ι ∈ *Inf-F*› **and**
concl-i-in: ‹*concl-of* ι ∈ *N*›
shows
‹ι ∈ *Red-I-*$\mathcal{G}$ *N* ›
proof −
have ‹ι ∈ *Inf-F* ⟹ $\mathcal{G}$-*I* ι ≠ *None* ⟹ *the* ($\mathcal{G}$-*I* ι) ⊆ *Red-I-G* ($\mathcal{G}$-*F* (*concl-of* ι))› **using** *inf-map* **by** *simp*
moreover have ‹*Red-I-G* ($\mathcal{G}$-*F* (*concl-of* ι)) ⊆ *Red-I-G* ($\mathcal{G}$-*Fset N*)›
using *concl-i-in ground.Red-I-of-subset* **by** *blast*
moreover have ι ∈ *Inf-F* ⟹ $\mathcal{G}$-*I* ι = *None* ⟹ *concl-of* ι ∈ *N* ⟹ $\mathcal{G}$-*F* (*concl-of* ι) ⊆ $\mathcal{G}$-*Fset N*
by *blast*
ultimately show *?thesis* **using** *i-in concl-i-in* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *auto*
qed

sublocale *calculus Bot-F Inf-F entails-*$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
proof

fix B N N' ι
show ‹*Red-I-$\mathcal{G}$* $N \subseteq$ *Inf-F*› **unfolding** *Red-I-$\mathcal{G}$-def* **by** *blast*
show ‹$B \in$ *Bot-F* $\Longrightarrow$ $N \models_{\mathcal{G}} \{B\} \Longrightarrow$ $N -$ *Red-F-$\mathcal{G}$* $N \models_{\mathcal{G}} \{B\}$› **using** *Red-F-Bot-F* **by** *simp*
show ‹$N \subseteq N' \Longrightarrow$ *Red-F-$\mathcal{G}$* $N \subseteq$ *Red-F-$\mathcal{G}$* N'› **using** *Red-F-of-subset-F* **by** *simp*
show ‹$N \subseteq N' \Longrightarrow$ *Red-I-$\mathcal{G}$* $N \subseteq$ *Red-I-$\mathcal{G}$* N'› **using** *Red-I-of-subset-F* **by** *simp*
show ‹$N' \subseteq$ *Red-F-$\mathcal{G}$* $N \Longrightarrow$ *Red-F-$\mathcal{G}$* $N \subseteq$ *Red-F-$\mathcal{G}$* $(N - N')$› **using** *Red-F-of-Red-F-subset-F* **by** *simp*
show ‹$N' \subseteq$ *Red-F-$\mathcal{G}$* $N \Longrightarrow$ *Red-I-$\mathcal{G}$* $N \subseteq$ *Red-I-$\mathcal{G}$* $(N - N')$› **using** *Red-I-of-Red-F-subset-F* **by** *simp*
show ‹$\iota \in$ *Inf-F* $\Longrightarrow$ *concl-of* $\iota \in N \Longrightarrow \iota \in$ *Red-I-$\mathcal{G}$* N› **using** *Red-I-of-Inf-to-N-F* **by** *simp*
qed

end

lemma *standard-empty-tiebreaker-equiv*: *standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G Red-F-G Bot-F $\mathcal{G}$-F $\mathcal{G}$-I* = *tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G Red-F-G $\mathcal{G}$-F $\mathcal{G}$-I* ($\lambda g\ C\ C'$. *False*)
proof −
have *tiebreaker-lifting-axioms* ($\lambda g\ C\ C'$. *False*)
unfolding *tiebreaker-lifting-axioms-def*
by *auto*

then show *?thesis*
unfolding *standard-lifting-def tiebreaker-lifting-def* **by** *blast*
qed

context *standard-lifting*
begin

interpretation *empt-ord*: *tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G Red-F-G $\mathcal{G}$-F $\mathcal{G}$-I* $\lambda g\ C\ C'$. *False*
using *standard-empty-tiebreaker-equiv* **using** *standard-lifting-axioms* **by** *blast*

lemma *red-f-equiv*: *empt-ord.Red-F-$\mathcal{G}$* = *Red-F-$\mathcal{G}$*
unfolding *Red-F-$\mathcal{G}$-def empt-ord.Red-F-$\mathcal{G}$-def* **by** *simp*

sublocale *calc?*: *calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$*
using *empt-ord.calculus-axioms red-f-equiv* **by** *fastforce*

lemma *grounded-inf-in-ground-inf*: $\iota \in$ *Inf-F* $\Longrightarrow$ *$\mathcal{G}$-I* $\iota \neq$ *None* $\Longrightarrow$ *the* (*$\mathcal{G}$-I* ι) $\subseteq$ *Inf-G*
using *inf-map ground.Red-I-to-Inf* **by** *blast*

abbreviation *ground-Inf-overapproximated* :: *'f set* ⇒ *bool* **where**
ground-Inf-overapproximated $N \equiv$ *ground.Inf-from* (*$\mathcal{G}$-Fset* N)
$\subseteq \{\iota.\ \exists \iota' \in$ *Inf-from* $N.$ *$\mathcal{G}$-I* $\iota' \neq$ *None* $\wedge\ \iota \in$ *the* (*$\mathcal{G}$-I* ι')$\}$ $\cup$ *Red-I-G* (*$\mathcal{G}$-Fset* N)

lemma *sat-inf-imp-ground-red*:
assumes
saturated N **and**
$\iota' \in$ *Inf-from* N **and**
$\mathcal{G}$-I $\iota' \neq$ *None* $\wedge\ \iota \in$ *the* (*$\mathcal{G}$-I* ι')
shows $\iota \in$ *Red-I-G* (*$\mathcal{G}$-Fset* N)
using *assms Red-I-$\mathcal{G}$-def* **unfolding** *saturated-def* **by** *auto*

lemma *sat-imp-ground-sat*:
saturated N $\Longrightarrow$ *ground-Inf-overapproximated N* $\Longrightarrow$ *ground.saturated* ($\mathcal{G}$*-Fset N*)
unfolding *ground.saturated-def* **using** *sat-inf-imp-ground-red* **by** *auto*

theorem *stat-ref-comp-to-non-ground*:
assumes
stat-ref-G: *statically-complete-calculus Bot-G Inf-G entails-G Red-I-G Red-F-G* **and**
sat-n-imp: $\bigwedge N$. *saturated N* $\Longrightarrow$ *ground-Inf-overapproximated N*
shows
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
proof
fix *B N*
assume
b-in: *B* $\in$ *Bot-F* **and**
sat-n: *saturated N* **and**
n-entails-bot: *N* $\models\mathcal{G}$ {*B*}
have *ground-n-entails*: $\mathcal{G}$*-Fset N* $\models G$ $\mathcal{G}$*-F B*
using *n-entails-bot* **by** *simp*
then obtain *BG* **where** *bg-in1*: *BG* $\in$ $\mathcal{G}$*-F B*
using *Bot-map-not-empty*[*OF b-in*] **by** *blast*
then have *bg-in*: *BG* $\in$ *Bot-G*
using *Bot-map*[*OF b-in*] **by** *blast*
have *ground-n-entails-bot*: $\mathcal{G}$*-Fset N* $\models G$ {*BG*}
using *ground-n-entails bg-in1 ground.entail-set-all-formulas* **by** *blast*
have *ground.Inf-from* ($\mathcal{G}$*-Fset N*) $\subseteq$
{ι. $\exists \iota' \in$ *Inf-from N*. $\mathcal{G}$*-I* $\iota' \neq$ *None* $\wedge$ $\iota \in$ *the* ($\mathcal{G}$*-I* ι')} $\cup$ *Red-I-G* ($\mathcal{G}$*-Fset N*)
using *sat-n-imp*[*OF sat-n*] **.**
have *ground.saturated* ($\mathcal{G}$*-Fset N*)
using *sat-imp-ground-sat*[*OF sat-n sat-n-imp*[*OF sat-n*]] **.**
then have $\exists$ *BG'*$\in$*Bot-G*. *BG'* $\in$ ($\mathcal{G}$*-Fset N*)
using *stat-ref-G ground.calculus-axioms bg-in ground-n-entails-bot*
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
by *blast*
then show $\exists$ *B'*$\in$ *Bot-F*. *B'* $\in$ *N*
using *bg-in Bot-cond Bot-map-not-empty Bot-cond*
by *blast*
qed

end

context *tiebreaker-lifting*
begin

lemma *saturated-empty-order-equiv-saturated*:
saturated N = *calc.saturated N*
by (*rule refl*)

lemma *static-empty-order-equiv-static*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$ =
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *empty-ord.Red-F-*$\mathcal{G}$
unfolding *statically-complete-calculus-def*

by (*rule iffI*) (*standard*,(*standard*)[],*simp*)+

theorem *static-to-dynamic*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ empty-ord.Red-F-$\mathcal{G}$ =
dynamically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$
using *dyn-equiv-stat static-empty-order-equiv-static*
by *blast*

end

4.4 Lifting with a Family of Redundancy Criteria

locale *lifting-intersection = inference-system Inf-F +*
ground: *inference-system-family Q Inf-G-q +*
ground: *consequence-relation-family Bot-G Q entails-q*
for
Inf-F :: 'f inference set **and**
Bot-G :: 'g set **and**
Q :: 'q set **and**
Inf-G-q :: ‹'q ⇒ 'g inference set› **and**
entails-q :: 'q ⇒ 'g set ⇒ 'g set ⇒ bool **and**
Red-I-q :: 'q ⇒ 'g set ⇒ 'g inference set **and**
Red-F-q :: 'q ⇒ 'g set ⇒ 'g set
+ **fixes**
Bot-F :: 'f set **and**
$\mathcal{G}$-F-q :: 'q ⇒ 'f ⇒ 'g set **and**
$\mathcal{G}$-I-q :: 'q ⇒ 'f inference ⇒ 'g inference set option **and**
Prec-F-g :: 'g ⇒ 'f ⇒ 'f ⇒ bool
assumes
standard-lifting-family:
∀ q ∈ Q. tiebreaker-lifting Bot-F Inf-F Bot-G (entails-q q) (Inf-G-q q) (Red-I-q q)
(Red-F-q q) ($\mathcal{G}$-F-q q) ($\mathcal{G}$-I-q q) Prec-F-g
begin

abbreviation *$\mathcal{G}$-Fset-q :: 'q ⇒ 'f set ⇒ 'g set* **where**
$\mathcal{G}$-Fset-q q N ≡ ⋃ ($\mathcal{G}$-F-q q ‘ N)

definition *Red-I-$\mathcal{G}$-q :: 'q ⇒ 'f set ⇒ 'f inference set* **where**
Red-I-$\mathcal{G}$-q q N = {ι ∈ Inf-F. ($\mathcal{G}$-I-q q ι ≠ None ∧ the ($\mathcal{G}$-I-q q ι) ⊆ Red-I-q q ($\mathcal{G}$-Fset-q q N))
∨ ($\mathcal{G}$-I-q q ι = None ∧ $\mathcal{G}$-F-q q (concl-of ι) ⊆ ($\mathcal{G}$-Fset-q q N ∪ Red-F-q q ($\mathcal{G}$-Fset-q q N)))}

definition *Red-F-$\mathcal{G}$-empty-q :: 'q ⇒ 'f set ⇒ 'f set* **where**
Red-F-$\mathcal{G}$-empty-q q N = {C. ∀ D ∈ $\mathcal{G}$-F-q q C. D ∈ Red-F-q q ($\mathcal{G}$-Fset-q q N)}

definition *Red-F-$\mathcal{G}$-q :: 'q ⇒ 'f set ⇒ 'f set* **where**
Red-F-$\mathcal{G}$-q q N =
{C. ∀ D ∈ $\mathcal{G}$-F-q q C. D ∈ Red-F-q q ($\mathcal{G}$-Fset-q q N) ∨ (∃ E ∈ N. Prec-F-g D E C ∧ D ∈ $\mathcal{G}$-F-q q E)}

abbreviation *entails-$\mathcal{G}$-q :: 'q ⇒ 'f set ⇒ 'f set ⇒ bool* **where**
entails-$\mathcal{G}$-q q N1 N2 ≡ entails-q q ($\mathcal{G}$-Fset-q q N1) ($\mathcal{G}$-Fset-q q N2)

lemma *red-crit-lifting-family*:
assumes *q-in*: *q ∈ Q*
shows *calculus Bot-F Inf-F (entails-$\mathcal{G}$-q q) (Red-I-$\mathcal{G}$-q q) (Red-F-$\mathcal{G}$-q q)*

proof −
interpret *wf-lift*:
tiebreaker-lifting Bot-F Inf-F Bot-G entails-q q Inf-G-q q Red-I-q q
Red-F-q q $\mathcal{G}$*-F-q q* $\mathcal{G}$*-I-q q Prec-F-g*
using *standard-lifting-family q-in* **by** *metis*
have *Red-I-*$\mathcal{G}$*-q q* = *wf-lift.Red-I-*$\mathcal{G}$
unfolding *Red-I-*$\mathcal{G}$*-q-def wf-lift.Red-I-*$\mathcal{G}$*-def* **by** *blast*
moreover have *Red-F-*$\mathcal{G}$*-q q* = *wf-lift.Red-F-*$\mathcal{G}$
unfolding *Red-F-*$\mathcal{G}$*-q-def wf-lift.Red-F-*$\mathcal{G}$*-def* **by** *blast*
ultimately show *?thesis*
using *wf-lift.calculus-axioms* **by** *simp*
qed

lemma *red-crit-lifting-family-empty-ord*:
assumes *q-in*: $q \in Q$
shows *calculus Bot-F Inf-F* (*entails-*$\mathcal{G}$*-q q*) (*Red-I-*$\mathcal{G}$*-q q*) (*Red-F-*$\mathcal{G}$*-empty-q q*)
proof −
interpret *wf-lift*:
tiebreaker-lifting Bot-F Inf-F Bot-G entails-q q Inf-G-q q Red-I-q q
Red-F-q q $\mathcal{G}$*-F-q q* $\mathcal{G}$*-I-q q Prec-F-g*
using *standard-lifting-family q-in* **by** *metis*
have *Red-I-*$\mathcal{G}$*-q q* = *wf-lift.Red-I-*$\mathcal{G}$
unfolding *Red-I-*$\mathcal{G}$*-q-def wf-lift.Red-I-*$\mathcal{G}$*-def* **by** *blast*
moreover have *Red-F-*$\mathcal{G}$*-empty-q q* = *wf-lift.empty-ord.Red-F-*$\mathcal{G}$
unfolding *Red-F-*$\mathcal{G}$*-empty-q-def wf-lift.empty-ord.Red-F-*$\mathcal{G}$*-def* **by** *blast*
ultimately show *?thesis*
using *wf-lift.calc.calculus-axioms* **by** *simp*
qed

sublocale *consequence-relation-family Bot-F Q entails-*$\mathcal{G}$*-q*
proof (*unfold-locales*; (*intro ballI*) *?*)
show $Q \neq \{\}$
by (*rule ground.Q-nonempty*)
next
fix *qi*
assume *qi-in*: $qi \in Q$

interpret *lift*: *tiebreaker-lifting Bot-F Inf-F Bot-G entails-q qi Inf-G-q qi*
Red-I-q qi Red-F-q qi $\mathcal{G}$*-F-q qi* $\mathcal{G}$*-I-q qi Prec-F-g*
using *qi-in* **by** (*metis standard-lifting-family*)

show *consequence-relation Bot-F* (*entails-*$\mathcal{G}$*-q qi*)
by *unfold-locales*
qed

sublocale *intersection-calculus Bot-F Inf-F Q entails-*$\mathcal{G}$*-q Red-I-*$\mathcal{G}$*-q Red-F-*$\mathcal{G}$*-q*
by *unfold-locales* (*auto simp*: *Q-nonempty red-crit-lifting-family*)

abbreviation *entails-*$\mathcal{G}$ :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ‹$\models\cap\mathcal{G}$› *50*) **where**
($\models\cap\mathcal{G}$) ≡ *entails*

abbreviation *Red-I-*$\mathcal{G}$ :: *'f set* ⇒ *'f inference set* **where**
Red-I-$\mathcal{G}$ ≡ *Red-I*

abbreviation *Red-F-*$\mathcal{G}$ :: *'f set* ⇒ *'f set* **where**

Red-F-$\mathcal{G}$ $\equiv$ *Red-F*

lemmas *entails-$\mathcal{G}$-def* = *entails-def*
lemmas *Red-I-$\mathcal{G}$-def* = *Red-I-def*
lemmas *Red-F-$\mathcal{G}$-def* = *Red-F-def*

sublocale *empty-ord*: *intersection-calculus Bot-F Inf-F Q entails-$\mathcal{G}$-q Red-I-$\mathcal{G}$-q Red-F-$\mathcal{G}$-empty-q*
by *unfold-locales* (*auto simp*: *Q-nonempty red-crit-lifting-family-empty-ord*)

abbreviation *Red-F-$\mathcal{G}$-empty* :: *'f set* $\Rightarrow$ *'f set* **where**
Red-F-$\mathcal{G}$-empty $\equiv$ *empty-ord.Red-F*

lemmas *Red-F-$\mathcal{G}$-empty-def* = *empty-ord.Red-F-def*

lemma *sat-inf-imp-ground-red-fam-inter*:
assumes
sat-n: *saturated N* **and**
ι'-in: $\iota' \in$ *Inf-from N* **and**
q-in: $q \in Q$ **and**
grounding: *$\mathcal{G}$-I-q q ι'* $\neq$ *None* $\wedge$ $\iota \in$ *the* (*$\mathcal{G}$-I-q q ι'*)
shows $\iota \in$ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)
proof −
have $\iota' \in$ *Red-I-$\mathcal{G}$-q q N*
using *sat-n ι'-in q-in all-red-crit calculus.saturated-def sat-int-to-sat-q*
by *blast*
then have *the* (*$\mathcal{G}$-I-q q ι'*) $\subseteq$ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)
by (*simp add*: *Red-I-$\mathcal{G}$-q-def grounding*)
then show *?thesis*
using *grounding* **by** *blast*
qed

abbreviation *ground-Inf-overapproximated* :: *'q* $\Rightarrow$ *'f set* $\Rightarrow$ *bool* **where**
ground-Inf-overapproximated q N $\equiv$
ground.Inf-from-q q (*$\mathcal{G}$-Fset-q q N*)
$\subseteq$ {ι. $\exists \iota' \in$ *Inf-from N*. *$\mathcal{G}$-I-q q ι'* $\neq$ *None* $\wedge$ $\iota \in$ *the* (*$\mathcal{G}$-I-q q ι'*)} $\cup$ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)

