Formalization of a Comprehensive Framework for Saturation Theorem Proving in Isabelle/HOL

Sophie Tourret
April 20, 2020

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 Consequence Relations and Inference Systems 2
  1.1 Consequence Relations ................................................. 2
  1.2 Families of Consequence Relations .................................. 2
  1.3 Inference Systems ................................................... 3

2 Calculi 3
  2.1 Calculi with a Redundancy Criterion ............................... 4
  2.2 Calculi with a Family of Redundancy Criteria .................... 8
  2.3 Variations on a Theme ............................................... 13

3 Lifting to Non-ground Calculi 21
  3.1 Standard Lifting ..................................................... 21
  3.2 Strong Standard Lifting ............................................. 22
  3.3 Lifting with a Family of Well-founded Orderings ................ 23
  3.4 Lifting with a Family of Redundancy Criteria .................... 31

4 Labeled Liftings 36
  4.1 Labeled Lifting with a Family of Well-founded Orderings .......... 37
  4.2 Labeled Lifting with a Family of Redundancy Criteria ............ 40
1 Consequence Relations and Inference Systems

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. This corresponds to section 2.1 of the report.

theory Consequence-Relations-and-Inference-Systems
imports Main
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 ≠ {} and bot-implies-all: B ∈ Bot ⇒ {B} |= N1 and subset-entailed: N2 ⊆ N1 ⇒ N1 |= N2 and all-formulas-entailed: (∀ C ∈ N2. N1 |= {C}) ⇒ N1 |= N2 and entails-trans [trans]: N1 |= N2 ⇒ N2 |= N3 ⇒ N1 |= N3
begin

lemma entail-set-all-formulas: N1 |= N2 ←→ (∀ C ∈ N2. N1 |= {C})
  by (meson all-formulas-entailed empty-subsetI insert-subset subset-entailed entails-trans)

lemma entail-union: N |= N1 ∧ N |= N2 ←→ N |= N1 ∪ 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: (∀ i ∈ I. N |= Ni i) ←→ N |= ∪ (Ni : 'I)
  using entail-set-all-formulas[of N ∪ (Ni : 'I)] entail-set-all-formulas[of N]
  Complete-Lattices.UN-ball-bex-simps(2)[of Ni : 'I ∧ C. N |= {C}, symmetric]
  by meson

lemma entail-all-bot: (∃ B ∈ Bot. N |= {B}) ⇒ (∀ B' ∈ Bot. N |= {B'})
  using bot-implies-all entails-trans by blast

end

1.2 Families of Consequence Relations

locale consequence-relation-family =
    fixes Bot :: 'f set and Q :: 'q itself and
    entails-q :: 'q ⇒ ('f set ⇒ 'f set ⇒ bool)
    assumes


Bot-not-empty: Bot ≠ {} and
q-cons-rel: consequence-relation Bot (entails-q q)
begin

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

lemma intersect-cons-rel-family: consequence-relation Bot entails-Q
unfolding consequence-relation-def
proof (intro conjI)
show (Bot ≠ {}) using Bot-not-empty.
next
show ∀ B N. B ∈ Bot −→ {B} |= Q N
unfolding entails-Q-def by (metis consequence-relation-def q-cons-rel)
next
show ∀ N2 N1. N2 ⊆ N1 −→ N1 |= Q N2
unfolding entails-Q-def by (metis consequence-relation-def q-cons-rel)
next
show ∀ N2 N1. (∀ C∈N2. N1 |= Q {C}) −→ N1 |= Q N2
unfolding entails-Q-def by (metis consequence-relation-def q-cons-rel)
next
show ∀ N1 N2 N3. N1 |= Q N2 −→ N2 |= Q N3 −→ N1 |= Q N3
unfolding entails-Q-def by (metis consequence-relation-def q-cons-rel)
qed

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-from2 :: 'f set ⇒ 'f set ⇒ 'f inference set where
Inf-from2 N M = Inf-from (N ∪ M) − Inf-from (N − M)

end

end

2 Calculi

In this section, the section 2.2 to 2.4 of the report are covered. This includes results on calculi equipped with a redundancy criterion or with a family of redundancy criteria, as well as a proof that various notions of redundancy are equivalent

theory Calculi
locale calculus-with-red-crit = inference-system Inf + consequence-relation Bot entails
  for
  Bot :: 'f set and
  Inf :: ('f inference set) and
  entails :: 'f set ⇒ 'f set ⇒ bool (infix |= 50)
+ fixes
  Red-Inf :: 'f set ⇒ 'f inference set and
  Red-F :: 'f set ⇒ 'f set
assumes
  Red-Inf-to-Inf: Red-Inf N ⊆ Inf and
  Red-F-Bot: B ∈ Bot ⇒ N |= {B} ⇒ N - Red-F N |= {B} and
  Red-F-of-subset: N ⊆ N' ⇒ Red-F N ⊆ Red-F N' and
  Red-Inf-of-subset: N ⊆ N' ⇒ Red-Inf N ⊆ Red-Inf N' and
  Red-F-of-Red-F-subset: N' ⊆ Red-F N ⇒ Red-F N ⊆ Red-F (N - N') and
  Red-Inf-of-Red-F-subset: N' ⊆ Red-F N ⇒ Red-Inf N ⊆ Red-Inf (N - N') and
begin
lemma Red-Inf-of-Inf-to-N-subset: {i ∈ Inf. (concl-of i ∈ N)} ⊆ Red-Inf N
  using Red-Inf-of-Inf-to-N by blast

lemma red-concl-to-red-inf:
  assumes
    i-in: i ∈ Inf and
    concl: concl-of i ∈ Red-F N
  shows i ∈ Red-Inf N
proof –
  have i ∈ Red-Inf (Red-F N) by (simp add: Red-Inf-of-Inf-to-N i-in concl)
  then have i-in-Red: i ∈ Red-Inf (N ∪ Red-F N) by (simp add: Red-Inf-of-Inf-to-N concl i-in)
  have red-n-subs: Red-F N ⊆ Red-F (N ∪ Red-F N) by (simp add: Red-F-of-subset)
  then have i ∈ Red-Inf ((N ∪ Red-F N) - (Red-F N - N)) using Red-Inf-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-with-red-crit.Red-Inf-of-subset calculus-with-red-crit-axioms sup-bot.right-neutral)
qed

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

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

lemma Red-Inf-without-red-F:
  Red-Inf (N - Red-F N) = Red-Inf N
  using Red-Inf-of-subset [of N - Red-F N N]
    and Red-Inf-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-Inf N using saturated unfolding saturated-def by auto
  also have Red-Inf N ⊆ Red-Inf (N – Red-F N) using Red-Inf-of-Red-F-subset by auto
  finally have Inf-from (N – Red-F N) ⊆ Red-Inf (N – Red-F N) by auto
  then show ?thesis unfolding saturated-def by auto
qed

definition Sup-Red-Inf-lolist :: 'f set lists ⇒ 'f inference set where
  Sup-Red-Inf-lolist D = (⋃ i ∈ {i. enat i < llens D}. Red-Inf (nth D i))

lemma Sup-Red-Inf-unit: Sup-Red-Inf-lolist (LCons X LNil) = Red-Inf X
  using Sup-Red-Inf-lolist-def enat-0-iff(1) by simp

definition fair :: 'f set lists ⇒ bool where
  fair D ≡ Inf-from (Liminf-lolist D) ⊆ Sup-Red-Inf-lolist D

inductive derive :: 'f set lists ⇒ bool (infix ▶ Red 50) where
  derive: M ▶ N ⊆ Red-F N ⇒ M ▶ Red N


lemma (in−) elem-Sup-lolist-imp-Sup-upto-lolist':
  x ∈ Sup-lolist Xs ≡ ∃ j < llens Xs. x ∈ Sup-upto-lolist Xs j
  unfolding Sup-lolist-def Sup-upto-lolist-def by blast

lemma gt-Max-notin: (finite A ⇒ A ≠ {}) ⇒ x > Max A ⇒ x ∉ A by auto

lemma equiv-Sup-Liminf:
  assumes
    in-Sup: C ∈ Sup-lolist D and
    not-in-Liminf: C ∉ Liminf-lolist D
  shows
    ∃ i ∈ {i. enat i < llens D}. C ∈ nth D i − nth D (Suc i)
proof –
  obtain i where C-D-i: C ∈ Sup-upto-lolist D i and i < llens D
    using elem-Sup-lolist-imp-Sup-upto-lolist' in-Sup by fast
  then obtain j where j ≥ i ∧ enat j < llens D ∧ C ∉ nth D j using not-in-Liminf
    unfolding Sup-lolist-def chain-def Liminf-lolist-def by auto
  obtain k where k: C ∈ nth D k enat k < llens D k ≤ i using C-D-i
    unfolding Sup-upto-lolist-def by auto
  let ?S = {i. i < j ∧ i ≥ k ∧ C ∈ nth D i}
  define l where l ≡ Max ?S
  have :k ∈ {i. i < j ∧ k ≤ i ∧ C ∈ nth D i} using k j by (auto simp: order.order-iff-strict)
  then have nempty: {i. i < j ∧ k ≤ i ∧ C ∈ nth D i} ≠ {} by auto
  then have l-prop: l < j ∧ l ≥ k ∧ C ∈ nth D l) using Max-in[of ?S, OF - nempty] unfolding l-def
    by auto
  then have C ∈ nth D l − nth D (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) < llens D}
        using l-prop j by (clarify, metisSuc-leI dual-order.order-iff-strict enat-ord-simps(2) less-trans)
lemma Red-in-Sup:
  assumes deriv: chain (▷Red) D
  shows Sup-list D − Liminf-list D ⊆ Red-F (Sup-list D)
proof
  fix C
  assume C-in-subset: C ∈ Sup-list D − Liminf-list D
  fix i
  assume in-ith-elem: C ∈ lnth D i − lnth D (Suc i) and
  i: enat i < llength D
  have lnth D i ▷Red lnth D (Suc i) using deriv in-ith-elem chain-lnth-rel by auto
  then have C ∈ Red-F (lnth D (Suc i)) using in-ith-elem derive cases by blast
  then have C ∈ Red-F (Sup-list D) using Red-F-of-subset
  then have C ∈ Red-F (Sup-list D) using equiv-Sup-Liminf[of C] C-in-subset by fast
qed

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

lemma Red-F-subset-Liminf:
  assumes deriv: ⟨ chain (▷Red) D ⟩ and
  i: ⟨ enat i < llength D ⟩
  shows ⟨ Red-F (lnth D i) ⊆ Red-F (Liminf-list D) ⟩
proof
  have Sup-in-diff: ⟨ Red-F (Sup-list D) ⊆ Red-F (Sup-list D − (Sup-list D − Liminf-list D)) ⟩
    using Red-F-of-Red-F-subset[OF Red-in-Sup] deriv by auto
  also have ⟨ Sup-list D − (Sup-list D − Liminf-list D) = Liminf-list D ⟩
    by (simp add: Liminf-list-subset-Sup-list double-diff)
  then have Red-F-Sup-in-Liminf: ⟨ Red-F (Sup-list D) ⊆ Red-F (Liminf-list D) ⟩
    using Sup-in-diff by auto
  have ⟨ lnth D i ⊆ Sup-list D ⟩ unfolding Sup-list-def using i by blast
  then have Red-F (lnth D i) ⊆ Red-F (Sup-list D) using Red-F-of-subset
unfolding Sup-list-def by auto
then show \textit{thesis} using Red-F-Sup-Liminf by auto
qed

\textbf{lemma} \texttt{i-in-Liminf-or-Red-F}:
\textbf{assumes}
\begin{itemize}
  \item deriv: \texttt{\langle chain (\triangleright Red) D \rangle and} \\
  \item i: \texttt{\langle enat i < llength D \rangle}
\end{itemize}
\textbf{shows} \texttt{\langle lnth D i \subseteq \text{Red-F} (\text{Liminf-llist D}) \cup \text{Liminf-llist D} \rangle}
\textbf{proof} (\texttt{rule,rule})
fix \textit{C}
\textbf{assume} \texttt{C: \langle C \in lnth D i \rangle}
\textbf{and} \texttt{C-not-Liminf: \langle C \notin \text{Liminf-llist D} \rangle}
\textbf{have} \texttt{\langle C \in \text{Sup-list D} \rangle}\text{ unfolding Sup-list-def using C i by auto}
\textbf{then obtain} \textit{j} \text{ where} \texttt{j: \langle C \in lnth D j - lnth D (Suc j) \rangle \langle enat (Suc j) < llength D \rangle}
\textbf{using} \texttt{equiv-Sup-Liminf[of C D] C-not-Liminf by auto}
\textbf{then have} \texttt{\langle C \in \text{Red-F} (\text{lnth D (Suc j)}) \rangle}
\textbf{using} \texttt{deriv by (meson chain-lnth-rel contra-subsetD derive.cases)}
\textbf{then show} \texttt{\langle C \in \text{Red-F} (\text{Liminf-llist D}) \rangle}\text{ using Red-F-subset-Liminf[of D Suc j] deriv j(2) by blast}
qed

\textbf{lemma} \texttt{fair-implies-Liminf-saturated}:
\textbf{assumes}
\begin{itemize}
  \item deriv: \texttt{\langle chain (\triangleright Red) D \rangle and} \\
  \item fair: \texttt{\langle fair D \rangle}
\end{itemize}
\textbf{shows} \texttt{\langle \text{saturated} (\text{Liminf-llist D}) \rangle}
\textbf{proof}
fix \textit{ι}
\textbf{assume} \texttt{ι: \langle ι \in \text{Inf-from} (\text{Liminf-llist D}) \rangle}
\textbf{have} \texttt{\langle ι \in \text{Sup-Red-Inf-llist D} \rangle}\text{ unfolding fair-def by auto}
\textbf{then obtain} \textit{i} \text{ where} \texttt{i: \langle enat i < llength D \rangle \langle ι \in \text{Red-Inf} (\text{lnth D i}) \rangle}
\textbf{using} \texttt{Sup-Red-Inf-llist-def by auto}
\textbf{then show} \texttt{\langle ι \in \text{Red-Inf} (\text{Liminf-llist D}) \rangle}
\textbf{using} \texttt{deriv i-in-Liminf-or-Red-F[of D i] Red-Inf-subset-Liminf by blast}
qed

end

locale \texttt{static-refutational-complete-calculus} = calculus-with-red-crit +
\textbf{assumes} static-refutational-complete: \texttt{\langle B \in Bot \Rightarrow saturated N \Rightarrow N \models \{ B \} \Rightarrow \exists B' \in \text{Bot.} B' \in N \rangle}

locale \texttt{dynamic-refutational-complete-calculus} = calculus-with-red-crit +
\textbf{assumes} dynamic-refutational-complete: \texttt{\langle B \in Bot \Rightarrow chain (\triangleright Red) D \Rightarrow fair D \Rightarrow lnth D 0 \models \{ B \} \Rightarrow \exists i \in \{ i. \text{enat i < llength D} \}. \exists B' \in \text{Bot.} B' \in lnth D i \rangle}
begin

sublocale \texttt{static-refutational-complete-calculus}
\textbf{proof}
fix \textit{B N}
\textbf{assume}
2.2 Calculi with a Family of Redundancy Criteria

locale calculus-with-red-crit-family = inference-system Inf + consequence-relation-family Bot Q entails-q for
  Bot :: 'f set and
  Inf :: (if inference set) and

bot-elem: ⟨B ∈ Bot⟩ and
saturated-N: saturated N and
refut-N: N ⊢ {B}
define D where D = LCons N LNil
have simp: (∃ lnth D) by (auto simp: D-def)
have deriv-D: (chain (Red) D) by (simp add: chain-chain-singleton D-def)
have liminf-is-N: Liminf-list D = N by (simp add: D-def Liminf-list-LCons)
have head-D: N = lnth D 0 by (simp add: D-def)
have Sup-Red-Inf-llist D = Red-Inf N by (simp add: D-def Sup-Red-Inf-unit)
then have fair-D: fair D using saturated-N by (simp add: fair-def saturated-def liminf-is-N)
obtain i B' where B'-is-bot: i B' ∈ Bot and B'-in: B' ∈ (lnth D i) and (i < llength D)
  using dynamic-refutational-complete[of B D] bot-elem fair-D head-D saturated-N deriv-D refut-N
by auto
then have i = 0
by (auto simp: D-def enat-0-iff)
show ⟨∃ B' ∈ Bot. B' ∈ N: B'-is-bot B'-in unfolding ⟨i = 0: head-D[symmetric] by auto
qed

end

sublocale static-refutational-complete-calculus ⊆ dynamic-refutational-complete-calculus
proof
fix B D
assume
  bot-elem: ⟨B ∈ Bot⟩ and
  deriv: ⟨chain (Red) D⟩ and
  fair: ⟨fair D⟩ and
  unsat: ⟨(lnth D 0) ⊢ {B}⟩
have non-empty: (∃ lnth D) using chain-not-lnull[OF deriv] .
have subs: ⟨(lnth D 0) ⊆ Sup-llist D⟩
  using lhd-subset-Sup-llist[of D] non-empty by (simp add: lhd-conv-lnth)
have Sup-llist D = {B}:
  using unsat subset-entailed[OF subs] entails-trans[of Sup-llist D lnth D 0] by auto
then have Sup-no-Red: ⟨Sup-llist D − Red-F (Sup-llist D) ⊆ Liminf-list D⟩
  using bot-elem Red-F-Bot by auto
have Sup-no-Red-in-Liminf: ⟨Sup-llist D − Red-F (Sup-llist D) ⊆ Liminf-list D⟩
  using deriv Red-in-Sup by auto
have Liminf-entails-Bot: ⟨Liminf-list D ⊆ {B}⟩
  using Sup-no-Red-subset-entailed[OF Sup-no-Red-in-Liminf] entails-trans by blast
have saturated (Liminf-list D):
  using deriv fair fair-implies-Liminf-saturated unfolding saturated-def by auto
then have ∃ B' ∈ Bot. B' ∈ (Liminf-list D)
  using bot-elem static-refutational-complete Liminf-entails-Bot by auto
then show ∃ i ∈ {i. enat i < llength D}. ∃ B' ∈ Bot. B' ∈ lnth D i
  unfolding Liminf-list-def by auto
qed


\[ Q \vdash \text{'q itself and} \]
\[ \text{entails-q} \vdash 'q \Rightarrow ('f set \Rightarrow 'f set \Rightarrow \text{bool}) \]
\[ + \text{fixes} \]
\[ \text{Red-Inf-q} \vdash 'q \Rightarrow ('f set \Rightarrow 'f inference set) \text{ and} \]
\[ \text{Red-F-q} \vdash 'q \Rightarrow ('f set \Rightarrow 'f set) \]
\[ \text{assumes} \]
\[ \text{all-red-crit: calculus-with-red-crit Bot Inf (entails-q q) (Red-Inf-q q) (Red-F-q q)} \]

\begin{definition}
\text{Red-Inf-Q} :: \('f set \Rightarrow \text{f inference set} \]
\[ \text{where} \]
\[ \text{Red-Inf-Q} N = \bigcap \{ X N \mid X \in \text{Red-Inf-q ' UNIV} \} \]
\end{definition}

\begin{definition}
\text{Red-F-Q} :: \('f set \Rightarrow \text{f set} \]
\[ \text{where} \]
\[ \text{Red-F-Q} N = \bigcap \{ X N \mid X \in \text{Red-F-q ' UNIV} \} \]
\end{definition}

\begin{lemma}
\text{inter-red-crit: calculus-with-red-crit Bot Inf entails-Q Red-Inf-Q Red-F-Q} \]
\end{lemma}
\[ \text{unfolding calculus-with-red-crit-def calculus-with-red-crit-axioms-def} \]
\[ \text{proof (intro conjI)} \]
\[ \text{show consequence-relation Bot entails-Q} \]
\[ \text{using intersect-cons-rel-family} , \]
\[ \text{next} \]
\[ \text{show } \forall N. \text{Red-Inf-Q N } \subseteq \text{Inf} \]
\[ \text{unfolding Red-Inf-Q-def} \]
\[ \text{proof} \]
\[ \text{fix } N \]
\[ \text{show } \bigcap \{ X N \mid X \in \text{Red-Inf-q ' UNIV} \} \subseteq \text{Inf} \]
\[ \text{proof (intro Inter-subset)} \]
\[ \text{fix Red-Inf} \]
\[ \text{assume one-red-inf: Red-Inf } \in \{ X N \mid X \in \text{Red-Inf-q ' UNIV} \} \]
\[ \text{show Red-Inf } \subseteq \text{Inf using one-red-inf} \]
\[ \text{proof} \]
\[ \text{assume } \exists \text{Red-Inf-qi. Red-Inf = Red-Inf-qi N } \land \text{Red-Inf-qi } \in \text{Red-Inf-q ' UNIV} \]
\[ \text{then obtain Red-Inf-qi where} \]
\[ \text{red-inf-def: Red-Inf } = \text{Red-Inf-qi N } \text{ and red-inf-qi-in: Red-Inf-qi } \in \text{Red-Inf-q ' UNIV} \]
\[ \text{by blast} \]
\[ \text{obtain qi where red-inf-qi-def: Red-Inf-qi } = \text{Red-Inf-q qi and qi-in: qi } \in \text{UNIV} \]
\[ \text{using red-inf-qi-in by blast} \]
\[ \text{show Red-Inf } \subseteq \text{Inf} \]
\[ \text{using all-red-crit calculus-with-red-crit.Red-Inf-to-Inf red-inf-qi-def} \]
\[ \text{red-inf-def by blast} \]
\[ \text{qed} \]
\[ \text{next} \]
\[ \text{show } \{ X N \mid X \in \text{Red-Inf-q ' UNIV} \} \neq \{ \} \text{ by blast} \]
\[ \text{qed} \]
\[ \text{qed} \]
\[ \text{next} \]
\[ \text{show } \forall B N. B \in \text{Bot } \rightarrow \text{ N } \models Q \{ B \} \rightarrow N - \text{Red-F-Q N } \models Q \{ B \} \]
\[ \text{proof (intro allI impI)} \]
\[ \text{fix B N} \]
\[ \text{assume} \]
\[ \text{B-in: B } \in \text{Bot and} \]
\[ \text{N-unsat: N } \models Q \{ B \} \]
\[ \text{show N - Red-F-Q N } \models Q \{ B \} \text{ unfolding entails-Q-def Red-F-Q-def} \]
\[ \text{proof} \]
fix qi

define entails-qi (infix \(\models q_i\)) where entails-qi = entails-q qi

have cons-rel-qi: consequence-relation Bot entails-qi

unfolding entails-qi-def using all-red-crit calculus-with-red-crit.axioms(1) by blast

define Red-F-qi where Red-F-qi = Red-F-q qi

have red-qi-in-Q: Red-F-Q N ⊆ Red-F-qi N

unfolding Red-F-Q-def Red-F-qi-def using image_iff by blast

then have N - (Red-F-qi N) ⊆ N - (Red-F-Q N) by blast

then have entails-1: (N - Red-F-Q N) \(\models q_i\) (N - Red-F-qi N)

using all-red-crit

unfolding calculus-with-red-crit-def consequence-relation-def entails-qi-def by metis

have N-unsat-qi: N \models q_i \{B\} using N-unsat unfolding entails-qi-def entails-Q-def by simp

then have N-unsat-qi: (N - Red-F-qi N) \models q_i \{B\}


by fastforce

show (N - \(\bigcap\) \{X N | X \in Red-F-q ' UNIV\}) \models q_i \{B\}

using consequence-relation.entails-trans[OF cons-rel-qi entails-1 N-unsat-qi]

unfolding Red-F-Q-def .