abbreviation *ground-saturated* :: *'q* $\Rightarrow$ *'f set* $\Rightarrow$ *bool* **where**
ground-saturated q N $\equiv$ *ground.Inf-from-q q* (*$\mathcal{G}$-Fset-q q N*) $\subseteq$ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)

lemma *sat-imp-ground-sat-fam-inter*:
saturated N $\Longrightarrow$ $q \in Q \Longrightarrow$ *ground-Inf-overapproximated q N* $\Longrightarrow$ *ground-saturated q N*
using *sat-inf-imp-ground-red-fam-inter* **by** *auto*

theorem *stat-ref-comp-to-non-ground-fam-inter*:
assumes
stat-ref-G:
$\forall q \in Q$. *statically-complete-calculus Bot-G* (*Inf-G-q q*) (*entails-q q*) (*Red-I-q q*)
(*Red-F-q q*) **and**
sat-n-imp: $\bigwedge N$. *saturated N* $\Longrightarrow$ $\exists q \in Q$. *ground-Inf-overapproximated q N*
shows
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$-empty
using *empty-ord.calculus-axioms* **unfolding** *statically-complete-calculus-def*
statically-complete-calculus-axioms-def
proof (*standard*, *clarify*)

fix B N
assume
$b\text{-}in$: $B \in Bot\text{-}F$ **and**
$sat\text{-}n$: $saturated\ N$ **and**
$entails\text{-}bot$: $N \models\cap\mathcal{G}\ \{B\}$
then obtain q **where**
$q\text{-}in$: $q \in Q$ **and**
$inf\text{-}subs$: $ground.Inf\text{-}from\text{-}q\ q\ (\mathcal{G}\text{-}Fset\text{-}q\ q\ N) \subseteq$
$\{\iota.\ \exists \iota' \in Inf\text{-}from\ N.\ \mathcal{G}\text{-}I\text{-}q\ q\ \iota' \neq None \wedge \iota \in the\ (\mathcal{G}\text{-}I\text{-}q\ q\ \iota')\}$
$\cup\ Red\text{-}I\text{-}q\ q\ (\mathcal{G}\text{-}Fset\text{-}q\ q\ N)$
using $sat\text{-}n\text{-}imp[of\ N]$ **by** $blast$
interpret $q\text{-}calc$: $calculus\ Bot\text{-}F\ Inf\text{-}F\ entails\text{-}\mathcal{G}\text{-}q\ q\ Red\text{-}I\text{-}\mathcal{G}\text{-}q\ q\ Red\text{-}F\text{-}\mathcal{G}\text{-}q\ q$
using $all\text{-}red\text{-}crit[rule\text{-}format,\ OF\ q\text{-}in]$ **.**
have $n\text{-}q\text{-}sat$: $q\text{-}calc.saturated\ N$
using $q\text{-}in\ sat\text{-}int\text{-}to\text{-}sat\text{-}q\ sat\text{-}n$ **by** $simp$
interpret $lifted\text{-}q\text{-}calc$:
$tiebreaker\text{-}lifting\ Bot\text{-}F\ Inf\text{-}F\ Bot\text{-}G\ entails\text{-}q\ q\ Inf\text{-}G\text{-}q\ q\ Red\text{-}I\text{-}q\ q$
$Red\text{-}F\text{-}q\ q\ \mathcal{G}\text{-}F\text{-}q\ q\ \mathcal{G}\text{-}I\text{-}q\ q$
using $q\text{-}in$ **by** $(simp\ add: standard\text{-}lifting\text{-}family)$
have $n\text{-}lift\text{-}sat$: $lifted\text{-}q\text{-}calc.calc.saturated\ N$
using $n\text{-}q\text{-}sat$ **unfolding** $Red\text{-}I\text{-}\mathcal{G}\text{-}q\text{-}def\ lifted\text{-}q\text{-}calc.Red\text{-}I\text{-}\mathcal{G}\text{-}def$
$lifted\text{-}q\text{-}calc.saturated\text{-}def\ q\text{-}calc.saturated\text{-}def$ **by** $auto$
have $ground\text{-}sat\text{-}n$: $lifted\text{-}q\text{-}calc.ground.saturated\ (\mathcal{G}\text{-}Fset\text{-}q\ q\ N)$
by $(rule\ lifted\text{-}q\text{-}calc.sat\text{-}imp\text{-}ground\text{-}sat[OF\ n\text{-}lift\text{-}sat])$
$(use\ n\text{-}lift\text{-}sat\ inf\text{-}subs\ ground.Inf\text{-}from\text{-}q\text{-}def$ **in** $auto)$
have $ground\text{-}n\text{-}entails\text{-}bot$: $entails\text{-}\mathcal{G}\text{-}q\ q\ N\ \{B\}$
using $q\text{-}in\ entails\text{-}bot$ **unfolding** $entails\text{-}\mathcal{G}\text{-}def$ **by** $simp$
interpret $statically\text{-}complete\text{-}calculus\ Bot\text{-}G\ Inf\text{-}G\text{-}q\ q\ entails\text{-}q\ q\ Red\text{-}I\text{-}q\ q$
$Red\text{-}F\text{-}q\ q$
using $stat\text{-}ref\text{-}G[rule\text{-}format,\ OF\ q\text{-}in]$ **.**
obtain BG **where** $bg\text{-}in$: $BG \in \mathcal{G}\text{-}F\text{-}q\ q\ B$
using $lifted\text{-}q\text{-}calc.Bot\text{-}map\text{-}not\text{-}empty[OF\ b\text{-}in]$ **by** $blast$
then have $BG \in Bot\text{-}G$ **using** $lifted\text{-}q\text{-}calc.Bot\text{-}map[OF\ b\text{-}in]$ **by** $blast$
then have $\exists BG' \in Bot\text{-}G.\ BG' \in \mathcal{G}\text{-}Fset\text{-}q\ q\ N$
using $ground\text{-}sat\text{-}n\ ground\text{-}n\text{-}entails\text{-}bot\ statically\text{-}complete[of\ BG,\ OF\ -\ ground\text{-}sat\text{-}n]$
$bg\text{-}in\ lifted\text{-}q\text{-}calc.ground.entail\text{-}set\text{-}all\text{-}formulas[of\ \mathcal{G}\text{-}Fset\text{-}q\ q\ N\ \mathcal{G}\text{-}Fset\text{-}q\ q\ \{B\}]$
by $simp$
then show $\exists B' \in Bot\text{-}F.\ B' \in N$ **using** $lifted\text{-}q\text{-}calc.Bot\text{-}cond$ **by** $blast$
qed

lemma $sat\text{-}eq\text{-}sat\text{-}empty\text{-}order$: $saturated\ N = empty\text{-}ord.saturated\ N$
by $(rule\ refl)$

lemma $static\text{-}empty\text{-}ord\text{-}inter\text{-}equiv\text{-}static\text{-}inter$:
$statically\text{-}complete\text{-}calculus\ Bot\text{-}F\ Inf\text{-}F\ entails\ Red\text{-}I\ Red\text{-}F =$
$statically\text{-}complete\text{-}calculus\ Bot\text{-}F\ Inf\text{-}F\ entails\ Red\text{-}I\ Red\text{-}F\text{-}\mathcal{G}\text{-}empty$
unfolding $statically\text{-}complete\text{-}calculus\text{-}def$
by $(simp\ add: empty\text{-}ord.calculus\text{-}axioms\ calculus\text{-}axioms)$

theorem $stat\text{-}eq\text{-}dyn\text{-}ref\text{-}comp\text{-}fam\text{-}inter$: $statically\text{-}complete\text{-}calculus\ Bot\text{-}F\ Inf\text{-}F$
$entails\ Red\text{-}I\ Red\text{-}F\text{-}\mathcal{G}\text{-}empty =$
$dynamically\text{-}complete\text{-}calculus\ Bot\text{-}F\ Inf\text{-}F\ entails\ Red\text{-}I\ Red\text{-}F$

using *dyn-equiv-stat static-empty-ord-inter-equiv-static-inter*
by *blast*

end

end

5 Labeled Lifting to Non-Ground Calculi

This section formalizes the extension of the lifting results to labeled calculi. This corresponds to section 3.4 of the report.

theory *Labeled-Lifting-to-Non-Ground-Calculi*
imports *Lifting-to-Non-Ground-Calculi*
begin

5.1 Labeled Lifting with a Family of Tiebreaker Orderings

locale *labeled-tiebreaker-lifting = no-labels*: *tiebreaker-lifting Bot-F Inf-F*
Bot-G entails-G Inf-G Red-I-G Red-F-G $\mathcal{G}$*-F* $\mathcal{G}$*-I Prec-F*
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
entails-G :: *'g set* ⇒ *'g set* ⇒ *bool* (**infix** ‹$\models G$› *50*) **and**
Inf-G :: *'g inference set* **and**
Red-I-G :: *'g set* ⇒ *'g inference set* **and**
Red-F-G :: *'g set* ⇒ *'g set* **and**
$\mathcal{G}$*-F* :: *'f* ⇒ *'g set* **and**
$\mathcal{G}$*-I* :: *'f inference* ⇒ *'g inference set option* **and**
Prec-F :: *'g* ⇒ *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹$\sqsubset$› *50*)
+ **fixes**
Inf-FL :: ‹*('f* × *'l) inference set*›
assumes
Inf-F-to-Inf-FL: ‹ι_F ∈ *Inf-F* ⟹ *length* (*Ll* :: *'l list*) = *length* (*prems-of* ι_F) ⟹
∃ *L0*. *Infer* (*zip* (*prems-of* ι_F) *Ll*) (*concl-of* ι_F, *L0*) ∈ *Inf-FL*› **and**
Inf-FL-to-Inf-F: ‹ι_{FL} ∈ *Inf-FL* ⟹ *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL})) ∈ *Inf-F*›
begin

definition *to-F* :: ‹*('f* × *'l) inference* ⇒ *'f inference*› **where**
‹*to-F* ι_{FL} = *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL}))›

abbreviation *Bot-FL* :: ‹*('f* × *'l) set*› **where**
‹*Bot-FL* ≡ *Bot-F* × *UNIV*›

abbreviation $\mathcal{G}$*-F-L* :: ‹*('f* × *'l)* ⇒ *'g set*› **where**
‹$\mathcal{G}$*-F-L CL* ≡ $\mathcal{G}$*-F* (*fst CL*)›

abbreviation $\mathcal{G}$*-I-L* :: ‹*('f* × *'l) inference* ⇒ *'g inference set option*› **where**
‹$\mathcal{G}$*-I-L* ι_{FL} ≡ $\mathcal{G}$*-I* (*to-F* ι_{FL})›

sublocale *standard-lifting Inf-FL Bot-G Inf-G* ($\models G$) *Red-I-G Red-F-G Bot-FL* $\mathcal{G}$*-F-L* $\mathcal{G}$*-I-L*
proof
show *Bot-FL* ≠ {}

using *no-labels.Bot-F-not-empty* **by** *simp*
next
show $B \in$ *Bot-FL* $\Longrightarrow \mathcal{G}$*-F-L* $B \neq \{\}$ **for** B
using *no-labels.Bot-map-not-empty* **by** *auto*
next
show $B \in$ *Bot-FL* $\Longrightarrow \mathcal{G}$*-F-L* $B \subseteq$ *Bot-G* **for** B
using *no-labels.Bot-map* **by** *force*
next
fix *CL*
show $\mathcal{G}$*-F-L CL* $\cap$ *Bot-G* $\neq \{\} \longrightarrow$ *CL* $\in$ *Bot-FL*
using *no-labels.Bot-cond* **by** (*metis SigmaE UNIV-I UNIV-Times-UNIV mem-Sigma-iff prod.sel*(*1*))
next
fix ι
assume
i-in: ‹$\iota \in$ *Inf-FL*› **and**
ground-not-none: ‹$\mathcal{G}$*-I-L* $\iota \neq$ *None*›
then show *the* ($\mathcal{G}$*-I-L* ι) $\subseteq$ *Red-I-G* ($\mathcal{G}$*-F-L* (*concl-of* ι))
unfolding *to-F-def* **using** *no-labels.inf-map Inf-FL-to-Inf-F* **by** *fastforce*
qed

notation *entails-*$\mathcal{G}$ (**infix** ‹$\models\mathcal{G}$*L*› *50*)

lemma *labeled-entailment-lifting*: *NL1* $\models\mathcal{G}$*L NL2* $\longleftrightarrow$ *fst* ‘ *NL1* $\models\mathcal{G}$ *fst* ‘ *NL2*
by *simp*

lemma *red-inf-impl*: $\iota \in$ *Red-I-*$\mathcal{G}$ *NL* $\Longrightarrow$ *to-F* $\iota \in$ *no-labels.Red-I-*$\mathcal{G}$ (*fst* ‘ *NL*)
unfolding *Red-I-*$\mathcal{G}$*-def no-labels.Red-I-*$\mathcal{G}$*-def* **using** *Inf-FL-to-Inf-F* **by** (*auto simp*: *to-F-def*)

lemma *labeled-saturation-lifting*: *saturated NL* $\Longrightarrow$ *no-labels.saturated* (*fst* ‘ *NL*)
unfolding *saturated-def no-labels.saturated-def Inf-from-def no-labels.Inf-from-def*
proof *clarify*
fix ι
assume
subs-Red-I: {$\iota \in$ *Inf-FL*. *set* (*prems-of* ι) $\subseteq$ *NL*} $\subseteq$ *Red-I-*$\mathcal{G}$ *NL* **and**
i-in: $\iota \in$ *Inf-F* **and**
i-prems: *set* (*prems-of* ι) $\subseteq$ *fst* ‘ *NL*
define *Lli* **where** *Lli i* = (*SOME x*. ((*prems-of* ι)!*i*,*x*) $\in$ *NL*) **for** *i*
have [*simp*]:((*prems-of* ι)!*i*,*Lli i*) $\in$ *NL* **if** *i* < *length* (*prems-of* ι) **for** *i*
using *that i-prems* **unfolding** *Lli-def* **by** (*metis nth-mem someI-ex DomainE Domain-fst subset-eq*)
define *Ll* **where** *Ll* = *map Lli* [*0*..<*length* (*prems-of* ι)]
have *Ll-length*: *length Ll* = *length* (*prems-of* ι) **unfolding** *Ll-def* **by** *auto*
have *subs-NL*: *set* (*zip* (*prems-of* ι) *Ll*) $\subseteq$ *NL* **unfolding** *Ll-def* **by** (*auto simp*:*in-set-zip*)
obtain *L0* **where** *L0*: *Infer* (*zip* (*prems-of* ι) *Ll*) (*concl-of* ι, *L0*) $\in$ *Inf-FL*
using *Inf-F-to-Inf-FL*[*OF i-in Ll-length*] **..**
define ι*-FL* **where** ι*-FL* = *Infer* (*zip* (*prems-of* ι) *Ll*) (*concl-of* ι, *L0*)
then have *set* (*prems-of* ι*-FL*) $\subseteq$ *NL* **using** *subs-NL* **by** *simp*
then have ι*-FL* $\in$ {$\iota \in$ *Inf-FL*. *set* (*prems-of* ι) $\subseteq$ *NL*} **unfolding** ι*-FL-def* **using** *L0* **by** *blast*
then have ι*-FL* $\in$ *Red-I-*$\mathcal{G}$ *NL* **using** *subs-Red-I* **by** *fast*
moreover have ι = *to-F* ι*-FL* **unfolding** *to-F-def* ι*-FL-def* **using** *Ll-length* **by** (*cases* ι) *auto*
ultimately show $\iota \in$ *no-labels.Red-I-*$\mathcal{G}$ (*fst* ‘ *NL*) **by** (*auto intro*: *red-inf-impl*)
qed

lemma *stat-ref-comp-to-labeled-sta-ref-comp*:
  assumes *static*:
    statically-complete-calculus Bot-F Inf-F ($\models\mathcal{G}$) *no-labels.Red-I-*$\mathcal{G}$ *no-labels.Red-F-*$\mathcal{G}$
  shows *statically-complete-calculus Bot-FL Inf-FL* ($\models\mathcal{G}L$) *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
proof
  fix *Bl* :: ‹$'f \times 'l$› **and** *Nl* :: ‹($'f \times 'l$) *set*›
  assume
    Bl-in: ‹*Bl* ∈ *Bot-FL*› **and**
    Nl-sat: ‹*saturated Nl*› **and**
    Nl-entails-Bl: ‹*Nl* $\models\mathcal{G}L$ {*Bl*}›
  define *B* **where** *B* = *fst Bl*
  have *B-in*: *B* ∈ *Bot-F* **using** *Bl-in B-def SigmaE* **by** *force*
  define *N* **where** *N* = *fst* ‘ *Nl*
  have *N-sat*: *no-labels.saturated N*
    using *N-def Nl-sat labeled-saturation-lifting* **by** *blast*
  have *N-entails-B*: *N* $\models\mathcal{G}$ {*B*}
    using *Nl-entails-Bl* **unfolding** *labeled-entailment-lifting N-def B-def* **by** *force*
  have ∃ *B′* ∈ *Bot-F*. *B′* ∈ *N* **using** *B-in N-sat N-entails-B*
    using *static*[*unfolded statically-complete-calculus-def*
      statically-complete-calculus-axioms-def] **by** *blast*
  then obtain *B′* **where** *in-Bot*: *B′* ∈ *Bot-F* **and** *in-N*: *B′* ∈ *N* **by** *force*
  then have *B′* ∈ *fst* ‘ *Bot-FL* **by** *fastforce*
  obtain *Bl′* **where** *in-Nl*: *Bl′* ∈ *Nl* **and** *fst-Bl′*: *fst Bl′* = *B′*
    using *in-N* **unfolding** *N-def* **by** *blast*
  have *Bl′* ∈ *Bot-FL* **using** *fst-Bl′ in-Bot vimage-fst* **by** *fastforce*
  then show ‹∃ *Bl′*∈*Bot-FL*. *Bl′* ∈ *Nl*› **using** *in-Nl* **by** *blast*
qed

end

5.2 Labeled Lifting with a Family of Redundancy Criteria

locale *labeled-lifting-intersection* = *no-labels*: *lifting-intersection Inf-F*
  Bot-G Q Inf-G-q entails-q Red-I-q Red-F-q Bot-F $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q* λ*g Cl Cl′. False*
  for
    Bot-F :: $'f$ *set* **and**
    Inf-F :: $'f$ *inference set* **and**
    Bot-G :: $'g$ *set* **and**
    Q :: $'q$ *set* **and**
    entails-q :: $'q \Rightarrow 'g$ *set* ⇒ $'g$ *set* ⇒ *bool* **and**
    Inf-G-q :: $'q \Rightarrow 'g$ *inference set* **and**
    Red-I-q :: $'q \Rightarrow 'g$ *set* ⇒ $'g$ *inference set* **and**
    Red-F-q :: $'q \Rightarrow 'g$ *set* ⇒ $'g$ *set* **and**
    $\mathcal{G}$*-F-q* :: $'q \Rightarrow 'f \Rightarrow 'g$ *set* **and**
    $\mathcal{G}$*-I-q* :: $'q \Rightarrow 'f$ *inference* ⇒ $'g$ *inference set option*
  + **fixes**
    Inf-FL :: ‹($'f \times 'l$) *inference set*›
  assumes
    Inf-F-to-Inf-FL:
      ‹ι_F ∈ *Inf-F* ⟹ *length* (*Ll* :: $'l$ *list*) = *length* (*prems-of* ι_F) ⟹
      ∃ *L0*. *Infer* (*zip* (*prems-of* ι_F) *Ll*) (*concl-of* ι_F, *L0*) ∈ *Inf-FL*› **and**
    Inf-FL-to-Inf-F: ‹ι_{FL} ∈ *Inf-FL* ⟹ *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL})) ∈ *Inf-F*›
begin

definition *to-F* :: ‹('*f* × '*l*) *inference* ⇒ '*f inference*› **where**
‹*to-F* ι_{FL} = *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL}))›

abbreviation *Bot-FL* :: ‹('*f* × '*l*) *set*› **where**
‹*Bot-FL* ≡ *Bot-F* × *UNIV*›

abbreviation $\mathcal{G}$*-F-L-q* :: ‹'*q* ⇒ ('*f* × '*l*) ⇒ '*g set*› **where**
‹$\mathcal{G}$*-F-L-q q CL* ≡ $\mathcal{G}$*-F-q q* (*fst CL*)›

abbreviation $\mathcal{G}$*-I-L-q* :: ‹'*q* ⇒ ('*f* × '*l*) *inference* ⇒ '*g inference set option*› **where**
‹$\mathcal{G}$*-I-L-q q* ι_{FL} ≡ $\mathcal{G}$*-I-q q* (*to-F* ι_{FL})›

abbreviation $\mathcal{G}$*-Fset-L-q* :: '*q* ⇒ ('*f* × '*l*) *set* ⇒ '*g set* **where**
$\mathcal{G}$*-Fset-L-q q N* ≡ $\bigcup$ ($\mathcal{G}$*-F-L-q q* ‘ *N*)

definition *Red-I-*$\mathcal{G}$*-L-q* :: '*q* ⇒ ('*f* × '*l*) *set* ⇒ ('*f* × '*l*) *inference set* **where**
Red-I-$\mathcal{G}$*-L-q q N* =
{ι ∈ *Inf-FL*. ($\mathcal{G}$*-I-L-q q* ι ≠ *None* ∧ *the* ($\mathcal{G}$*-I-L-q q* ι) ⊆ *Red-I-q q* ($\mathcal{G}$*-Fset-L-q q N*))
∨ ($\mathcal{G}$*-I-L-q q* ι = *None* ∧ $\mathcal{G}$*-F-L-q q* (*concl-of* ι) ⊆ $\mathcal{G}$*-Fset-L-q q N* ∪ *Red-F-q q* ($\mathcal{G}$*-Fset-L-q q N*))}

abbreviation *Red-I-*$\mathcal{G}$*-L* :: ('*f* × '*l*) *set* ⇒ ('*f* × '*l*) *inference set* **where**
Red-I-$\mathcal{G}$*-L N* ≡ ($\bigcap$ *q* ∈ *Q*. *Red-I-*$\mathcal{G}$*-L-q q N*)

abbreviation *entails-*$\mathcal{G}$*-L-q* :: '*q* ⇒ ('*f* × '*l*) *set* ⇒ ('*f* × '*l*) *set* ⇒ *bool* **where**
entails-$\mathcal{G}$*-L-q q N1 N2* ≡ *entails-q q* ($\mathcal{G}$*-Fset-L-q q N1*) ($\mathcal{G}$*-Fset-L-q q N2*)

lemma *lifting-q*:
assumes *q* ∈ *Q*
shows *labeled-tiebreaker-lifting Bot-F Inf-F Bot-G* (*entails-q q*) (*Inf-G-q q*) (*Red-I-q q*)
(*Red-F-q q*) ($\mathcal{G}$*-F-q q*) ($\mathcal{G}$*-I-q q*) (λ*g Cl Cl'*. *False*) *Inf-FL*
using *assms no-labels.standard-lifting-family Inf-F-to-Inf-FL Inf-FL-to-Inf-F*
by (*simp add*: *labeled-tiebreaker-lifting-axioms-def labeled-tiebreaker-lifting-def*)

lemma *lifted-q*:
assumes *q-in*: *q* ∈ *Q*
shows *standard-lifting Inf-FL Bot-G* (*Inf-G-q q*) (*entails-q q*) (*Red-I-q q*) (*Red-F-q q*)
Bot-FL ($\mathcal{G}$*-F-L-q q*) ($\mathcal{G}$*-I-L-q q*)
proof −
interpret *q-lifting*: *labeled-tiebreaker-lifting Bot-F Inf-F Bot-G entails-q q Inf-G-q q*
Red-I-q q Red-F-q q $\mathcal{G}$*-F-q q* $\mathcal{G}$*-I-q q* λ*g Cl Cl'*. *False Inf-FL*
using *lifting-q*[*OF q-in*] **.**
have $\mathcal{G}$*-I-L-q q* = *q-lifting.*$\mathcal{G}$*-I-L*
unfolding *to-F-def q-lifting.to-F-def* **by** *simp*
then show *?thesis*
using *q-lifting.standard-lifting-axioms* **by** *simp*
qed

lemma *ord-fam-lifted-q*:
assumes *q-in*: *q* ∈ *Q*
shows *tiebreaker-lifting Bot-FL Inf-FL Bot-G* (*entails-q q*) (*Inf-G-q q*) (*Red-I-q q*)
(*Red-F-q q*) ($\mathcal{G}$*-F-L-q q*) ($\mathcal{G}$*-I-L-q q*) (λ*g Cl Cl'*. *False*)
proof −
interpret *standard-q-lifting*: *standard-lifting Inf-FL Bot-G Inf-G-q q entails-q q*
Red-I-q q Red-F-q q Bot-FL $\mathcal{G}$*-F-L-q q* $\mathcal{G}$*-I-L-q q*
using *lifted-q*[*OF q-in*] **.**

show *?thesis*
by *unfold-locales auto*
qed

definition *Red-F-$\mathcal{G}$-empty-L-q* :: *'q* ⇒ (*'f* × *'l*) *set* ⇒ (*'f* × *'l*) *set* **where**
Red-F-$\mathcal{G}$-empty-L-q q N = {*C*. ∀ *D* ∈ *$\mathcal{G}$-F-L-q q C*. *D* ∈ *Red-F-q q* (*$\mathcal{G}$-Fset-L-q q N*) ∨
(∃ *E* ∈ *N*. *False* ∧ *D* ∈ *$\mathcal{G}$-F-L-q q E*)}

abbreviation *Red-F-$\mathcal{G}$-empty-L* :: (*'f* × *'l*) *set* ⇒ (*'f* × *'l*) *set* **where**
Red-F-$\mathcal{G}$-empty-L N ≡ (⋂ *q* ∈ *Q*. *Red-F-$\mathcal{G}$-empty-L-q q N*)

lemma *all-lifted-red-crit*:
assumes *q-in*: *q* ∈ *Q*
shows *calculus Bot-FL Inf-FL* (*entails-$\mathcal{G}$-L-q q*) (*Red-I-$\mathcal{G}$-L-q q*) (*Red-F-$\mathcal{G}$-empty-L-q q*)
proof −
interpret *ord-q-lifting*: *tiebreaker-lifting Bot-FL Inf-FL Bot-G entails-q q*
Inf-G-q q Red-I-q q Red-F-q q $\mathcal{G}$-F-L-q q $\mathcal{G}$-I-L-q q λ*g Cl Cl'*. *False*
using *ord-fam-lifted-q*[*OF q-in*] **.**
have *Red-I-$\mathcal{G}$-L-q q* = *ord-q-lifting.Red-I-$\mathcal{G}$*
unfolding *Red-I-$\mathcal{G}$-L-q-def ord-q-lifting.Red-I-$\mathcal{G}$-def* **by** *simp*
moreover have *Red-F-$\mathcal{G}$-empty-L-q q* = *ord-q-lifting.Red-F-$\mathcal{G}$*
unfolding *Red-F-$\mathcal{G}$-empty-L-q-def ord-q-lifting.Red-F-$\mathcal{G}$-def* **by** *simp*
ultimately show *?thesis*
using *ord-q-lifting.calculus-axioms* **by** *argo*
qed

lemma *all-lifted-cons-rel*:
assumes *q-in*: *q* ∈ *Q*
shows *consequence-relation Bot-FL* (*entails-$\mathcal{G}$-L-q q*)
using *all-lifted-red-crit calculus-def q-in* **by** *blast*

sublocale *consequence-relation-family Bot-FL Q entails-$\mathcal{G}$-L-q*
using *all-lifted-cons-rel* **by** (*simp add*: *consequence-relation-family.intro no-labels.Q-nonempty*)

sublocale *intersection-calculus Bot-FL Inf-FL Q entails-$\mathcal{G}$-L-q Red-I-$\mathcal{G}$-L-q Red-F-$\mathcal{G}$-empty-L-q*
using *intersection-calculus.intro*[*OF consequence-relation-family-axioms*]
by (*simp add*: *all-lifted-red-crit intersection-calculus-axioms-def no-labels.Q-nonempty*)

lemma *in-Inf-FL-imp-to-F-in-Inf-F*: *ι* ∈ *Inf-FL* ⟹ *to-F ι* ∈ *Inf-F*
by (*simp add*: *Inf-FL-to-Inf-F to-F-def*)

lemma *in-Inf-from-imp-to-F-in-Inf-from*: *ι* ∈ *Inf-from N* ⟹ *to-F ι* ∈ *no-labels.Inf-from* (*fst* ‘ *N*)
unfolding *Inf-from-def no-labels.Inf-from-def to-F-def* **by** (*auto intro*: *Inf-FL-to-Inf-F*)

notation *no-labels.entails-$\mathcal{G}$* (**infix** ‹⊨∩$\mathcal{G}$› *50*)

abbreviation *entails-$\mathcal{G}$-L* :: (*'f* × *'l*) *set* ⇒ (*'f* × *'l*) *set* ⇒ *bool* (**infix** ‹⊨∩$\mathcal{G}$*L*› *50*) **where**
(⊨∩$\mathcal{G}$*L*) ≡ *entails*

lemmas *entails-$\mathcal{G}$-L-def* = *entails-def*

lemma *labeled-entailment-lifting*: *NL1* ⊨∩$\mathcal{G}$*L NL2* ⟷ *fst* ‘ *NL1* ⊨∩$\mathcal{G}$ *fst* ‘ *NL2*
unfolding *no-labels.entails-$\mathcal{G}$-def entails-$\mathcal{G}$-L-def* **by** *force*

lemma *red-inf-impl*: $\iota \in$ *Red-I NL* $\Longrightarrow$ *to-F* $\iota \in$ *no-labels.Red-I-*$\mathcal{G}$ (*fst* ‘ *NL*)
unfolding *no-labels.Red-I-*$\mathcal{G}$*-def Red-I-def*
proof *clarify*
fix *X Xa q*
assume
q-in: $q \in Q$ **and**
i-in-inter: $\iota \in (\bigcap q \in Q.$ *Red-I-*$\mathcal{G}$*-L-q q NL*)
have *i-in-q*: $\iota \in$ *Red-I-*$\mathcal{G}$*-L-q q NL* **using** *q-in i-in-inter image-eqI* **by** *blast*
then have *i-in*: $\iota \in$ *Inf-FL* **unfolding** *Red-I-*$\mathcal{G}$*-L-q-def* **by** *blast*
have *to-F-in*: *to-F* $\iota \in$ *Inf-F* **unfolding** *to-F-def* **using** *Inf-FL-to-Inf-F*[*OF i-in*] **.**
have *rephrase1*: $(\bigcup$ *CL*$\in$*NL.* $\mathcal{G}$*-F-q q* (*fst CL*)) = $(\bigcup$ ($\mathcal{G}$*-F-q q* ‘ *fst* ‘ *NL*)) **by** *blast*
have *rephrase2*: *fst* (*concl-of* ι) = *concl-of* (*to-F* ι)
unfolding *concl-of-def to-F-def* **by** *simp*
have *subs-red*: ($\mathcal{G}$*-I-L-q q* $\iota \neq$ *None* $\wedge$ *the* ($\mathcal{G}$*-I-L-q q* ι) $\subseteq$ *Red-I-q q* ($\mathcal{G}$*-Fset-L-q q NL*))
$\vee$ ($\mathcal{G}$*-I-L-q q* ι = *None* $\wedge$ $\mathcal{G}$*-F-L-q q* (*concl-of* ι) $\subseteq$ $\mathcal{G}$*-Fset-L-q q NL* $\cup$ *Red-F-q q* ($\mathcal{G}$*-Fset-L-q q NL*))
using *i-in-q* **unfolding** *Red-I-*$\mathcal{G}$*-L-q-def* **by** *blast*
then have *to-F-subs-red*: ($\mathcal{G}$*-I-q q* (*to-F* ι) $\neq$ *None* $\wedge$
the ($\mathcal{G}$*-I-q q* (*to-F* ι)) $\subseteq$ *Red-I-q q* (*no-labels.*$\mathcal{G}$*-Fset-q q* (*fst* ‘ *NL*)))
$\vee$ ($\mathcal{G}$*-I-q q* (*to-F* ι) = *None* $\wedge$
$\mathcal{G}$*-F-q q* (*concl-of* (*to-F* ι))
$\subseteq$ *no-labels.*$\mathcal{G}$*-Fset-q q* (*fst* ‘ *NL*) $\cup$ *Red-F-q q* (*no-labels.*$\mathcal{G}$*-Fset-q q* (*fst* ‘ *NL*)))
using *rephrase1 rephrase2* **by** *metis*
then show *to-F* $\iota \in$ *no-labels.Red-I-*$\mathcal{G}$*-q q* (*fst* ‘ *NL*)
using *to-F-in* **unfolding** *no-labels.Red-I-*$\mathcal{G}$*-q-def* **by** *simp*
qed

lemma *labeled-family-saturation-lifting*: *saturated NL* $\Longrightarrow$ *no-labels.saturated* (*fst* ‘ *NL*)
unfolding *saturated-def no-labels.saturated-def Inf-from-def no-labels.Inf-from-def*
proof *clarify*
fix ιF
assume
labeled-sat: {$\iota \in$ *Inf-FL. set* (*prems-of* ι) $\subseteq$ *NL*} $\subseteq$ *Red-I NL* **and**
iF-in: $\iota F \in$ *Inf-F* **and**
iF-prems: *set* (*prems-of* ιF) $\subseteq$ *fst* ‘ *NL*
define *Lli* **where** *Lli i* = (*SOME x.* ((*prems-of* ιF)!*i*,*x*) $\in$ *NL*) **for** *i*
have [*simp*]:((*prems-of* ιF)!*i*,*Lli i*) $\in$ *NL* **if** *i* < *length* (*prems-of* ιF) **for** *i*
using *that iF-prems nth-mem someI-ex* **unfolding** *Lli-def* **by** (*metis DomainE Domain-fst subset-eq*)
define *Ll* **where** *Ll* = *map Lli* [*0*..<*length* (*prems-of* ιF)]
have *Ll-length*: *length Ll* = *length* (*prems-of* ιF) **unfolding** *Ll-def* **by** *auto*
have *subs-NL*: *set* (*zip* (*prems-of* ιF) *Ll*) $\subseteq$ *NL* **unfolding** *Ll-def* **by** (*auto simp*:*in-set-zip*)
obtain *L0* **where** *L0*: *Infer* (*zip* (*prems-of* ιF) *Ll*) (*concl-of* ιF, *L0*) $\in$ *Inf-FL*
using *Inf-F-to-Inf-FL*[*OF iF-in Ll-length*] **..**
define ιFL **where** ιFL = *Infer* (*zip* (*prems-of* ιF) *Ll*) (*concl-of* ιF, *L0*)
then have *set* (*prems-of* ιFL) $\subseteq$ *NL* **using** *subs-NL* **by** *simp*
then have $\iota FL \in$ {$\iota \in$ *Inf-FL. set* (*prems-of* ι) $\subseteq$ *NL*} **unfolding** ιFL*-def* **using** *L0* **by** *blast*
then have $\iota FL \in$ *Red-I NL* **using** *labeled-sat* **by** *fast*
moreover have ιF = *to-F* ιFL **unfolding** *to-F-def* ιFL*-def* **using** *Ll-length* **by** (*cases* ιF) *auto*
ultimately show $\iota F \in$ *no-labels.Red-I-*$\mathcal{G}$ (*fst* ‘ *NL*)
by (*auto intro*: *red-inf-impl*)
qed

theorem *labeled-static-ref*:

assumes *calc*: *statically-complete-calculus Bot-F Inf-F* ($\models\cap\mathcal{G}$) *no-labels.Red-I-*$\mathcal{G}$
no-labels.Red-F-$\mathcal{G}$*-empty*
shows *statically-complete-calculus Bot-FL Inf-FL* ($\models\cap\mathcal{G}L$) *Red-I Red-F*
proof
fix *Bl* :: ‹$'f \times 'l$› **and** *Nl* :: ‹($'f \times 'l$) *set*›
assume
Bl-in: ‹*Bl* ∈ *Bot-FL*› **and**
Nl-sat: ‹*saturated Nl*› **and**
Nl-entails-Bl: ‹*Nl* $\models\cap\mathcal{G}L$ {*Bl*}›
define *B* **where** *B* = *fst Bl*
have *B-in*: *B* ∈ *Bot-F* **using** *Bl-in B-def SigmaE* **by** *force*
define *N* **where** *N* = *fst* ‘ *Nl*
have *N-sat*: *no-labels.saturated N*
using *N-def Nl-sat labeled-family-saturation-lifting* **by** *blast*
have *N-entails-B*: *N* $\models\cap\mathcal{G}$ {*B*}
using *Nl-entails-Bl* **unfolding** *labeled-entailment-lifting N-def B-def* **by** *force*
have ∃ *B'* ∈ *Bot-F*. *B'* ∈ *N* **using** *B-in N-sat N-entails-B*
calc[*unfolded statically-complete-calculus-def*
statically-complete-calculus-axioms-def]
by *blast*
then obtain *B'* **where** *in-Bot*: *B'* ∈ *Bot-F* **and** *in-N*: *B'* ∈ *N* **by** *force*
then have *B'* ∈ *fst* ‘ *Bot-FL* **by** *fastforce*
obtain *Bl'* **where** *in-Nl*: *Bl'* ∈ *Nl* **and** *fst-Bl'*: *fst Bl'* = *B'*
using *in-N* **unfolding** *N-def* **by** *blast*
have *Bl'* ∈ *Bot-FL* **using** *fst-Bl' in-Bot vimage-fst* **by** *fastforce*
then show ‹∃ *Bl'*∈*Bot-FL*. *Bl'* ∈ *Nl*› **using** *in-Nl* **by** *blast*
qed

end

end

6 Given Clause Prover Architectures

This section covers all the results presented in the section 4 of the report. This is where abstract architectures of provers are defined and proven dynamically refutationally complete.