qed

qed

next

show \(\forall\) N1 N2. N1 ⊆ N2 \(\rightarrow\) Red-F-Q N1 ⊆ Red-F-Q N2

proof (intro allI impI)

fix N1 :: 'f set

and N2 :: 'f set

assume N1-in-N2: N1 ⊆ N2

show Red-F-Q N1 ⊆ Red-F-Q N2

proof

fix x

assume x-in: x \in Red-F-Q N1

then have \(\forall\) qi. x \in Red-F-q qi N1 unfolding Red-F-Q-def by blast

then have \(\forall\) qi. x \in Red-F-qi qi N2


by blast

then show x \in Red-F-Q N2 unfolding Red-F-Q-def by blast

qed

qed

next

show \(\forall\) N1 N2. N1 ⊆ N2 \(\rightarrow\) Red-Inf-Q N1 ⊆ Red-Inf-Q N2

proof (intro allI impI)

fix N1 :: 'f set

and N2 :: 'f set

assume N1-in-N2: N1 ⊆ N2

show Red-Inf-Q N1 ⊆ Red-Inf-Q N2

proof

fix x

assume x-in: x \in Red-Inf-Q N1

then have \(\forall\) qi. x \in Red-Inf-q qi N1 unfolding Red-Inf-Q-def by blast

then have \(\forall\) qi. x \in Red-Inf-qi qi N2


by blast

then show x \in Red-Inf-Q N2 unfolding Red-Inf-Q-def by blast

qed
qed

next

show \( \forall N2 \, N1. \ N2 \subseteq \text{Red-F-Q} \ N1 \rightarrow \text{Red-F-Q} \ N1 \subseteq \text{Red-F-Q} \ (N1 - N2) \)

proof (intro allI impI)

fix \( N2 \ N1 \)

assume \( N2\text{-in-Red-N1} : \ N2 \subseteq \text{Red-F-Q} \ N1 \)

show \( \text{Red-F-Q} \ N1 \subseteq \text{Red-F-Q} \ (N1 - N2) \)

proof

fix \( x \)

assume \( x\text{-in} : x \in \text{Red-F-Q} \ N1 \)

then have \( \forall qi. \ x \in \text{Red-F-q qi} \ N1 \) unfolding \( \text{Red-F-Q-def} \) by blast

moreover have \( \forall qi. \ N2 \subseteq \text{Red-F-q qi} \ N1 \) using \( N2\text{-in-Red-N1} \) unfolding \( \text{Red-F-Q-def} \) by blast

ultimately have \( \forall qi. \ x \in \text{Red-F-q qi} \ (N1 - N2) \) using all-red-crit calculus-with-red-crit.\( \text{axioms(2)} \) calculus-with-red-crit.\( \text{Red-F-of-Red-F-subset by blast} \)

then show \( x \in \text{Red-F-Q} \ (N1 - N2) \) unfolding \( \text{Red-F-Q-def by blast} \)

qed

qed

next

show \( \forall N2 \, N1. \ N2 \subseteq \text{Red-F-Q} \ N1 \rightarrow \text{Red-Inf-Q} \ N1 \subseteq \text{Red-Inf-Q} \ (N1 - N2) \)

proof (intro allI impI)

fix \( N2 \ N1 \)

assume \( N2\text{-in-Red-N1} : \ N2 \subseteq \text{Red-F-Q} \ N1 \)

show \( \text{Red-Inf-Q} \ N1 \subseteq \text{Red-Inf-Q} \ (N1 - N2) \)

proof

fix \( x \)

assume \( x\text{-in} : x \in \text{Red-Inf-Q} \ N1 \)

then have \( \forall qi. \ x \in \text{Red-Inf-q qi} \ N1 \) unfolding \( \text{Red-Inf-Q-def} \) by blast

moreover have \( \forall qi. \ N2 \subseteq \text{Red-F-q qi} \ N1 \) using \( N2\text{-in-Red-N1} \) unfolding \( \text{Red-F-Q-def} \) by blast

ultimately have \( \forall qi. \ x \in \text{Red-Inf-q qi} \ (N1 - N2) \) using all-red-crit calculus-with-red-crit.\( \text{axioms(2)} \) calculus-with-red-crit.\( \text{Red-Inf-of-Red-F-subset by blast} \)

then show \( x \in \text{Red-Inf-Q} \ (N1 - N2) \) unfolding \( \text{Red-Inf-Q-def by blast} \)

qed

qed

next

show \( \forall \iota \, N. \ \iota \in \text{Inf} \rightarrow \text{concl-of } \iota \in N \rightarrow \iota \in \text{Red-Inf-Q} \ N \)

proof (intro allI impI)

fix \( \iota \ N \)

assume \( \iota\text{-in: } \iota \in \text{Inf} \) and

\( \text{concl-in: concl-of } \iota \in N \)

then have \( \forall qi. \ \iota \in \text{Red-Inf-q qi} \ N \)

using all-red-crit calculus-with-red-crit.\( \text{axioms(2)} \) calculus-with-red-crit.\( \text{Red-Inf-of-Inf-to-N by blast} \)

then show \( \iota \in \text{Red-Inf-Q} \ N \) unfolding \( \text{Red-Inf-Q-def by blast} \)

qed

qed

sublocale inter-red-crit-calculus: calculus-with-red-crit

where \( \text{Bot=Bot} \) and \( \text{Inf=Inf} \) and \( \text{entails=entails-Q} \) and \( \text{Red-Inf=Red-Inf-Q} \) and \( \text{Red-F=Red-F-Q} \)

using \( \text{inter-red-crit .} \).
lemma sat-int-to-sat-q: calculus-with-red-crit.saturated Inf Red-Inf-Q N \iff
(\forall qi. \text{calculus-with-red-crit.saturated Inf (Red-Inf-q qi) N}) \text{ for } N

proof
fix N
assume inter-sat: calculus-with-red-crit.saturated Inf Red-Inf-Q N
show \forall qi. \text{calculus-with-red-crit.saturated Inf (Red-Inf-q qi) N}
proof
fix qi
interpret one: calculus-with-red-crit Bot Inf entails-q qi Red-Inf-q qi Red-F-q qi
by (rule all-red-crit)
show one.saturated N
using inter-sat unfolding one.saturated-def inter-red-crit-calculus.saturated-def Red-Inf-Q-def
by blast
qed

next
fix N
assume all-sat: \forall qi. \text{calculus-with-red-crit.saturated Inf (Red-Inf-q qi) N}
show inter-red-crit-calculus.saturated N unfolding inter-red-crit-calculus.saturated-def Red-Inf-Q-def
proof
fix x
assume x-in: x \in Inf-from N
have \forall Red-Inf-qi \in Red-Inf-q \ ' \ \text{UNIV}. \ x \in Red-Inf-qi N
proof
fix Red-Inf-qi
assume red-inf-in: Red-Inf-qi \in Red-Inf-q \ ' \ \text{UNIV}
then obtain qi where red-inf-qi-def: Red-Inf-qi = Red-Inf-q qi \text{ by blast}
interpret one: calculus-with-red-crit Bot Inf entails-q qi Red-Inf-q qi Red-F-q qi
by (rule all-red-crit)
have one.saturated N using all-sat red-inf-qi-def by blast
then show x \in Red-Inf-qi N unfolding one.saturated-def using x-in red-inf-qi-def by blast
qed
then show x \in \bigcap \{X \ | \ X. \ x \in Red-Inf-q \ ' \ \text{UNIV}\} by blast
qed

qed

lemma stat-ref-comp-from-bot-in-sat:
\forall N. (\text{calculus-with-red-crit.saturated Inf Red-Inf-Q N} \land (\forall B \in \text{Bot}. B \notin N)) \rightarrow
(\exists B \in \text{Bot}. \exists qi. \neg \text{entails-q qi N } \{B\}) \land
\neg \text{stat-ref-comp} \rightarrow \text{static-refutational-complete-calculus Bot Inf entails-Q Red-Inf-Q Red-F-Q}

proof (rule ccontr)
assume
N-saturated: \forall N. (\text{calculus-with-red-crit.saturated Inf Red-Inf-Q N} \land (\forall B \in \text{Bot}. B \notin N)) \rightarrow
(\exists B \in \text{Bot}. \exists qi. \neg \text{entails-q qi N } \{B\}) \land
no-stat-ref-comp: \neg \text{static-refutational-complete-calculus Bot Inf (|=Q) Red-Inf-Q Red-F-Q}

obtain N1 B1 where B1-in:
B1 \in \text{Bot and N1-saturated: calculus-with-red-crit.saturated Inf Red-Inf-Q N1 and}
N1-unsat: N1 \models Q \{B1\} and no-B-in-N1: \forall B \in \text{Bot}. B \notin N1
using no-stat-ref-comp by (metis inter-red-crit static-refutational-complete-calculus.intro
static-refutational-complete-calculus-axioms.intro)

obtain B2 qi where no-qi:\neg \text{entails-q qi N1 } \{B2\} using N-saturated
N1-saturated no-B-in-N1 by blast
have N1 \models 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-Q-def by blast
then show False using no-qi by simp
qed

2.3 Variations on a Theme

locale calculus-with-reduced-red-crit = calculus-with-red-crit Bot Inf entails Red-Inf Red-F
for
Bot :: 'f set and
Inf :: ('f inference set) and
entails :: 'f set ⇒ 'f set ⇒ bool (infix ⊨ 50) and
Red-Inf :: 'f set ⇒ 'f inference set and
Red-F :: 'f set ⇒ 'f set
+ assumes
inf-in-red-inf: Inf-from2 UNIV (Red-F N) ⊆ Red-Inf 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-Inf-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
proof
fix ι
assume i-in: ι ∈ Inf-from N
show ι ∈ Red-Inf N
using i-in red-sat-n inf-in-red-inf unfolding reduc-saturated-def Inf-from-def Inf-from2-def by blast
qed

end

locale reduc-static-refutational-complete-calculus = calculus-with-red-crit +
assumes reduc-static-refutational-complete: B ∈ Bot ⇒ reduc-saturated N ⇒ N \vdash \{B\} ⇒ ∃B'∈Bot. B' ∈ N

locale reduc-static-refutational-complete-reduc-calculus = calculus-with-reduced-red-crit +
assumes reduc-static-refutational-complete: B ∈ Bot ⇒ reduc-saturated N ⇒ N \vdash \{B\} ⇒ ∃B'∈Bot. B' ∈ N
begin

sublocale reduc-static-refutational-complete-calculus
by (simp add: calculus-with-red-crit-axioms reduc-static-refutational-complete
reduc-static-refutational-complete-calculus-axioms.intro reduc-static-refutational-complete-calculus-def)
sublocale static-refutational-complete-calculus
proof
fix B N
assume
  bot-elem: \( B \in \text{Bot} \)
  saturated-N: saturated N
  refut-N: \( N \vdash \{ B \} \)
have reduc-saturated-N: reduc-saturated N using saturated-N sat-eq-reduc-sat by blast
show \( \exists B' \in \text{Bot}. \ B' \in N \) using reduc-static-refutational-complete[OF bot-elem reduc-saturated-N refut-N]
.
qed
end

context calculus-with-reduced-red-crit
begin

lemma stat-ref-comp-imp-red-stat-ref-comp: static-refutational-complete-calculus Bot Inf entails Red-Inf
Red-F \implies reduc-static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
proof
fix B N
assume
  stat-ref-comp: static-refutational-complete-calculus Bot Inf \( (\vdash) \) Red-Inf Red-F
  bot-elem: \( B \in \text{Bot} \)
  saturated-N: reduc-saturated N
  refut-N: \( N \vdash \{ B \} \)
have reduc-saturated-N: saturated N using saturated-N sat-eq-reduc-sat by blast
show \( \exists B' \in \text{Bot}. \ B' \in N \)
  using Calculi.static-refutational-complete-calculus.static-refutational-complete[OF stat-ref-comp
  bot-elem reduc-saturated-N refut-N]
.
qed
end

context calculus-with-red-crit
begin

definition Red-Red-Inf ::= \( \{ F \} \) inference set where
Red-Red-Inf N = Red-Inf N \cup Inf-from2 UNIV (Red-F N)

lemma reduced-calc-is-calc: calculus-with-red-crit Bot Inf entails Red-Red-Inf Red-F
proof
fix N
show Red-Red-Inf N \subseteq Inf
  unfolding Red-Red-Inf-def Inf-from2-def Inf-from-def using Red-Inf-to-Inf by auto
next
fix B N
assume
  b-in: \( B \in \text{Bot} \)
  n-entails: \( N \vdash \{ B \} \)
show \( N - \text{Red-F} \ N \vdash \{ B \} \)
  by (simp add: Red-F-Bot b-in n-entails)
next
fix $N N'$ :: 'f set
assume $N \subseteq N'$
then show $\text{Red-F } N \subseteq \text{Red-F } N'$ by (simp add: Red-F-of-subset)

next
fix $N N'$ :: 'f set
assume $n$-in: $N \subseteq N'$
then show $\text{Red-Red-Inf } N \subseteq \text{Red-Red-Inf } N'$
  unfolding $\text{Red-Inf-of-subset}$ by blast

next
fix $N N'$ :: 'f set
assume $N' \subseteq \text{Red-F } N$
then show $\text{Red-F } N \subseteq \text{Red-F } (N - N' \cap N)$ by (simp add: Red-F-of-Red-F-subset)

next
fix $\iota N$
assume $\iota \in \text{Inf}$
then show $\iota \in \text{Red-Red-Inf } N$
  unfolding $\text{Red-Inf-of-Inf-to-N}$ Red-Red-Inf-def by blast

qed

lemma inf-subs-reduced-red-inf: $\text{Inf-from2 } \text{UNIV } \text{Red-F } N \subseteq \text{Red-Red-Inf } N$
  unfolding $\text{Red-Red-Inf-def}$ 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: calculus-with-reduced-red-crit Bot Inf entails $\text{Red-Red-Inf } \text{Red-F}$
  using inf-subs-reduced-red-inf reduced-calc-is-calc
  by (simp add: calculus-with-reduced-red-crit.intro calculus-with-reduced-red-crit-axioms-def)

interpretation reduc-calc : calculus-with-reduced-red-crit Bot Inf entails $\text{Red-Red-Inf } \text{Red-F}$
  using reduc-calc by simp

lemma sat-imp-red-calc-sat: saturated $N \implies$ reduc-calc.saturated $N$
  unfolding saturated-def reduc-calc.saturated-def Red-Red-Inf-def by blast

lemma red-sat-eq-red-calc-sat: reduc-saturated $N \iff$ reduc-calc.saturated $N$
proof
assume red-sat-n: reduc-saturated $N$
show reduc-calc.saturated $N$

15
unfolding reduc-calc.saturated-def
proof
  fix \( \iota \)
  assume i-in: \( \iota \in \text{Inf-from } N \)
  show \( \iota \in \text{Red-Red-Inf } N \)
    using i-in red-sat-n unfolding reduc-saturated-def Inf-from2-def Inf-from-def Red-Red-Inf-def by blast
qed
next
assume red-sat-n: reduc-calc.saturated N
show reduc-saturated N
  unfolding reduc-saturated-def
proof
  fix \( \iota \)
  assume i-in: \( \iota \in \text{Inf-from } (N - \text{Red-F } N) \)
  show \( \iota \in \text{Red-Inf } N \)
    using i-in red-sat-n unfolding Inf-from-def reduc-calc.saturated-def Red-Red-Inf-def Inf-from2-def
by blast
qed
qed

lemma red-sat-eq-sat: reduc-saturated N \iff saturated (N - Red-F N)
  unfolding reduc-saturated-def saturated-def by (simp add: Red-Inf-without-red-F)

theorem stat-is-stat-red: static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F \iff static-refutational-complete-calculus Bot Inf entails Red-Red-Inf Red-F
proof
  assume stat-ref1: static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  show static-refutational-complete-calculus Bot Inf entails Red-Red-Inf Red-F
    using reduc-calc.calculus-with-red-crit-axioms
  unfolding static-refutational-complete-calculus-def static-refutational-complete-calculus-axioms-def
proof
  show \( \forall B N. \ B \in \text{Bot } \longrightarrow \text{reduc-calc.saturated } N \longrightarrow N \models \{ B \} \longrightarrow (\exists B' \in \text{Bot. } B' \in N) \)
  proof (clarify)
  fix B N
  assume b-in: \( B \in \text{Bot } \) and
  n-sat: reduc-calc.saturated N and
  n-imp-b: \( N \models \{ B \} \)
  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 \( (N - \text{Red-F } N) \models \{ B \} \)
    using n-imp-b b-in by (simp add: reduc-calc.Red-F-Bot)
  ultimately show \( \exists B' \in \text{Bot. } B' \in N \)
    using stat-ref1 by (meson DiffD1 b-in static-refutational-complete-calculus.static-refutational-complete)
  qed
  qed
next
assume stat-ref3: static-refutational-complete-calculus Bot Inf entails Red-Red-Inf Red-F
show static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  unfolding static-refutational-complete-calculus-def static-refutational-complete-calculus-axioms-def
  using calculus-with-red-crit-axioms


proof
  show \( \forall B \, N. \, B \in \text{Bot} \rightarrow \text{saturated} \, N \rightarrow N \models \{B\} \rightarrow (\exists B' \in \text{Bot}. \, B' \in N) \)
proof clarify
  fix \( B \, N \)
  assume
    \( b\text{-in}: \, B \in \text{Bot} \) and
    \( n\text{-sat}: \, \text{saturated} \, N \) and
    \( n\text{-imp-b}: \, N \models \{B\} \)
  then show \( \exists B' \in \text{Bot}, \, B' \in N \)
using stat-ref3 sat-imp-red-calc-sat[OF n-sat]
by (meson static-refutational-complete-calculus.static-refutational-complete)
qed
qed

theorem red-stat-red-is-stat-red: reduc-static-refutational-complete-calculus Bot Inf entails Red-Red-Inf
Red-F \(\iff\) 
static-refutational-complete-calculus Bot Inf entails Red-Red-Inf Red-F
using reduc-calc.stat-ref-comp-imp-red-stat-ref-comp
by (metis reduc-calc.sat-eq-reduc-sat reduc-static-refutational-complete-calculus.axioms(2)
   reduc-static-refutational-complete-calculus-axioms-def reduc-static-refutational-complete-calculus.axioms)

definition Sup-Red-F-llist :: \('f set llist \Rightarrow 'f set\) where
  Sup-Red-F-llist D = \(\bigcup i \in \{i. \, \text{enat} \, i < \text{llength} \, D\}. \, \text{Red-F} \, (\text{lnth} \, D \, i)\)

lemma Sup-Red-F-unit: Sup-Red-F-llist (LCons X LNil) = Red-F X
using Sup-Red-F-llist-def enat-0-iff(1) by simp

lemma sup-red-f-in-red-liminf: chain derive D \(\Rightarrow\) Sup-Red-F-llist D \(\subseteq\) Red-F (Liminf-llist D)
proof
  fix \( N \)
  assume
    deriv: chain derive D and
    n-in-sup: \( N \in \text{Sup-Red-F-llist} \, D \)
  obtain i0 where i-smaller: \( \text{enat} \, i0 < \text{llength} \, D \) and n-in: \( N \in \text{Red-F} \, (\text{lnth} \, D \, i0) \)
  using n-in-sup unfolding Sup-Red-F-llist-def by blast
  have Red-F (lnth D i0) \(\subseteq\) Red-F (Liminf-llist D)
  using i-smaller by (simp add: deriv Red-F-subset-Liminf)
  then show \( N \in \text{Red-F} \, (\text{Liminf-llist} \, D) \)
    using n-in by fast
qed

lemma sup-red-inf-in-red-liminf: chain derive D \(\Rightarrow\) Sup-Red-Inf-llist D \(\subseteq\) Red-Inf (Liminf-llist D)
proof
fix \( i \)

assume

deriv: \( \text{chain derive } D \) and

i-in-sup: \( i \in \text{Sup-Red-Inf-llist } D \)

obtain \( i0 \) where i-smaller: \( \text{enat } i0 < l\text{length } D \) and n-in: \( i \in \text{Red-Inf } (\text{lnth } D i0) \)

using i-in-sup unfolding Sup-Red-Inf-llist-def by blast

have Red-Inf (\text{lnth } D i0) \subseteq Red-Inf (\text{Liminf-llist } D)

using i-smaller by (simp add: deriv Red-Inf-subset-Liminf)

then show \( i \in \text{Red-Inf } (\text{Liminf-llist } D) \)

using n-in by fast

qed

definition reduc-fair :: \( 'f \text{ set llist } \Rightarrow bool \) where

reduc-fair D \equiv Inf-from (\text{Liminf-llist } D \setminus (\text{Sup-Red-F-llist } D)) \subseteq \text{Sup-Red-Inf-llist } D

lemma reduc-fair-imp-Liminf-reduc-sat: chain derive D \Rightarrow reduc-fair D \Rightarrow reduc-saturated (\text{Liminf-llist } D)

unfolding reduc-saturated-def

proof

fix D

assume

deriv: \( \text{chain derive } D \) and

red-fair: reduc-fair D

have Inf-from (\text{Liminf-llist } D \setminus \text{Red-F } (\text{Liminf-llist } D)) \subseteq Inf-from (\text{Liminf-llist } D \setminus \text{Sup-Red-F-llist } D)

using sup-red-f-in-red-liminf[of deriv] unfolding Inf-from-def by blast

then have Inf-from (\text{Liminf-llist } D \setminus \text{Red-F } (\text{Liminf-llist } D)) \subseteq \text{Sup-Red-Inf-llist } D

using reduc-fair unfolding reduc-fair-def by simp

then show Inf-from (\text{Liminf-llist } D \setminus \text{Red-F } (\text{Liminf-llist } D)) \subseteq \text{Red-Inf } (\text{Liminf-llist } D)

using sup-red-inf-in-red-liminf[of deriv] by fast

qed

end

locale reduc-dynamic-refutational-complete-calculus = calculus-with-red-crit +

assumes

reduc-dynamic-refutational-complete: \( B \in \text{Bot } \Rightarrow \text{chain derive } D \Rightarrow \text{reduc-fair } D \Rightarrow \text{lnth } D 0 \models \{B\} \Rightarrow \exists i. \text{enat } i < l\text{length } D \}. \exists B'\in\text{Bot}. B' \in \text{lnth } D i

begin

begin

locale reduc-static-refutational-complete-calculus

proof

fix B N

assume

bot-elem: \( B \in \text{Bot} \) and

saturated-N: reduc-saturated N and

refut-N: \( N \models \{B\} \)

define D where D = LCons N LNil

have[simp]: \( \lnot \text{lnull } D \) by (auto simp: D-def)

have deriv-D: \( \text{chain } (\gg \text{Red }) D \) by (simp add: chain-chain-singleton D-def)

have liminf-is-N: \( \text{Liminf-llist } D = N \) by (simp add: D-def Liminf-llist-LCons)

have head-D: \( N = \text{lnth } D 0 \) by (simp add: D-def)

have Sup-Red-F-llist D = Red-F N by (simp add: D-def Sup-Red-F-unit)

moreover have Sup-Red-Inf-llist D = Red-Inf N by (simp add: D-def Sup-Red-Inf-unit)

end

end
ultimately have fair-D: reduc-fair D 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: \( B' \in \bot \) and \( B' \)-in: \( B' \in (\text{lnth } D \ i) \) and \( i < \text{llength } D \)
using reduc-dynamic-refutational-complete[of \( B \) \( D \)] bot-elem fair-D head-D saturated-N deriv-D refut-N
by auto
then have \( i = 0 \)
by (auto simp: D-def enat-0-iff)
show \( \exists B' \in \bot. B' \in (\text{lnth } D \ 0) \)
using reduc-static-refutational-complete unfolding \( i = 0 \): head-D[symmetric] by auto
qed

end

sublocale reduc-static-refutational-complete-calculus \( \subseteq \) reduc-dynamic-refutational-complete-calculus proof
fix \( B \) \( D \)
assume
\( \text{bot-elem: } (B \in \bot) \) and
\( \text{deriv: chain } (\Rightarrow \text{Red}) \) \( D \) and
\( \text{fair: } \text{reduc-fair } D \) and
\( \text{unsat: } (\text{lnth } D \ 0) \models \{ B \} \)
have non-empty: \( \neg \text{null } D \)
using chain-not-null[OF deriv]
have sub: \( (\text{lnth } D \ 0) \subseteq \text{Sup-llist } D \)
using lhd-subset-Sup-llist
have \( \text{Sup-llist } D \models \{ B \} \)
using unsat subset-entailed[OF sub]
then have Sup-no-Red: \( \text{Sup-llist } D \models \neg \text{Red-F }\) \( \text{Sup-llist } D \)
using deriv Red-in-Sup
have Liminf-entails-Bot: \( \text{Liminf-llist } D \models \{ B \} \)
using deriv fair reduc-fair-imp-Liminf-reduc-sat unfolding reduc-saturated-def
by auto
then have \( \exists B' \in \bot. B' \in (\text{Liminf-llist } D) \)
using bot-elem reduc-static-refutational-complete Liminf-entails-Bot
by auto
then show \( \exists i \in \{ i. \text{enat } i < \text{llength } D \}. \exists B' \in \bot. B' \in (\text{lnth } D \ i) \)
using Liminf-llist-def
by auto
qed

context calculus-with-red-crit

begin

proof
assume dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
then interpret dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
by simp
show static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
by (simp add: static-refutational-complete-calculus-axioms)
next
assume static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
then interpret static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by simp
show dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by (simp add: dynamic-refutational-complete-calculus-axioms)
qed

proof
assume reduc-dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
then interpret reduc-dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by simp
show reduc-static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by (simp add: reduc-static-refutational-complete-calculus-axioms)
next
assume reduc-static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
then interpret reduc-static-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by simp
show reduc-dynamic-refutational-complete-calculus Bot Inf entails Red-Inf Red-F
  by (simp add: reduc-dynamic-refutational-complete-calculus-axioms)
qed

interpretation reduc-calc : calculus-with-reduced-red-crit Bot Inf entails Red-Red-Inf Red-F
using reduc-calc by simp

  using dyn-equiv-stat stat-is-stat-red reduc-calc dyn-equiv-stat by meson


end

end
3 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
  Calculi
  Well-Quasi-Orders.Minimal-Elements
begin

3.1 Standard Lifting
locale standard-lifting =
  Non-ground: inference-system Inf-F +
  for
  Bot-F :: 'f set and
  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-Inf-G :: 'g inference set and
  Red-F-G :: 'g set ⇒ 'g set
+ fixes
  G-F :: 'f ⇒ 'g set and
  G-Inf :: '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-cond: G-F C ∩ Bot-G ≠ {} ⇒ C ∈ Bot-F and
  inf-map: ι ∈ Inf-F ⇒ G-Inf ι ≠ None ⇒ the (G-Inf ι) ⊆ Red-Inf-G (G-F (concl-of ι))
begin
abbreviation G-set :: 'f set ⇒ 'g set where
  ⟨G-set N≡⋃(G-F · N)⟩
lemma G-subset: ⟨N1 ⊆ N2 ⇒ G-set N1 ⊆ G-set N2⟩ by auto
definition entails-G :: 'f set ⇒ 'f set ⇒ bool (infix |=G 50) where
  ⟨N1 |=G N2 ≡ G-set N1 |=G G-set 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-implies-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 lifted-consequence-relation: consequence-relation
  where Bot=F and entails=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-implies-all[of - G-set N] subs-Bot-G-entails Bot-map-not-empty
    unfolding entails-G-def
    by auto
  qed
next
  fix N1 N2 :: ⟨'f set⟩
  assume ⟨N2 ⊆ N1⟩
  then show ⟨N1 |=G N2} using entails-G-def G-subset Ground.subset-entailed by auto
next
  fix N1 N2 N3
  assume ⟨N1 |=G N2} and ⟨N2 |=G N3⟩
  then show ⟨N1 |=G N3} using entails-G-def Ground.entails-trans by blast
  qed
end