theory *Given-Clause-Architectures*
imports
Lambda-Free-RPOs.Lambda-Free-Util
Labeled-Lifting-to-Non-Ground-Calculi
begin

6.1 Basis of the Given Clause Prover Architectures

locale *given-clause-basis* = *std?*: *labeled-lifting-intersection Bot-F Inf-F Bot-G Q*
entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q*
{ι_{FL} :: ($'f \times 'l$) *inference*. *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL})) ∈ *Inf-F*}
for
Bot-F :: $'f$ *set*
and *Inf-F* :: $'f$ *inference set*
and *Bot-G* :: $'g$ *set*
and *Q* :: $'q$ *set*
and *entails-q* :: $'q \Rightarrow 'g$ *set* ⇒ $'g$ *set* ⇒ *bool*
and *Inf-G-q* :: ‹$'q \Rightarrow 'g$ *inference set*›

and *Red-I-q* :: $'q \Rightarrow 'g\ set \Rightarrow 'g\ inference\ set$
and *Red-F-q* :: $'q \Rightarrow 'g\ set \Rightarrow 'g\ set$
and $\mathcal{G}$*-F-q* :: $'q \Rightarrow 'f \Rightarrow 'g\ set$
and $\mathcal{G}$*-I-q* :: $'q \Rightarrow 'f\ inference \Rightarrow 'g\ inference\ set\ option$
+ **fixes**
Equiv-F :: $'f \Rightarrow 'f \Rightarrow bool$ (**infix** $\langle \doteq \rangle$ *50*) **and**
Prec-F :: $'f \Rightarrow 'f \Rightarrow bool$ (**infix** $\langle \prec\cdot \rangle$ *50*) **and**
Prec-L :: $'l \Rightarrow 'l \Rightarrow bool$ (**infix** $\langle \sqsubset L \rangle$ *50*) **and**
active :: $'l$
assumes
equiv-equiv-F: *equivp* $(\doteq)$ **and**
wf-prec-F: *wfp* $(\prec\cdot)$ *transp* $(\prec\cdot)$ **and**
wf-prec-L: *wfp* $(\sqsubset L)$ *transp* $(\sqsubset L)$ **and**
compat-equiv-prec: $C1 \doteq D1 \Longrightarrow C2 \doteq D2 \Longrightarrow C1 \prec\cdot C2 \Longrightarrow D1 \prec\cdot D2$ **and**
equiv-F-grounding: $q \in Q \Longrightarrow C1 \doteq C2 \Longrightarrow \mathcal{G}\text{-}F\text{-}q\ q\ C1 \subseteq \mathcal{G}\text{-}F\text{-}q\ q\ C2$ **and**
prec-F-grounding: $q \in Q \Longrightarrow C2 \prec\cdot C1 \Longrightarrow \mathcal{G}\text{-}F\text{-}q\ q\ C1 \subseteq \mathcal{G}\text{-}F\text{-}q\ q\ C2$ **and**
active-minimal: $l2 \neq active \Longrightarrow active \sqsubset L\ l2$ **and**
at-least-two-labels: $\exists\, l2.\ active \sqsubset L\ l2$ **and**
static-ref-comp: *statically-complete-calculus Bot-F Inf-F* $(\models \cap \mathcal{G})$
no-labels.Red-I-$\mathcal{G}$ *no-labels.Red-F-*$\mathcal{G}$*-empty*
begin

abbreviation *Inf-FL* :: $('f \times 'l)$ *inference set* **where**
Inf-FL $\equiv \{\iota_{FL}.\ Infer\ (map\ fst\ (prems\text{-}of\ \iota_{FL}))\ (fst\ (concl\text{-}of\ \iota_{FL})) \in Inf\text{-}F\}$

abbreviation *Prec-eq-F* :: $'f \Rightarrow 'f \Rightarrow bool$ (**infix** $\langle \preceq\cdot \rangle$ *50*) **where**
$C \preceq\cdot D \equiv C \doteq D \vee C \prec\cdot D$

definition *Prec-FL* :: $('f \times 'l) \Rightarrow ('f \times 'l) \Rightarrow bool$ (**infix** $\langle \sqsubset \rangle$ *50*) **where**
$Cl1 \sqsubset Cl2 \longleftrightarrow fst\ Cl1 \prec\cdot fst\ Cl2 \vee (fst\ Cl1 \doteq fst\ Cl2 \wedge snd\ Cl1 \sqsubset L\ snd\ Cl2)$

lemma *irrefl-prec-F*: $\neg\ C \prec\cdot C$
using *wf-prec-F*
by (*meson asympD wfp-imp-asymp*)

lemma *trans-prec-F*: $C1 \prec\cdot C2 \Longrightarrow C2 \prec\cdot C3 \Longrightarrow C1 \prec\cdot C3$
using *wf-prec-F*
by (*meson transpE*)

lemma *wf-prec-FL*: *wfp* $(\sqsubset)$ *transp* $(\sqsubset)$
proof −
show *transp* $(\sqsubset)$ **unfolding** *Prec-FL-def*
proof (*rule transpI*)
fix *Cl1 Cl2 Cl3*
assume *trans-hyps*:
$(fst\ Cl1 \prec\cdot fst\ Cl2 \vee fst\ Cl1 \doteq fst\ Cl2 \wedge snd\ Cl1 \sqsubset L\ snd\ Cl2)$
$(fst\ Cl2 \prec\cdot fst\ Cl3 \vee fst\ Cl2 \doteq fst\ Cl3 \wedge snd\ Cl2 \sqsubset L\ snd\ Cl3)$
have $fst\ Cl1 \prec\cdot fst\ Cl2 \Longrightarrow fst\ Cl2 \doteq fst\ Cl3 \Longrightarrow fst\ Cl1 \prec\cdot fst\ Cl3$
using *compat-equiv-prec* **by** (*metis equiv-equiv-F equivp-def*)
moreover have $fst\ Cl1 \doteq fst\ Cl2 \Longrightarrow fst\ Cl2 \prec\cdot fst\ Cl3 \Longrightarrow fst\ Cl1 \prec\cdot fst\ Cl3$
using *compat-equiv-prec* **by** (*metis equiv-equiv-F equivp-def*)
moreover have $snd\ Cl1 \sqsubset L\ snd\ Cl2 \Longrightarrow snd\ Cl2 \sqsubset L\ snd\ Cl3 \Longrightarrow snd\ Cl1 \sqsubset L\ snd\ Cl3$
using *wf-prec-L* **unfolding** *transp-def* **by** (*meson UNIV-I*)
moreover have $fst\ Cl1 \doteq fst\ Cl2 \Longrightarrow fst\ Cl2 \doteq fst\ Cl3 \Longrightarrow fst\ Cl1 \doteq fst\ Cl3$
using *equiv-equiv-F* **by** (*meson equivp-transp*)

```
    ultimately show fst Cl1 ≺· fst Cl3 ∨ fst Cl1 ≐ fst Cl3 ∧ snd Cl1 ⊏L snd Cl3
      using trans-hyps trans-prec-F by blast
  qed
next
  {
    assume contra: ∃ f. ∀ i. f (Suc i) ⊏ f i
    then obtain f where
      f-suc: ∀ i. f (Suc i) ⊏ f i
      by blast

    define R :: (('f × 'l) × ('f × 'l)) set where
      R = {(Cl1, Cl2). fst Cl1 ≺· fst Cl2}
    define S :: (('f × 'l) × ('f × 'l)) set where
      S = {(Cl1, Cl2). fst Cl1 ≐ fst Cl2 ∧ snd Cl1 ⊏L snd Cl2}

    obtain k where
      f-chain: ∀ i. (f (Suc (i + k)), f (i + k)) ∈ S
    proof (atomize-elim, rule wf-infinite-down-chain-compatible[of R f S])
      show wf R
        using wfP-app wf-prec-F
        unfolding R-def wfp-def
        by auto
    next
      show ∀ i. (f (Suc i), f i) ∈ R ∪ S
        using f-suc unfolding R-def S-def Prec-FL-def by blast
    next
      show R O S ⊆ R
        unfolding R-def S-def using compat-equiv-prec equiv-equiv-F equivp-reflp by fastforce
    qed

    define g where
      ⋀i. g i = f (i + k)

    have g-chain: ∀ i. (g (Suc i), g i) ∈ S
      unfolding g-def using f-chain by simp
    have wf-s: wf S
      unfolding S-def
      by (rule wf-subset[OF wf-app[OF wf-prec-L(1)[ unfolded wfp-def], of snd]]) fast
    have False
      using g-chain[unfolded S-def] wf-s[unfolded S-def, folded wfp-def, unfolded wfp-on-def]
      by (meson g-chain wf-no-infinite-down-chainE wf-s)
  }

  then show wfp (⊏)
    unfolding infinite-chain-function-iff-infinite-chain-llist wfP-iff-no-infinite-down-chain-llist
    by argo
qed

definition active-subset :: ('f × 'l) set ⇒ ('f × 'l) set where
  active-subset M = {CL ∈ M. snd CL = active}

definition passive-subset :: ('f × 'l) set ⇒ ('f × 'l) set where
  passive-subset M = {CL ∈ M. snd CL ≠ active}

lemma active-subset-insert[simp]:
```

active-subset (*insert Cl N*) = (*if snd Cl = active then* {*Cl*} *else* {}) ∪ *active-subset N*
unfolding *active-subset-def* **by** *auto*

lemma *active-subset-union*[*simp*]: *active-subset* (*M* ∪ *N*) = *active-subset M* ∪ *active-subset N*
unfolding *active-subset-def* **by** *auto*

lemma *passive-subset-insert*[*simp*]:
passive-subset (*insert Cl N*) = (*if snd Cl* ≠ *active then* {*Cl*} *else* {}) ∪ *passive-subset N*
unfolding *passive-subset-def* **by** *auto*

lemma *passive-subset-union*[*simp*]: *passive-subset* (*M* ∪ *N*) = *passive-subset M* ∪ *passive-subset N*
unfolding *passive-subset-def* **by** *auto*

sublocale *std?*: *statically-complete-calculus Bot-FL Inf-FL* (⊨∩*GL*) *Red-I Red-F*
using *labeled-static-ref*[*OF static-ref-comp*] **.**

lemma *labeled-tiebreaker-lifting*:
assumes *q-in*: *q* ∈ *Q*
shows *tiebreaker-lifting Bot-FL Inf-FL Bot-G* (*entails-q q*) (*Inf-G-q q*)
(*Red-I-q q*) (*Red-F-q q*) (*G-F-L-q q*) (*G-I-L-q q*) (λ*g*. *Prec-FL*)
proof −
have *tiebreaker-lifting Bot-FL Inf-FL Bot-G* (*entails-q q*) (*Inf-G-q q*)
(*Red-I-q q*) (*Red-F-q q*) (*G-F-L-q q*) (*G-I-L-q q*) (λ*g Cl Cl*′. *False*)
using *ord-fam-lifted-q*[*OF q-in*] **.**
then have *standard-lifting Inf-FL Bot-G* (*Inf-G-q q*) (*entails-q q*) (*Red-I-q q*)
(*Red-F-q q*) *Bot-FL* (*G-F-L-q q*) (*G-I-L-q q*)
using *lifted-q*[*OF q-in*] **by** *blast*
then show *tiebreaker-lifting Bot-FL Inf-FL Bot-G* (*entails-q q*) (*Inf-G-q q*)
(*Red-I-q q*) (*Red-F-q q*) (*G-F-L-q q*) (*G-I-L-q q*) (λ*g*. *Prec-FL*)
using *wf-prec-FL* **by** (*simp add*: *tiebreaker-lifting.intro tiebreaker-lifting-axioms.intro*)
qed

sublocale *lifting-intersection Inf-FL Bot-G Q Inf-G-q entails-q Red-I-q Red-F-q*
Bot-FL G-F-L-q G-I-L-q λ*g*. *Prec-FL*
using *labeled-tiebreaker-lifting* **unfolding** *lifting-intersection-def*
by (*simp add*: *lifting-intersection-axioms.intro*
no-labels.ground.consequence-relation-family-axioms
no-labels.ground.inference-system-family-axioms)

notation *derive* (**infix** ‹⊳*L*› *50*)

lemma *std-Red-I-eq*: *std.Red-I* = *Red-I-G*
unfolding *Red-I-G-q-def Red-I-G-L-q-def* **by** *simp*

lemma *std-Red-F-eq*: *std.Red-F* = *Red-F-G-empty*
unfolding *Red-F-G-empty-q-def Red-F-G-empty-L-q-def* **by** *simp*

sublocale *statically-complete-calculus Bot-FL Inf-FL* (⊨∩*GL*) *Red-I Red-F*
by *unfold-locales* (*use statically-complete std-Red-I-eq* **in** *auto*)

lemma *labeled-red-inf-eq-red-inf*:
assumes *i-in*: ι ∈ *Inf-FL*
shows ι ∈ *Red-I N* ⟷ *to-F* ι ∈ *no-labels.Red-I-G* (*fst* ‘ *N*)
proof

assume *i-in2*: $\iota \in$ *Red-I N*
then have $X \in$ *Red-I-$\mathcal{G}$-q ‘ Q* $\Longrightarrow \iota \in X\ N$ **for** X
unfolding *Red-I-def* **by** *blast*
obtain *X0* **where** *X0* $\in$ *Red-I-$\mathcal{G}$-q ‘ Q*
using *Q-nonempty* **by** *blast*
then obtain *q0* **where** *x0-is*: *X0 N* = *Red-I-$\mathcal{G}$-q q0 N* **by** *blast*
then obtain *Y0* **where** *y0-is*: *Y0* (*fst ‘ N*) = *to-F ‘* (*X0 N*) **by** *auto*
have *Y0* (*fst ‘ N*) = *no-labels.Red-I-$\mathcal{G}$-q q0* (*fst ‘ N*)
unfolding *y0-is*
proof
show *to-F ‘ X0 N* $\subseteq$ *no-labels.Red-I-$\mathcal{G}$-q q0* (*fst ‘ N*)
proof
fix $\iota 0$
assume *i0-in*: $\iota 0 \in$ *to-F ‘ X0 N*
then have *i0-in2*: $\iota 0 \in$ *to-F ‘ Red-I-$\mathcal{G}$-q q0 N*
using *x0-is* **by** *argo*
then obtain $\iota 0$*-FL* **where** *i0-FL-in*: $\iota 0$*-FL* $\in$ *Inf-FL* **and** *i0-to-i0-FL*: $\iota 0$ = *to-F* $\iota 0$*-FL* **and**
subs1: ((*$\mathcal{G}$-I-L-q q0* $\iota 0$*-FL*) $\neq$ *None* $\wedge$
the (*$\mathcal{G}$-I-L-q q0* $\iota 0$*-FL*) $\subseteq$ *Red-I-q q0* (*$\mathcal{G}$-Fset-q q0 N*))
$\vee$ ((*$\mathcal{G}$-I-L-q q0* $\iota 0$*-FL* = *None*) $\wedge$
$\mathcal{G}$-F-L-q q0 (*concl-of* $\iota 0$*-FL*) $\subseteq$ *$\mathcal{G}$-Fset-q q0 N* $\cup$ *Red-F-q q0* (*$\mathcal{G}$-Fset-q q0 N*))
unfolding *Red-I-$\mathcal{G}$-q-def* **by** *blast*
have *concl-swap*: *fst* (*concl-of* $\iota 0$*-FL*) = *concl-of* $\iota 0$
unfolding *concl-of-def i0-to-i0-FL to-F-def* **by** *simp*
have *i0-in3*: $\iota 0 \in$ *Inf-F*
using *i0-to-i0-FL Inf-FL-to-Inf-F*[*OF i0-FL-in*] **unfolding** *to-F-def* **by** *blast*
{
assume
not-none: *$\mathcal{G}$-I-q q0* $\iota 0 \neq$ *None* **and**
the (*$\mathcal{G}$-I-q q0* $\iota 0$) $\neq \{\}$
then obtain $\iota 1$ **where** *i1-in*: $\iota 1 \in$ *the* (*$\mathcal{G}$-I-q q0* $\iota 0$) **by** *blast*
have *the* (*$\mathcal{G}$-I-q q0* $\iota 0$) $\subseteq$ *Red-I-q q0* (*no-labels.$\mathcal{G}$-Fset-q q0* (*fst ‘ N*))
using *subs1 i0-to-i0-FL not-none* **by** *auto*
}
moreover {
assume
is-none: *$\mathcal{G}$-I-q q0* $\iota 0$ = *None*
then have *$\mathcal{G}$-F-q q0* (*concl-of* $\iota 0$) $\subseteq$ *no-labels.$\mathcal{G}$-Fset-q q0* (*fst ‘ N*)
$\cup$ *Red-F-q q0* (*no-labels.$\mathcal{G}$-Fset-q q0* (*fst ‘ N*))
using *subs1 i0-to-i0-FL concl-swap* **by** *simp*
}
ultimately show $\iota 0 \in$ *no-labels.Red-I-$\mathcal{G}$-q q0* (*fst ‘ N*)
unfolding *no-labels.Red-I-$\mathcal{G}$-q-def* **using** *i0-in3* **by** *auto*
qed
next
show *no-labels.Red-I-$\mathcal{G}$-q q0* (*fst ‘ N*) $\subseteq$ *to-F ‘ X0 N*
proof
fix $\iota 0$
assume *i0-in*: $\iota 0 \in$ *no-labels.Red-I-$\mathcal{G}$-q q0* (*fst ‘ N*)
then have *i0-in2*: $\iota 0 \in$ *Inf-F*
unfolding *no-labels.Red-I-$\mathcal{G}$-q-def* **by** *blast*
obtain $\iota 0$*-FL* **where** *i0-FL-in*: $\iota 0$*-FL* $\in$ *Inf-FL* **and** *i0-to-i0-FL*: $\iota 0$ = *to-F* $\iota 0$*-FL*
using *Inf-F-to-Inf-FL*[*OF i0-in2*] **unfolding** *to-F-def*
by (*smt* (*verit*) *Ex-list-of-length fst-conv inference.exhaust-sel inference.sel*(*1*)
inference.sel(*2*) *map-fst-zip*)

```
    have concl-swap: fst (concl-of ι0-FL) = concl-of ι0
      unfolding concl-of-def i0-to-i0-FL to-F-def by simp
    have subs1: ((𝒢-I-L-q q0 ι0-FL) ≠ None ∧
        the (𝒢-I-L-q q0 ι0-FL) ⊆ Red-I-q q0 (𝒢-Fset-q q0 N))
        ∨ ((𝒢-I-L-q q0 ι0-FL = None) ∧
        𝒢-F-L-q q0 (concl-of ι0-FL) ⊆ (𝒢-Fset-q q0 N ∪ Red-F-q q0 (𝒢-Fset-q q0 N)))
      using i0-in i0-to-i0-FL concl-swap unfolding no-labels.Red-I-𝒢-q-def by simp
    then have ι0-FL ∈ Red-I-𝒢-q q0 N
      using i0-FL-in unfolding Red-I-𝒢-q-def by simp
    then show ι0 ∈ to-F ‘ X0 N
      using x0-is i0-to-i0-FL i0-in2 by blast
  qed
 qed
 then have Y ∈ no-labels.Red-I-𝒢-q ‘ Q ⟹ to-F ι ∈ Y (fst ‘ N) for Y
  using i-in2 no-labels.Red-I-def std-Red-I-eq red-inf-impl by force
 then show to-F ι ∈ no-labels.Red-I-𝒢 (fst ‘ N)
  unfolding Red-I-def no-labels.Red-I-𝒢-def by blast
next
 assume to-F-in: to-F ι ∈ no-labels.Red-I-𝒢 (fst ‘ N)
 have imp-to-F: X ∈ no-labels.Red-I-𝒢-q ‘ Q ⟹ to-F ι ∈ X (fst ‘ N) for X
  using to-F-in unfolding no-labels.Red-I-𝒢-def by blast
 then have to-F-in2: to-F ι ∈ no-labels.Red-I-𝒢-q q (fst ‘ N) if q ∈ Q for q
  using that by auto
 have Red-I-𝒢-q q N = {ι0-FL ∈ Inf-FL. to-F ι0-FL ∈ no-labels.Red-I-𝒢-q q (fst ‘ N)} for q
 proof
  show Red-I-𝒢-q q N ⊆ {ι0-FL ∈ Inf-FL. to-F ι0-FL ∈ no-labels.Red-I-𝒢-q q (fst ‘ N)}
  proof
   fix q0 ι1
   assume
    i1-in: ι1 ∈ Red-I-𝒢-q q0 N
   have i1-in2: ι1 ∈ Inf-FL
    using i1-in unfolding Red-I-𝒢-q-def by blast
   then have to-F-i1-in: to-F ι1 ∈ Inf-F
    using Inf-FL-to-Inf-F unfolding to-F-def by blast
   have concl-swap: fst (concl-of ι1) = concl-of (to-F ι1)
    unfolding concl-of-def to-F-def by simp
   then have i1-to-F-in: to-F ι1 ∈ no-labels.Red-I-𝒢-q q0 (fst ‘ N)
    using i1-in to-F-i1-in unfolding Red-I-𝒢-q-def no-labels.Red-I-𝒢-q-def by force
   show ι1 ∈ {ι0-FL ∈ Inf-FL. to-F ι0-FL ∈ no-labels.Red-I-𝒢-q q0 (fst ‘ N)}
    using i1-in2 i1-to-F-in by blast
  qed
 next
  show {ι0-FL ∈ Inf-FL. to-F ι0-FL ∈ no-labels.Red-I-𝒢-q q (fst ‘ N)} ⊆ Red-I-𝒢-q q N
  proof
   fix q0 ι1
   assume
    i1-in: ι1 ∈ {ι0-FL ∈ Inf-FL. to-F ι0-FL ∈ no-labels.Red-I-𝒢-q q0 (fst ‘ N)}
   then have i1-in2: ι1 ∈ Inf-FL by blast
   then have to-F-i1-in: to-F ι1 ∈ Inf-F
    using Inf-FL-to-Inf-F unfolding to-F-def by blast
   have concl-swap: fst (concl-of ι1) = concl-of (to-F ι1)
    unfolding concl-of-def to-F-def by simp
   then have ((𝒢-I-L-q q0 ι1) ≠ None ∧ the (𝒢-I-L-q q0 ι1) ⊆ Red-I-q q0 (𝒢-Fset-q q0 N))
    ∨ (𝒢-I-L-q q0 ι1 = None ∧
      𝒢-F-L-q q0 (concl-of ι1) ⊆ 𝒢-Fset-q q0 N ∪ Red-F-q q0 (𝒢-Fset-q q0 N))
```

```
      using i1-in unfolding no-labels.Red-I-G-q-def by auto
    then show ι1 ∈ Red-I-G-q q0 N
      using i1-in2 unfolding Red-I-G-q-def by blast
  qed
 qed
 then have ι ∈ Red-I-G-q q N if q ∈ Q for q
  using that to-F-in2 i-in unfolding Red-I-G-q-def no-labels.Red-I-G-q-def by auto
 then show ι ∈ Red-I-G N
  unfolding Red-I-G-def by blast
qed


lemma red-labeled-clauses:
 assumes ‹C ∈ no-labels.Red-F-G-empty (fst ‘ N) ∨
  (∃ C' ∈ fst ‘ N. C' ≺· C) ∨ (∃(C', L') ∈ N. L' ⊏L L ∧ C' ⪯· C)›
 shows ‹(C, L) ∈ Red-F N›
proof −
 note assms
 moreover have i: ‹C ∈ no-labels.Red-F-G-empty (fst ‘ N) ⟹ (C, L) ∈ Red-F N›
 proof −
  assume C ∈ no-labels.Red-F-G-empty (fst ‘ N)
  then have C ∈ no-labels.Red-F-G-empty-q q (fst ‘ N) if q ∈ Q for q
   unfolding no-labels.Red-F-G-empty-def using that by fast
  then have g-in-red: G-F-q q C ⊆ Red-F-q q (no-labels.G-Fset-q q (fst ‘ N)) if q ∈ Q for q
   unfolding no-labels.Red-F-G-empty-q-def using that by blast
  have G-F-L-q q (C, L) ⊆ Red-F-q q (G-Fset-q q N) if q ∈ Q for q
   using that g-in-red by simp
  then show ?thesis
   unfolding Red-F-def Red-F-G-q-def by blast
 qed
 moreover have ii: ‹∃ C' ∈ fst ‘ N. C' ≺· C ⟹ (C, L) ∈ Red-F N›
 proof −
  assume ∃ C' ∈ fst ‘ N. C' ≺· C
  then obtain C' where c'-in: C' ∈ fst ‘ N and c-prec-c': C' ≺· C by blast
  obtain L' where c'-l'-in: (C', L') ∈ N using c'-in by auto
  have c'-l'-prec: (C', L') ⊏ (C, L)
   using c-prec-c' unfolding Prec-FL-def by simp
  have c-in-c'-g: G-F-q q C ⊆ G-F-q q C' if q ∈ Q for q
   using prec-F-grounding[OF that c-prec-c'] by presburger
  then have G-F-L-q q (C, L) ⊆ G-F-L-q q (C', L') if q ∈ Q for q
   using that by auto
  then have (C, L) ∈ Red-F-G-q q N if q ∈ Q for q
   unfolding Red-F-G-q-def using that c'-l'-in c'-l'-prec by blast
  then show ?thesis
   unfolding Red-F-def by blast
 qed
 moreover have iii: ‹∃(C', L') ∈ N. L' ⊏L L ∧ C' ⪯· C ⟹ (C, L) ∈ Red-F N›
 proof −
  assume ∃(C', L') ∈ N. L' ⊏L L ∧ C' ⪯· C
  then obtain C' L' where c'-l'-in: (C', L') ∈ N and l'-sub-l: L' ⊏L L and c'-sub-c: C' ⪯· C
   by fast
  have (C, L) ∈ Red-F N if C' ≺· C
   using that c'-l'-in ii by fastforce
  moreover {
   assume equiv-c-c': C ≐ C'
```

then have *equiv-c′-c*: $C' \doteq C$
using *equiv-equiv-F* **by** (*simp add*: *equivp-symp*)
then have *c′-l′-prec*: $(C', L') \sqsubset (C, L)$
using *l′-sub-l* **unfolding** *Prec-FL-def* **by** *simp*
have $\mathcal{G}$*-F-q q C* $=$ $\mathcal{G}$*-F-q q C′* **if** $q \in Q$ **for** q
using *that equiv-F-grounding equiv-c-c′ equiv-c′-c* **by** (*simp add*: *set-eq-subset*)
then have $\mathcal{G}$*-F-L-q q* $(C, L) = \mathcal{G}$*-F-L-q q* (C', L') **if** $q \in Q$ **for** q
using *that* **by** *auto*
then have $(C, L) \in$ *Red-F-*$\mathcal{G}$*-q q N* **if** $q \in Q$ **for** q
unfolding *Red-F-*$\mathcal{G}$*-q-def* **using** *that c′-l′-in c′-l′-prec* **by** *blast*
then have *?thesis*
unfolding *Red-F-def* **by** *blast*
}
ultimately show *?thesis*
using *c′-sub-c equiv-equiv-F equivp-symp* **by** *fastforce*
qed
ultimately show *?thesis*
by *blast*
qed

end

6.2 Given Clause Procedure

locale *given-clause* = *given-clause-basis Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q*
Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F Prec-L active*
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
entails-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* $\Rightarrow$ *bool* **and**
Inf-G-q :: $\langle$*'q* $\Rightarrow$ *'g inference set*$\rangle$ **and**
Red-I-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g inference set* **and**
Red-F-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-F-q* :: *'q* $\Rightarrow$ *'f* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-I-q* :: *'q* $\Rightarrow$ *'f inference* $\Rightarrow$ *'g inference set option* **and**
Equiv-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\langle\doteq\rangle$ *50*) **and**
Prec-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\langle\prec\cdot\rangle$ *50*) **and**
Prec-L :: *'l* $\Rightarrow$ *'l* $\Rightarrow$ *bool* (**infix** $\langle\sqsubset L\rangle$ *50*) **and**
active :: *'l* +
assumes
inf-have-prems: $\iota F \in$ *Inf-F* $\Longrightarrow$ *prems-of* $\iota F \neq []$
begin

lemma *labeled-inf-have-prems*: $\iota \in$ *Inf-FL* $\Longrightarrow$ *prems-of* $\iota \neq []$
using *inf-have-prems* **by** *fastforce*

inductive *step* :: (*'f* $\times$ *'l*) *set* $\Rightarrow$ (*'f* $\times$ *'l*) *set* $\Rightarrow$ *bool* (**infix** $\langle\rightsquigarrow GC\rangle$ *50*) **where**
process: *N1* $= N \cup M \Longrightarrow$ *N2* $= N \cup M' \Longrightarrow M \subseteq$ *Red-F* $(N \cup M') \Longrightarrow$
active-subset $M' = \{\} \Longrightarrow$ *N1* $\rightsquigarrow$*GC N2*
| *infer*: *N1* $= N \cup \{(C, L)\} \Longrightarrow$ *N2* $= N \cup \{(C,$ *active*$)\} \cup M \Longrightarrow L \neq$ *active* $\Longrightarrow$
active-subset $M = \{\} \Longrightarrow$
no-labels.Inf-between (*fst* ‘ *active-subset N*) $\{C\}$
$\subseteq$ *no-labels.Red-I* (*fst* ‘ $(N \cup \{(C,$ *active*$)\} \cup M)) \Longrightarrow$
N1 $\rightsquigarrow$*GC N2*

lemma *one-step-equiv*: $N1 \rightsquigarrow GC\ N2 \Longrightarrow N1 \rhd L\ N2$
proof (*cases N1 N2 rule*: *step.cases*)
show $N1 \rightsquigarrow GC\ N2 \Longrightarrow N1 \rightsquigarrow GC\ N2$ **by** *blast*
next
fix $N\ M\ M'$
assume
gc-step: $N1 \rightsquigarrow GC\ N2$ **and**
n1-is: $N1 = N \cup M$ **and**
n2-is: $N2 = N \cup M'$ **and**
m-red: $M \subseteq Red\text{-}F\ (N \cup M')$ **and**
active-empty: $active\text{-}subset\ M' = \{\}$
have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *n1-is n2-is m-red* **by** *auto*
then show $N1 \rhd L\ N2$
unfolding *derive.simps* **by** *blast*
next
fix $N\ C\ L\ M$
assume
gc-step: $N1 \rightsquigarrow GC\ N2$ **and**
n1-is: $N1 = N \cup \{(C,\ L)\}$ **and**
not-active: $L \neq active$ **and**
n2-is: $N2 = N \cup \{(C,\ active)\} \cup M$ **and**
active-empty: $active\text{-}subset\ M = \{\}$
have $(C,\ active) \in N2$ **using** *n2-is* **by** *auto*
moreover have $C \preceq\cdot C$ **using** *equiv-equiv-F* **by** (*metis equivp-def*)
moreover have $active \sqsubset L\ L$ **using** *active-minimal*[*OF not-active*] **.**
ultimately have $\{(C,\ L)\} \subseteq Red\text{-}F\ N2$
using *red-labeled-clauses* **by** *blast*
moreover have $N1 - N2 = \{\} \vee N1 - N2 = \{(C,\ L)\}$ **using** *n1-is n2-is* **by** *blast*
ultimately have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *std-Red-F-eq* **by** *blast*
then show $N1 \rhd L\ N2$
unfolding *derive.simps* **by** *blast*
qed

lemma *gc-to-red*: $chain\ (\rightsquigarrow GC)\ Ns \Longrightarrow chain\ (\rhd L)\ Ns$
using *one-step-equiv Lazy-List-Chain.chain-mono* **by** *blast*

lemma (**in**−) *all-ex-finite-set*: $(\forall\,(j{::}nat){\in}\{0..{<}m\}.\ \exists\,(n{::}nat).\ P\ j\ n) \Longrightarrow$
$(\forall\, n1\ n2.\ \forall\, j{\in}\{0..{<}m\}.\ P\ j\ n1 \longrightarrow P\ j\ n2 \longrightarrow n1 = n2) \Longrightarrow finite\ \{n.\ \exists j \in \{0..{<}m\}.\ P\ j\ n\}$ **for** m P
proof −
fix $m{::}nat$ **and** $P{::}\ nat \Rightarrow nat \Rightarrow bool$
assume
allj-exn: $\forall\, j{\in}\{0..{<}m\}.\ \exists\, n.\ P\ j\ n$ **and**
uniq-n: $\forall\, n1\ n2.\ \forall\, j{\in}\{0..{<}m\}.\ P\ j\ n1 \longrightarrow P\ j\ n2 \longrightarrow n1 = n2$
have $\{n.\ \exists j \in \{0..{<}m\}.\ P\ j\ n\} = (\bigcup((\lambda j.\ \{n.\ P\ j\ n\})\ `\ \{0..{<}m\}))$ **by** *blast*
then have *imp-finite*: $(\forall\, j{\in}\{0..{<}m\}.\ finite\ \{n.\ P\ j\ n\}) \Longrightarrow finite\ \{n.\ \exists j \in \{0..{<}m\}.\ P\ j\ n\}$
using *finite-UN*[*of* $\{0..{<}m\}$ $\lambda j.\ \{n.\ P\ j\ n\}$] **by** *simp*
have $\forall\, j{\in}\{0..{<}m\}.\ \exists!n.\ P\ j\ n$ **using** *allj-exn uniq-n* **by** *blast*
then have $\forall\, j{\in}\{0..{<}m\}.\ finite\ \{n.\ P\ j\ n\}$ **by** (*metis bounded-nat-set-is-finite lessI mem-Collect-eq*)
then show $finite\ \{n.\ \exists j \in \{0..{<}m\}.\ P\ j\ n\}$ **using** *imp-finite* **by** *simp*
qed