3.2 Strong Standard Lifting

locale strong-standard-lifting = Non-ground: inference-system Inf-F +
for
  Bot-F :: ⟨'f set⟩ and
  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-Inf-G :: ⟨'g set ⇒ 'g inference set} and
  Red-F-G :: ⟨'g set ⇒ 'g set⟩
+ fixes
  G'-F :: ⟨'f ⇒ 'g set⟩ and
  G'-Inf :: ⟨'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: u ∈ Inf-F ⟹ G-Inf u ≠ None ⟹ concl-of ′ ⟨the (G-Inf u)⟩ ⊆ (G-F (concl-of ′ u))⟩
and
inf-map-in-Inf: \(a \in \Inf-F \implies \mathcal{G}-\Inf \ i \neq \text{None} \implies (\mathcal{G}-\Inf \ i) \subseteq \Inf-G\).

begin

sublocale standard-lifting
proof
  show \(\Bot-F \neq \{\}\ \text{using} \ \Bot-F\text{-not-empty}\).
next
  fix \(B\)
  assume \(b\text{-in}: B \in \Bot-F\)
  show \(\mathcal{G}\cdot F \ B \neq \{\}\ \text{using} \ \Bot-map\text{-not-empty}[OF \ b\text{-in}]\).
next
  show \(\bigwedge C.\ \mathcal{G}\cdot F \ C \cap \Bot-G \neq \{\} \implies C \in \Bot-F\ \text{using} \ \Bot-cond\).
next
  fix \(\i\)
  assume \(i\text{-in}: \i \in \Inf-F\) and \(\text{some-g}: \mathcal{G}\cdot\Inf \ i \neq \text{None}\)
  show the \((\mathcal{G}\cdot\Inf \ i) \subseteq \Red-\Inf-G\ (\mathcal{G}\cdot F (\text{concl-of} \ \i))\)
  proof
    fix \(\i G\)
    assume \(ig\text{-in1}: \i G \in \text{the} (\mathcal{G}\cdot\Inf \ i)\)
    then have \(ig\text{-in2}: \i G \in \Inf-G\ \text{using} \ \inf-map\text{-in-Inf}[OF \ i\text{-in} \ \text{some-g}]\ \text{by blast}\)
    show \(\i G \in \Red-\Inf-G\ (\mathcal{G}\cdot F (\text{concl-of} \ \i))\)
      using \(\text{strong-inf-map}[OF \ i\text{-in} \ \text{some-g}] \ \Ground.\Red-\Inf\text{-of-Inf-to-N}[OF \ ig\text{-in2}]\ \ i\text{-in1} \ \text{by blast}\)
  qed
qed

end

3.3 Lifting with a Family of Well-founded Orderings

locale lifting-with-wf-ordering-family = standard-lifting \(\Bot-F \ Inf-F \ Bot-G \ Inf-G \ entails-G \ Red-\Inf-G \ Red-F-G \ \mathcal{G}\cdot F \ \mathcal{G}\cdot\Inf\)
for
  \(\Bot-F :: (\'f set)\) and
  \(\Inf-F :: (\'f inference set)\) and
  \(\Bot-G :: (\'g set)\) and
  \(\entails-G :: (\'g set \Rightarrow (\'g inference set))\) and
  \(\Inf-G :: (\'g inference set)\) and
  \(\Red-\Inf-G :: (\'g set \Rightarrow (\'g inference set))\) and
  \(\Red-F-G :: (\'g set \Rightarrow (\'g set))\) and
  \(\mathcal{G}\cdot F :: (\'f \Rightarrow (\'g set))\) and
  \(\mathcal{G}\cdot\Inf :: (\'f inference \Rightarrow (\'g inference set \ \text{option}))\)
+ fixes
  \(\Prec-F-g :: (\'g \Rightarrow (\'f \Rightarrow \text{bool})\)
assumes
  all-wf: minimal-element (\(\Prec-F-g \ g\)) \ UNIV
begin

definition \(\Red-\Inf-G :: (\'f set \Rightarrow (\'f inference set))\) where
\(\Red-\Inf-G \ N = \{\i \in \Inf-F.\ \(\mathcal{G}\cdot\Inf \ i \neq \text{None} \land \text{the} (\mathcal{G}\cdot\Inf \ i) \subseteq \Red-\Inf-G\ (\mathcal{G}\cdot set \ N)\)\)
\(\lor (\mathcal{G}\cdot\Inf \ i = \text{None} \land \mathcal{G}\cdot F (\text{concl-of} \ i) \subseteq (\mathcal{G}\cdot set \ N \cup \Red-F-G (\mathcal{G}\cdot set \ N)))\})$
**definition** Red-F-G :: 'f set ⇒ 'f set where
\(\text{Red-F-G } N = \{ C. \forall D \in \mathcal{G}-F C. D \in \text{Red-F-G } (\mathcal{G}-\text{set } N) \lor (\exists E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E)\}\)

**lemma** Prec-trans:
**assumes**
\(\langle \text{Prec-F-g } D A B \rangle \) and \(\langle \text{Prec-F-g } D B C \rangle\)
**shows**
\(\langle \text{Prec-F-g } D A C \rangle\)
**using** minimal-element.po assms unfolding po-on-def transp-on-def by (smt UNIV-I alt-wf)

**lemma** prop-nested-in-set: \(D \in P C \implies C \in \{ C. \forall D \in P C. A D \lor B C D \} \implies A D \lor B C D\)
by blast

**lemma** Red-F-G-equiv-def:
\(\text{Red-F-G } N = \{ C. \forall D \in \mathcal{G}-F C. D \in \text{Red-F-G } (\mathcal{G}-\text{set } N) \lor (\exists E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E)\}\)
**proof** (rule;clarisimp)
**fix** \(C D\)
**assume**
\(C\)-in: \(\langle C \in \text{Red-F-G } N \rangle\) and
\(D\)-in: \(\langle D \in \mathcal{G}-F C \rangle\) and
not-sec-case: \(\langle \exists E \in N - \text{Red-F-G } N. \text{Prec-F-g } D E C \implies D \notin \mathcal{G}-F E\rangle\)
**have** C-in-unfolded: \(\langle C \in \{ C. \forall D \in \mathcal{G}-F C. D \in \text{Red-F-G } (\mathcal{G}-\text{set } N) \lor (\exists E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E)\rangle\)
using C-in unfolding Red-F-G-def .
**have** neg-not-sec-case: \(\langle \exists E \in N - \text{Red-F-G } N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E\rangle\)
using not-sec-case by clarsimp
**have** unfol-C-D: \(\langle D \in \text{Red-F-G } (\mathcal{G}-\text{set } N) \lor (\exists E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E)\rangle\)
using prop-nested-in-set[of D \(\mathcal{G}-F C\) \(\lambda x. x \in \text{Red-F-G } (\bigcup \{ \mathcal{G}-F \} )\) \(\lambda x. \exists E \in N. \text{Prec-F-g } y E x \land y \in \mathcal{G}-F E\), OF D-in C-in-unfolded] by blast
**show** \(\langle D \in \text{Red-F-G } (\mathcal{G}-\text{set } N)\rangle\)
**proof** (rule cocontr)
**assume** contraD: \(\langle D \notin \text{Red-F-G } (\mathcal{G}-\text{set } N)\rangle\)
**have** non-empty: \(\langle \exists E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E\rangle\)
using contra unfol-C-D by auto
**define** \(B\) where \(\langle B = \{ E \in N. \text{Prec-F-g } D E C \land D \in \mathcal{G}-F E\}\rangle\)
**then have** B-non-empty: \(\langle B \neq \{\} \rangle\)
using non-empty by auto
**interpret** minimal-element Prec-F-g D UNIV using all-wf[of D] .
**obtain** \(F::'f where\ F::'f = \text{min-elt } B\) by auto
**then have** D-in-F: \(\langle D \in \mathcal{G}-F F\rangle\)
**unfolding** B-def using non-empty
by (smt Sup-UNIV Sup-upper UNIV-I contra-subsetD empty-iff empty-subsetI mem-Collect-eq min-elt-map cofold-C-D)
**have** F-prec: \(\langle \text{Prec-F-g } D F C\rangle\)
using F min-elt-map[of B, OF - B-non-empty] unfolding B-def by auto
**have** F-not-in: \(\langle F \notin \text{Red-F-G } N\rangle\)
**proof**
**assume** F-in: \(\langle F \in \text{Red-F-G } N\rangle\)
**have** unfol-F-D: \(\langle D \in \text{Red-F-G } (\mathcal{G}-\text{set } N) \lor (\exists G \in N. \text{Prec-F-g } D G F \land D \in \mathcal{G}-F G)\rangle\)
using F-in D-in-F unfolding Red-F-G-def by auto
**then have** GMap: \(\langle \exists G \in N. \text{Prec-F-g } D G F \land D \in \mathcal{G}-F G\rangle\)
using contra D-in unfolding Red-F-G-def by auto
**then obtain** G where G-in: \(\langle G \in N\rangle\) and G-prec: \(\langle \text{Prec-F-g } D G F\rangle\) and G-map: \(\langle D \in \mathcal{G}-F G\rangle\)
by auto

have ⟨Prec-F-g D G C⟩ using G-prec F-prec Prec-trans by blast
then have ⟨G ∈ B⟩ unfolding B-def using G-in G-map by auto
then show False using F G-prec min-elt-minimal[of B G, OF - B-non-empty] by auto
qed

have ⟨F ∈ N⟩ using F by (metis B-def B-non-empty mem-Collect-eq min-elt-mem top-greatest)
then have ⟨F ∈ N − Red-F-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: (∀ D ∈ F G C. D ∈ Red-F-G (G-set N) ∨ (∃ E ∈ N − Red-F-G N. Prec-F-g D E C ∧ D ∈ G-F E))
show ⟨C ∈ Red-F-G N⟩ unfolding Red-F-G-set-def using only-if by auto
qed

proof
fix D
assume D-hyp: ⟨D ∈ G-set N − Red-F-G (G-set N)⟩
interpret minimal-element Prec-F-g D UNIV using all-wf[of D] .
have D-in-C: ⟨D ∈ G-set N⟩ using D-hyp by blast
have D-not-in-C: ⟨D /∈ Red-F-G (G-set N)⟩ using D-hyp by blast
have C-not-in: ⟨C /∈ Red-F-G N⟩
proof
assume C-in: ⟨C ∈ Red-F-G N⟩
show ⟨D ∈ Red-F-G (G-set N) ∨ (∃ E ∈ N. Prec-F-g D E C ∧ D ∈ G-F E)⟩ using C-in D-in-C unfolding Red-F-G-set-def by auto
then show False using D-not-in by simp
next
assume E ∉ N. Prec-F-g D E C ∧ D ∈ G-F E
then show False using C by (metis (no-types, lifting) B-def UNIV-I empty-iff mem-Collect-eq min-elt-minimal top-greatest)
qed
qed

show ⟨D ∈ G-set (N − Red-F-G N)⟩ using D-in-C C-not-in C-in-N by blast
qed

lemma Red-F-Bot-F: ⟨B ∈ Bot-F ⟹ N ⊨ G {B} ⟹ N − Red-F-G N ⊨ G {B}⟩
proof –
\[
\begin{align*}
\text{fix } B & \ N \\
\text{assume } & \quad B \in Bot-F \text{ and } \\
N \text{-entails: } & \quad \langle N \models G \{ B \} \rangle \\
\text{then have } & \quad \text{to-bot: } G \text{-set } N \to Red-F-G (G \text{-set } N) \models G F B; \\
\text{using } & \quad \text{Ground.Red-F-Bot Bot-map unfolding entails-G-def by (smt cSup-singleton Ground.entail-set-all-formulas image-insert image-is-empty subsetCE)} \\
\text{have } & \quad \text{from-f: } G \text{-set } (N - Red-F-G N) \models G \text{-set } N - Red-F-G (G \text{-set } N); \\
\text{using } & \quad \text{Ground.subset-entailed[of not-red-map-in-map-not-red] by blast} \\
\text{then have } & \quad G \text{-set } (N - Red-F-G N) \models G F B; \\
\text{using } & \quad \text{to-bot Ground.entails-trans by blast} \\
\text{then show } & \quad \langle N - Red-F-G N \models G \{ B \} \rangle; \\
\text{using } & \quad \text{Bot-map unfolding entails-G-def by simp qed} \\
\end{align*}
\]

**Lemma** Red-F-of-subset-F: \( \langle N \subseteq N' \Rightarrow Red-F-G N \subseteq Red-F-G N' \rangle \)

**using** Ground.Red-F-of-subset unfolding Red-F-G-def by (smt Collect-mono G-subset subset-iff)

**Lemma** Red-Inf-of-subset-F: \( \langle N \subseteq N' \Rightarrow Red-Inf-G N \subseteq Red-Inf-G N' \rangle \)


**Lemma** Red-F-of-Red-F-subset-F: \( \langle N' \subseteq Red-F-G N \Rightarrow Red-F-G N \subseteq Red-F-G (N - N') \rangle \)

**proof**

- **fix** \( N \ N' \ C \)
- **assume** \( N' \text{-in-Red-F-N}: \langle N' \subseteq Red-F-G N \rangle \text{ and } \)
  \( C \text{-in-Red-F-N}: \langle C \in Red-F-G N \rangle \)
- **have** \( \text{lem8}: \forall D \in G\text{-F C}. D \in Red-F-G (G \text{-set } N) \lor (\exists E \in (N - Red-F-G N). Prec-F-g D E C \land D \in G\text{-F E})\)
  **using** Red-F-G-equiv-def C-in-red-F-N by blast
- **show** \( \langle C \in Red-F-G (N - N') \rangle; \text{ unfolding Red-F-G-def} \)
  **proof** (rule,rule)
  - **fix \( D \in G\text{-F C} \),**
  - **then have** \( \langle D \in Red-F-G (G \text{-set } N) \lor (\exists E \in (N - Red-F-G N). Prec-F-g D E C \land D \in G\text{-F E}) \rangle; \)
    **using** lem8 by auto
  - **then show** \( \langle D \in Red-F-G (G \text{-set } (N - N')) \lor (\exists E \in (N - N'). Prec-F-g D E C \land D \in G\text{-F E}) \rangle; \)
    **proof** (rule,rule)
    - **assume** \( \langle D \in Red-F-G (G \text{-set } N) \rangle; \)
    - **then have** \( \langle D \in Red-F-G (G \text{-set } N - Red-F-G (G \text{-set } N)) \rangle; \)
      **using** Ground.Red-F-of-Red-F-subset[of Red-F-G (G \text{-set } N) G \text{-set } N] by auto
    - **then have** \( \langle D \in Red-F-G (G \text{-set } (N - Red-F-G N)) \rangle; \)
      **using** Ground.Red-F-of-subset[[OF not-red-map-in-map-not-red][of N]] by auto
    - **then have** \( \langle D \in Red-F-G (G \text{-set } (N - N')) \rangle; \)
      **using** N'-in-Red-F-G G-subset[of N - Red-F-G N N - N'] by (smt DiffE DiffI Ground.Red-F-of-subset subsetCE subsetI)
    - **then show** \( \text{thesis by blast} \)
  - **next**
    - **assume** \( \exists E \in (N - Red-F-G N). Prec-F-g D E C \land D \in G\text{-F E} \)
    - **then obtain \( E \) where**
      \( E\text{-in: } \langle E \in (N - Red-F-G N) \rangle \text{ and } \)
      \( E\text{-prec-C: } \langle Prec-F-g D E C \rangle \text{ and } \)
      \( D\text{-in: } \langle D \in G\text{-F E} \rangle \)

26
proof

\begin{enumerate}
\item \textbf{assumption:} \(N \in \text{Red-Inf-N}\) and \(i \in \text{Red-Inf-N}\)
\item \textbf{have:} \(\forall i \in i \in \text{Red-Inf-N}\) unfolding \(\text{Red-Inf-G-def}\) by \textbf{blasting}
\begin{enumerate}
\item \textbf{assume not-none:} \(\text{Red-Inf-G-def}\) by \textbf{auto}
\item \textbf{then show:} \(\text{Red-Inf-G-def}\) by \textbf{auto}
\end{enumerate}
\item \textbf{moreover:} \(\forall i \in i \in \text{Red-Inf-N}\) unfolding \(\text{Red-Inf-G-def}\) by \textbf{blasting}
\begin{enumerate}
\item \textbf{have:} \(\text{Red-Inf-G-def}\) by \textbf{auto}
\item \textbf{then show:} \(\text{Red-Inf-G-def}\) by \textbf{auto}
\end{enumerate}
\item \textbf{ultimately show:} \(\forall i \in i \in \text{Red-Inf-N}\) unfolding \(\text{Red-Inf-G-def}\) by \textbf{auto}
\end{enumerate}
lemma Red-Inf-of-Inf-to-N-F:
  assumes
  i-in: u ∈ Inf-F and
  concl-i-in: ⟨concl-of i ∈ N⟩
  shows
  ⟨i ∈ Red-Inf-G N⟩
proof
  have ⟨i ∈ Inf-F ⟹ G-inf i ≠ None ⟹ the (G-inf i) ⊆ Red-Inf-G (G-F ⟨concl-of i⟩)⟩ using inf-map
  by simp
  moreover have ⟨Red-Inf-G (G-F ⟨concl-of i⟩) ⊆ Red-Inf-G (G-set N)⟩ using concl-i-in Ground.Red-Inf-of-subset-F by blast
  moreover have i ∈ Inf-F ⟹ G-inf i = None ⟹ concl-of i ∈ N ⟹ G-F ⟨concl-of i⟩ ⊆ G-set N by blast
  ultimately show ?thesis using i-in concl-i-in unfolding Red-Inf-G-def by auto
qed

sublocale lifted-calculus-with-red-crit: calculus-with-red-crit
  where
  Bot = Bot-F and Inf = Inf-F and entails = entails-G and
  Red-Inf = Red-Inf-G and Red-F = Red-F-G
proof
  fix B N N' i
  show ⟨Red-Inf-G N ⊆ Inf-F⟩ unfolding Red-Inf-G-def by blast
  show ⟨B ∈ Bot-F ⟹ N |=G {B} ⟹ N - Red-F-G N |=G {B}⟩ using Red-F-Bot-F by simp
  show ⟨N ⊆ N' ⟹ Red-F-G N ⊆ Red-F-G N'⟩ using Red-F-of-subset-F by simp
  show ⟨N ⊆ N' ⟹ Red-Inf-G N ⊆ Red-Inf-G N'⟩ using Red-Inf-of-subset-F by simp
  show ⟨N' ⊆ Red-F-G N ⟹ Red-F-G N ⊆ Red-F-G (N - N')⟩ using Red-F-of-Red-F-subset-F by simp
  show ⟨N' ⊆ Red-F-G N ⟹ Red-Inf-G N ⊆ Red-Inf-G (N - N')⟩ using Red-Inf-of-Red-F-subset-F by simp
  show ⟨i ∈ Inf-F ⟹ concl-of i ∈ N ⟹ i ∈ Red-Inf-G N⟩ using Red-Inf-of-Inf-to-N-F by simp
qed

lemma lifted-calc-is-calc: calculus-with-red-crit Bot-F Inf-F entails-G Red-Inf-G Red-F-G

lemma grounded-inf-in-ground-inf: i ∈ Inf-F ⟹ G-inf i ≠ None ⟹ the (G-inf i) ⊆ Inf-G
  using inf-map Ground.Red-Inf-to-Inf by blast

  ⟨{i. ∃ i' Non-ground.Inf-from N. G-inf i' ≠ None ∧ i ∈ the (G-inf i')} ∪ Red-Inf-G (G-set N)}⟩ ⟹
  Ground.saturated (G-set N)
proof
  fix N
  assume
  ⟨sat-n: lifted-calc-is-calc-with-red-crit.saturated N and
  inf-grounded-in: Ground.Inf-from (G-set N) ⊆
  ⟨{i. ∃ i' Non-ground.Inf-from N. G-inf i' ≠ None ∧ i ∈ the (G-inf i')} ∪ Red-Inf-G (G-set N)}⟩ ⟹
  Ground.saturated (G-set N)}⟩ unfolding Ground.saturated-def
proof
  fix i
  assume i-in: i ∈ Ground.Inf-from (G-set N)
  {
assume $\iota \in \{ \iota. \exists \iota' \in \text{Non-ground.} \text{Inf-from N. G-Inf } \iota' \neq \text{None } \wedge \iota \in \text{the (G-Inf } \iota') \}$
then obtain $\iota'$ where $\iota' \in \text{Non-ground.} \times \text{Inf-from N. G-Inf } \iota' \neq \text{None } \iota \in \text{the (G-Inf } \iota')$ by blast
then have $\iota \in \text{Red-Inf-G (G-set N)}$
  using Red-Inf-G-def sat-n unfolding lifted-calculus-with-red-crit.saturated-def by auto
} then show $\iota \in \text{Red-Inf-G (G-set N)}$ using inf-grounded-in i-in by blast
qed

**theorem** stat-ref-comp-to-non-ground:

assumes

stat-ref-G: static-refutational-complete-calculus Bot-G Inf-G entails-G Red-Inf-G Red-F-G and

sat-n-imp: $\bigwedge N. (\text{lifted-calculus-with-red-crit.saturated N } \Rightarrow \text{Ground.} \text{Inf-from (G-set N)}) \subseteq (\{ \iota. \exists \iota' \in \text{Non-ground.} \text{Inf-from N. G-Inf } \iota' \neq \text{None } \wedge \iota \in \text{the (G-Inf } \iota') \} \cup \text{Red-Inf-G (G-set N)})$

shows

static-refutational-complete-calculus Bot-F Inf-F entails-G Red-Inf-G Red-F-G

**proof**

fix $B N$

assume

$\text{b-in: } B \in \text{Bot-F and}$

$\text{sat-n: lifted-calculus-with-red-crit.saturated N and}$

$n\text{-entails-bot: } N \models G \{B\}$

have $\text{ground-n-entails: G-set N } \models G F B$

using $n\text{-entails-bot unfolding entails-G-def by simp}$

then obtain $BG$ where $bg\text{-in1: } BG \in G F B$

using Bot-map-not-empty[OF b-in] by blast

then have $\text{bg\text{-in: } BG } \in \text{Bot-G}$

using Bot-map[OF b-in] by blast

have $\text{ground-n-entails-bot: G-set N } \models G \{BG\}$

using $\text{ground-n-entails bg\text{-in1 Ground.} entail\text{-set-all-formulas by blast}$

have $\text{Ground. Inf-from (G-set N) } \subseteq (\{ \iota. \exists \iota' \in \text{Non-ground.} \text{Inf-from N. G-Inf } \iota' \neq \text{None } \wedge \iota \in \text{the (G-Inf } \iota') \} \cup \text{Red-Inf-G (G-set N)})$

using $\text{sat-n-imp[OF sat-n]}$.

have $\text{Ground. saturated (G-set N)}$

using $\text{sat-imp-ground-sat[OF sat-n sat-n-imp[OF sat-n]}$.

then have $\exists BG' \in \text{Bot-F. } BG' \in (G-set N)$

using $\text{stat-ref-G} \text{ Ground. calculus-with-red-crit-axioms bg\text{-in ground-n-entails-bot}$

unfolding static-refutational-complete-calculus-def static-refutational-complete-calculus-axioms-def by blast

then show $\exists B' \in \text{Bot-F. } B' \in N$

using $\text{bg\text{-in Bot-cond Bot-map-not-empty Bot-cond by blast}$

qed

end

**abbreviation** Empty-Order where

Empty-Order $C1 C2 \equiv \text{False}$

**lemma** any-to-empty-order-lifting:


$G-\text{Inf Prec-F-g } \Rightarrow \text{lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-G Inf-G Red-Inf-G Red-F-G G-F}$

proof –

fix $\text{Bot-F Inf-F Bot-G entails-G Inf-G Red-Inf-G Red-F-G G-F G-Inf Prec-F-g}$