lemma *gc-fair*:
assumes
deriv: *chain* (⇝*GC*) *Ns* **and**
init-state: *active-subset* (*lhd Ns*) = {} **and**
final-state: *passive-subset* (*Liminf-llist Ns*) = {}
shows *fair Ns*
unfolding *fair-def*
proof
fix ι
assume *i-in*: ι ∈ *Inf-from* (*Liminf-llist Ns*)
note *lhd-is* = *lhd-conv-lnth*[*OF chain-not-lnull*[*OF deriv*]]
have *i-in-inf-fl*: ι ∈ *Inf-FL* **using** *i-in* **unfolding** *Inf-from-def* **by** *blast*
have *Liminf-llist Ns* = *active-subset* (*Liminf-llist Ns*)
using *final-state* **unfolding** *passive-subset-def active-subset-def* **by** *blast*
then have *i-in2*: ι ∈ *Inf-from* (*active-subset* (*Liminf-llist Ns*)) **using** *i-in* **by** *simp*
define *m* **where** *m* = *length* (*prems-of* ι)
then have *m-def-F*: *m* = *length* (*prems-of* (*to-F* ι)) **unfolding** *to-F-def* **by** *simp*
have *i-in-F*: *to-F* ι ∈ *Inf-F*
using *i-in Inf-FL-to-Inf-F* **unfolding** *Inf-from-def to-F-def* **by** *blast*
then have *m-pos*: *m* > *0* **using** *m-def-F* **using** *inf-have-prems* **by** *blast*
have *exist-nj*: ∀ *j* ∈ {*0*..<*m*}. (∃ *nj*. *enat* (*Suc nj*) < *llength Ns* ∧
prems-of ι ! *j* ∉ *active-subset* (*lnth Ns nj*) ∧
(∀ *k*. *k* > *nj* ⟶ *enat k* < *llength Ns* ⟶ *prems-of* ι ! *j* ∈ *active-subset* (*lnth Ns k*)))
proof *clarify*
fix *j*
assume *j-in*: *j* ∈ {*0*..<*m*}
then obtain *C* **where** *c-is*: (*C*, *active*) = *prems-of* ι ! *j*
using *i-in2* **unfolding** *m-def Inf-from-def active-subset-def*
by (*smt* (*verit*, *ccfv-threshold*) *atLeastLessThan-iff mem-Collect-eq nth-mem snd-conv subset-iff surj-pair*)
then have (*C*, *active*) ∈ *Liminf-llist Ns*
using *j-in i-in* **unfolding** *m-def Inf-from-def* **by** *force*
then obtain *nj* **where** *nj-is*: *enat nj* < *llength Ns* **and**
c-in2: (*C*, *active*) ∈ ⋂ (*lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*})
unfolding *Liminf-llist-def* **using** *init-state* **by** *blast*
then have *c-in3*: ∀ *k*. *k* ≥ *nj* ⟶ *enat k* < *llength Ns* ⟶ (*C*, *active*) ∈ *lnth Ns k* **by** *blast*
have *nj-pos*: *nj* > *0* **using** *init-state c-in2 nj-is*
unfolding *active-subset-def lhd-is* **by** *force*
obtain *nj-min* **where** *nj-min-is*: *nj-min* = (*LEAST nj*. *enat nj* < *llength Ns* ∧
(*C*, *active*) ∈ ⋂ (*lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*})) **by** *blast*
then have *in-allk*: ∀ *k*. *k* ≥ *nj-min* ⟶ *enat k* < *llength Ns* ⟶ (*C*, *active*) ∈ (*lnth Ns k*)
using *c-in3 nj-is c-in2*
by (*metis* (*mono-tags*, *lifting*) *INT-E LeastI-ex mem-Collect-eq*)
have *njm-smaller-D*: *enat nj-min* < *llength Ns*
using *nj-min-is*
by (*metis* (*lifting*) *ext LeastI*
‹⋀*thesis*. (⋀*nj*. *enat nj* < *llength Ns* ⟹ (*C*, *active*) ∈ ⋂ (*lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*}) ⟹ *thesis*) ⟹ *thesis*›)
have *nj-min* > *0*
using *nj-is c-in2 nj-pos nj-min-is lhd-is*
by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq* ‹(*C*, *active*) ∈ *Liminf-llist Ns*›
‹*Liminf-llist Ns* = *active-subset* (*Liminf-llist Ns*)›
‹∀ *k*≥*nj-min*. *enat k* < *llength Ns* ⟶ (*C*, *active*) ∈ *lnth Ns k*› *active-subset-def init-state*

```
      linorder-not-less mem-Collect-eq zero-enat-def chain-length-pos[OF deriv])
  then obtain njm-prec where nj-prec-is: Suc njm-prec = nj-min using gr0-conv-Suc by auto
  then have njm-prec-njm: njm-prec < nj-min by blast
  then have njm-prec-njm-enat: enat njm-prec < enat nj-min by simp
  have njm-prec-smaller-d: njm-prec < llength Ns
    by (rule less-trans[OF njm-prec-njm-enat njm-smaller-D])
  have njm-prec-all-suc: ∀ k>njm-prec. enat k < llength Ns ⟶ (C, active) ∈ lnth Ns k
    using nj-prec-is in-allk by simp
  have notin-njm-prec: (C, active) ∉ lnth Ns njm-prec
  proof (rule ccontr)
    assume ¬ (C, active) ∉ lnth Ns njm-prec
    then have absurd-hyp: (C, active) ∈ lnth Ns njm-prec by simp
    have prec-smaller: enat njm-prec < llength Ns using nj-min-is nj-prec-is
      using njm-prec-smaller-d by fastforce
    have (C, active) ∈ ⋂ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns})
    proof −
      {
        fix k
        assume k-in: njm-prec ≤ k ∧ enat k < llength Ns
        have k = njm-prec ⟹ (C, active) ∈ lnth Ns k using absurd-hyp by simp
        moreover have njm-prec < k ⟹ (C, active) ∈ lnth Ns k
          using nj-prec-is in-allk k-in by simp
        ultimately have (C, active) ∈ lnth Ns k using k-in by fastforce
      }
      then show (C, active) ∈ ⋂ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns}) by blast
    qed
    then have enat njm-prec < llength Ns ∧
        (C, active) ∈ ⋂ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns})
      using prec-smaller by blast
    then show False
      using nj-min-is nj-prec-is Orderings.wellorder-class.not-less-Least njm-prec-njm by blast
  qed
  then have notin-active-subs-njm-prec: (C, active) ∉ active-subset (lnth Ns njm-prec)
    unfolding active-subset-def by blast
  then show ∃ nj. enat (Suc nj) < llength Ns ∧ prems-of ι ! j ∉ active-subset (lnth Ns nj) ∧
      (∀ k. k > nj ⟶ enat k < llength Ns ⟶ prems-of ι ! j ∈ active-subset (lnth Ns k))
    using c-is njm-prec-all-suc njm-prec-smaller-d by (metis (mono-tags, lifting)
      active-subset-def mem-Collect-eq nj-prec-is njm-smaller-D snd-conv)
qed
define nj-set where nj-set = {nj. (∃ j∈{0..<m}. enat (Suc nj) < llength Ns ∧
    prems-of ι ! j ∉ active-subset (lnth Ns nj) ∧
    (∀ k. k > nj ⟶ enat k < llength Ns ⟶ prems-of ι ! j ∈ active-subset (lnth Ns k)))}
then have nj-not-empty: nj-set ≠ {}
proof −
  have zero-in: 0 ∈ {0..<m} using m-pos by simp
  then obtain n0 where enat (Suc n0) < llength Ns and
    prems-of ι ! 0 ∉ active-subset (lnth Ns n0) and
    ∀ k>n0. enat k < llength Ns ⟶ prems-of ι ! 0 ∈ active-subset (lnth Ns k)
    using exist-nj by fast
  then have n0 ∈ nj-set unfolding nj-set-def using zero-in by blast
  then show nj-set ≠ {} by auto
qed
have nj-finite: finite nj-set
  using all-ex-finite-set[OF exist-nj]
  by (metis (no-types, lifting) Suc-ile-eq dual-order.strict-implies-order
```

linorder-neqE-nat nj-set-def)

have $\exists\, n \in$ *nj-set*. $\forall\, nj \in$ *nj-set*. $nj \leq n$
using *nj-not-empty nj-finite* **using** *Max-ge Max-in* **by** *blast*
then obtain *n* **where** *n-in*: $n \in$ *nj-set* **and** *n-bigger*: $\forall\, nj \in$ *nj-set*. $nj \leq n$ **by** *blast*
then obtain *j0* **where** *j0-in*: $j0 \in \{0..<m\}$ **and** *suc-n-length*: *enat* (*Suc n*) < *llength Ns* **and**
j0-notin: *prems-of* ι ! *j0* $\notin$ *active-subset* (*lnth Ns n*) **and**
j0-allin: $(\forall\, k.\ k > n \longrightarrow$ *enat k* < *llength Ns* $\longrightarrow$ *prems-of* ι ! *j0* $\in$ *active-subset* (*lnth Ns k*))
unfolding *nj-set-def* **by** *blast*
obtain *C0* **where** *C0-is*: *prems-of* ι ! *j0* = (*C0*, *active*) **using** *j0-in*
using *i-in2* **unfolding** *m-def Inf-from-def active-subset-def*
by (*metis* (*mono-tags*, *lifting*)
‹$\bigwedge$*thesis*. ($\bigwedge$*j0*. *j0* $\in \{0..<m\} \Longrightarrow$ *enat* (*Suc n*) < *llength Ns* $\Longrightarrow$ *prems-of* ι ! *j0* $\notin$ *active-subset* (*lnth Ns n*) $\Longrightarrow \forall\, k{>}n.$ *enat k* < *llength Ns* $\longrightarrow$ *prems-of* ι ! *j0* $\in$ *active-subset* (*lnth Ns k*) $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›
active-subset-def j0-allin lessI mem-Collect-eq split-pairs)
then have *C0-prems-i*: (*C0*, *active*) $\in$ *set* (*prems-of* ι) **using** *in-set-conv-nth j0-in m-def* **by** *force*
have *C0-in*: (*C0*, *active*) $\in$ (*lnth Ns* (*Suc n*))
using *C0-is j0-allin suc-n-length* **by** (*simp add*: *active-subset-def*)
have *C0-notin*: (*C0*, *active*) $\notin$ (*lnth Ns n*) **using** *C0-is j0-notin* **unfolding** *active-subset-def* **by** *simp*
have *step-n*: *lnth Ns n* $\rightsquigarrow_{GC}$ *lnth Ns* (*Suc n*)
using *deriv chain-lnth-rel n-in* **unfolding** *nj-set-def* **by** *blast*
have $\exists\, N\ C\ L\ M.$ (*lnth Ns n* = $N \cup \{(C,\ L)\} \wedge$
lnth Ns (*Suc n*) = $N \cup \{(C,$ *active*$)\} \cup M \wedge L \neq$ *active* $\wedge$ *active-subset M* = $\{\} \wedge$
no-labels.Inf-between (*fst* ‘ *active-subset N*) $\{C\}$
$\subseteq$ *no-labels.Red-I* (*fst* ‘ ($N \cup \{(C,$ *active*$)\} \cup M$)))
proof −
have *proc-or-infer*: ($\exists$ *N1 N M N2 M'*. *lnth Ns n* = *N1* $\wedge$ *lnth Ns* (*Suc n*) = *N2* $\wedge$ *N1* = $N \cup M \wedge$
N2 = $N \cup M' \wedge M \subseteq$ *Red-F* ($N \cup M'$) $\wedge$ *active-subset M'* = $\{\}$) $\vee$
($\exists$ *N1 N C L N2 M*. *lnth Ns n* = *N1* $\wedge$ *lnth Ns* (*Suc n*) = *N2* $\wedge$ *N1* = $N \cup \{(C,\ L)\} \wedge$
N2 = $N \cup \{(C,$ *active*$)\} \cup M \wedge L \neq$ *active* $\wedge$ *active-subset M* = $\{\} \wedge$
no-labels.Inf-between (*fst* ‘ *active-subset N*) $\{C\} \subseteq$
no-labels.Red-I (*fst* ‘ ($N \cup \{(C,$ *active*$)\} \cup M$)))
using *step.simps*[*of lnth Ns n lnth Ns* (*Suc n*)] *step-n* **by** *blast*
show *?thesis*
using *C0-in C0-notin proc-or-infer j0-in C0-is*
by (*smt* (*verit*, *ccfv-threshold*) *UnI1*
‹$\bigwedge$*thesis*. ($\bigwedge$*j0*. *j0* $\in \{0..<m\} \Longrightarrow$ *enat* (*Suc n*) < *llength Ns* $\Longrightarrow$ *prems-of* ι ! *j0* $\notin$ *active-subset* (*lnth Ns n*) $\Longrightarrow \forall\, k{>}n.$ *enat k* < *llength Ns* $\longrightarrow$ *prems-of* ι ! *j0* $\in$ *active-subset* (*lnth Ns k*) $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›
active-subset-union lessI sup-bot.right-neutral)
qed
then obtain *N M L* **where** *inf-from-subs*:
no-labels.Inf-between (*fst* ‘ *active-subset N*) $\{C0\}$
$\subseteq$ *no-labels.Red-I* (*fst* ‘ ($N \cup \{(C0,$ *active*$)\} \cup M$)) **and**
nth-d-is: *lnth Ns n* = $N \cup \{(C0,\ L)\}$ **and**
suc-nth-d-is: *lnth Ns* (*Suc n*) = $N \cup \{(C0,$ *active*$)\} \cup M$ **and**
l-not-active: $L \neq$ *active*
using *C0-in C0-notin j0-in C0-is* **using** *active-subset-def* **by** *fastforce*
have $j \in \{0..<m\} \Longrightarrow$ *prems-of* ι ! $j \neq$ *prems-of* ι ! *j0* $\Longrightarrow$ *prems-of* ι ! $j \in$ (*active-subset N*) **for** *j*
proof −
fix *j*
assume *j-in*: $j \in \{0..<m\}$ **and**
j-not-j0: *prems-of* ι ! $j \neq$ *prems-of* ι ! *j0*
obtain *nj* **where** *nj-len*: *enat* (*Suc nj*) < *llength Ns* **and**

nj-prems: *prems-of* ι ! $j \notin$ *active-subset* (*lnth Ns nj*) **and**
nj-greater: ($\forall$ *k*. $k > nj \longrightarrow$ *enat k* < *llength Ns* $\longrightarrow$ *prems-of* ι ! $j \in$ *active-subset* (*lnth Ns k*))
using *exist-nj j-in* **by** *blast*
then have $nj \in$ *nj-set* **unfolding** *nj-set-def* **using** *j-in* **by** *blast*
moreover have $nj \neq n$
proof
assume $nj = n$
then have *prems-of* ι ! j = (*C0*, *active*)
using *C0-in C0-notin step.simps*[*of lnth Ns n lnth Ns* (*Suc n*)] *step-n*
nj-greater nj-prems snd-conv
by (*simp add*: *suc-n-length*)
(*smt* (*verit*, *del-insts*) *C0-notin Un-iff active-subset-def insertCI insertE lessI mem-Collect-eq snd-conv suc-n-length*
sup-bot.right-neutral)
then show *False* **using** *j-not-j0 C0-is* **by** *simp*
qed
ultimately have $nj < n$ **using** *n-bigger* **by** *force*
then have *prems-of* ι ! $j \in$ (*active-subset* (*lnth Ns n*))
using *nj-greater n-in Suc-ile-eq dual-order.strict-implies-order* **unfolding** *nj-set-def* **by** *blast*
then show *prems-of* ι ! $j \in$ (*active-subset N*)
using *nth-d-is l-not-active* **unfolding** *active-subset-def* **by** *force*
qed
then have *set* (*prems-of* ι) $\subseteq$ *active-subset N* $\cup$ {(*C0*, *active*)}
using *C0-prems-i C0-is m-def*
by (*metis Un-iff atLeast0LessThan in-set-conv-nth insertCI lessThan-iff subrelI*)
moreover have $\neg$ (*set* (*prems-of* ι) $\subseteq$ *active-subset N* − {(*C0*, *active*)}) **using** *C0-prems-i* **by** *blast*
ultimately have $\iota \in$ *Inf-between* (*active-subset N*) {(*C0*, *active*)}
using *i-in-inf-fl* **unfolding** *Inf-between-def Inf-from-def* **by** *blast*
then have *to-F* $\iota \in$ *no-labels.Inf-between* (*fst* ‘ *active-subset N*) {*C0*}
unfolding *to-F-def Inf-between-def Inf-from-def*
no-labels.Inf-between-def no-labels.Inf-from-def **using** *Inf-FL-to-Inf-F*
by *force*
then have *to-F* $\iota \in$ *no-labels.Red-I* (*fst* ‘ (*lnth Ns* (*Suc n*)))
using *suc-nth-d-is inf-from-subs* **by** *fastforce*
then have $\forall$ *q* $\in$ *Q*. (*G-I-q q* (*to-F* ι) $\neq$ *None* $\wedge$
the (*G-I-q q* (*to-F* ι)) $\subseteq$ *Red-I-q q* ($\bigcup$ (*G-F-q q* ‘ *fst* ‘ *lnth Ns* (*Suc n*))))
$\vee$ (*G-I-q q* (*to-F* ι) = *None* $\wedge$
G-F-q q (*concl-of* (*to-F* ι)) $\subseteq \bigcup$ (*G-F-q q* ‘ *fst* ‘ *lnth Ns* (*Suc n*)) $\cup$
Red-F-q q ($\bigcup$ (*G-F-q q* ‘ *fst* ‘ *lnth Ns* (*Suc n*))))
unfolding *to-F-def no-labels.Red-I-def no-labels.Red-I-G-q-def* **by** *blast*
then have $\iota \in$ *Red-I-G* (*lnth Ns* (*Suc n*))
using *i-in-inf-fl* **unfolding** *Red-I-G-def Red-I-G-q-def* **by** (*simp add*: *to-F-def*)
then show $\iota \in$ *Sup-llist* (*lmap Red-I-G Ns*)
unfolding *Sup-llist-def* **using** *suc-n-length* **by** *auto*
qed

theorem *gc-complete-Liminf*:
assumes
deriv: *chain* ($\rightsquigarrow$*GC*) *Ns* **and**
init-state: *active-subset* (*lhd Ns*) = {} **and**
final-state: *passive-subset* (*Liminf-llist Ns*) = {} **and**
b-in: *B* $\in$ *Bot-F* **and**
bot-entailed: *no-labels.entails-G* (*fst* ‘ *lhd Ns*) {*B*}
shows $\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *Liminf-llist Ns*
proof −

note *lhd-is* = *lhd-conv-lnth*[*OF chain-not-lnull*[*OF deriv*]]
have *labeled-b-in*: (*B*, *active*) ∈ *Bot-FL* **using** *b-in* **by** *simp*
have *labeled-bot-entailed*: *entails-G-L* (*lhd Ns*) {(*B*, *active*)}
using *labeled-entailment-lifting bot-entailed lhd-is* **by** *fastforce*
have *fair*: *fair Ns* **using** *gc-fair*[*OF deriv init-state final-state*] **.**
then show *?thesis*
using *dynamically-complete-Liminf*[*OF labeled-b-in gc-to-red*[*OF deriv*] *fair labeled-bot-entailed*]
by *blast*
qed

theorem *gc-complete*:
assumes
deriv: *chain* (⇝*GC*) *Ns* **and**
init-state: *active-subset* (*lhd Ns*) = {} **and**
final-state: *passive-subset* (*Liminf-llist Ns*) = {} **and**
b-in: *B* ∈ *Bot-F* **and**
bot-entailed: *no-labels.entails-G* (*fst* ‘ *lhd Ns*) {*B*}
shows ∃ *i*. *enat i* < *llength Ns* ∧ (∃ *BL* ∈ *Bot-FL*. *BL* ∈ *lnth Ns i*)
proof −
note *lhd-is* = *lhd-conv-lnth*[*OF chain-not-lnull*[*OF deriv*]]
have ∃ *BL*∈*Bot-FL*. *BL* ∈ *Liminf-llist Ns*
using *assms* **by** (*rule gc-complete-Liminf*)
then show *?thesis*
unfolding *Liminf-llist-def* **by** *auto*
qed

end

6.3 Lazy Given Clause Procedure

locale *lazy-given-clause* = *given-clause-basis Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q G-F-q G-I-q Equiv-F Prec-F Prec-L active*
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
entails-q :: *'q* ⇒ *'g set* ⇒ *'g set* ⇒ *bool* **and**
Inf-G-q :: ‹*'q* ⇒ *'g inference set*› **and**
Red-I-q :: *'q* ⇒ *'g set* ⇒ *'g inference set* **and**
Red-F-q :: *'q* ⇒ *'g set* ⇒ *'g set* **and**
G-F-q :: *'q* ⇒ *'f* ⇒ *'g set* **and**
G-I-q :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**
Equiv-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≐› *50*) **and**
Prec-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺·› *50*) **and**
Prec-L :: *'l* ⇒ *'l* ⇒ *bool* (**infix** ‹⊏*L*› *50*) **and**
active :: *'l*
begin

inductive *step* :: *'f inference set* × (*'f* × *'l*) *set* ⇒
'f inference set × (*'f* × *'l*) *set* ⇒ *bool* (**infix** ‹⇝*LGC*› *50*) **where**
process: *N1* = *N* ∪ *M* ⟹ *N2* = *N* ∪ *M'* ⟹ *M* ⊆ *Red-F* (*N* ∪ *M'*) ⟹
active-subset M' = {} ⟹ (*T*, *N1*) ⇝*LGC* (*T*, *N2*) |
schedule-infer: *T2* = *T1* ∪ *T'* ⟹ *N1* = *N* ∪ {(*C*, *L*)} ⟹ *N2* = *N* ∪ {(*C*, *active*)} ⟹

$L \neq active \Longrightarrow T' = no\text{-}labels.Inf\text{-}between\ (fst\ `\ active\text{-}subset\ N)\ \{C\} \Longrightarrow$
$(T1,\ N1) \rightsquigarrow LGC\ (T2,\ N2)\ |$
compute-infer: $T1 = T2 \cup \{\iota\} \Longrightarrow N2 = N1 \cup M \Longrightarrow active\text{-}subset\ M = \{\} \Longrightarrow$
$\iota \in no\text{-}labels.Red\text{-}I\ (fst\ `\ (N1 \cup M)) \Longrightarrow (T1,\ N1) \rightsquigarrow LGC\ (T2,\ N2)\ |$
delete-orphan-infers: $T1 = T2 \cup T' \Longrightarrow$
$T' \cap no\text{-}labels.Inf\text{-}from\ (fst\ `\ active\text{-}subset\ N) = \{\} \Longrightarrow (T1,\ N) \rightsquigarrow LGC\ (T2,\ N)$

lemma *premise-free-inf-always-from*: $\iota \in Inf\text{-}F \Longrightarrow prems\text{-}of\ \iota = [] \Longrightarrow \iota \in no\text{-}labels.Inf\text{-}from\ N$
unfolding *no-labels.Inf-from-def* **by** *simp*

lemma *one-step-equiv*: $(T1,\ N1) \rightsquigarrow LGC\ (T2,\ N2) \Longrightarrow N1 \triangleright L\ N2$
proof (*cases* $(T1,\ N1)\ (T2,\ N2)$ *rule*: *step.cases*)
show $(T1,\ N1) \rightsquigarrow LGC\ (T2,\ N2) \Longrightarrow (T1,\ N1) \rightsquigarrow LGC\ (T2,\ N2)$ **by** *blast*
next
fix $N\ M\ M'$
assume
n1-is: $N1 = N \cup M$ **and**
n2-is: $N2 = N \cup M'$ **and**
m-red: $M \subseteq Red\text{-}F\ (N \cup M')$
have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *n1-is n2-is m-red* **by** *auto*
then show $N1 \triangleright L\ N2$
unfolding *derive.simps* **by** *blast*
next
fix $N\ C\ L\ M$
assume
n1-is: $N1 = N \cup \{(C,\ L)\}$ **and**
not-active: $L \neq active$ **and**
n2-is: $N2 = N \cup \{(C,\ active)\}$
have $(C,\ active) \in N2$ **using** *n2-is* **by** *auto*
moreover have $C \preceq\cdot\ C$ **by** (*metis equivp-def equiv-equiv-F*)
moreover have $active \sqsubset L\ L$ **using** *active-minimal*[*OF not-active*] **.**
ultimately have $\{(C,\ L)\} \subseteq Red\text{-}F\ N2$
using *red-labeled-clauses* **by** *blast*
then have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *std-Red-F-eq* **using** *n1-is n2-is* **by** *blast*
then show $N1 \triangleright L\ N2$
unfolding *derive.simps* **by** *blast*
next
fix M
assume
n2-is: $N2 = N1 \cup M$
have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *n2-is* **by** *blast*
then show $N1 \triangleright L\ N2$
unfolding *derive.simps* **by** *blast*
next
assume *n2-is*: $N2 = N1$
have $N1 - N2 \subseteq Red\text{-}F\ N2$
using *n2-is* **by** *blast*
then show $N1 \triangleright L\ N2$
unfolding *derive.simps* **by** *blast*
qed

lemma *lgc-to-red*: *chain* (⇝*LGC*) *Ns* ⟹ *chain* (▷*L*) (*lmap snd Ns*)
using *one-step-equiv Lazy-List-Chain.chain-mono*
by (*smt* (*verit*) *chain-lmap prod.exhaust-sel*)

lemma *lgc-fair*:
assumes
deriv: *chain* (⇝*LGC*) *Ns* **and**
init-state: *active-subset* (*snd* (*lhd Ns*)) = {} **and**
final-state: *passive-subset* (*Liminf-llist* (*lmap snd Ns*)) = {} **and**
no-prems-init: ∀ ι ∈ *Inf-F*. *prems-of* ι = [] ⟶ ι ∈ *fst* (*lhd Ns*) **and**
final-schedule: *Liminf-llist* (*lmap fst Ns*) = {}
shows *fair* (*lmap snd Ns*)
unfolding *fair-def*
proof
fix ι
assume *i-in*: ι ∈ *Inf-from* (*Liminf-llist* (*lmap snd Ns*))
note *lhd-is* = *lhd-conv-lnth*[*OF chain-not-lnull*[*OF deriv*]]
have *i-in-inf-fl*: ι ∈ *Inf-FL* **using** *i-in* **unfolding** *Inf-from-def* **by** *blast*
have *Liminf-llist* (*lmap snd Ns*) = *active-subset* (*Liminf-llist* (*lmap snd Ns*))
using *final-state* **unfolding** *passive-subset-def active-subset-def* **by** *blast*
then have *i-in2*: ι ∈ *Inf-from* (*active-subset* (*Liminf-llist* (*lmap snd Ns*)))
using *i-in* **by** *simp*
define *m* **where** *m* = *length* (*prems-of* ι)
then have *m-def-F*: *m* = *length* (*prems-of* (*to-F* ι)) **unfolding** *to-F-def* **by** *simp*
have *i-in-F*: *to-F* ι ∈ *Inf-F*
using *i-in Inf-FL-to-Inf-F* **unfolding** *Inf-from-def to-F-def* **by** *blast*
have *exist-nj*: ∀ *j* ∈ {*0*..<*m*}. (∃ *nj*. *enat* (*Suc nj*) < *llength Ns* ∧
prems-of ι ! *j* ∉ *active-subset* (*snd* (*lnth Ns nj*)) ∧
(∀ *k*. *k* > *nj* ⟶ *enat k* < *llength Ns* ⟶ *prems-of* ι ! *j* ∈ *active-subset* (*snd* (*lnth Ns k*))))
proof *clarify*
fix *j*
assume *j-in*: *j* ∈ {*0*..<*m*}
then obtain *C* **where** *c-is*: (*C*, *active*) = *prems-of* ι ! *j*
using *i-in2* **unfolding** *m-def Inf-from-def active-subset-def*
by (*smt* (*verit*) *atLeastLessThan-iff mem-Collect-eq nth-mem snd-eqD subsetD surj-pair*)
then have (*C*, *active*) ∈ *Liminf-llist* (*lmap snd Ns*)
using *j-in i-in* **unfolding** *m-def Inf-from-def* **by** *force*
then obtain *nj* **where** *nj-is*: *enat nj* < *llength Ns* **and**
c-in2: (*C*, *active*) ∈ ⋂ (*snd* ‘ (*lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*}))
unfolding *Liminf-llist-def* **using** *init-state* **by** *fastforce*
then have *c-in3*: ∀ *k*. *k* ≥ *nj* ⟶ *enat k* < *llength Ns* ⟶ (*C*, *active*) ∈ *snd* (*lnth Ns k*) **by** *blast*
have *nj-pos*: *nj* > *0* **using** *init-state c-in2 nj-is* **unfolding** *active-subset-def lhd-is* **by** *fastforce*
obtain *nj-min* **where** *nj-min-is*: *nj-min* = (*LEAST nj*. *enat nj* < *llength Ns* ∧
(*C*, *active*) ∈ ⋂ (*snd* ‘ (*lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*}))) **by** *blast*
then have *in-allk*: ∀ *k*. *k* ≥ *nj-min* ⟶ *enat k* < *llength Ns* ⟶ (*C*, *active*) ∈ *snd* (*lnth Ns k*)
using *c-in3 nj-is c-in2 INT-E LeastI-ex*
[*of* λ*n*. *enat n* < *llength Ns*
∧ (*C*, *active*) ∈ ⋂ (*snd* ‘ *lnth Ns* ‘ {*na*. *n* ≤ *na* ∧ *enat na* < *llength Ns*})]
by *blast*
have *njm-smaller-D*: *enat nj-min* < *llength Ns*
using *nj-min-is*
by (*metis* (*no-types*, *lifting*) *ext LeastI*
‹⋀*thesis*. (⋀*nj*. *enat nj* < *llength Ns* ⟹ (*C*, *active*) ∈ ⋂ (*snd* ‘ *lnth Ns* ‘ {*k*. *nj* ≤ *k* ∧ *enat k* < *llength Ns*}) ⟹ *thesis*) ⟹ *thesis*›)

```
  have nj-min > 0
    using nj-is c-in2 nj-pos nj-min-is lhd-is
    by (metis (mono-tags, lifting) active-subset-def emptyE in-allk init-state mem-Collect-eq
        not-less snd-conv zero-enat-def chain-length-pos[OF deriv])
  then obtain njm-prec where nj-prec-is: Suc njm-prec = nj-min using gr0-conv-Suc by auto
  then have njm-prec-njm: njm-prec < nj-min by blast
  then have njm-prec-njm-enat: enat njm-prec < enat nj-min by simp
  have njm-prec-smaller-d: njm-prec < llength Ns
    by (rule less-trans[OF njm-prec-njm-enat njm-smaller-D])
  have njm-prec-all-suc: ∀ k>njm-prec. enat k < llength Ns ⟶ (C, active) ∈ snd (lnth Ns k)
    using nj-prec-is in-allk by simp
  have notin-njm-prec: (C, active) ∉ snd (lnth Ns njm-prec)
  proof (rule ccontr)
    assume ¬ (C, active) ∉ snd (lnth Ns njm-prec)
    then have absurd-hyp: (C, active) ∈ snd (lnth Ns njm-prec) by simp
    have prec-smaller: enat njm-prec < llength Ns using nj-min-is nj-prec-is
      using njm-prec-smaller-d by blast
    have (C, active) ∈ ⋂ (snd ‘ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns}))
    proof −
      {
        fix k
        assume k-in: njm-prec ≤ k ∧ enat k < llength Ns
        have k = njm-prec ⟹ (C, active) ∈ snd (lnth Ns k) using absurd-hyp by simp
        moreover have njm-prec < k ⟹ (C, active) ∈ snd (lnth Ns k)
          using nj-prec-is in-allk k-in by simp
        ultimately have (C, active) ∈ snd (lnth Ns k) using k-in by fastforce
      }
      then show (C, active) ∈ ⋂ (snd ‘ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns}))
        by blast
    qed
    then have enat njm-prec < llength Ns ∧
        (C, active) ∈ ⋂ (snd ‘ (lnth Ns ‘ {k. njm-prec ≤ k ∧ enat k < llength Ns}))
      using prec-smaller by blast
    then show False
      using nj-min-is nj-prec-is Orderings.wellorder-class.not-less-Least njm-prec-njm by blast
  qed
  then have notin-active-subs-njm-prec: (C, active) ∉ active-subset (snd (lnth Ns njm-prec))
    unfolding active-subset-def by blast
  then show ∃ nj. enat (Suc nj) < llength Ns ∧ prems-of ι ! j ∉ active-subset (snd (lnth Ns nj)) ∧
      (∀ k. k > nj ⟶ enat k < llength Ns ⟶ prems-of ι ! j ∈ active-subset (snd (lnth Ns k)))
    using c-is njm-prec-all-suc njm-prec-smaller-d by (metis (mono-tags, lifting)
        active-subset-def mem-Collect-eq nj-prec-is njm-smaller-D snd-conv)
qed
define nj-set where nj-set = {nj. (∃ j∈{0..<m}. enat (Suc nj) < llength Ns ∧
    prems-of ι ! j ∉ active-subset (snd (lnth Ns nj)) ∧
    (∀ k. k > nj ⟶ enat k < llength Ns ⟶ prems-of ι ! j ∈ active-subset (snd (lnth Ns k))))}
{
  assume m-null: m = 0
  then have enat 0 < llength Ns ∧ to-F ι ∈ fst (lhd Ns)
    using no-prems-init i-in-F m-def-F zero-enat-def chain-length-pos[OF deriv] by auto
  then have ∃ n. enat n < llength Ns ∧ to-F ι ∈ fst (lnth Ns n)
    unfolding lhd-is by blast
}
moreover {
  assume m-pos: m > 0
```

have *nj-not-empty*: $nj\text{-}set \neq \{\}$
proof −
have *zero-in*: $0 \in \{0..<m\}$ **using** *m-pos* **by** *simp*
then obtain *n0* **where** $enat\ (Suc\ n0) < llength\ Ns$ **and**
$prems\text{-}of\ \iota\ !\ 0 \notin active\text{-}subset\ (snd\ (lnth\ Ns\ n0))$ **and**
$\forall\, k{>}n0.\ enat\ k < llength\ Ns \longrightarrow prems\text{-}of\ \iota\ !\ 0 \in active\text{-}subset\ (snd\ (lnth\ Ns\ k))$
using *exist-nj* **by** *fast*
then have $n0 \in nj\text{-}set$ **unfolding** *nj-set-def* **using** *zero-in* **by** *blast*
then show $nj\text{-}set \neq \{\}$ **by** *auto*
qed
have *nj-finite*: *finite nj-set*
using *all-ex-finite-set*[*OF exist-nj*] **by** (*metis* (*no-types*, *lifting*) *Suc-ile-eq*
dual-order.strict-implies-order linorder-neqE-nat nj-set-def)
have $\exists\, n \in nj\text{-}set.\ \forall\, nj \in nj\text{-}set.\ nj \leq n$
using *nj-not-empty nj-finite* **using** *Max-ge Max-in* **by** *blast*
then obtain *n* **where** *n-in*: $n \in nj\text{-}set$ **and** *n-bigger*: $\forall\, nj \in nj\text{-}set.\ nj \leq n$ **by** *blast*
then obtain *j0* **where** *j0-in*: $j0 \in \{0..<m\}$ **and** *suc-n-length*: $enat\ (Suc\ n) < llength\ Ns$ **and**
j0-notin: $prems\text{-}of\ \iota\ !\ j0 \notin active\text{-}subset\ (snd\ (lnth\ Ns\ n))$ **and**
j0-allin: $(\forall\, k.\ k > n \longrightarrow enat\ k < llength\ Ns \longrightarrow$
$prems\text{-}of\ \iota\ !\ j0 \in active\text{-}subset\ (snd\ (lnth\ Ns\ k)))$
unfolding *nj-set-def* **by** *blast*
obtain *C0* **where** *C0-is*: $prems\text{-}of\ \iota\ !\ j0 = (C0,\ active)$
using *j0-in i-in2* **unfolding** *m-def Inf-from-def active-subset-def*
by (*metis* (*mono-tags*, *lifting*) *active-subset-def j0-allin less-Suc-eq mem-Collect-eq split-pairs suc-n-length*)
then have *C0-prems-i*: $(C0,\ active) \in set\ (prems\text{-}of\ \iota)$ **using** *in-set-conv-nth j0-in m-def* **by** *force*
have *C0-in*: $(C0,\ active) \in (snd\ (lnth\ Ns\ (Suc\ n)))$
using *C0-is j0-allin suc-n-length* **by** (*simp add*: *active-subset-def*)
have *C0-notin*: $(C0,\ active) \notin (snd\ (lnth\ Ns\ n))$
using *C0-is j0-notin* **unfolding** *active-subset-def* **by** *simp*
have *step-n*: $lnth\ Ns\ n \rightsquigarrow LGC\ lnth\ Ns\ (Suc\ n)$
using *deriv chain-lnth-rel n-in* **unfolding** *nj-set-def* **by** *blast*
have *is-scheduled*: $\exists\, T2\ T1\ T'\ N1\ N\ C\ L\ N2.\ lnth\ Ns\ n = (T1,\ N1) \wedge lnth\ Ns\ (Suc\ n) = (T2,\ N2) \wedge$
$T2 = T1 \cup T' \wedge N1 = N \cup \{(C,\ L)\} \wedge N2 = N \cup \{(C,\ active)\} \wedge L \neq active \wedge$
$T' = no\text{-}labels.Inf\text{-}between\ (fst\ `\ active\text{-}subset\ N)\ \{C\}$
using *step.simps*[*of lnth Ns n lnth Ns* (*Suc n*)] *step-n C0-in C0-notin*
unfolding *active-subset-def* **by** *fastforce*
then obtain *T2 T1 T' N1 N L N2* **where** *nth-d-is*: $lnth\ Ns\ n = (T1,\ N1)$ **and**
suc-nth-d-is: $lnth\ Ns\ (Suc\ n) = (T2,\ N2)$ **and** *t2-is*: $T2 = T1 \cup T'$ **and**
n1-is: $N1 = N \cup \{(C0,\ L)\}\ N2 = N \cup \{(C0,\ active)\}$ **and**
l-not-active: $L \neq active$ **and**
tp-is: $T' = no\text{-}labels.Inf\text{-}between\ (fst\ `\ active\text{-}subset\ N)\ \{C0\}$
using *C0-in C0-notin j0-in C0-is* **using** *active-subset-def* **by** *fastforce*
have $j \in \{0..<m\} \Longrightarrow prems\text{-}of\ \iota\ !\ j \neq prems\text{-}of\ \iota\ !\ j0 \Longrightarrow prems\text{-}of\ \iota\ !\ j \in (active\text{-}subset\ N)$
for *j*
proof −
fix *j*
assume *j-in*: $j \in \{0..<m\}$ **and**
j-not-j0: $prems\text{-}of\ \iota\ !\ j \neq prems\text{-}of\ \iota\ !\ j0$
obtain *nj* **where** *nj-len*: $enat\ (Suc\ nj) < llength\ Ns$ **and**
nj-prems: $prems\text{-}of\ \iota\ !\ j \notin active\text{-}subset\ (snd\ (lnth\ Ns\ nj))$ **and**
nj-greater: $(\forall\, k.\ k > nj \longrightarrow enat\ k < llength\ Ns \longrightarrow$
$prems\text{-}of\ \iota\ !\ j \in active\text{-}subset\ (snd\ (lnth\ Ns\ k)))$
using *exist-nj j-in* **by** *blast*