29
Red-F-G G-F G-Inf Prec-F-g
then interpret lift-g:
lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-G Inf-G Red-Inf-G Red-F-G G-F G-Inf Prec-F-g
by auto
have empty-wf: minimal-element ((λg. Empty-Order) g) UNIV
by (simp add: lift-g.all-wf minimal-element.intro po-on-def transp-on-def wfp-on-def wfp-on-imp-irrefl-on)
by (simp add: empty-wf lift-g.standard-lifting-axioms
lifting-with-wf-ordering-family-axioms.intro lifting-with-wf-ordering-family-def)
qed

locale lifting-equivalence-with-empty-order
for
G-F :: (f ⇒ 'g set) and
G-Inf :: (f inference ⇒ 'g inference set option) and
Bot-F :: (f set) and
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-Inf-G :: (g set ⇒ 'g inference set) and
Red-F-G :: (g set ⇒ 'g set) and
Prec-F-g :: (f ⇒ 'f ⇒ 'f ⇒ bool)

sublocale lifting-with-wf-ordering-family ⊆ lifting-equivalence-with-empty-order
proof
show po-on Empty-Order UNIV
unfolding po-on-def by (simp add: transp-onI wfp-on-imp-irrefl-on)
show wfp-on Empty-Order UNIV
unfolding wfp-on-def by simp
qed

context lifting-equivalence-with-empty-order
begin

lemma saturated-empty-order-equiv-saturated:
any-order-lifting.lifted-calculus-with-red-crit.saturated N =
empty-order-lifting.lifted-calculus-with-red-crit.saturated N by standard

lemma static-empty-order-equiv-static:
static-refutational-complete-calculus Bot-F Inf-F
static-refutational-complete-calculus Bot-F Inf-F any-order-lifting.entails-G
any-order-lifting.Red-Inf-G any-order-lifting.Red-F-G
unfolding static-refutational-complete-calculus-def
by (rule iffI) (standard, (standard)[], simp)+

theorem static-to-dynamic:
  static-refutational-complete-calculus Bot-F Inf-F
  any-order-lifting, entails-G empty-order-lifting, Red-Inf-G empty-order-lifting, Red-F-G =
  dynamic-refutational-complete-calculus Bot-F Inf-F
  any-order-lifting, entails-G any-order-lifting, Red-Inf-G any-order-lifting, Red-F-G
(is ?static=?dynamic)
proof
  assume ?static
  then have static-general:
    static-refutational-complete-calculus Bot-F Inf-F any-order-lifting, entails-G
    any-order-lifting, Red-Inf-G any-order-lifting, Red-F-G (is ?static-gen)
    using static-empty-order-equiv-static by simp
  interpret static-refutational-complete-calculus Bot-F Inf-F any-order-lifting, entails-G
  any-order-lifting, Red-Inf-G any-order-lifting, Red-F-G
  using static-general .
  show ?dynamic by standard
next
  assume dynamic-gen: ?dynamic
  interpret dynamic-refutational-complete-calculus Bot-F Inf-F any-order-lifting, entails-G
  any-order-lifting, Red-Inf-G any-order-lifting, Red-F-G
  using dynamic-gen .
  have static-refutational-complete-calculus Bot-F Inf-F any-order-lifting, entails-G
  any-order-lifting, Red-Inf-G any-order-lifting, Red-F-G
  by standard
  then show ?static using static-empty-order-equiv-static by simp
qed

3.4 Lifting with a Family of Redundancy Criteria

locale standard-lifting-with-red-crit-family =
  Non-ground: inference-system Inf-F
  + Ground-family: calculus-with-red-crit-family Bot-G Inf-G Q entails-q Red-Inf-q Red-F-q
for
  Inf-F :: 'f inference set and
  Bot-G :: 'g set and
  Inf-G :: 'g inference set and
  Q :: 'q itself and
  entails-q :: 'q ⇒ ('g set ⇒ 'g set ⇒ bool) and
  Red-Inf-q :: 'q ⇒ ('g set ⇒ 'g inference set) and
  Red-F-q :: 'q ⇒ ('g set ⇒ 'g set)
+ fixes
  Bot-F :: 'f set and
  G-F-q :: 'q ⇒ 'f ⇒ 'g set and
  G-Inf-q :: 'q ⇒ 'f inference ⇒ 'g inference set option and
  Prec-F-q :: 'g ⇒ 'f ⇒ 'f ⇒ bool
assumes
  standard-lifting-family: lifting-with-uf-ordering-family Bot-F Inf-F Bot-G (entails-q q)
  Inf-G (Red-Inf-q q) (Red-F-q q) (G-F-q q) (G-Inf-q q) Prec-F-q
begin

definition G-set-q :: 'q ⇒ 'f set ⇒ 'g set where
  G-set-q q N ≡ ∪ (G-F-q q ' N)
definition Red-Inf-\mathcal{G}-q :: 'q \Rightarrow 'q set \Rightarrow 'q inference set where
\begin{align*}
\text{Red-Inf-}\mathcal{G}-q \ N = \{ t \in \text{Inf-F}. (\mathcal{G}-\text{Inf-} q \ t \neq \text{None} \land \text{the} (\mathcal{G}-\text{Inf-} q \ t) \subseteq \text{Red-Inf-} q \ (\mathcal{G}-\text{set-} q \ N) \} \\
\lor (\mathcal{G}-\text{Inf-} q \ t = \text{None} \land \text{F-} q \ t \quad \text{conc-of-} t \subseteq (\mathcal{G}-\text{set-} q \ N \cup \text{Red-} F\ q \ (\mathcal{G}-\text{set-} q \ N)))
\end{align*}

definition Red-Inf-\mathcal{G}-Q :: 'q set \Rightarrow 'q inference set where
\text{Red-Inf-}\mathcal{G}-Q \ N = \cap \{ X \ N \mid X \in (\text{Red-Inf-}\mathcal{G}-q \cdot \text{UNIV}) \}

definition Red-F-\mathcal{G}\cdot\text{-empty-q} :: 'q \Rightarrow 'q set \Rightarrow 'q set where
\text{Red-F-}\mathcal{G}\cdot\text{-empty-q} \ N = \{ C. \forall D \in \mathcal{G}\cdot F\ q \ C. D \in \text{Red-} F\ q \ (\mathcal{G}-\text{set-} q \ N) \} \\
\lor \exists E \in N. \text{Empty-Order E C} \land D \in \mathcal{G}\cdot F\ q \ E)

definition Red-F-\mathcal{G}\cdot\text{-empty} :: 'q set \Rightarrow 'q set where
\text{Red-F-}\mathcal{G}\cdot\text{-empty} \ N = \cap \{ X \ N \mid X \in (\text{Red-F-}\mathcal{G}\cdot\text{-empty-q} \cdot \text{UNIV}) \}

definition Red-F-\mathcal{G}\cdot q :: 'q \Rightarrow 'q set \Rightarrow 'q set where
\text{Red-F-}\mathcal{G}\cdot q \ N = \{ C. \forall D \in \mathcal{G}\cdot F\ q \ C. D \in \text{Red-} F\ q \ (\mathcal{G}-\text{set-} q \ N) \} \\
\lor \exists E \in N. \text{Prec-} F\ q D E C \land D \in \mathcal{G}\cdot F\ q \ E)

definition entails-\mathcal{G}\cdot q :: 'q \Rightarrow 'q set \Rightarrow bool where
entails-\mathcal{G}\cdot q \ N_1 N_2 \equiv \text{entails-q q (}\mathcal{G}\cdot\text{-set-q q N1}) \ (\mathcal{G}\cdot\text{-set-q q N2})

definition entails-\mathcal{G}\cdot Q :: 'q set \Rightarrow 'q set \Rightarrow bool (\text{infix} \models 50) where
entails-\mathcal{G}\cdot Q \ N_1 N_2 \equiv \forall q. \text{entails-} \mathcal{G}\cdot q q N_1 N_2

lemma red-crit-lifting-family:
\text{calculus-with-red-crit Bot-F Inf-F (entails-}\mathcal{G}\cdot q q) \ (\text{Red-Inf-}\mathcal{G}\cdot q q) \ (\text{Red-F-}\mathcal{G}\cdot q q)

proof
- fix q
  interpret \text{wf-lift}:
    \text{lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-q q Inf-G Red-Inf-q q Red-F-q q}
    \mathcal{G}\cdot F\ q q \mathcal{G}\cdot\text{-Inf-q q} \mathcal{G}\cdot\text{-set-q q Prec-F-q}
    using \text{standard-lifting-family} .
  have \text{entails-} \mathcal{G}\cdot q q = \text{wf-lift.} \text{entails-} \mathcal{G}
    unfolding \text{entails-} \mathcal{G}\cdot\text{-def wf-lift.} \text{entails-} \mathcal{G}\cdot\text{-def} \mathcal{G}\cdot\text{-set-q-def} \text{ by blast}
  moreover have \text{Red-Inf-} \mathcal{G}\cdot q q = \text{wf-lift.} \text{Red-Inf-} \mathcal{G}
    unfolding \text{Red-Inf-} \mathcal{G}\cdot q q \mathcal{G}\cdot\text{-set-q-def} \text{ wf-lift.} \text{Red-Inf-} \mathcal{G}\cdot\text{-def} \text{ by blast}
  moreover have \text{Red-F-} \mathcal{G}\cdot q q q = \text{wf-lift.} \text{Red-F-} \mathcal{G}
    unfolding \text{Red-F-} \mathcal{G}\cdot q q \mathcal{G}\cdot\text{-set-q-def} \text{ wf-lift.} \text{Red-F-} \mathcal{G}\cdot\text{-def} \text{ by blast}
  ultimately show \text{calculus-with-red-crit Bot-F Inf-F (entails-} \mathcal{G}\cdot q q) \ (\text{Red-Inf-}\mathcal{G}\cdot q q) \ (\text{Red-F-}\mathcal{G}\cdot q q)
    using \text{wf-lift.} \text{lifted-calculus-with-red-crit.} \text{calculus-with-red-crit-axioms by simp}
qed

lemma red-crit-lifting-family-empty-ord:
\text{calculus-with-red-crit Bot-F Inf-F (entails-} \mathcal{G}\cdot q q) \ (\text{Red-Inf-}\mathcal{G}\cdot q q) \ (\text{Red-F-}\mathcal{G}\cdot\text{-empty-q q})

proof
- fix q
  interpret \text{wf-lift}:
    \text{lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-q q Inf-G Red-Inf-q q Red-F-q q}
    \mathcal{G}\cdot F\ q q \mathcal{G}\cdot\text{-Inf-q q} \mathcal{G}\cdot\text{-set-q q Prec-F-q}
    using \text{standard-lifting-family} .
  have \text{entails-} \mathcal{G}\cdot q q = \text{wf-lift.} \text{entails-} \mathcal{G}

32
unfolding entails-G-q-def wf-lift.entails-G-def G-set-q-def by blast
moreover have Red-Inf-G-q q = wf-lift.Red-Inf-G
moreover have Red-Inf-G-q-def G-set-q-def wf-lift.Red-Inf-G-def by blast
unfolding Red-F-G-empty-q-def G-set-q-def by blast
ultimately show calculus-with-red-crit Bot-F Inf-F (entails-G-q q) (Red-Inf-G-q q) (Red-F-G-empty-q q)
  by simp
qed

lemma cons-rel-fam-Q-lem: (consequence-relation-family Bot-F entails-G-q)

proof
  show Bot-F \neq \{\}
  using standard-lifting-family
  by (meson ex-in-conv lifting-with-wf-ordering-family.axioms(1) standard-lifting.Bot-F-not-empty)
next
fix qi
  show Bot-F \neq \{\}
  using standard-lifting-family
  by (meson ex-in-conv lifting-with-wf-ordering-family.axioms(1) standard-lifting.Bot-F-not-empty)
next
fix qi B N1
  assume
  B-in: B \in Bot-F
interpret lift: lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-q qi Inf-G Red-Inf-q qi Red-F-q qi G-F-q qi G-Inf-q qi Prec-F-g
  by (rule standard-lifting-family)
have (entails-G-q qi) = lift.entails-G
  unfolding entails-G-q-def lift.entails-G-def G-set-q-def by simp
then show entails-G-q qi \{B\} N1
  using B-in lift.lifted-consequence-relation.bot.implies.all by auto
next
fix qi and N2 N1::'f set
  assume
  N-incl: N2 \subseteq N1
interpret lift: lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-q qi Inf-G Red-Inf-q qi Red-F-q qi G-F-q qi G-Inf-q qi Prec-F-g
  by (rule standard-lifting-family)
have (entails-G-q qi) = lift.entails-G
  unfolding entails-G-q-def lift.entails-G-def G-set-q-def by simp
then show entails-G-q qi N1 N2
  using N-incl by (simp add: lift.lifted-consequence-relation.subset.entailed)
next
fix qi N1 N2
  assume
  all-C: \forall C \in N2. entails-G-q qi N1 \{C\}
interpret lift: lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-q qi Inf-G Red-Inf-q qi Red-F-q qi G-F-q qi G-Inf-q qi Prec-F-g
  by (rule standard-lifting-family)
have (entails-G-q qi) = lift.entails-G
  unfolding entails-G-q-def lift.entails-G-def G-set-q-def by simp
then show entails-G-q qi N1 N2
  using all-C lift.lifted-consequence-relation.all.formulas.entailed by presburger
next
fix \( qi \) \( N1 \) \( N2 \) \( N3 \)

assume

\( \text{entails}^{12}: \text{entails}-G-q \) \( qi \) \( N1 \) \( N2 \) \( N3 \)

interpret \( \text{lift}: \text{lifting-with-wf-ordering-family} \) \( \text{Bot-F Inf-F Bot-G} \) \( \text{entails-q} \) \( \text{Inf-G} \) \( \text{Red-Inf-q} \) \( \text{qi} \) \( \text{Red-F-q} \) \( \text{qi} \) \( \text{G-Inf-q} \) \( \text{qi} \) \( \text{Prec-F-g} \)

by (rule standard-lifting-family)

have \( (\text{entails-G-q} \) \( qi) = \text{lift.entails-G} \)

unfolding \( \text{entails-G-q-def lift.entails-G-def g-set-q-def} \) by simp

then show \( \text{entails-G-q} \) \( qi \) \( N1 \) \( N3 \)

using \( \text{entails12} \) \( \text{entails23} \) \( \text{lift.lifted-consequence-relation.entails-trans} \) by presburger

qed

interpretation \( \text{cons-rel-Q}: \text{consequence-relation-family} \) \( \text{Bot-F} \) \( \text{entails-G-Q} \)

proof --

interpret \( \text{cons-rel-fam}: \text{consequence-relation-family} \) \( \text{Bot-F Q} \) \( \text{entails-G-q} \)

by (rule cons-rel-fam-Q-lem)

have \( \text{consequence-relation-family.entails-Q} \) \( \text{entails-G-q = entails-G-Q} \)

unfolding \( \text{entails-G-Q-def cons-rel-fam.entails-Q-def} \) by (simp add: entails-G-q-def)

then show \( \text{consequence-relation Bot-F entails-G-Q} \)

using \( \text{consequence-relation-family.intersect-cons-rel-family[OF cons-rel-fam-Q-lem]} \) by simp

qed

sublocale \( \text{lifted-calc-w-red-crit-family}: \text{calculus-with-red-crit-family} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-q Red-Inf-G-Q Red-F-G-g} \)

using \( \text{cons-rel-fam-Q-lem red-crit-lifting-family} \)

by (simp add: calculus-with-red-crit-family.intro calculus-with-red-crit-family-axioms-def)

lemma \( \text{lifted-calc-family-is-calc}: \text{calculus-with-red-crit} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-Q Red-Inf-G-Q Red-F-G-g} \)

proof --

have \( \text{lifted-calc-w-red-crit-family.entails-Q} \) \( \text{entails-G-Q} \)

unfolding \( \text{entails-G-Q-def lifted-calc-w-red-crit-family.entails-Q-def} \) by simp

moreover have \( \text{lifted-calc-w-red-crit-family.Red-Inf-Q = Red-Inf-G-Q} \)

moreover have \( \text{lifted-calc-w-red-crit-family.Red-F-Q = Red-F-G-g} \)

moreover have \( \text{lifted-calc-w-red-crit-family.Red-F-G-Q-def lifted-calc-w-red-crit-family.Red-F-G-Q-def} \) by simp

ultimately show \( \text{calculus-with-red-crit} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-Q Red-Inf-G-Q Red-F-G-g} \)

using \( \text{lifted-calc-w-red-crit-family.inter-red-crit} \) by simp

qed

sublocale \( \text{empty-ord-lifted-calc-w-red-crit-family}: \text{calculus-with-red-crit-family} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-Q Red-Inf-G-Q Red-F-G-empty} \)

using \( \text{cons-rel-fam-Q-lem red-crit-lifting-family-empty-ord} \)

by (simp add: calculus-with-red-crit-family.intro calculus-with-red-crit-family-axioms-def)

lemma \( \text{inter-calc}: \text{calculus-with-red-crit} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-Q Red-Inf-G-Q Red-F-G-empty} \)

proof --

have \( \text{lifted-calc-w-red-crit-family.entails-Q} \) \( \text{entails-G-Q} \)

unfolding \( \text{entails-G-Q-def lifted-calc-w-red-crit-family.entails-Q-def} \) by simp

moreover have \( \text{empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q = Red-Inf-G-Q} \)

moreover have \( \text{empty-ord-lifted-calc-w-red-crit-family.Red-F-Q = Red-F-G-empty} \)

moreover have \( \text{empty-ord-lifted-calc-w-red-crit-family.Red-F-G-empty-def empty-ord-lifted-calc-w-red-crit-family.Red-F-G-empty} \) by simp

ultimately show \( \text{calculus-with-red-crit} \) \( \text{Bot-F Inf-F Q} \) \( \text{entails-G-Q Red-Inf-G-Q Red-F-G-empty} \)

using \( \text{empty-ord-lifted-calc-w-red-crit-family.inter-red-crit} \) by simp

qed

34
theorem stat-ref-comp-to-non-ground-fam-inter:
  assumes
  stat-ref-G: \( \forall q. \text{static-refutational-complete-calculus} \text{Bot-G Inf-G} (\text{entails-}q \text{ G}) (\text{Red-Inf-}q \text{ G}) (\text{Red-F-}q \text{ G}) \)
  and
  sat-n-imp: \( \forall N. (\text{empty-ord-lifted-calculc-w-red-crit-family} \text{. inter-red-crit-calculus}. \text{saturated} \text{ N} \implies \exists q. \text{Ground-family}. \text{Inf-from} \text{ (G-set-q} q \text{ N)} \subseteq \{\text{\{t. } \exists t' \in \text{Non-ground}. \text{Inf-from} \text{ N. G-Inf-q q t'} \neq \text{None} \land r \in \text{the} \text{ (G-Inf-q q t')}} \} \cup \text{Red-Inf-q q G-set-q q N}\))
  shows
  \( \text{static-refutational-complete-calculus} \text{Bot-F Inf-F} \text{ entails-G Q Red-Inf-G Q Red-F-G empty} \)
  using inter-calc
  unfolding static-refutational-complete-calculus-def static-refutational-complete-calculus-axioms-def
proof
  (standard, clarify)
  fix B N
  assume
  b-in: \( B \in \text{Bot-F} \) and
  sat-n: \( \text{calculus-with-red-crit-saturated Inf-F Red-Inf-G-Q N} \) and
  entails-bot: \( N = \cap \{B\} \)
  interpret calculus-with-red-crit Bot-F Inf-F entails-G-Q Red-Inf-G-Q Red-F-G empty
  using inter-calc by blast
  have empty-ord-lifted-calculc-w-red-crit-family.Red-Inf-Q = Red-Inf-G-Q
  unfolding Red-Inf-G-Q-def lifted-calculc-w-red-crit-family.Red-Inf-Q-def by simp
  then have empty-ord-sat-n: \( \text{empty-ord-lifted-calculc-w-red-crit-family}. \text{inter-red-crit-calculus}. \text{saturated} \text{ N} \)
  using sat-n
  unfolding saturated-def empty-ord-lifted-calculc-w-red-crit-family.inter-red-crit-calculus.saturated-def
  by simp
  then obtain q where inf-subs: \( \text{Ground-family}. \text{Inf-from} \text{ (G-set-q} q \text{ N)} \subseteq \{\text{\{t. } \exists t' \in \text{Non-ground}. \text{Inf-from} \text{ N. G-Inf-q q t'} \neq \text{None} \land r \in \text{the} \text{ (G-Inf-q q t')}} \} \cup \text{Red-Inf-q q G-set-q q N}\)
  using sat-n-imp[of N] by blast
  interpret q-calc: calculus-with-red-crit Bot-F Inf-F entail-G-q q Red-Inf-G-q q Red-F-G-q q
  using lifted-calc-w-red-crit-family.all-red-crit[of q]
  have n-q-sat: \( q\text{.calc}.saturated \text{ N} \) using lifted-calc-w-red-crit-family.sat-int-to-sat-q empty-ord-sat-n by simp
  interpret lifted-q-calc: lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entail-q q Inf-G Red-Inf-q q Red-F-q q G-F-q q G-Inf-q q
  by (simp add: standard-lifting-family)
  have lifted-q-calc.empty-order-lifting.lifted-calculus-with-red-crit.saturated N
  lifted-q-calc.lifted-calculus-with-red-crit.saturated-def q-calc.saturated-def by auto
  then have ground-sat-n: \( \text{lifted-q-calc}. \text{Ground}. \text{saturated} \text{ (G-set-q q N)} \)
  using lifted-q-calc.sat-imp-ground-sat[of N] inf-subs unfolding G-set-q-def by blast
  have entail-G-q q N \( \{B\} \) using entail-G-q q Red-Inf-G-q q Red-F-q q
  unfolding entail-G-Q-def by simp
  then have ground-n-entails-bot: entail-q q \( G-set-q q N \) \( \{B\} \) unfolding entails-G-Q-def .
  interpret static-refutational-complete-calculus Bot-G Inf-G entail-q q Red-Inf-q q Red-F-q q
  using stat-ref-G[of q]
  .
  obtain BG where bg-in: \( BG \in G-F-q q B \)
  using lifted-q-calc.Bot-map-not-empty[OF b-in] by blast
  then have BG \( \in \text{Bot-G} \) using lifted-q-calc.Bot-map[of b-in] by blast
  then have \( \exists BG' \in G-set-q q N \)
  using ground-sat-n ground-n-entails-bot static-refutational-complete[of BG, OF - ground-sat-n]
by simp
then show \( \exists B' \in \text{Bot-F}. B' \in N \) using \( \text{lifted-q-calc.\text{Bot-cond unfolding G-set-q-def by blast} \text{ qed}} \)

lemma \( \text{sat-eq-sat-empty-order: lifted-calc-w-red-crit-family.\text{inter-red-crit-calculus.saturated N = empty-ord-lifted-calc-w-red-crit-family.\text{inter-red-crit-calculus.saturated N}} \) by simp


assume ?static

next
assume dynamic-gen: ?dynamic
then show ?static using \( \text{static-empty-ord-inter-equiv-static-inter by simp \) qed

end

end

4 Labeled Liftings

This section formalizes the extension of the lifting results to labeled calculi. This corresponds to section 3.4 of the report.
4.1 Labeled Lifting with a Family of Well-founded Orderings

locale labeled-lifting-w-wf-ord-family =
lifting-with-wf-ordering-family Bot-F Inf-F Bot-G entails-G Inf-G Red-Inf-G Red-F-G G-F Inf-F g entail G entail G entail G entail \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 ⊨ G 50) and
Inf-G :: 'g inference set and
Red-Inf-G :: 'g set ⇒ 'g inference set and
Red-F-G :: 'g set ⇒ 'g set and
G-F :: 'f ⇒ 'g set and
G-Inf :: 'f inference ⇒ 'g inference set option and
\Prec-F :: 'g ⇒ 'f ⇒ 'f ⇒ bool (infix ▷ 50)
\fixes
l :: 'l itself and
Inf-FL :: ⟨'f × 'l⟩ inference set
\assumes
Inf-F-to-Inf-FL :: \ιF ∈ Inf-F =⇒ \length (Ll :: 'l list) = \length (prems-of \ιF) =⇒
\exists L0. \Infer (\map \fst (\prems-of \ιF), L0) ∈ Inf-FL and
Inf-FL-to-Inf-F :: \ιFL ∈ Inf-FL =⇒ \Infer (\map \fst (\prems-of \ιFL)) (\fst (\concl-of \ιFL)) ∈ Inf-F
begin
\definition to-F :: ⟨'f × 'l⟩ inference ⇒ 'f inference where
\to-F \ιFL = \Infer (\map \fst (\prems-of \ιFL)) (\fst (\concl-of \ιFL))
\definition Bot-FL :: ⟨'f × 'l⟩ set where \Bot-FL = Bot-F × UNIV
\definition G-F-L :: ⟨'f × 'l⟩ ⇒ 'g set where \G-F-L CL = G-F (\fst CL)
\definition G-Inf-L :: ⟨'f × 'l⟩ inference ⇒ 'g inference set option where \G-Inf-L \ιFL = G-Inf (\to-F \ιFL)
\sublocale labeled-standard-lifting: standard-lifting where
\Bot-F = Bot-FL and
\Inf-F = Inf-FL and
\G-F = G-F-L and
\G-Inf = G-Inf-L
\proof
\show \Bot-FL ≠ {} unfolding Bot-FL-def using Bot-F-not-empty by simp
next
\show B ∈ Bot-FL =⇒ \G-F-L B ≠ {} for B unfolding G-F-L-def Bot-FL-def using Bot-map-not-empty by auto
next
\show B ∈ Bot-FL =⇒ \G-F-L B ⊆ Bot-G for B unfolding G-F-L-def Bot-FL-def using Bot-map by force
next
\fix CL
show $G\cap F\not=\emptyset\implies CL\in Bot-FL$

unfolding $G\cap F\not=\emptyset$ by metis SigmaE UNIV-I UNIV-Times-UNIV mem-Sigma-iff prod.sel(1)

next
fix $i$
assume
  $i\in Inf-FL$ and
  $G\cap F\not=\emptyset$

then show $(G\cap F)i\subseteq Red-Inf-G (G\cap F (\text{concl-of } i))$

unfolding $G\cap F\not=\emptyset$ by fastforce

abbreviation Labeled-Empty-Order :: $(f \times l)\Rightarrow (f \times l)\Rightarrow \text{bool}$ where
Labeled-Empty-Order $C1 C2 \equiv False$

sublocale labeled-lifting-w-empty-ord-family :
  lifting-with-wf-ordering-family Bot-FL Inf-FL Bot-G entails-G Inf-G Red-Inf-G Red-F-G

proof
  show po-on Labeled-Empty-Order UNIV unfolding po-on-def by simp add: transp-onI wfp-on-imp-irreflp-on

  show wfp-on Labeled-Empty-Order UNIV unfolding wfp-on-def by simp

qed

notation labeled-standard-lifting.entails-$G$ (infix $|$=GL 50)

lemma labeled-entailment-lifting: $NL1 \Rightarrow GL NL2 \iff fint\ NL1 \Rightarrow GL fint\ NL2$

unfolding labeled-standard-lifting.entails-$G$ unfolding $G\cap F\not=\emptyset$ by auto

lemma (in−) subset-fst: $A \subseteq \text{fst } AB \implies \forall x \in A. \exists y. (x,y) \in AB$ by fastforce

lemma red-inf-impl: $i \in \text{labeled-lifting-w-empty-ord-family}.\text{Red-Inf-}G NL \Rightarrow \text{to-F } i \in \text{Red-Inf-}G (\text{fst } NL)$

unfolding labeled-lifting-w-empty-ord-family.Red-Inf-$G$ Red-Inf-$G$ unfolding $G\cap F\not=\emptyset$ to-F-def

using Inf-FL-to-Inf-F by auto

lemma labeled-saturation-lifting:
  labeled-lifting-w-empty-ord-family.lifted-calculus-with-red-crit.saturated $NL \implies$
  empty-order-lifting.lifted-calculus-with-red-crit.saturated (fst $\times NL$)

unfolding labeled-lifting-w-empty-ord-family.lifted-calculus-with-red-crit.saturated-def
  empty-order-lifting.lifted-calculus-with-red-crit.saturated-def

proof clarify
fix $i$
assume
  $i\in Inf-FL$ and
  $i\prems$: set (prems-of $i$) $\subseteq NL$

define Lli where $Lli\ i \equiv (\text{SOME } x. (\text{prems-of } i)!i,x) \in NL)$ for $i$

have $[(\text{simp}):((\text{prems-of } i)!i,Lli\ i) \in NL \text{ if } i < \text{length } (\text{prems-of } i)$ for $i$

using that subset-fst[of $i\prems$] unfolding Lli-def by (meson nth-mem someI-ex)
define \( \text{Ll} \) where \( \text{Ll} \equiv \text{map} \text{Lli} [0..<\text{length (prems-of } \text{i})] \)

have \( \text{Ll-length: length \( \text{Ll} \) = length (prems-of } \text{i}) \) unfolding \( \text{Ll}\text{-def by auto} \)

have \( \text{subs-NL: set (zip (prems-of } \text{i}) \text{Ll}) \subseteq \text{NL} \) unfolding \( \text{Ll}\text{-def by (auto simp:in-set-zip}) \)

obtain \( \text{L0} \) where \( \text{L0: Infer (zip (prems-of } \text{i}) \text{Ll}) \) (concl-of \( \text{i} \), \( \text{L0} \)) \( \in \text{Inf-FL} \)

using \( \text{Inf-F-to-Inf-FL}(\text{OF } \text{i-in \text{Ll-length}}) \) ..