then have *nj* ∈ *nj-set* **unfolding** *nj-set-def* **using** *j-in* **by** *blast*
moreover have *nj* ≠ *n*
proof (*rule ccontr*)
assume ¬ *nj* ≠ *n*
then have *prems-of* ι ! *j* = (*C0*, *active*)
using *C0-in C0-notin step.simps*[*of lnth Ns n lnth Ns* (*Suc n*)] *step-n*
active-subset-def is-scheduled nj-greater nj-prems suc-n-length **by** *auto*
then show *False* **using** *j-not-j0 C0-is* **by** *simp*
qed
ultimately have *nj* < *n* **using** *n-bigger* **by** *force*
then have *prems-of* ι ! *j* ∈ (*active-subset* (*snd* (*lnth Ns n*)))
using *nj-greater n-in Suc-ile-eq dual-order.strict-implies-order*
unfolding *nj-set-def* **by** *blast*
then show *prems-of* ι ! *j* ∈ (*active-subset N*)
using *nth-d-is l-not-active n1-is* **unfolding** *active-subset-def* **by** *force*
qed
then have *prems-i-active*: *set* (*prems-of* ι) ⊆ *active-subset N* ∪ {(*C0*, *active*)}
using *C0-prems-i C0-is m-def*
by (*metis Un-iff atLeast0LessThan in-set-conv-nth insertCI lessThan-iff subrelI*)
moreover have ¬ (*set* (*prems-of* ι) ⊆ *active-subset N* − {(*C0*, *active*)}) **using** *C0-prems-i* **by** *blast*
ultimately have ι ∈ *Inf-between* (*active-subset N*) {(*C0*, *active*)}
using *i-in-inf-fl prems-i-active* **unfolding** *Inf-between-def Inf-from-def* **by** *blast*
then have *to-F* ι ∈ *no-labels.Inf-between* (*fst* ‘ *active-subset N*) {*C0*}
unfolding *to-F-def Inf-between-def Inf-from-def*
no-labels.Inf-between-def no-labels.Inf-from-def
using *Inf-FL-to-Inf-F* **by** *force*
then have *i-in-t2*: *to-F* ι ∈ *T2* **using** *tp-is t2-is* **by** *simp*
have *j* ∈ {*0*..<*m*} ⟹ (∀ *k*. *k* > *n* ⟶ *enat k* < *llength Ns* ⟶
prems-of ι ! *j* ∈ *active-subset* (*snd* (*lnth Ns k*))) **for** *j*
proof (*cases j* = *j0*)
case *True*
assume *j* = *j0*
then show (∀ *k*. *k* > *n* ⟶ *enat k* < *llength Ns* ⟶
prems-of ι ! *j* ∈ *active-subset* (*snd* (*lnth Ns k*))) **using** *j0-allin* **by** *simp*
next
case *False*
assume *j-in*: *j* ∈ {*0*..<*m*} **and**
j ≠ *j0*
obtain *nj* **where** *nj-len*: *enat* (*Suc nj*) < *llength Ns* **and**
nj-prems: *prems-of* ι ! *j* ∉ *active-subset* (*snd* (*lnth Ns nj*)) **and**
nj-greater: (∀ *k*. *k* > *nj* ⟶ *enat k* < *llength Ns* ⟶
prems-of ι ! *j* ∈ *active-subset* (*snd* (*lnth Ns k*)))
using *exist-nj j-in* **by** *blast*
then have *nj* ∈ *nj-set* **unfolding** *nj-set-def* **using** *j-in* **by** *blast*
then show (∀ *k*. *k* > *n* ⟶ *enat k* < *llength Ns* ⟶
prems-of ι ! *j* ∈ *active-subset* (*snd* (*lnth Ns k*)))
using *nj-greater n-bigger* **by** *auto*
qed
then have *allj-allk*: (∀ *c*∈ *set* (*prems-of* ι). (∀ *k*. *k* > *n* ⟶ *enat k* < *llength Ns* ⟶
c ∈ *active-subset* (*snd* (*lnth Ns k*))))
using *m-def* **by** (*metis atLeast0LessThan in-set-conv-nth lessThan-iff*)
have ∀ *c*∈ *set* (*prems-of* ι). *snd c* = *active*
using *prems-i-active* **unfolding** *active-subset-def* **by** *auto*
then have *ex-n-i-in*: ∃ *n*. *enat* (*Suc n*) < *llength Ns* ∧ *to-F* ι ∈ *fst* (*lnth Ns* (*Suc n*)) ∧
(∀ *c*∈ *set* (*prems-of* ι). *snd c* = *active*) ∧

```
    (∀ c∈ set (prems-of ι). (∀ k. k > n ⟶ enat k < llength Ns ⟶
      c ∈ active-subset (snd (lnth Ns k))))
   using allj-allk i-in-t2 suc-nth-d-is fstI n-in nj-set-def
   by auto
 then have ∃ n. enat n < llength Ns ∧ to-F ι ∈ fst (lnth Ns n) ∧
    (∀ c∈ set (prems-of ι). snd c = active) ∧ (∀ c∈ set (prems-of ι). (∀ k. k ≥ n ⟶
      enat k < llength Ns ⟶ c ∈ active-subset (snd (lnth Ns k))))
   by auto
}
ultimately obtain n T2 N2 where i-in-suc-n: to-F ι ∈ fst (lnth Ns n) and
 all-prems-active-after: m > 0 ⟹ (∀ c∈ set (prems-of ι). (∀ k. k ≥ n ⟶ enat k < llength Ns ⟶
          c ∈ active-subset (snd (lnth Ns k)))) and
 suc-n-length: enat n < llength Ns and suc-nth-d-is: lnth Ns n = (T2, N2)
 by (metis less-antisym old.prod.exhaust zero-less-Suc)
then have i-in-t2: to-F ι ∈ T2 by simp
have ∃ p≥n. enat (Suc p) < llength Ns ∧ to-F ι ∈ (fst (lnth Ns p)) ∧ to-F ι ∉ (fst (lnth Ns (Suc p)))
proof (rule ccontr)
 assume
   contra: ¬ (∃ p≥n. enat (Suc p) < llength Ns ∧ to-F ι ∈ (fst (lnth Ns p)) ∧
            to-F ι ∉ (fst (lnth Ns (Suc p))))
 then have i-in-suc: p0 ≥ n ⟹ enat (Suc p0) < llength Ns ⟹ to-F ι ∈ (fst (lnth Ns p0)) ⟹
   to-F ι ∈ (fst (lnth Ns (Suc p0))) for p0
   by blast
 have p0 ≥ n ⟹ enat p0 < llength Ns ⟹ to-F ι ∈ (fst (lnth Ns p0)) for p0
 proof (induction rule: nat-induct-at-least)
   case base
   then show ?case using i-in-t2 suc-nth-d-is
     by simp
 next
   case (Suc p0)
   assume p-bigger-n: n ≤ p0 and
     induct-hyp: enat p0 < llength Ns ⟹ to-F ι ∈ fst (lnth Ns p0) and
     sucsuc-smaller-d: enat (Suc p0) < llength Ns
   have suc-p-bigger-n: n ≤ p0 using p-bigger-n by simp
   have suc-smaller-d: enat p0 < llength Ns
     using sucsuc-smaller-d Suc-ile-eq dual-order.strict-implies-order by blast
   then have to-F ι ∈ fst (lnth Ns p0) using induct-hyp by blast
   then show ?case using i-in-suc[OF suc-p-bigger-n sucsuc-smaller-d] by blast
 qed
 then have i-in-all-bigger-n: ∀ j. j ≥ n ∧ enat j < llength Ns ⟶ to-F ι ∈ (fst (lnth Ns j))
   by presburger
 have llength (lmap fst Ns) = llength Ns by force
 then have to-F ι ∈ ⋂ (lnth (lmap fst Ns) ‘ {j. n ≤ j ∧ enat j < llength (lmap fst Ns)})
   using i-in-all-bigger-n using Suc-le-D by auto
 then have to-F ι ∈ Liminf-llist (lmap fst Ns)
   unfolding Liminf-llist-def using suc-n-length by auto
 then show False using final-schedule by fast
qed
then obtain p where p-greater-n: p ≥ n and p-smaller-d: enat (Suc p) < llength Ns and
 i-in-p: to-F ι ∈ (fst (lnth Ns p)) and i-notin-suc-p: to-F ι ∉ (fst (lnth Ns (Suc p)))
 by blast
have p-neq-n: Suc p ≠ n using i-notin-suc-p i-in-suc-n by blast
have step-p: lnth Ns p ⇝LGC lnth Ns (Suc p) using deriv p-smaller-d chain-lnth-rel by blast
then have ∃ T1 T2 ι N2 N1 M. lnth Ns p = (T1, N1) ∧ lnth Ns (Suc p) = (T2, N2) ∧
   T1 = T2 ∪ {ι} ∧ N2 = N1 ∪ M ∧ active-subset M = {} ∧
```

```
    ι ∈ no-labels.Red-I-G (fst ‘ (N1 ∪ M))
proof −
  have ci-or-do: (∃ T1 T2 ι N2 N1 M. lnth Ns p = (T1, N1) ∧ lnth Ns (Suc p) = (T2, N2) ∧
      T1 = T2 ∪ {ι} ∧ N2 = N1 ∪ M ∧ active-subset M = {} ∧
      ι ∈ no-labels.Red-I-G (fst ‘ (N1 ∪ M))) ∨
      (∃ T1 T2 T' N. lnth Ns p = (T1, N) ∧ lnth Ns (Suc p) = (T2, N) ∧
      T1 = T2 ∪ T' ∧ T' ∩ no-labels.Inf-from (fst ‘ active-subset N) = {})
    using step.simps[of lnth Ns p lnth Ns (Suc p)] step-p i-in-p i-notin-suc-p by fastforce
  then have p-greater-n-strict: n < Suc p
    using suc-nth-d-is p-greater-n i-in-t2 i-notin-suc-p le-eq-less-or-eq by force
  have m > 0 ⟹ j ∈ {0..<m} ⟹ prems-of (to-F ι) ! j ∈ fst ‘ active-subset (snd (lnth Ns p))
    for j
  proof −
    fix j
    assume
      m-pos: m > 0 and
      j-in: j ∈ {0..<m}
    then have prems-of ι ! j ∈ (active-subset (snd (lnth Ns p)))
      using all-prems-active-after[OF m-pos] p-smaller-d m-def p-greater-n p-neq-n
      by (meson Suc-ile-eq atLeastLessThan-iff dual-order.strict-implies-order nth-mem
          p-greater-n-strict)
    then have fst (prems-of ι ! j) ∈ fst ‘ active-subset (snd (lnth Ns p))
      by blast
    then show prems-of (to-F ι) ! j ∈ fst ‘ active-subset (snd (lnth Ns p))
      unfolding to-F-def using j-in m-def by simp
  qed
  then have prems-i-active-p: m > 0 ⟹
      to-F ι ∈ no-labels.Inf-from (fst ‘ active-subset (snd (lnth Ns p)))
    using i-in-F unfolding no-labels.Inf-from-def
    by (simp add: atLeast0LessThan m-def-F) (metis in-set-conv-nth subsetI)
  have m = 0 ⟹ (∃ T1 T2 ι N2 N1 M. lnth Ns p = (T1, N1) ∧ lnth Ns (Suc p) = (T2, N2) ∧
      T1 = T2 ∪ {ι} ∧ N2 = N1 ∪ M ∧ active-subset M = {} ∧
      ι ∈ no-labels.Red-I-G (fst ‘ (N1 ∪ M)))
    using ci-or-do premise-free-inf-always-from[of to-F ι fst ‘ active-subset -, OF i-in-F]
      m-def i-in-p i-notin-suc-p m-def-F by auto
  then show (∃ T1 T2 ι N2 N1 M. lnth Ns p = (T1, N1) ∧ lnth Ns (Suc p) = (T2, N2) ∧
      T1 = T2 ∪ {ι} ∧ N2 = N1 ∪ M ∧ active-subset M = {} ∧
      ι ∈ no-labels.Red-I-G (fst ‘ (N1 ∪ M)))
    using ci-or-do i-in-p i-notin-suc-p prems-i-active-p unfolding active-subset-def by force
qed
then obtain T1p T2p N1p N2p Mp where lnth Ns p = (T1p, N1p) and
  suc-p-is: lnth Ns (Suc p) = (T2p, N2p) and T1p = T2p ∪ {to-F ι} and T2p ∩ {to-F ι} = {} and
  n2p-is: N2p = N1p ∪ Mpand active-subset Mp = {} and
  i-in-red-inf: to-F ι ∈ no-labels.Red-I-G
      (fst ‘ (N1p ∪ Mp))
  using i-in-p i-notin-suc-p by fastforce
have to-F ι ∈ no-labels.Red-I (fst ‘ (snd (lnth Ns (Suc p))))
  using i-in-red-inf suc-p-is n2p-is by fastforce
then have ∀ q ∈ Q. (G-I-q q (to-F ι) ≠ None ∧
    the (G-I-q q (to-F ι)) ⊆ Red-I-q q (⋃ (G-F-q q ‘ fst ‘ snd (lnth Ns (Suc p)))))
    ∨ (G-I-q q (to-F ι) = None ∧
    G-F-q q (concl-of (to-F ι)) ⊆ ⋃ (G-F-q q ‘ fst ‘ snd (lnth Ns (Suc p))) ∪
      Red-F-q q (⋃ (G-F-q q ‘ fst ‘ snd (lnth Ns (Suc p)))))
  unfolding to-F-def no-labels.Red-I-def no-labels.Red-I-G-q-def by blast
then have ι ∈ Red-I-G (snd (lnth Ns (Suc p)))
```

using *i-in-inf-fl* **unfolding** *Red-I-$\mathcal{G}$-def Red-I-$\mathcal{G}$-q-def* **by** (*simp add*: *to-F-def*)
then show $\iota \in$ *Sup-llist* (*lmap Red-I-$\mathcal{G}$* (*lmap snd Ns*))
unfolding *Sup-llist-def* **using** *suc-n-length p-smaller-d* **by** *auto*
qed

theorem *lgc-complete-Liminf*:
assumes
deriv: *chain* ($\rightsquigarrow$*LGC*) *Ns* **and**
init-state: *active-subset* (*snd* (*lhd Ns*)) = {} **and**
final-state: *passive-subset* (*Liminf-llist* (*lmap snd Ns*)) = {} **and**
no-prems-init: $\forall \iota \in$ *Inf-F*. *prems-of* $\iota = [] \longrightarrow \iota \in$ *fst* (*lhd Ns*) **and**
final-schedule: *Liminf-llist* (*lmap fst Ns*) = {} **and**
b-in: *B* $\in$ *Bot-F* **and**
bot-entailed: *no-labels.entails-$\mathcal{G}$* (*fst* ‘ *snd* (*lhd Ns*)) {*B*}
shows $\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *Liminf-llist* (*lmap snd Ns*)
proof −
have *labeled-b-in*: (*B*, *active*) $\in$ *Bot-FL* **using** *b-in* **by** *simp*
have *simp-snd-lmap*: *lhd* (*lmap snd Ns*) = *snd* (*lhd Ns*)
by (*rule llist.map-sel*(*1*)[*OF chain-not-lnull*[*OF deriv*]])
have *labeled-bot-entailed*: *entails-$\mathcal{G}$-L* (*snd* (*lhd Ns*)) {(*B*, *active*)}
using *labeled-entailment-lifting bot-entailed* **by** *fastforce*
have *fair* (*lmap snd Ns*)
using *lgc-fair*[*OF deriv init-state final-state no-prems-init final-schedule*] **.**
then show *?thesis*
using *dynamically-complete-Liminf labeled-b-in lgc-to-red*[*OF deriv*]
labeled-bot-entailed simp-snd-lmap std-Red-I-eq
by *presburger*
qed

theorem *lgc-complete*:
assumes
deriv: *chain* ($\rightsquigarrow$*LGC*) *Ns* **and**
init-state: *active-subset* (*snd* (*lhd Ns*)) = {} **and**
final-state: *passive-subset* (*Liminf-llist* (*lmap snd Ns*)) = {} **and**
no-prems-init: $\forall \iota \in$ *Inf-F*. *prems-of* $\iota = [] \longrightarrow \iota \in$ *fst* (*lhd Ns*) **and**
final-schedule: *Liminf-llist* (*lmap fst Ns*) = {} **and**
b-in: *B* $\in$ *Bot-F* **and**
bot-entailed: *no-labels.entails-$\mathcal{G}$* (*fst* ‘ *snd* (*lhd Ns*)) {*B*}
shows $\exists$ *i*. *enat i* < *llength Ns* $\wedge$ ($\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *snd* (*lnth Ns i*))
proof −
have $\exists$ *BL*$\in$*Bot-FL*. *BL* $\in$ *Liminf-llist* (*lmap snd Ns*)
using *assms* **by** (*rule lgc-complete-Liminf*)
then show *?thesis*
unfolding *Liminf-llist-def* **by** *auto*
qed

end

end