define \( \text{t-FL} \) where \( \text{t-FL} = \text{Infer (zip (prems-of } \text{i}) \text{Ll}) \) (concl-of \( \text{i} \), \( \text{L0} \))

then have set (prems-of \( \text{t-FL} \)) \( \subseteq \text{NL} \) using \( \text{subs-NL by simp} \)

then have \( \text{t-FL} \in \{ \text{i} \in \text{Inf-FL. set (prems-of } \text{i}) \subseteq \text{NL} \} \) unfolding \( \text{t-FL}\text{-def using } \text{L0 by blast} \)

then have \( \text{t-FL} \in \text{labeled-lifting-w-empty-ord-family.Red-Inf-\text{G} NL using } \text{subs-Red-Inf by fast} \)

moreover have \( \text{t = to-F } \text{t-FL unfolding to-F\text{-def } t-FL\text{-def using } \text{Ll-length by (cases } \text{i}) \) auto \n
ultimately show \( \text{i} \in \text{Red-Inf-\text{G} } (\text{fst } ' \text{NL}) \) by (auto intro:red-inf-impl)

qed


lemma \( \text{stat-ref-comp-to-labeled-sta-ref-comp: static-refutational-complete-calculus Bot-F Inf-F (=} G) \text{ Red-Inf-} G \text{ Red-F-}\text{G} \Rightarrow \text{static-refutational-complete-calculus Bot-FL Inf-FL (=} GL) \text{ labeled-lifting-w-empty-ord-family.Red-Inf-\text{G} labeled-lifting-w-empty-ord-family.Red-F-}\text{G} \text{ unfolding static-refutational-complete-calculus-def} \)

proof (rule \text{ conjI impI; clarify} )


show calculus-with-red-crit Bot-FL Inf-FL (=} GL labeled-lifting-w-empty-ord-family.Red-Inf-\text{G} labeled-lifting-w-empty-ord-family.Red-F-\text{G}

by standard

next

assume

calc: calculus-with-red-crit Bot-F Inf-F (=} G) Red-Inf-\text{G} Red-F-\text{G} and

static: static-refutational-complete-calculus-axioms Bot-F Inf-F (=} G) Red-Inf-\text{G} Red-F-\text{G}

show static-refutational-complete-calculus-axioms Bot-FL Inf-FL (=} GL)

labeled-lifting-w-empty-ord-family.Red-Inf-\text{G}

unfolding static-refutational-complete-calculus-axioms-def

proof (intro \text{ conjI impI allI})

fix \( \text{Bl} :: (\text{f} \times \text{f}) \text{ and } \text{NI} :: (\text{f} \times \text{f}) \text{ set})

assume

\( \text{Bl-in: } \text{Bl} \in \text{Bot-FL, and} \text{NI-sat: } (\text{labeled-lifting-w-empty-ord-family.lifted-calculus-with-red-crit.saturated NI}) \text{ and} \text{NI-entails-Bl: } (\text{NI } \vdash \text{G } (\text{Bl}')) \)

have static-axioms: \( \text{B} \in \text{Bot-F } \rightarrow \text{empty-order-lifting.lifted-calculus-with-red-crit.saturated } \text{N } \rightarrow N \vdash \text{G } (\text{B}) \rightarrow (\exists \text{B}' \in \text{Bot-F. } B' \in \text{N}) \text{ for } B \in \text{N} \)

using \( \text{static[unfolded static-refutational-complete-calculus-axioms-def] by fast} \)

define \( \text{B where } \text{B = fst Bl} \)

have \( \text{B-in: } \text{B} \in \text{Bot-F using } \text{Bl-in Bot-FL-def B-def SigmaE by force} \)

define \( \text{N where } \text{N = fst } ' \text{NI} \)

have \( \text{N-sat: empty-order-lifting.lifted-calculus-with-red-crit.saturated N} \)

using \( \text{N-def NI-sat labeled-saturation-lifting by blast} \)

have \( \text{N-entails-B: } N \Rightarrow G (B) \)

using \( \text{NI-entails-Bl unfolding labeled-entailment-lifting N-def B-def by force} \)

have \( \exists \text{B}' \in \text{Bot-F. } B' \in \text{N using } \text{B-in N-sat N-entails-B static-axioms[of B N] by blast} \)

then obtain \( \text{B' where in-Bot: } B' \in \text{Bot-F and in-N: } B' \in \text{N by force} \)

then have \( \text{B' \in fst ' Bot-FL unfolding Bot-FL-def by fastforce} \)

obtain \( \text{Bl' where in-NI: } \text{Bl'} \in \text{NI and fst-Bl': } \text{fst Bl'} = B' \)

using \( \text{in-N unfolding N-def by blast} \)
have $Bl' \in Bot-FL$ unfolding Bot-FL-def using fst-$Bl'$ in-Bot vimage-fst by fastforce
then show $\exists Bl' \in Bot-FL. Bl' \in Nb$ using in-Nl by blast
qed
qed

end

4.2 Labeled Lifting with a Family of Redundancy Criteria

locale labeled-lifting-with-red-crit-family = no-labels: standard-lifting-with-red-crit-family Inf-F Bot-G Inf-G Q entails-q Red-Inf-q Red-F-q Bot-F G-F-q G-Inf-q \lambda g. Empty-Order
for
Bot-F :: 'f set and
Inf-F :: 'f inference set and
Bot-G :: 'g set and
Q :: 'q itself and
entails-q :: 'q \Rightarrow 'g set \Rightarrow 'g set \Rightarrow bool and
Inf-G :: 'g inference set and
Red-Inf-q :: 'q \Rightarrow 'g set \Rightarrow 'g inference set and
Red-F-q :: 'q \Rightarrow 'g set \Rightarrow 'g set and
G-F-q :: 'q \Rightarrow 'f \Rightarrow 'g set and
G-Inf-q :: 'q \Rightarrow 'f inference \Rightarrow 'g inference set option
+ fixes
l :: 'l itself and
Inf-FL :: (\langle 'f \times 'l \rangle inference set)
assumes
Inf-F-to-Inf-FL: \langle \varepsilon_F \in Inf-F \Rightarrow length (Ll :: 'l list) = length (prems-of \varepsilon_F) \Rightarrow \exists L0. Infer (zip (prems-of \varepsilon_F) Ll) (concl-of \varepsilon_F, L0) \in Inf-FL) and
Inf-FL-to-Inf-F: \langle \varepsilon_FL \in Inf-FL \Rightarrow Infer (map fst (prems-of \varepsilon_FL)) (fst (concl-of \varepsilon_FL)) \in Inf-F
begin

definition to-F :: (\langle 'f \times 'l \rangle inference set \Rightarrow 'f inference) where
to-F \varepsilon_FL = Infer (map fst (prems-of \varepsilon_FL)) (fst (concl-of \varepsilon_FL))

definition Bot-FL :: (\langle 'f \times 'l \rangle set) where Bot-FL = Bot-F \times UNIV

definition G-F-L-q :: 'q \Rightarrow \langle 'f \times 'l \rangle \Rightarrow 'g set where G-F-L-q q CL = G-F-q q (fst CL)

definition G-Inf-L-q :: 'q \Rightarrow \langle 'f \times 'l \rangle inference set \Rightarrow 'g inference set option where
G-Inf-L-q q \varepsilon_FL = G-Inf-q q (to-F \varepsilon_FL)

definition G-set-L-q :: 'q \Rightarrow \langle 'f \times 'l \rangle set \Rightarrow 'g set where
G-set-L-q q N \equiv \bigcup (G-F-L-q q \times N)

definition Red-Inf-G-L-q :: 'q \Rightarrow \langle 'f \times 'l \rangle set \Rightarrow \langle 'f \times 'l \rangle inference set where
Red-Inf-G-L-q q N = \{ \varepsilon \in Inf-FL. ((G-Inf-L-q q \varepsilon) \neq None \land (G-Inf-L-q q \varepsilon) \subseteq Red-Inf-q q (G-set-L-q q N)) \}
\lor ((G-Inf-L-q q \varepsilon = None) \land G-F-L-q q (concl-of \varepsilon) \subseteq (G-set-L-q q N \cup Red-F-q q (G-set-L-q q N))))

definition Red-Inf-G-L-Q :: \langle 'f \times 'l \rangle set \Rightarrow \langle 'f \times 'l \rangle inference set where
Red-Inf-G-L-Q N = \bigcap \{ X N | X. X \in (Red-Inf-G-L-q : UNIV)\}

definition Labeled-Empty-Order :: \langle 'f \times 'l \rangle \Rightarrow \langle 'f \times 'l \rangle \Rightarrow bool where
Labeled-Empty-Order C1 C2 \equiv False

definition Red-F-G-empty-L-q :: 'q \Rightarrow \langle 'f \times 'l \rangle set \Rightarrow \langle 'f \times 'l \rangle set where
Red-F-G-empty-L \ q N = \{ C. \ \forall D \in G-F-L-q \ q C. \ D \in Red-F-q \ q (G-set-L-q \ q N) \ \vee \\
(\exists E \in N. \text{Labeled-Empty-Order} \ E \ C \ \land D \in G-F-L-q \ q E)\}

\textbf{definition} \ Red-F-G-empty-L :: \'(f \times 't) \set \Rightarrow \'(f \times 't) \set \text{ where} \\
Red-F-G-empty-L \ q N = \bigcap \{ X \ \set | X. \ X \in (Red-F-G-empty-L-q : \ \bot) \}\}

\textbf{definition} \ entails-G-L-q :: 'q \Rightarrow \'(f \times 't) \set \Rightarrow \'(f \times 't) \set \Rightarrow \text{bool where} \\
entails-G-L-q \ q N1 N2 \equiv \text{entails-q} \ q \ (G-set-L-q \ q N1) \ (G-set-L-q \ q N2)\)

\textbf{definition} \ entails-G-L-Q :: \'(f \times 't) \set \Rightarrow \'(f \times 't) \set \Rightarrow \text{bool (infix \ \inL \ 50) where} \\
entails-G-L-Q \ q N1 N2 \equiv \forall \ q. \ \text{entails-G-L-q} \ q N1 N2\)

\textbf{lemma} \ lifting-q: \text{-labeled-lifting-w-wf-ord-family} \ Bot-F \ Inf-F \ Bot-G \ \text{(entails-q} \ q) \ \text{Inf-G} \ \text{(Red-Inf-q} \ q) \\
(\text{Red-F-q} \ q) \ (G-F-L-q \ q) \ (G-Inf-q \ q) \ (\lambda g. \ \text{Empty-Order}) \ \text{Inf-FL}\)
\textbf{proof} – \\
\text{fix} \ q \\
\text{show} \ \text{labeled-lifting-w-wf-ord-family} \ Bot-F \ Inf-F \ Bot-G \ \text{(entails-q} \ q) \ \text{Inf-G} \ \text{(Red-Inf-q} \ q) \\
(\text{Red-F-q} \ q) \ (G-F-L-q \ q) \ (G-Inf-q \ q) \ (\lambda g. \ \text{Empty-Order}) \ \text{Inf-FL}\)
\text{using} \ \text{no-labels.standard-lifting-family} \ \text{Inf-F-to-Inf-FL} \ \text{Inf-FL-to-Inf-F}\)
\text{by} \ (\text{simp add: labeled-lifting-w-wf-ord-family-axioms-def labeled-lifting-w-wf-ord-family-def)}
\textbf{qed}

\textbf{lemma} \ lifted-q: \text{-standard-lifting} \ Bot-FL \ Inf-FL \ Bot-G \ Inf-G \ \text{(entails-q} \ q) \ \text{(Red-Inf-q} \ q) \\
(\text{Red-F-q} \ q) \ (G-F-L-q \ q) \ (G-Inf-L-q \ q)\)
\textbf{proof} – \\
\text{fix} \ q \\
\text{interpret} \ q\text{-lifting:} \ \text{labeled-lifting-w-wf-ord-family} \ Bot-F \ Inf-F \ Bot-G \ \text{entails-q} \ q \ \text{Inf-G} \\
\text{Red-Inf-q} \ q \ \text{Red-F-q} \ q \ G-F-L-q \ q \ G-Inf-q \ q \ (\lambda g. \ \text{Empty-Order}) \ \text{Inf-FL}\)
\text{using} \ \text{lifting-q}\).
\text{have} \ G-F-L-q \ q = q\text{-lifting} \ G-F-L\)
\text{unfolding} \ G-F-L-q\text{-def q\text{-lifting}G-F-L-def by simp}
\text{moreover have} \ G-Inf-L-q \ q = q\text{-lifting} \ G-Inf-L\)
\text{unfolding} \ G-Inf-L-q\text{-def q\text{-lifting}G-Inf-L-def to-F-def q\text{-lifting}to-F-def by simp}
\text{moreover have} \ Bot-FL = q\text{-lifting} \ Bot-FL\)
\text{unfolding} \ Bot-FL\text{-def q\text{-lifting}Bot-FL-def by simp}
\text{ultimately show} \ \text{standard-lifting} \ Bot-FL \ Inf-FL \ Bot-G \ \text{Inf-G} \ \text{(entails-q} \ q) \ \text{(Red-Inf-q} \ q) \ \text{(Red-F-q} \ q) \\
(G-F-L-q \ q) \ (G-Inf-L-q \ q)\)
\text{using} \ q\text{-lifting.labeled-standard-lifting.standard-lifting-axioms by simp}
\textbf{qed}

\textbf{lemma} \ ord-fam-lifted-q: \text{-lifting-with-wf-ordering-family} \ Bot-FL \ Inf-FL \ Bot-G \ \text{(entails-q} \ q) \ \text{Inf-G} \\
(\text{Red-Inf-q} \ q) \ (\text{Red-F-q} \ q) \ (G-F-L-q \ q) \ (G-Inf-L-q \ q) \ (\lambda g. \ \text{Labeled-Empty-Order})\)
\textbf{proof} – \\
\text{fix} \ q \\
\text{interpret} \ standard-q\text{-lifting:} \ \text{standard-lifting} \ Bot-FL \ Inf-FL \ Bot-G \ \text{entails-q} \ q \\
\text{Red-Inf-q} \ q \ \text{Red-F-q} \ q \ G-F-L-q \ q \ G-Inf-L-q \ q\)
\text{using} \ \text{lifted-q}\).
\text{have} \ \text{minimal-element} \ \text{Labeled-Empty-Order} \ \text{UNIV}
\text{unfolding} \ \text{Labeled-Empty-Order-def}
\text{by} \ (\text{simp add: minimal-element.intro po-on-def transp-onI wp-on-imp irreflp-on)}
\text{then show} \ \text{lifting-with-wf-ordering-family} \ Bot-FL \ Inf-FL \ Bot-G \ \text{(entails-q} \ q) \ \text{Inf-G} \\
(\text{Red-Inf-q} \ q) \ (\text{Red-F-q} \ q) \ (G-F-L-q \ q) \ (G-Inf-L-q \ q) \ (\lambda g. \ \text{Labeled-Empty-Order})\)
\text{using} \ \text{standard-q-lifting.standard-lifting-axioms}
\text{by} \ (\text{simp add: lifting-with-wf-ordering-family-axioms.intro lifting-with-wf-ordering-family-def)}
\textbf{qed}
lemma all-lifted-red-crit: calculus-with-red-crit Bot-FL Inf-FL (entails-G-L-q q) (Red-Inf-G-L-q q)
(Red-F-G-empty-L-q q)

proof –
  fix q
  interpret ord-q-lifting: lifting-with-uf-ordering-family Bot-FL Inf-FL Bot-G entails-q q Inf-G
  Red-Inf-G-L-q q Red-F-G-empty-L-q q G F L q q G Inf L q q Lambda. Labeled-Empty-Order
  using ord-fam.lifted-q .
  have entails-G-L-q q = ord-q-lifting. entails-G
  unfolding entails-G-L-q-def G set-L-q-def ord-q-lifting. entails-G-def by simp
  moreover have Red-Inf-G-L-q-q q = ord-q-lifting. Red-Inf-G
  unfolding Red-Inf-G-L-q-q-def ord-q-lifting. Red-Inf-G-def G set-L-q-def by simp
  moreover have Red-F-G-empty-L-q-q q = ord-q-lifting. Red-F-G
  unfolding Red-F-G-empty-L-q-q-def ord-q-lifting. Red-F-G-def G set-L-q-def by simp
  ultimately show calculus-with-red-crit Bot-FL Inf-FL (entails-G-L-q q) (Red-Inf-G-L-q q)
  (Red-F-G-empty-L-q q)
  using ord-q-lifting. lifted-calculus-with-red-crit. calculus-with-red-crit-axioms by argo
qed

lemma all-lifted-cons-rel: consequence-relation Bot-FL (entails-G-L-q q)

proof –
  fix q
  interpret q-red-crit: calculus-with-red-crit Bot-FL Inf-FL entails-G-L-q q Red-Inf-G-L-q q
  Red-F-G-empty-L-q q
  using all-lifted-red-crit .
  show consequence-relation Bot-FL (entails-G-L-q q)
  using q-red-crit.consequence-relation-axioms .
qed

sublocale labeled-cons-rel-family: consequence-relation-family Bot-FL Q entails-G-L-q
  using all-lifted-cons-rel no-labels.lifted-calc-w-red-crit-family. Bot-not-empty
  unfolding Bot-FL-def
  by (simp add: consequence-relation-family.intro)

sublocale with-labels: calculus-with-red-crit-family Bot-FL Inf-FL Q entails-G-L-q Red-Inf-G-L-q
  Red-F-G-empty-L-q
  using calculus-with-red-crit-family.intro[OF labeled-cons-rel-family.consequence-relation-family-axioms]
  all-lifted-cons-rel
  by (simp add: all-lifted-red-crit calculus-with-red-crit-family-axioms-def)

notation no-labels.entails-G-Q (infix \|-\cap 50)

lemma labeled-entailment-lifting: NL1 |-\cap NL2 \iff fst ' NL1 |-\cap fst ' NL2
  entails-G-L-Q-def entails-G-L-Q-def G set-L-q-def G F L q-def
  by force

lemma subset-fst: A \subseteq fst ' AB \implies \forall x \in A. \exists y. (x,y) \in AB by fastforce

lemma red-inf-impl: \iota \in with-labels.Red-Inf-Q NL \implies
  to-F \iota \in no-labels.empty-ord-lifted-calc-w-red-crit-family. Red-Inf-Q (fst ' NL)
proof clarify
  fix X Xa q
assume
\[ i \in \{ X \mid X \in \text{Red-Inf-G-L-q} \wedge i \in \text{UNIV} \} \]

have [i-in-q]: \( i \in \text{Red-Inf-G-L-q} \) \( \subseteq \text{NL} \) using \( \text{i-in-inter image-eqI by blast} \)

then have [i-in]: \( i \in \text{Inf-FL unfolding Red-Inf-G-L-q-def by blast} \)

have to-F-in: \( \text{to-F} \circ i \in \text{Inf-F unfolding to-F-def using Inf-F-to-Inf-F[OF i-in]} \) .

have rephrase1: \( \bigcup C \subseteq \text{NL} \) \( \text{G-q ~ (fsl ~ C)} \) \( = \bigcup (\text{G-q ~ (fsl ~ NL)}) \) by blast

have rephrase2: \( \text{fst (concl-of i) ~ (concl-of (to-F i))} \) unfolding concl-of-def to-F-def by simp

using i-in unfolding Red-Inf-G-L-q-def by blast

then have to-F-sub-red: \( (G-q ~ (to-F i)) \neq \text{None} \wedge (G-q ~ (to-F i)) \subseteq \text{Red-Inf-q ~ (G-set-q ~ NL))} \)

\( \vee (G-q ~ (to-F i) = \text{None} \wedge (G-q ~ (concl-of (to-F i)) \subseteq (\text{no-labels.G-set-q ~ (fsl ~ NL)} \cup \text{Red-F-q ~ (G-set-q ~ NL)})) \) \)

using i-in unfolding Red-Inf-G-L-q-def by blast

then show to-F-in [i-in-q]: \( \text{to-F} \circ i \notin \text{no-labels.Red-Inf-G-q} \) \( \subseteq \text{NL} \) using to-F-in unfolding no-labels.Red-Inf-G-q-def by simp

qed

lemma labeled-family-saturation-lifting: with-labels.inter-red-crit-calculus.saturated NL \( \Rightarrow \)

no-labels.lifted-calc-w-red-crit-family.inter-red-crit-calculus.saturated (fsl ~ NL)

unfolding with-labels.inter-red-crit-calculus.saturated-def

with-labels.lifted-calc-w-red-crit-family.inter-red-crit-calculus.saturated-def

with-labels.Inf-from-def no-labels.Non-ground.Inf-from-def

proof clarify

fix i F

assume

labeled-sat: \( \{ i \in \text{Inf-FL. set (prems-of i) ~ NL} \} \subseteq \text{with-labels.Red-Inf-Q ~ NL and} \)

i-F-in: \( i \in \text{Inf-F and} \)

i-F-prems: \( \text{set (prems-of iF) ~ fst ~ NL} \)

define Ll where Ll i \( \equiv (\text{SOME x. (prems-of iF)|x ~ NL}) \) for i

have [simp]: \( \text{(prems-of iF)|i ~ NL \text{ if i} ~ \text{< length (prems-of iF) for i}} \)

using that subset-fst[OF iF-prems] nth-mem some-el-ex unfolding Ll-def

by metis

define Ll where Ll i \( \equiv \text{map Lli [0..<length (prems-of iF)]} \)

have Ll-length: \( \text{length Ll} = \text{length (prems-of iF)} \) unfolding Ll-def by auto

have subs-NL: \( \text{set (zip (prems-of iF) Ll) ~ NL unfolding Ll-def by (auto simp:in-set-zip)} \)

obtain L0 where L0: \( \text{Inf (zip (prems-of iF) Ll) ~ (concl-of iF, L0) ~ Inf-FL} \)

using Inf-FL-to-Inf-FL[OF iF-in Ll-length] ..

define iF FL where iF FL \( \equiv \text{Inf (zip (prems-of iF) Ll) ~ (concl-of iF, L0) ~ Inf-FL} \)

then have set [prems-of iF] \( \subseteq \) \( \text{Inf-FL unfolding iF-def using L0 by blast} \)

then have iF FL \( \in \) \( \{ i \in \text{Inf-FL. set (prems-of i) ~ NL} \} \) unfolding iF-def by fast

moreover have iF = to-F iF unfolding to-F-def iF-def using Ll-length by (cases iF) auto

ultimately show iF \( \in \) \( \text{no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fsl ~ NL)} \)

by (auto intro:red-inf-impl)

qed

theorem labeled-static-ref: static-refutational-complete-calculus Bot-F Inf-F (\( \equiv \cap \))
5 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.
5.1 Basis of the Prover Architectures

locale Prover-Architecture-Basis = labeled-lifting-with-red-crit-family Bot-F Inf-F Bot-G Q entails-q Inf-G

Red-Inf-q Red-F-q G-F-q G-Inf-q \l Inf-FL

for

Bot-F :: 'f set
and Inf-F :: 'f inference set
and Bot-G :: 'g set
and Q :: 'q itself
and entails-q :: 'q ⇒ ('g set ⇒ 'g set ⇒ bool)
and Inf-G :: ('g inference set)
and Red-Inf-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-Inf-q :: 'q ⇒ 'f inference ⇒ 'g inference set option
and l :: 'l itself
and Inf-FL :: ('f × 'l) inference set:

+ fixes

Equiv-F :: ('f × 'f) set and
Prec-F :: 'f ⇒ 'f ⇒ bool (infix \triangleright 50) and
Prec-l :: 'l ⇒ 'l ⇒ bool (infix \sqsubset 50)

assumes

equiv-F-is-eqv-rel: equiv UNIV Equiv-F and
wf-pred-F: minimal-element (Prec-F) UNIV and
wf-pred-l: minimal-element (Prec-l) UNIV and
compat-eqv-pred: (C1,D1) ∈ equiv-F ⇒ (C2,D2) ∈ equiv-F ⇒ C1 ↦ C2 ⇒ D1 ↦ D2 and
equiv-F-grounding: (C1,C2) ∈ equiv-F ⇒ G-F-q q C1 = G-F-q q C2 and
prec-F-grounding: C1 ↦ C2 ⇒ G-F-q q C1 \subseteq G-F-q q C2 and
static-ref-comp: static-refutational-complete-calculus Bot-F Inf-F (|=\cap)

no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q
no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-F-Q


definition equiv-F-fun :: 'f ⇒ 'f ⇒ bool (infix \equiv 50) where
equiv-F-fun C D ≡ (C,D) ∈ Equiv-F


definition Prec-eq-F :: 'f ⇒ 'f ⇒ bool (infix \equiv 50) where
Prec-eq-F C D ≡ ((C,D) ∈ Equiv-F \∨ C ↦ D)


definition Prec-FL :: ('f × 'l) ⇒ ('f × 'l) ⇒ bool (infix \sqsubset 50) where
Prec-FL C1 C2 ≡ (fst C1 ↦ fst C2) \∨ (fst C1 \eq fst C2 \∧ snd C1 \sqsubset l snd C2)

lemma wf-pred-FL: minimal-element (\sqsubset) UNIV
proof

show po-on (\sqsubset) UNIV unfolding po-on-def
proof

show irreflp-on (\sqsubset) UNIV unfolding irreflp-on-def Prec-FL-def
proof

fix a

assume a-in: a ∈ (UNIV::('f × 'l) set)

have \neg (fst a ↦ fst a) using wf-pred-F minimal-element.min-elt-ex by force

moreover have \neg (snd a \sqsubset l snd a) using wf-pred-l minimal-element.min-elt-ex by force

ultimately show \neg (fst a ↦ fst a \∨ fst a \eq fst a \∧ snd a \sqsubset l snd a) by blast
**qed**

**next**

**show** transp-on (\(\Box\)) UNIV unfolding transp-on-def Prec-FL-def

**proof** (simp, intro allI impI)

fix \(a\), \(b\), \(a_2\), \(b_2\)

assume trans-hyp: \(a_1 \rightarrow a_2 \lor a_1 \not\leq a_2 \land b_1 \sqsubset b_2\) \& \((a_2 \not\leq a_3 \lor a_2 \not\leq a_3 \land b_2 \sqsubset b_3)\)

have \(a_1 \rightarrow a_2 \Rightarrow a_2 \rightarrow a_3 \Rightarrow a_1 \rightarrow a_3\) using wf-prec-F compat-equiv-prec by blast

moreover have \(a_1 \rightarrow a_2 \Rightarrow a_2 \rightarrow a_3 \Rightarrow a_1 \rightarrow a_3\) using wf-prec-F compat-equiv-prec by blast

moreover have \(\forall (a_1 \not\leq a_2 \Rightarrow a_2 \not\leq a_3 \Rightarrow a_1 \not\leq a_3)\) using equi-F-is-equiv-rel equiv-class-eq unfolding equiv-F-fun-def by fastforce

ultimately show \((a_1 \not\leq a_3 \lor a_1 \not\leq a_3 \land b_1 \sqsubset b_3)\) using trans-hyp by blast

**qed**

**qed**

**next**

**show** wfp-on (\(\Box\)) UNIV unfolding wfp-on-def

**proof**

assume contra: \(\exists f. \forall i. f \in \text{UNIV} \land f (\text{Suc}\ i) \not\leq f\ i\)

then obtain \(f\) where \(\text{f-in}\): \(\forall i. f \in \text{UNIV} \land f \text{-suc}: \forall i. f (\text{Suc}\ i) \not\leq f\ i\) by blast

define \(f\)-F where \(f\text{-F} = (\lambda i.\ f\ i)\)

define \(f\)-L where \(f\text{-L} = (\lambda i.\ \text{snd} (f\ i))\)

have uni-F: \(\forall i. f\text{-F} i \in \text{UNIV unfolding f-in by simp}\)

have uni-L: \(\forall i. f\text{-L} i \in \text{UNIV unfolding f-in by simp}\)

have decomp: \(\forall i. f\text{-F} (\text{Suc}\ i) \not\leq f\text{-F} i \land f\text{-L} (\text{Suc}\ i) \not\leq f\text{-L} i\)

using f-suc unfolding Prec-FL-def f-prec-F f-prec-L-def by blast

define I-F where \(I\text{-F} = \{ i | f\text{-F} (\text{Suc}\ i) \not\leq f\text{-F} i\}\)

define I-L where \(I\text{-L} = \{ i | f\text{-L} (\text{Suc}\ i) \not\leq f\text{-L} i\}\)

have I-F \& I-L = UNIV using decomp unfolding I-def I-L-def by blast

then have finite I-F \& finite I-L by (metis finite-UnI infinite-UNIV-nat)

moreover have infinite I-F \& infinite I-L by (meson compat-equiv-prec iso-tuple-UNIV-I not-finite-existsD)

moreover have infinite I-L \& infinite I-F by (meson finite I-F finite I-L)

ultimately show \(\text{False}\) using wf-prec-F wf-prec-L by (metis minimal-element-def wfp-on-def)

**qed**

**lemma** labeled-static-ref-comp:


**lemma** standard-labeled-lifting-family: lifting-with-wf-ordering-family Bot-FL Inf-FL Bot-G (entails-q q) Inf-G (Red-Inf-q q) (Red-F-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q)

**proof**

fix q

have lifting-with-wf-ordering-family Bot-FL Inf-FL Bot-G (entails-q q) Inf-G (Red-Inf-q q) (Red-F-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q)

using ord-fam-lifted-q.

then have standard-lifting Bot-FL Inf-FL Bot-G (entails-q q) (Red-Inf-q q) (Red-F-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q)

using lifted-q by blast

then show lifting-with-wf-ordering-family Bot-FL Inf-FL Bot-G (entails-q q) Inf-G (Red-Inf-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q) (G-Inf-L-q q)
(Red-F-q q) (G-F-L-q q) (G-Inf-L-q q) (λg. Prec-FL)
using wf-prec-FL
by (simp add: lifting-with-wf-ordering-family.intro lifting-with-wf-ordering-family-axioms.intro)
qed

entails-q Red-Inf-q Red-F-q
Bot-FL G-F-L-q G-Inf-L-q λg. Prec-FL
using standard-labeled-lifting-family
  no-labels.Ground-family.calculus-with-red-crit-family-axioms
by (simp add: standard-labeled-lifting-family.intro standard-lifting-with-red-crit-family-axioms.intro)

lemma entail-equiv:
labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.entails-Q N1 N2 = (N1 ≦∩L N2)
unfolding labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.entails-Q-def
  entails-G-L-Q-def entails-G-L-q-def labeled-ord-red-crit-fam.entails-G-q-def
by simp

lemma entail-equiv2: labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.entails-Q = (≡∩L)
using entail-equiv by auto

  with-labels.Red-Inf-Q N
unfolding labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-Inf-Q-def
by simp

  with-labels.Red-Inf-Q
using red-inf-equiv by auto

  with-labels.Red-F-Q N
by simp

  with-labels.Red-F-Q
using empty-red-f-equiv by auto

lemma labeled-ordered-static-ref-comp:
  static-refutational-complete-calculus Bot-FL Inf-FL
labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.entails-Q
labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-Inf-Q
labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q
using labeled-ord-red-crit-fam.static-empty-ord-inter-equiv-static-inter empty-red-f-equiv2
  red-inf-equiv2 entail-equiv2 labeled-static-ref-comp
by argo

interpretation stat-ref-calc: static-refutational-complete-calculus Bot-FL Inf-FL
lemma labeled-ordered-dynamic-ref-comp:
  dynamic-refutational-complete-calculus Bot-FL Inf-FL
labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.
by (rule labeled-ordered-static-ref-comp)

lemma labeled-red-inf-eq-red-inf: \( \tau \in \text{Inf-FL} \implies \tau \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.} \text{Red-Inf-Q} N \equiv \)
(to-F \( \tau \)) \in no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fst \( \cdot \) N) for \( \tau \)

proof –
  fix \( \tau \)
  assume \( \tau \)-in: \( \tau \in \text{Inf-FL} \)
  have \( \tau \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.} \text{Red-Inf-Q} N \equiv \)
  (to-F \( \tau \)) \in no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fst \( \cdot \) N)

proof –
  assume \( \tau \)-in2: \( \tau \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.} \text{Red-Inf-Q} N \)
  then have \( X \in \text{labeled-ord-red-crit-fam.} \text{Red-Inf-G-q } \cdot \text{UNIV} \implies \tau \in X \) for \( X \)
  unfolding labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-Inf-Q-def by blast

obtain \( X0 \) where \( X0 \in \text{labeled-ord-red-crit-fam.} \text{Red-Inf-G-q } \cdot \text{UNIV} \) by blast
then obtain \( q0 \) where \( x0-is: X0 \subseteq \text{labeled-ord-red-crit-fam.} \text{Red-Inf-G-q q0 N} \) by blast
then obtain \( Y0 \) where \( y0-is: Y0 (\text{fst } \cdot \) N) = to-F \( \cdot (X0 \) N) by auto

have \( Y0 (\text{fst } \cdot \) N) = no-labels.Red-Inf-G-q q0 (fst \( \cdot \) N)

unfolding \( y0-is \)

proof
  show to-F \( \cdot X0 \subseteq \text{no-labels.Red-Inf-G-q q0 (fst } \cdot \) N)

proof
  fix \( i0 \)
  assume \( i0\)-in: \( i0 \in \text{to-F } \cdot X0 \) N
  then have \( i0\)-in2: \( i0 \in \text{to-F } \cdot \) (labeled-ord-red-crit-fam.Red-Inf-G-q q0 N)
  using \( x0-is \) by argo

then obtain \( i0-FL \) where \( i0-FL-in: i0-FL \in \text{Inf-FL and i0-to-i0-FL: i0 = to-F } i0-FL \) and
subs1: \( ((G-Inf-L-q q0 i0-FL) \neq \text{None}) \land \)
  the \( (G-Inf-L-q q0 i0-FL) \subseteq \text{Red-Inf-q q0 (labeled-ord-red-crit-fam.G-set-q q0 N)})
  \lor \( ((G-Inf-L-q q0 i0-FL = \text{None}) \land \)
  \text{G-F-L-q q0 (concl-of } i0-FL) \subseteq \text{labeled-ord-red-crit-fam.G-set-q q0 N} \cup \text{Red-F-q q0 (labeled-ord-red-crit-fam.G-set-q q0 N)})

unfolding labeled-ord-red-crit-fam.Red-Inf-G-q-def by blast

have concl-swap: \( \text{fst \ (concl-of } i0-FL) = \text{concl-of } i0 \)

unfolding concl-of-def \( i0\)-to-i0-FL to-F-def by simp

have \( i0\)-in3: \( i0 \in \text{Inf-F} \)

using \( \text{i0\)-to-i0-FL Inf-FL-to-Inf-F} \{ \text{OF i0-FL-in} \} \) unfolding to-F-def by blast
{ 
  assume
  not-none: \( G-Inf-q q0 i0 \neq \text{None and} \)
  the \( (G-Inf-q q0 i0) \neq \{ \}) \)
then obtain \( i1 \) where \( i1\)-in: \( i1 \in \text{(G-Inf-q q0 i0) by blast} \)

have \( (G-Inf-q q0 i0) \subseteq \text{Red-Inf-q q0 (no-labels.G-set-q q0 (fst } \cdot \) N)} \)

using subs1 \( i0\)-to-i0-FL not-none
proof

moreover {  
  assume is-none: G-Inf-q q0 i0 = None  
  then have G-F-q q0 (concl-of i0) ⊆ no-labels.G-set-q q0 (fst i N) ∪ Red-F-q q0 (no-labels.G-set-q q0 (fst i N))  
  using subs1 i0-to-i0-FL concl-swap  
  G-Inf-L-q-def G-F-L-q-def by simp  
}

ultimately show i0 ∈ no-labels.Red-Inf-G-q q0 (fst i N)  
  unfolding no-labels.Red-Inf-G-q-def using i0-in3 by auto
qed

next  
show no-labels.Red-Inf-G-q q0 (fst i N) ⊆ to-F i X0 N
proof  
fix i0
  assume i0-in: i0 ∈ no-labels.Red-Inf-G-q q0 (fst i N)
  then have i0-in2: i0 ∈ Inf-F  
  unfolding no-labels.Red-Inf-G-q-def by blast
obtain i0-FL where i0-FL-in: i0-FL ∈ Inf-FL and i0-to-i0-FL: i0 = to-F i0-FL  
  using Inf-F-to-Inf-FL[OF i0-in2] unfolding to-F-def  
  by (metis Ex-list-of-lengthfst-consexhaust-set inference.inject map-fst-zip)
  have concl-swap: fst (concl-of i0-FL) = concl-of i0  
  unfolding concl-of-def i0-to-i0-FL to-F-def by simp
  have subs1: ((G-Inf-L-q q0 i0-FL) ≠ None ∧ the ((G-Inf-L-q q0 i0-FL) ⊆ Red-Inf-q q0 (labeled-ord-red-crit-fam.G-set-q q0 N)) ∨ ((G-Inf-L-q q0 i0-FL = None) ∧ G-F-L-q q0 (concl-of i0-FL) ⊆ (labeled-ord-red-crit-fam.G-set-q q0 N ∪ Red-F-q q0 (labeled-ord-red-crit-fam.G-set-q q0 N))))  
  using i0-in i0-to-i0-FL concl-swap  
  labeled-ord-red-crit-fam.G-set-q-def G-F-L-q-def  
  by simp
  then have i0-FL ∈ labeled-ord-red-crit-fam.Red-Inf-G-q q0 N
  using i0-FL-in unfolding labeled-ord-red-crit-fam.Red-Inf-G-q-def  
  by simp
  then show i0 ∈ to-F i X0 N
    using x0-is i0-to-i0-FL i0-in2 by blast
qed

qed

then have Y ∈ no-labels.Red-Inf-G-q ' UNIV ⇒ (to-F i) ∈ Y (fst i N) for Y
  using i0-in2 no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q-def red-inf-eqv2 red-inf-impl by fastforce
then show (to-F i) ∈ no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fst i N)
  unfolding labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-Inf-Q-def
  no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q-def
  by blast
qed

moreover have (to-F i) ∈ no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fst i N) ⇒ i ∈ labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-Inf-Q N
proof –
  assume to-F-in: to-F i ∈ no-labels.empty-ord-lifted-calc-w-red-crit-family.Red-Inf-Q (fst i N)
have \( \text{imp-to-F}: X \in \text{no-labels}. \text{Red-Inf-}G\cdot q \leftrightarrow \text{UNIV} \implies \text{to-F} \ i \in X \ (\text{fst} \ i \ N) \ \text{for} \ X \)

using \( \text{to-F-in unfolding} \ \text{no-labels. empty-ord-lifted-calc-w-red-crit-family}. \text{Red-Inf-}Q\cdot \text{def by blast} \)
then have \( \text{to-F-in2}: \text{to-F} \ i \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q (\text{fst} \ i \ N) \ \text{for} \ q \)
by fast
have \( \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q \ N = \{\text{i0-FL} \in \text{Inf-FL}. \ \text{to-F} \ \text{i0-FL} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q (\text{fst} \ i \ N)\} \ \text{for} \ q \)
proof
\( \text{show} \ \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q \ N \subseteq \{\text{i0-FL} \in \text{Inf-FL}. \ \text{to-F} \ \text{i0-FL} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q (\text{fst} \ i \ N)\} \)
proof
fix \( q0 \ i1 \)
assume
\( \text{i1-in}: \text{i1} \in \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q0 \ N \)
have \( \text{i1-in2}: \text{i1} \in \text{Inf-FL} \)
using \( \text{i1-in unfolding} \ \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q\cdot \text{def by blast} \)
then have \( \text{to-F-i1-in}: \text{to-F} \ \text{i1} \in \text{Inf-F} \)
using \( \text{Inf-FL-to-Inf-F unfolding} \ \text{to-F-def by simp} \)
have \( \text{concl-swap}: \text{fst} (\text{concl-of} \ \text{i1}) = \text{concl-of} (\text{to-F} \ \text{i1}) \)
unfolding \( \text{concl-of-def to-F-def by simp} \)
then have \( \text{i1-to-F-in}: \text{to-F} \ \text{i1} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q0 \ (\text{fst} \ i \ N) \)
using \( \text{i1-in to-F-i1-in} \)
unfolding \( \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q\cdot \text{def no-labels}. \text{Red-Inf-}G\cdot q\cdot \text{def} \)
\( \text{G-Inf-L-q-def labeled-ord-red-crit-fam} \text{G-set-q-def no-labels.G-set-q-def G-F-L-q-def} \)
by force
\( \text{show} \ \text{i1} \in \{\text{i0-FL} \in \text{Inf-FL}. \ \text{to-F} \ \text{i0-FL} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q0 \ (\text{fst} \ i \ N)\} \)
using \( \text{i1-in2 i1-to-F-in by blast} \)
qed
next
\( \text{show} \ \{\text{i0-FL} \in \text{Inf-FL}. \ \text{to-F} \ \text{i0-FL} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q (\text{fst} \ i \ N)\} \subseteq \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q \ N \)
proof
fix \( q0 \ i1 \)
assume
\( \text{i1-in}: \text{i1} \in \{\text{i0-FL} \in \text{Inf-FL}. \ \text{to-F} \ \text{i0-FL} \in \text{no-labels}. \text{Red-Inf-}G\cdot q \ q0 \ (\text{fst} \ i \ N)\} \)
then have \( \text{i1-in2}: \text{i1} \in \text{Inf-FL} \ \text{by blast} \)
then have \( \text{to-F-i1-in}: \text{to-F} \ \text{i1} \in \text{Inf-F} \)
using \( \text{Inf-FL-to-Inf-F unfolding} \ \text{to-F-def by simp} \)
have \( \text{concl-swap}: \text{fst} (\text{concl-of} \ \text{i1}) = \text{concl-of} (\text{to-F} \ \text{i1}) \)
unfolding \( \text{concl-of-def to-F-def by simp} \)
then have \( \text{G-Inf-L-q q0 \ i1 = None} \land \text{the} (\text{G-Inf-L-q q0 \ i1) \subseteq Red-Inf-q q0} (\text{labeled-ord-red-crit-fam.G-set-q q0 N}) \)
\( \lor (\text{G-Inf-L-q q0 \ i1 = None}) \land \text{G-F-L-q q0} (\text{concl-of} \ \text{i1}) \subseteq (\text{labeled-ord-red-crit-fam.G-set-q q0 N} \cup \text{Red-F-q q0} (\text{labeled-ord-red-crit-fam.G-set-q q0 N})) \)
using \( \text{i1-in unfolding} \ \text{no-labels. Red-Inf-G-q-def G-Inf-L-q-def} \)
by \text{auto} \)
then show \( \text{i1} \in \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q0 \ N \)
using \( \text{i1-in2 unfolding labeled-ord-red-crit-fam.Red-Inf-G-q-def} \)
by blast
qed
qed
then have \( \text{i} \in \text{labeled-ord-red-crit-fam}. \text{Red-Inf-}G\cdot q \ q \ N \ \text{for} \ q \)
using \( \text{to-F-in2 i-in} \)
proof

moreover have 
qed

ultimately show \( \epsilon \in \text{labeled-ord-crit-fam. empty-ord-lifted-calc-w-red-crit-family. } \text{Red-Inf-Q } N \)

by argo

qed

lemma red-labeled-clauses : \( \forall C \in \text{no-labels. } \text{Red-F-G-empty } (\text{fst } N) \) and \( \exists C' \in \text{(fst } N) \) and \( C \succ C' \) and \( \exists C', L' \in N. (L' \sqsubseteq L \land C \sqsupset C') \)

\( (C, L) \in \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q } N \)

proof -

assume \( C \in \text{no-labels. } \text{Red-F-G-empty } (\text{fst } N) \)
then have \( C \in \text{no-labels. } \text{Red-F-G-empty } q (\text{fst } N) \)

by fast

unfolding \( \text{no-labels. } \text{Red-F-G-empty-def} \)
then have \( \text{g-in-red. } \text{Red-F-q q C } \subseteq \text{Red-F-q q (no-labels. } \text{G-set-q q (fst } N) \) \)

by fast

unfolding \( \text{no-labels. } \text{Red-F-G-empty-q-def} \)

have \( \text{no-labels. } \text{G-set-q q (fst } N) = \text{labeled-ord-crit-fam. } \text{G-set-q q N} \)

by fast

unfolding \( \text{no-labels. } \text{G-set-q-q-def} \)

then have \( \text{G-F-L-q q (C, L) } \subseteq \text{G-F-q q (labeled-ord-crit-fam. } \text{G-set-q q N} \)

by fast

using \( \text{g-in-red. } \text{Red-F-L-q-def} \)

then show \( (C, L) \in \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q } N \)

by fast

unfolding \( \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q-def} \)

by fast

qed

moreover have \( \exists C' \in (\text{fst } N) \).

\( C \succ C' \)

\( (C, L) \in \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q } N \)

proof -

assume \( \exists C' \in (\text{fst } N) \).

then obtain \( C' \) where \( c'-\text{in. } C' \in (\text{fst } N) \)

and \( c'-\text{prec. } C \succ C' \)

by fast

obtain \( L' \) where \( c'\cdot\text{-in. } (C', L') \in N \)

using \( c'\cdot\text{-in-def} \)

have \( c'\cdot\text{-prec. } (C', L') \subseteq (C, L) \)

using \( c'\cdot\text{-prec-def} \)

then have \( \text{G-F-L-q q (C, L) } \subseteq \text{Red-F-q q (C', L') } \)

by fast

unfolding \( \text{no-labels. } \text{G-set-q-q-def} \)

then have \( (C, L) \in \text{labeled-ord-crit-fam. } \text{Red-F-G-q-q-def} \)

by fast

unfolding \( \text{labeled-ord-crit-fam. } \text{Red-F-G-q-q-def} \)

by fast

then show \( (C, L) \in \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q } N \)

by fast

unfolding \( \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q-def} \)

by fast

qed

moreover have \( \exists (C', L') \in N. (L' \sqsubseteq L \land C \succ C') \)

\( (C, L) \in \text{labeled-ord-crit-fam. lifted-calc-w-red-crit-family. } \text{Red-F-Q } N \)
proof
  assume \( \exists (C', L) \in N. (L' \sqsubseteq L \land C \vdash C') \)
  then obtain \( C' \vdash L' \) where \( c'.l'\text{-in}: (C',L') \in N \) and \( l'\text{-sub-l}: L' \sqsubseteq L \) and \( c'.l'\text{-sub-c}: C \vdash C' \)
  by fast
  have \((C,L) \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q} N\) if \( C \vdash C' \)
  using that \( c'.l'\text{-in ii by fastforce} \)
  moreover \{ 
    assume \( \text{equiv-c-c':} C \equiv C' \)
    then have \( \text{equiv-c'-c':} C' \equiv C \)
      using \( \text{equiv-F-is-equiv-rel equiv-F-fun-def equiv-class-eq-iff by fastforce} \)
    then have \( c'.l'\text{-prec}: (C',L') \sqsubseteq (C,L) \)
      using \( l'\text{-sub-l unfolding Prec-FL-def by simp} \)
    have \( G-F-q q C = G-F-q q C' \)
      by \( \text{equiv-F-grounding equiv-c'-c by blast} \)
    then have \( G-F-L-q q (C,L) = G-F-L-q q (C',L') \)
    then have \((C,L) \in \text{labeled-ord-red-crit-fam.Red-F-Q-q-q N}\) for \( q \)
      unfolding \( \text{labeled-ord-red-crit-fam.Red-F-Q-q-def using c'.l'-in c'.l'-prec by blast} \)
    then have \((C,L) \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q} N\)
      unfolding \( \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q-def by blast} \)
  \}
  ultimately show \((C,L) \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q} N\)
    by unfolding \( \text{Prec-eq-F-def equiv-F-fun-def equiv-F-is-equiv-rel by blast} \)
  qed
  ultimately show \((C,L) \in \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q} N\)
    by blast
  qed

end

5.2 Given Clause Architecture

locale Given-Clause = Prover-Architecture-Basis Bot-F Inf-F Bot-G G entails-q Inf-G Red-Inf-q Red-F-q G-F-q G-Inf-q l Inf-FL Equiv-F Prec-F Prec-l 
  for 
  Bot-F :: 'f set and 
  Inf-F :: 'f inference set and 
  Bot-G :: 'g set and 
  Q :: 'q itself and 
  entails-q :: 'q ⇒ ('g set ⇒ 'g set ⇒ bool) and 
  Inf-G :: 'g inference set and 
  Red-Inf-q :: 'q ⇒ ('g set ⇒ 'g set ⇒ bool) and 
  Red-F-q :: 'q ⇒ ('g set ⇒ 'g set) and 
  G-F-q :: 'q ⇒ 'f ⇒ 'g set and 
  G-Inf-q :: 'q ⇒ 'f inference ⇒ 'g inference set option and 
  l :: 'l itself and 
  Inf-FL :: (('f × 'l) inference set) and 
  Equiv-F :: (('f × 'f) set) and 
  Prec-F :: 'f ⇒ 'f ⇒ bool (infix ⇒ 50) and 
  Prec-l :: 'l ⇒ 'l ⇒ bool (infix □ l 50) 
+ fixes 
  active :: 'l 

assumes 
  inf-have-premises: 'f F \in Inf-F \imp length (prems-of 'f F) > 0 and 
  active-minimal: l2 ≠ active \imp active □ l l2 and 
  at-least-two-labels: \( \exists l2. active □ l l2 \) and 

52
\textbf{inf-never-active}: \( \iota \in \text{Inf-FL} \implies \text{snd \ (concl-of \ \iota)} \neq \text{active} \)

\begin{verbatim}
lemma labeled-inf-have-premises: \( \iota \in \text{Inf-FL} \implies \text{set \ (prems-of \ \iota)} \neq \{ \} \)
  using \( \text{inf-have-premises} \ \text{Inf-FL-to-Inf-F} \) by fastforce

definition active-subset :: \( ('f \times 'l) \) set \( \Rightarrow ('f \times 'l) \) set \ where
  \( \text{active-subset} \ M = \{ \text{CL} \in M, \ \text{snd \ CL} = \text{active} \} \)

definition non-active-subset :: \( ('f \times 'l) \) set \( \Rightarrow ('f \times 'l) \) set \ where
  \( \text{non-active-subset} \ M = \{ \text{CL} \in M, \ \text{snd \ CL} \neq \text{active} \} \)

inductive Given-Clause-step :: \( ('f \times 'l) \) set \( \Rightarrow ('f \times 'l) \) set \( \Rightarrow \text{bool} \ ((\text{infix} \ \Rightarrow \text{GC} \ 50) \) where
  \begin{itemize}
    \item \( M \subseteq \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{Red-F-Q} \ (N \cup M') \) \implies
      \( \text{active-subset} \ M' = \{ \} \implies N1 \Rightarrow \text{GC} N2 \)
    \item \( N1 = N \cup M \implies N2 = N \cup M' \implies N \cap M = \{ \} \implies \)
      \( \text{active-subset} \ M' = \{ \} \implies N1 \Rightarrow \text{GC} N2 \)
    \item \( M \subseteq \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{Red-F-Q} \ (N \cup \{ (C, \text{active}) \} \cup M) \) \implies \)
      \( N1 \Rightarrow \text{GC} N2 \)
  \end{itemize}

abbreviation derive :: \( ('f \times 'l) \) set \( \Rightarrow ('f \times 'l) \) set \( \Rightarrow \text{bool} \ (\text{infix} \ \Rightarrow \text{RedL} \ 50) \) where
  \( \text{derive} \equiv \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{inter-red-crit-calculus}.\text{derive} \)

lemma one-step-equiv: \( N1 \Rightarrow \text{GC} N2 \\Rightarrow N1 \Rightarrow \text{RedL} N2 \)
  proof (cases \( N1 N2 \) rule: Given-Clause-step.cases)
    show \( N1 \Rightarrow \text{GC} N2 \Rightarrow N1 \Rightarrow \text{GC} N2 \) by blast

next
fix \( N M M' \)
assume
  \text{ge-step}: \( N1 \Rightarrow \text{GC} N2 \) \ and
  \text{n1-is}: \( N1 = N \cup M \) \ and
  \text{n2-is}: \( N2 = N \cup M' \) \ and
  \text{empty-inter}: \( N \cap M = \{ \} \) \ and
  \text{m-red}: \( M \subseteq \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{Red-F-Q} \ (N \cup M') \) \ and
  \text{active-empty}: \( \text{active-subset} \ M' = \{ \} \)

have \( N1 = N2 \subseteq \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{Red-F-Q} N2 \)
  using \( \text{n1-is} \ n2-is \) \text{empty-inter} \text{m-red} \ by auto

then show \( \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{inter-red-crit-calculus}.\text{derive} N1 N2 \)
  unfolding \( \text{labeled-ord-red-crit-fam}, \text{lifted-calc-w-red-crit-family}.\text{inter-red-crit-calculus}.\text{derive} \) \text{simps} \ by blast

next
fix \( N C L M \)
assume
  \text{ge-step}: \( N1 \Rightarrow \text{GC} N2 \) \ and
  \text{n1-is}: \( N1 = N \cup \{ (C, L) \} \) \ and
  \text{not-active}: \( L \neq \text{active} \) \ and
  \text{n2-is}: \( N2 = N \cup \{ (C, \text{active}) \} \cup M \) \ and
  \text{empty-inter}: \( \{ (C, L) \} \cap N = \{ \} \) \ and
  \text{active-empty}: \( \text{active-subset} \ M = \{ \} \)

have \( (C, \text{active}) \in N2 \) using \( \text{n2-is} \) \ by auto
moreover have \( C \triangleright C \) using \( \text{Prec-eq-F-def equiv-F-is-equiv-rel equiv-class-eq-iff} \) \ by fastforce
moreover have \( \text{active} \ \Box \ L \) using \( \text{active-minimal}[\text{OF not-active}] \) .

\end{verbatim}
ultimately have \(\{(C,L)\} \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q N2}\)
using red-labeled-clauses by blast
moreover have \((C,L) \not\in M \implies N1 - N2 = \{(C,L)\}\) using n1-is n2-is empty-inter not-active by auto
moreover have \((C,L) \in M \implies N1 - N2 = \{\}\) using n1-is n2-is by auto
ultimately have \(N1 - N2 \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q N2}\)
using empty-red-f-equiv[of N2] by blast
then show \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.derive} N1 \\cap N2\)
unfolding \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.derive}.\text{simps}\)
by blast
qed

abbreviation fair :: \('f \times 'l\) set llist \Rightarrow bool where
fair \(\equiv \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.fair}\)

lemma gc-to-red: chain \((\Rightarrow \text{GC})\) D \Rightarrow chain \((\succ\text{Red})\) D
using one-step-equiv Lazy-List-Chain.chain-monono by blast

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

lemma gc-fair: chain \((\Rightarrow \text{GC})\) D \Rightarrow \text{lenght D} > 0 \Rightarrow \text{active-subset (lnth D} 0) = \{\}\ \Rightarrow \text{non-active-subset (Liminf-llist D)} = \{\}\ \Rightarrow \text{fair D}\)
proof –
assume
drive: chain \((\Rightarrow \text{GC})\) D and
\(\text{non-empty:} \text{lenght D} > 0 \text{ and}\\\ \text{init-state:} \text{active-subset (lnth D} 0) = \{\}\ \text{and}\\\ \text{final-state:} \text{non-active-subset (Liminf-llist D)} = \{\}\)
show fair D
unfolding \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.fair-def}\)
proof
fix \(i\)
assume i-in: \(i \in \text{with-labels.Inf-from (Liminf-llist D)}\)
have i-in-inf-L: \(i \in \text{Inf-FL using i-in unfolding with-labels.Inf-from-def by blast}\)
have \(\text{Liminf-llist D} = \text{active-subset} (\text{Liminf-llist D})\)
using final-state unfolding non-active-subset-def active-subset-def by blast
then have i-in2: \(i \in \text{with-labels.Inf-from (active-subset (Liminf-llist D)) using i-in by simp}\)
define \(m\) where \(m = \text{lenght} \ (\text{prems-of} \ i)\)
then have m-def-F: \(m = \text{lenght} \ (\text{prems-of} \ (\text{to-F} i))\) unfolding to-F-def by simp

54
have {i-in-F': to-F i ∈ Inf-F}
  using i-in Inf-FL-to-Inf-F unfolding with-labels.Inf-from-def to-F-def by blast
then have m-pos: m > 0 using m-def-F using inf-have-premises by blast
have exist-nj: ∀j ∈ {0..<m}. (∃nj. enat (Suc nj) < llength D ∧ (prems-of i)!j ∉ active-subset (lnth D nj)) ∧
  (∀k. k > nj → enat k < llength D → (prems-of i)!j ∈ active-subset (lnth D k))
proof clarify
  fix j
  assume j-in: j ∈ {0..<m}
  then obtain C where c-is: (C, active) = (prems-of i)!j
    using i-in2 unfolding m-def with-labels.Inf-from-def active-subset-def
    by (smt Collect-mem-eq Collect-mono-iff LeastLessThan_iff nth_mem old.prod.exhaust smd-cone)
  then have (C, active) ∈ Liminf-llist D
    using j-in i-in unfolding m-def with-labels.Inf-from-def by force
  then obtain nj where nj-is: enat nj < llength D and
    c-in2: (C, active) ∈ ∩ (lnth D i {k. nj ≤ k ∧ enat k < llength D})
unfolding Liminf-llist-def using init-state by blast
then have c-in3: ∀k. k ≥ nj → enat k < llength D → (C, active) ∈ (lnth D k) by blast
have nj-pos: nj > 0 using init-state-c-in2 nj-is unfolding active-subset-def by fastforce
obtain nj-min where nj-min-is: nj-min = (LEAST nj. enat nj < llength D ∧ (C, active) ∈ (lnth D i {k. nj ≤ k ∧ enat k < llength D})) by blast
then have in-allk: ∀k. k ≥ nj-min → enat k < llength D → (C, active) ∈ (lnth D 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 D
  using nj-min-is
  by (smt LeastI-ex thesis. (∃nj. ¬ [enat nj < llength D; (C, active) ∈ ∩ (lnth D i {k. nj ≤ k ∧ enat k < llength D})]) → thesis) → thesis)
have nj-min > 0
  using nj-is c-in2 nj-pos nj-min-is
  by (metis (mono-tags, lifting) Collect-empty-eq (∃C. active) ∈ Liminf-llist D)
  (Liminf-llist D = active-subset (Liminf-llist D))
  (∀k≥nj-min. enat k < llength D → (C, active) ∈ lnth D k) active-subset-def init-state
  linorder-not-less_mem-Collect-eq non-empty zero-enat-def
then obtain njm-prec where njm-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-smaller-d: njm-prec < llength D
  using HOL.no-atp(15)[OF njm-smaller-D njm-prec-njm-enat].
have njm-prec-all-suc: ∀k>njm-prec. enat k < llength D → (C, active) ∈ lnth D k
  using njm-prec-is in-allk by simp
have notin-njm-prec: (C, active) ∉ lnth D njm-prec
proof (rule ccontr)
  assume ¬ (C, active) ∉ lnth D njm-prec
  then have absurd-hyp: (C, active) ∈ lnth D njm-prec by simp
  have prems-smaller: enat njm-prec < llength D using njm-min-is njm-prec-is
    by (smt LeastI-ex Suc-leD ¬ thesis. (∃nj. ¬ [enat nj < llength D; (C, active) ∈ ∩ (lnth D i {k. nj ≤ k ∧ enat k < llength D})]) → thesis) → thesis)
enat-ord-simps(1) le-eq-less-or-eq le-less-trans
have (C, active) ∈ ∩ (lnth D i {k. njm-prec ≤ k ∧ enat k < llength D})
proof
  { fix k
  assume k-in: njm-prec ≤ k ∧ enat k < llength D
  have k = njm-prec ⊢ (C, active) ∈ lnth D k using absurd-hyp by simp
}

55
moreover have njm-prec < k \implies (C,active) \in \text{lth} D k
  using nj-prec-is in-alkk k-in by simp
ultimately have (C,active) \in \text{lth} D k using k-in by fastforce

\[
\text{then show } (C,active) \in \bigcap (\text{lth} D \cdot \{k. \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D\}) \text{ by blast}
\]

qed

then have enat njm-prec < \text{llength} D \land
  (C,active) \in \bigcap (\text{lth} D \cdot \{k. \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D\})
  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-sub-njm-prec: (C, active) \notin \text{active-subset} (\text{lth} D \text{njm-prec})

unfolding active-subset-def by blast

then show \exists nj. \text{enat} (Suc nj) < \text{llength} D \land (\text{prems-of} \ i)!j \notin \text{active-subset} (\text{lth} D nj) \land
  (\forall k. k > nj \implies \text{enat} k < \text{llength} D \implies (\text{prems-of} \ i)!j \in \text{active-subset} (\text{lth} D 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-cone)

qed

have uniq-nj: j \in \{0..<m\} \implies
  (\text{enat} (Suc nj1) < \text{llength} D \land
  (\text{prems-of} \ i)!j \notin \text{active-subset} (\text{lth} D nj1) \land
  (\forall k. k > nj1 \implies \text{enat} k < \text{llength} D \implies (\text{prems-of} \ i)!j \in \text{active-subset} (\text{lth} D k))) \implies
  (\text{enat} (Suc nj2) < \text{llength} D \land
  (\text{prems-of} \ i)!j \notin \text{active-subset} (\text{lth} D nj2) \land
  (\forall k. k > nj2 \implies \text{enat} k < \text{llength} D \implies (\text{prems-of} \ i)!j \in \text{active-subset} (\text{lth} D k))) \implies nj1=nj2

proof (clarify, rule contr)

fix j nj1 nj2

assume j \in \{0..<m\} and
  nj1-d: \text{enat} (Suc nj1) < \text{llength} D and
  nj2-d: \text{enat} (Suc nj2) < \text{llength} D and
  nj1-notin: \text{prems-of} \ i! j \notin \text{active-subset} (\text{lth} D nj1) and
  k-nj1: \forall k>nj1. \text{enat} k < \text{llength} D \implies \text{prems-of} \ i! j \in \text{active-subset} (\text{lth} D k) and
  nj2-notin: \text{prems-of} \ i! j \notin \text{active-subset} (\text{lth} D nj2) and
  k-nj2: \forall k>nj2. \text{enat} k < \text{llength} D \implies \text{prems-of} \ i! j \in \text{active-subset} (\text{lth} D k) and
  diff-12: nj1 \neq nj2

have nj1 < nj2 \implies False

proof –
  assume prec-12: nj1 < nj2
  have enat nj2 < \text{llength} D using nj2-d using Suc-ile-eq less-trans by blast
  then have \text{prems-of} \ i! j \in \text{active-subset} (\text{lth} D nj2)
    using k-nj1 prec-12 by simp
  then show False using nj2-notin by simp

qed

moreover have nj1 > nj2 \implies False

proof –
  assume prec-21: nj2 < nj1
  have enat nj1 < \text{llength} D using nj1-d using Suc-ile-eq less-trans by blast
  then have \text{prems-of} \ i! j \in \text{active-subset} (\text{lth} D nj1)
    using k-nj2 prec-21

  then show False using nj1-notin by simp

qed

ultimately show False using diff-12 by linarith

qed
have \(\forall j \in \{0..<m\}. \text{enat } (\text{Suc } nj) < \text{llength } D \land (\text{prems-of } i)[j \notin \text{active-subset } (\text{lnth } D \text{ nj}) \land (\forall k. k > nj \to \text{enat } k < \text{llength } D \to (\text{prems-of } i)[j \in \text{active-subset } (\text{lnth } D \text{k})])\)
then have \(\text{nj-not-empty: } \text{nj-set} \neq \{\}\)

proof ~
have zero-in: \(0 \in \{0..<m\}\) using m-pos by simp
then obtain \(n0\) where \(\text{enat } (\text{Suc } n0) < \text{llength } D \land (\text{prems-of } i)[0 \notin \text{active-subset } (\text{lnth } D n0) \land (\forall k>n0. \text{enat } k < \text{llength } D \to (\text{prems-of } i)[0 \in \text{active-subset } (\text{lnth } D k))\)
using exist-nj by fast
then have \(n0 \in \text{nj-set}\) unfolding \(\text{nj-set-def}\) using zero-in by blast
then show \(\text{nj-set} \neq \{\}\) by auto
qed

have \(\exists n \in \text{nj-set}. \forall nj \in \text{nj-set. } nj \leq n\)
using \(\text{nj-not-empty}\) \(\text{nj-finite}\) using \(\text{Max-ge}\) \(\text{Max-in}\) by blast
then obtain \(n\) where \(\text{n-in: } n \in \text{nj-set}\) and \(\text{n-bigger: } \forall nj \in \text{nj-set. } nj \leq n\) by blast
then obtain \(j0\) where \(j0\)-in: \(j0 \in \{0..<m\}\) and \(\text{suc-n-length: } \text{enat } (\text{Suc } n) < \text{llength } D \land (\text{nj-notin: } \text{prems-of } i)[j0 \notin \text{active-subset } (\text{lnth } D n) \land \text{nj-allin: } (\forall k. k > n \to \text{enat } k < \text{llength } D \to (\text{prems-of } i)[j0 \in \text{active-subset } (\text{lnth } D k))\)
unfolding \(\text{nj-set-def}\) by blast
obtain \(C0\) where \(C0\)-is: \(\text{prems-of } i)[j0 = (C0,\text{active})\) using \(\text{j0-in}\)
using \(i\in2\) unfolding \(m\)-def with-labels.\(\text{Inf-from-def}\) active-subset-def-set
by (smt \(\text{Collect-mem}\) \(\text{Collect-mono-iiff}\) \(\text{atLeastLessThan-iiff}\) \(\text{nth-mem}\) \(\text{old.prod}\) \(\text{exhaust}\) \(\text{snd-cone}\)
then have \(\text{C0-prems-i\!: } (C0,\text{active}) \in \text{set } (\text{prems-of } i)\) using \(\text{in-set-conv-nth}\) \(\text{j0-in}\) \(m\)-def by force
have \(\text{C0-in: } (C0,\text{active}) \in (\text{lnth } D (\text{Suc } n))\)
using \(\text{C0-is}\) \(\text{j0-allin}\) \(\text{suc-n-length}\) by (simp add: active-subset-def-set)
have \(\text{C0-notin: } (C0,\text{active}) \notin (\text{lnth } D n)\) using \(\text{C0-is}\) \(\text{j0-notin}\) unfolding active-subset-def-set by simp
have \(\text{step-n\!: } \text{lnth } D n \to\text{GC } \text{lnth } D (\text{Suc } n)\)
using deriv chain-lnth-rel \(n\)-in unfolding active-subset-def-set by blast
have \(\exists N C L M. (\text{lnth } D n = N \cup \{(C,L)\} \land \{(C,L)\} \cap N = \{\} \land \text{lnth } D (\text{Suc } n) = N \cup \{(C,\text{active})\} \cup M \land \langle M \neq \text{active} \land \text{active-subset } M = \{\}\) \land \(\text{no-labels}\).\(\text{Non-ground}\).\(\text{Inf-from2}\) \(\langle\text{fst} \cdot (\text{active-subset } N)\rangle \subseteq \text{no-labels.\text{lft-calc-w-red-crit-family.\text{Red-Inf-Q}} (\text{fst} \cdot (N \cup \{(C,\text{active})\} \cup M))\)
proof ~
have \(\text{proc-or-infer: } (\exists N1 N M N2 M'). \text{lnth } D n = N1 \land \text{lnth } D (\text{Suc } n) = N2 \land N1 = N \cup M \land N2 = N \cup M' \land M \cap N = \{\} \land M \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.\text{Red-F-Q} (N \cup M') \land \text{active-subset } M' = \{\}) \land (\exists N1 N C L N2 M. (\text{lnth } D n = N1 \land \text{lnth } D (\text{Suc } n) = N2 \land N1 = N \cup \{(C, L)\} \land (\{(C, L)\} \cap N = \{\} \land N2 = N \cup \{(C, \text{active})\} \cup M \land L \neq \text{active} \land \text{active-subset } M = \{\} \land \text{no-labels}.\text{Non-ground}.\text{Inf-from2} (\text{fst} \cdot (\text{active-subset } N)) \subseteq \text{no-labels.\text{lft-calc-w-red-crit-family.\text{Red-Inf-Q}} (\text{fst} \cdot (N \cup \{(C,\text{active})\} \cup M))\)
using Given-Clause-step-simps[\langle\text{lnth } D n \text{ lnth } D (\text{Suc } n)\rangle \text{ step-n}\) by blast
show \(\text{thesis}\)
using \(\text{C0-in}\) \(\text{C0-notin}\) \(\text{proc-or-infer}\) \(\text{j0-in}\) \(\text{C0-is}\)
by (smt \(\text{Un-iff}\) active-subset-def mem-Collect-eq snd-conv sup-bot.right-neutral)
qed
then obtain \(N M L\) where \(\text{inf-from-sub}\):
no-labels.Non-ground.Inf-from2 (fst · (active-subset N)) \{ C0 \} ⊆
no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q (fst · (N ∪ \{(C0,active)\} ∪ M)) and

nth-d-is: lnth D n = N ∪ \{(C0,L)\} and

suc-nth-d-is: lnth D (Suc n) = N ∪ \{(C0,active)\} ∪ M and

l-not-active: L ≠ active

using C0-in C0-notin j0-in C0-is using active-subset-def by fastforce

have j ∈ \{0..<m\} \implies (prems-of i)!j ≠ (prems-of i)!j0 \implies (prems-of i)!j ∈ (active-subset N) for j

proof –

fix j

assume j-in: j ∈ \{0..<m\} and

j-not-j0: (prems-of i)!j ≠ (prems-of i)!j0

obtain nj where nj-len: enat (Suc nj) < llength D and

nj-prems: (prems-of i)!j ∈ active-subset (lnth D nj) and

nj-greater: (\forall k. k > nj \implies enat k < llength D \implies (prems-of i)!j ∈ active-subset (lnth D 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 i)!j = C0,active

using C0-in C0-notin Given-Clause-step.simps[of lnth D n lnth D (Suc n)] step-n

by (smt Un-iff Un-insert-right nj-greater nj-prems active-subset-def empty-Collect-eq
insertE lessI mem-Collect-eq prod.sel(2) suc-n-length)

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 i)!j ∈ active-subset (lnth D n)

using nj-greater n-in Suc-ile-eq dual-order.strict-implies-order unfolding nj-set-def by blast

then show (prems-of i)!j ∈ (active-subset N)

using nth-d-is l-not-active unfolding active-subset-def by force

qed

then have set (prems-of i) ⊆ 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 i) ⊆ active-subset N − \{(C0, active)\}) using C0-prems-i by blast

ultimately have i ∈ with-labels.Inf-from2 (active-subset N) \{(C0,active)\}

using i-in-inf-fl unfolding with-labels.Inf-from2-def with-labels.Inf-from-def by blast

then have to-F i ∈ no-labels.Non-ground.Inf-from2 (fst · (active-subset N)) \{ C0 \}

unfolding to-F-def with-labels.Inf-from2-def with-labels.Inf-from-def


by force

then have to-F i ∈ no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q (fst · (lnth D (Suc n)))

using suc-nth-d-is inf-from-sub by fastforce

then have \forall q. (G-Inf-q q (to-F i) ≠ None ∧

the (G-Inf-q q (to-F i)) ⊆ Red-Inf-q q (\{ (G-F-q q · (fst · (lnth D (Suc n)))) \}))

\forall (G-Inf-q q (to-F i)) = None ∧

G-F-q q (concl-of (to-F i)) ⊆ (\{ (G-F-q q · (fst · (lnth D (Suc n)))) \})

\cup Red-F-q q (\{ (G-F-q q · (fst · (lnth D (Suc n)))) \})

unfolding to-F-def no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q-def


by fastforce

then have i ∈ with-labels.Red-Inf-Q (lnth D (Suc n))


G-F-L-q-def using i-in-inf-fl by auto

58
then show $i \in$
labeled-ord-red-crit-fam. empty-ord-lifted-calc-w-red-crit-family. inter-red-crit-calculus. Sup-Red-Inf-list

D

unfolding
labeled-ord-red-crit-fam. empty-ord-lifted-calc-w-red-crit-family. inter-red-crit-calculus. Sup-Red-Inf-list-def

using red-inf-equiv2 suc-n-length by auto

qed

theorem gc-complete: chain ($\implies$GC) D $\implies$ llengh D $> 0$ $\implies$ active-subset (lnth D 0) = {} $\implies$
non-active-subset (Lliminf-llist D) = {} $\implies$ B $\in$ Bot-F $\implies$

\exists i. enat i $<$ llengh D $\land$ (\exists BL $\in$ Bot-FL. BL $\in$ (lnth D i))

proof

fix B
assume

deriv: chain ($\implies$GC) D and

not-empty-d: llengh D $> 0$ and

init-state: active-subset (lnth D 0) = {} and

final-state: non-active-subset (Lliminf-llist D) = {} and

b-in: B $\in$ Bot-F and

bot-entailed: no-labels.entails-G-Q (fst ' (lnth D 0)) \{ B \}

have labeled-b-in: (B, active) $\in$ Bot-FL unfolding Bot-FL-def using b-in by simp

have not-empty-d2: $\neg$ lnull D using not-empty-d by force

have labeled-bot-entailed: entails-G-L-Q (lnth D 0) \{(B, active)\}

using labeled-entail-lifting-bot-entailed by fastforce

have fair D using gc-fair[OF deriv not-empty-d init-state final-state] .

then have \exists i $\in$ \{ i. enat i $<$ llengh D \}. \exists BL $\in$ Bot-FL. BL $\in$ lnth D i

using labeled-ordered-dynamic-ref-comp labeled-b-in not-empty-d2 gc-to-red[OF deriv]

labeled-bot-entailed entail-equiv

unfolding dynamic-refutational-complete-calculus-def
dynamic-refutational-complete-calculus-axioms-def by blast

then show ?thesis by blast

qed

end

5.3 Lazy Given Clause Architecture

locale Lazy-Given-Clause = Prover-Architecture-Basis Bot-F Inf-F Bot-G Q entails-q Inf-G Red-Inf-q

Red-Inf-q G-F-q G-Inf-q l Inf-FL Equiv-F Prec-F Prec-l

for

Bot-F :: 'f set and
Inf-F :: 'f inference set and
Bot-G :: 'g set and
Q :: 'q itself and
entails-q :: 'q $\Rightarrow$ ('g set $\Rightarrow$ 'g set $\Rightarrow$ bool) and
Inf-G :: 'q inference set and
Red-Inf-q :: 'q $\Rightarrow$ ('g set $\Rightarrow$ 'g inference set) and
Red-F-q :: 'q $\Rightarrow$ ('g set $\Rightarrow$ 'g set) and
G-F-q :: 'q $\Rightarrow$ 'f $\Rightarrow$ 'g set and
G-Inf-q :: 'q $\Rightarrow$ 'f inference $\Rightarrow$ 'g inference set option and
l :: 'l itself and
Inf-FL :: ('f $\times$ 'l) inference set and
Equiv-F :: ('f $\times$ 'f) set and
\[\text{Prec-}F :: \forall \text{ } f \Rightarrow f \Rightarrow \text{bool (infix }\Rightarrow \text{ 50) and} \]
\[\text{Prec-l :: } l \Rightarrow l \Rightarrow \text{bool (infix }\subseteq \text{l 50)}\]

**fixes**

\[\text{active :: } l\]

**assumes**

\[\text{active-minimal: } l_2 \neq \text{active }\implies \text{active }\subseteq l \subseteq l_2 \text{ and} \]

\[\text{at-least-two-labels: } \exists l_2. \text{active }\subseteq l \subseteq l_2 \text{ and} \]

\[\text{inf-never-active: } i \in \text{Inf-FL }\implies \text{snd (concl-of } i) \neq \text{active}\]

**begin**

\[\text{definition active-subset :: } (f \times l) \text{ set }\Rightarrow (f \times l) \text{ set where} \]
\[\text{ active-subset } M = \{ CL \in M. \text{snd } CL = \text{active}\}\]

\[\text{definition non-active-subset :: } (f \times l) \text{ set }\Rightarrow (f \times l) \text{ set where} \]
\[\text{ non-active-subset } M = \{ CL \in M. \text{snd } CL \neq \text{active}\}\]

**inductive** \Lzy-
\[\text{Lazy-Given-Clause-step :: } (f \text{ inference set}) \times ((f \times l) \text{ set }) \Rightarrow\]
\[\text{ (f inference set) }\times ((f \times l) \text{ set }) \Rightarrow \text{bool (infix }\Rightarrow \text{LGC 50) where} \]

\[\text{process: } N_1 = N \cup M \implies N_2 = N \cup M' \implies N \cap M = \{\} \implies\]
\[\text{active-subset } M' = \{\} \implies (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2)\]

\[\text{schedule-infer: } T_2 = T_1 \cup T' \implies N_1 = N \cup \{(C, L)\} \implies \{(C, L)\} \cap N = \{\} \implies N_2 = N \cup\]
\[\{(C, \text{active})\} \implies\]
\[\text{L }\neq \text{active }\implies T' = \text{no-labels. Non-ground.Inf-from2 (fst }\cdot\text{ (active-subset } N)) \{C\} \implies\]
\[\text{(T}_1, N_1) \Rightarrow \text{LGC } (T_2, N_2)\]

\[\text{compute-infer: } T_1 = T_2 \cup \{i\} \implies T_2 \cap \{i\} = \{\} \implies N_2 = N_1 \cup M \implies \text{active-subset } M = \{\} \implies\]
\[\text{active-subset } M' = \{\} \implies (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2)\]

\[\text{delete-orphans: } T_1 = T_2 \cup T' \implies T_2 \cap T' = \{\} \implies\]
\[\text{T'} \cap \text{no-labels. Non-ground.Inf-from2 (fst }\cdot\text{ (active-subset } N)) = \{\} \implies (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2)\]

**abbreviation** derive :: \( (f \times l) \text{ set }\Rightarrow (f \times l) \text{ set }\Rightarrow \text{bool (infix }\Rightarrow \text{RedL 50) where} \)
\[\text{derive }\equiv \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.derive}\]

**lemma** premise-free-inf-always-from: \( i \in \text{Inf-F }\Rightarrow \text{length (prems-of } i) = 0 \implies\)
\[\text{i }\in\text{no-labels. Non-ground.Inf-from } N\]

**unfolding** no-labels. Non-ground.Inf-from-def by simp

**lemma** one-step-equiv: \( (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2) \Rightarrow N_1 \Rightarrow \text{RedL } N_2\)
**proof** (cases \( (T_1, N_1) \) \( (T_2, N_2) \) rule: \text{Lazy-Given-Clause-step.cases)\)

\[\text{show } (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2) \Rightarrow (T_1, N_1) \Rightarrow \text{LGC } (T_2, N_2) \text{ by blast}\]

**next**

**fix N M M'**

**assume**

\[\text{n1-is: } N_1 = N \cup M \text{ and}\]
\[\text{n2-is: } N_2 = N \cup M' \text{ and}\]
\[\text{empty-inter: } N \cap M = \{\} \text{ and}\]
\[\text{m-red: } M \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family. Red-F-Q } (N \cup M')\]

**have** \( N_1 - N_2 \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family. Red-F-Q } N_2\)

**using** n1-is n2-is empty-inter m-red by auto

**then** \( N_1 \Rightarrow \text{RedL } N_2\)

**unfolding** labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.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 \(\text{auto}\)
moreover have \(C \preceq C\) using \(\text{Prec-eq-F-def equiv-F-is-equiv rel equiv-class-eq-iff}\) by \(\text{fastforce}\)
moreover have \(active \models L\) using \(\text{active-minimal[OF not-active]}\)
ultimately have \(\{(C, L)\} \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q N2}\)
using \(\text{red-labeled-clauses by blast}\)
then have \(N1 \supseteq N2\) using \(\text{empty-red-f-equiv[of N2]}\) using \(n1-is\) \(n2-is\) by \(\text{blast}\)
then show \(N1 \Rightarrow RedL N2\)
unfolding \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.derive.simps by blast}\)

next
fix \(M\)
assume
\( n2-is: N2 = N1 \cup M \)

have \(N1 - N2 \subseteq \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.Red-F-Q N2}\)
using \(n2-is\) by \(\text{blast}\)
then show \(N1 \Rightarrow RedL N2\)
unfolding \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.derive.simps by blast}\)

qed

abbreviation fair :: \((f \times l)\) set llist \Rightarrow bool where
fair \(\equiv \text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.fair}\)

lemma lgc-to-red: \(\Longrightarrow\text{LGC} \) \(D \Longrightarrow\text{chain (\(\Rightarrow\text{RedL}\)) (\text{lmap snd D})}\)
using \(\text{one-step-equiv Lazy-List-Chain.chain-mono by (smt chain-lmap prod-collapse)}\)

lemma lgc-fair: \(\Longrightarrow\text{LGC} \) \(D \Longrightarrow llength D > 0 \Longrightarrow \text{active-subset (snd (\text{lnth D 0})) = \{\} \Longrightarrow \text{non-active-subset (\text{Liminf-llist (lmap snd D)}) = \{\} \Rightarrow (\forall i \in \text{Inf-F}. length (\text{prems-of i}) = 0 \Longrightarrow i \in (\text{fst (\text{lnth D 0})})) \Rightarrow \text{Liminf-llist (lmap fst D)}) = \{\} \Longrightarrow fair (\text{lmap snd D})}\)
proof –
assume
deriv: \(\Longrightarrow\text{LGC} \) \(D\) and
non-empty: \(llength D > 0\) and
init-state: \(\text{active-subset (snd (\text{lnth D 0})) = \{\} \) and
final-state: \(\text{non-active-subset (\text{Liminf-llist (lmap snd D)}) = \{\} \) and
no-prems-init-active: \(\forall i \in \text{Inf-F}. length (\text{prems-of i}) = 0 \Longrightarrow i \in (\text{fst (\text{lnth D 0})})\) and
final-schedule: \(\text{Liminf-llist (lmap fst D)}) = \{\}
show fair (\text{lmap snd D})
unfolding \(\text{labeled-ord-red-crit-fam.lifted-calc-w-red-crit-family.inter-red-crit-calculus.fair-def}\)
proof
```plaintext
fix i
assume i-in: i ∈ with-labels.Inf-from (Liminf-llist (lmap snd D))
have i-in-inf-F: i ∈ Inf-FL using i-in unfolding with-labels.Inf-from-def by blast
have Liminf-llist (lmap snd D) = active-subset (Liminf-llist (lmap snd D))
  using final-state unfolding non-active-subset-def active-subset-def by blast
then have i-in2: i ∈ with-labels.Inf-from (active-subset (Liminf-llist (lmap snd D)))
  using i-in by simp
define m where m = length (prems-of i)
then have m-def-F: m = length (prems-of (to-F i)) unfolding to-F-def by simp
have i-in-F: to-F i ∈ Inf-F
  using i-in Inf-FL-to-Inf-F unfolding with-labels.Inf-from-def to-F-def by blast
have exist-nj: ∀ j ∈ {0…<m}. (∃ nj. enat (Suc nj) < length D ∧
  (prems-of i)!j ∉ active-subset (snd (lnth D nj)) ∧
  (∀ k, k > nj → enat k < length D → (prems-of i)!j ∈ active-subset (snd (lnth D k))))
proof clarify
fix j
assume j-in: j ∈ {0…<m}
then obtain C where c-is: (C, active) = (prems-of i)!j
  using i-in unfolding m-def with-labels.Inf-from-def active-subset-def
by (smt Collect-mem-eq Collect-mono iff LeastLessThan iff nth-mem old.prod.exhaust snd-conv)
then have (C, active) ∈ Liminf-llist (lmap snd D)
  using i-in i-in unfolding m-def with-labels.Inf-from-def by force
then obtain nj where nj-is: enat nj < length D and
  c-in2: (C, active) ∈ (∃ snd (lnth D τ k. nj ≤ k ∧ enat k < length D))
liminfLiminf-llist-def using init-state by fastforce
then have c-in3: ∀ k, k ≥ nj → enat k < length D → (C, active) ∈ snd (lnth D k)
  by blast
have nj-pos: nj > 0 using init-state c-in2 nj-is unfolding active-subset-def by fastforce
obtain nj-min where nj-min-is: nj-min = (LEAST nj. enat nj < length D ∧
  (C, active) ∈ (∃ snd (lnth D τ k. nj ≤ k ∧ enat k < length D)))
  by blast
then have in-allk: ∀ k, k ≥ nj-min → enat k < length D → (C, active) ∈ snd (lnth D k)
  using c-in3 nj-is c-in2 INT-E LeastI-ex
by (smt INT-iff INT-simps(10) c-is image-eqI mem-Collect-eq)
have njm-smaller-D: enat nj-min < length D
  using nj-min-is
by (smt LeastI-ex "thesis. (∃ nj. [enat nj < length D;
  (C, active) ∈ (∃ snd (lnth D τ k. nj ≤ k ∧ enat k < length D))] → thesis) → thesis)
have nj-min > 0
  using nj-is c-in2 nj-pos nj-min-is
by (metis (mono-tags, lifting) active-subset-def emptyE in-allk init-state mem-Collect-eq
  non-empty not-less snd-conv zero-enat-def)
then obtain njm-prec where nj-prec-is: Suc njm-prec = nj-min using gr0-conv-Suc by auto
then have njm-prec-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 < length D
  using HOL.no-atp(15)[OF njm-smaller-D njm-prec-njm-enat]
have njm-prec-all-suc: ∀ k>njm-prec. enat k < length D → (C, active) ∈ snd (lnth D k)
  using njm-prec-is in-allk by simp
have notin-njm-prec: (C, active) ∉ snd (lnth D njm-prec)
proof (rule ccontr)
assume ¬ (C, active) ∉ snd (lnth D njm-prec)
then have absurd-hyp: (C, active) ∈ snd (lnth D njm-prec) by simp
have prec-smaller: enat njm-prec < length D using nj-min-is njm-prec-is
  by (smt LeastI-ex Suc-leD "thesis. (∃ nj. [enat nj < length D;
  (C, active) ∈ (∃ snd (lnth D τ k. nj ≤ k ∧ enat k < length D))] → thesis) → thesis
  enat-ord-simps(1) le-ord-le-or-or-ord le-less-trans)
```

\begin{verbatim}

have \((C, \text{active}) \in \bigcap (\text{snd} \cdot (\text{lnth} D \cdot \{k. \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D\}))\)
  proof -
  \{ 
  \begin{align*}
    & \text{fix } k \\
    & \text{assume } k\text{-in: } \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D \\
    & \text{have } k = \text{njm-prec} \implies (C, \text{active}) \in \text{snd} (\text{lnth} D \ k) \text{ using } \text{absurd-hyp by simp} \\
    & \text{moreover have } \text{njm-prec} < k \implies (C, \text{active}) \in \text{snd} (\text{lnth} D \ k) \\
    & \text{using } \text{nj-prec-is in-allk } k\text{-in by simp} \\
    & \text{ultimately have } (C, \text{active}) \in \text{snd} (\text{lnth} D \ k) \text{ using } k\text{-in by fastforce} \\
  \} \\
  \text{then show } (C, \text{active}) \in \bigcap (\text{snd} \cdot (\text{lnth} D \cdot \{k. \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D\})) \\
  \text{by blast} \\
  \text{qed} \\
  \text{then have } \text{enat } \text{njm-prec} < \text{llength} D \land \\
  (C, \text{active}) \in \bigcap (\text{snd} \cdot (\text{lnth} D \cdot \{k. \text{njm-prec} \leq k \land \text{enat} k < \text{llength} D\})) \\
  \text{using } \text{nj-pre smaller by blast} \\
  \text{then show } \text{False} \\
  \text{using } \text{nj-min-is nj-prec-is Orderings.wellorder-class.not-less-Least njm-prec-njm by blast} \\
  \text{qed} \\
  \text{then have } \text{not in active subs njm-prec: } (C, \text{active}) \notin \text{active-subset} (\text{snd} (\text{lnth} D \ \text{njm-prec})) \\
  \text{unfolding active-subset-def by blast} \\
  \text{then show } \exists n. \text{enat } (\text{Suc } n) < \text{llength } D \land (\text{prems-of } i)! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ n)) \\
  \land (\forall k. k > n \implies \text{enat } k < \text{llength } D \implies (\text{prems-of } i)! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k))) \\
  \text{using } c\text{-is njm-prec-all-suc njm-prec-smaller-d by (metis (mono-tags, lifting) active-subset-def mem-Collect-eq njm-prec-is njm-smaller-D snd-conv)} \\
  \text{qed} \\

define nj-set where nj-set = \{nj. (\exists j \in \{0..<m\}. \text{enat} (\text{Suc } n) < \text{llength } D \land \\
(\text{prems-of } i)! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ n)) \land \\
(\forall k. k > n \implies \text{enat } k < \text{llength } D \implies (\text{prems-of } i)! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k)))\}
  \} \\
  \begin{align*}
    & \text{assume } m\text{-null: } m = 0 \\
    & \text{then have } \text{enat } 0 < \text{llength } D \land \to-F i \in \text{fst} (\text{lnth} D \ 0) \\
    & \text{using } \text{no-prems-init-active i-in-F non-empty m-def-F zero-enat-def by auto} \\
    & \text{then have } \exists n. \text{enat } n < \text{llength } D \land \to-F i \in \text{fst} (\text{lnth} D \ n) \\
    & \text{by blast} \\
  \end{align*} \\
  \text{moreover } \{ 
  \begin{align*}
    & \text{assume } m\text{-pos: } m > 0 \\
    & \text{have uniq-nj: } j \in \{0..<m\} \implies \\
    & (\text{enat } (\text{Suc } n) < \text{llength } D \land \\
     (\text{prems-of } i)! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ n)) \land \\
     (\forall k. k > n \implies \text{enat } k < \text{llength } D \implies (\text{prems-of } i)! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k)))\) \implies \\
     (\text{enat } (\text{Suc } n) < \text{llength } D \land \\
      (\text{prems-of } i)! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ n)) \land \\
      (\forall k. k > n \implies \text{enat } k < \text{llength } D \implies (\text{prems-of } i)! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k)))\) \\
    \text{nj1=nj2} \\
    \text{proof (clarify, rule ccontr) } \\
    \text{fix } j \text{ nj1} \text{ nj2} \\
    \text{assume } j \in \{0..<m\} \text{ and } \\
    \text{nj1-d: enat } (\text{Suc } n) < \text{llength } D \text{ and } \\
    \text{nj2-d: enat } (\text{Suc } n) < \text{llength } D \text{ and } \\
    \text{nj1-notin: prems-of } i \; ! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ nj1)) \text{ and } \\
    \text{k-nj1: } \forall k>nj1. \text{enat } k < \text{llength } D \implies \text{prems-of } i \; ! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k)) \text{ and } \\
    \text{nj2-notin: prems-of } i \; ! j \notin \text{active-subset} (\text{snd} (\text{lnth} D \ nj2)) \text{ and } \\
    \text{k-nj2: } \forall k>nj2. \text{enat } k < \text{llength } D \implies \text{prems-of } i \; ! j \in \text{active-subset} (\text{snd} (\text{lnth} D \ k)) \text{ and }
  \end{align*}
\end{verbatim}

63
diff-12: nj1 ≠ nj2
have nj1 < nj2 ⇒ False
proof –
  assume prec-12: nj1 < nj2
  have enat nj2 < llength D using nj2-d using Suc-ile-eq less-trans by blast
  then have prems-of i ! j ∈ active-subset (snd (lnth D nj2))
    using k-nj1 prec-12 by simp
  then show False using nj2-notin by simp
qed
moreover have nj1 > nj2 ⇒ False
proof –
  assume prec-21: nj2 < nj1
  have enat nj1 < llength D using nj1-d using Suc-ile-eq less-trans by blast
  then have prems-of i ! j ∈ active-subset (snd (lnth D nj1))
    using k-nj2 prec-21 by simp
  then show False using nj1-notin by simp
qed
ultimately show False using diff-12 by linarith
qed
have nj-not-empty: nj-set ≠ {} by auto
proof –
  have zero-in: 0 ∈ {0..<m} using m-pos by simp
  then obtain n0 where enat (Suc n0) < llength D and
    prems-of i ! θ ∈ active-subset (snd (lnth D n0)) and
    ∀ k>n0. enat k < llength D ⇒ prems-of i ! θ ∈ active-subset (snd (lnth D 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 uniq-nj 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 ∃ n ∈ nj-set. ∀ nj ∈ nj-set. nj ≤ n
  using nj-not-empty nj-finite using Max-ge Max-in by blast
then obtain n where n-in: n ∈ nj-set and n-bigger: ∀ nj ∈ nj-set. nj ≤ n by blast
then obtain j0 where j0-in: j0 ∈ {0..<m} and suc-n-length: enat (Suc n) < llength D and
  j0-notin: (prems-of i)!j0 !∈ active-subset (snd (lnth D n)) and
  j0-allin: (∀ k. k > n → enat k < llength D →
    (prems-of i)!j0 ∈ active-subset (snd (lnth D k))
) unfolding nj-set-def by blast
obtain C0 where C0-is: (prems-of i)!j0 = (C0.active)
  j0-in i-in2 unfolding m-def with-labels.Inf-from-def active-subset-def
  by (smt Collect-mem-eq Collect-mono-iff atLeastLessThan iff nth-mem old.prod.exhaust snd-cone)
then have C0-prems-i: (C0.active) ∈ set (prems-of i) using in-set-conv-nth j0-in m-def by force
have C0-in: (C0.active) ∈ (snd (lnth D (Suc n)))
  using C0-is j0-allin suc-n-length by (simp add: active-subset-def)
have C0-notin: (C0.active) !∈ (snd (lnth D n))
  using C0-is j0-notin unfolding active-subset-def by simp
have step-n: lnth D n ⇒ LGC lnth D (Suc n)
  using deriv-chain-lnth-rel n-in unfolding nj-set-def by blast
have is-scheduled: ∃ T2 T1 T′ N1 N C L N2. lnth D n = (T1, N1) ∧ lnth D (Suc n) = (T2, N2)
∧ T2 = T1 ∪ T′ ∧ N1 = N ∪ {(C, L)} ∧ (C, L) ∩ N = {} ∧ N2 = N ∪ {(C, active)} ∧ L ≠ active ∧
\[ T' = \text{no-labels}.\text{Non-ground}.\text{Inf-from2} \left( \text{fst} \circ \text{active-subset} \ N \right) \{ C \} \]
using \text{Lazy-Given-Clause-step}.\text{simp}s[\text{of lnth} \ D \ n \ \text{lnth} \ D \ (\text{Suc} \ n)] \text{step-n} \ C0-in \ C0-notin
unfolding active-subset-def by fastforce

then obtain \( T2 \ T1 \ T' \ N \ L \ N2 \) \text{where nth-d-is: lnth} \ D \ n = (T1, N1) and
\( \text{suc-nth-d-is: lnth} \ D \ (\text{Suc} \ n) = (T2, N2) \) \text{and t2-is:} \ T2 = T1 \cup T' and
\( n1-is: \ N1 = N \cup \{(C0, L)\} \{C0, L\} \cap N = \{ \} \ N2 = N \cup \{(C0, \text{active})\} \) and
l-not-active: \( L \neq \text{active} \) and
tp-is: \( T' = \text{no-labels}.\text{Non-ground}.\text{Inf-from2} \left( \text{fst} \circ \text{active-subset} \ N \right) \{ C0 \} \)
using \( C0-in \ C0-notin \ j0-in \ C0-is \) using active-subset-def by fastforce

have \( j \in \{ 0..<m \} \Rightarrow \) \( \text{prems-of} \ i \| j \neq \) \( \text{prems-of} \ i \| j0 \Rightarrow \) \( \text{prems-of} \ i \| j \in \text{(active-subset} \ N) \) for \( j \)
proof –
fix \( j \)
assume \( j\text{-in:} \ j \in \{ 0..<m \} \) and
\( j\text{-not-j0:} \) \( \text{prems-of} \ i \| j \neq \) \( \text{prems-of} \ i \| j0 \)

obtain \( nj \text{ where} \ nj\text{-len:} \ enat \ (\text{Suc} \ nj) < \text{length} \ D \) and
\( \text{nj-prems:} \) \( \text{prems-of} \ i \| j \notin \text{active-subset} \ (\text{snd} \ (\text{lnth} \ D \ nj)) \) and
\( \text{nj-greater:} \) \( \forall \ k. \ k > nj \Rightarrow enat \ k < \text{length} \ D \) \( \Rightarrow \) \( \text{prems-of} \ i \| j \in \text{active-subset} \ (\text{snd} \ (\text{lnth} \ D \ k)) \)

using exist-nj \ j\text{-in} by blast

then have \( nj \notin \text{nj-set unfolding} \ nj\text{-set-def using} \ j\text{-in by blast} \)
moreover have \( nj \neq n \)
proof (rule contr)
assume \( \neg nj \neq n \)
then have \( \text{prems-of} \ i \| j = (C0,\text{active}) \)
using \( C0-in \ C0-notin \text{Lazy-Given-Clause-step}.\text{simp}s[\text{of lnth} \ D \ n \ \text{lnth} \ D \ (\text{Suc} \ n)] \text{step-n} \)
active-subset-def is-scheduled nj-prems suc-n-length by auto

then show False using \( j\text{-not-j0 C0-is by simp} \)
qed

ultimately have \( nj < n \) using \( n\text{-bigger by force} \)
then have \( \text{prems-of} \ i \| j \in \text{(active-subset} \ (\text{snd} \ (\text{lnth} \ D \ n)) \)

using \( nj\text{-greater n-in} \ Suc\text{-ile-eq dual-order.strict-implies-order} \)

unfolding \( nj\text{-set-def by blast} \)
then show \( \text{prems-of} \ i \| j \in \text{(active-subset} \ N) \)
using nth-d-is l-not-active n1-is unfolding active-subset-def by force

qed

then have \( \text{prems-i-active:} \) \( \text{set} \ (\text{prems-of} \ i) \subseteq \text{active-subset} \ N \cup \{(C0, \text{active})\} \)
using \( C0\text{-prems-i C0-is m-def} \)
by \( \) \text{metis Un-iff atleast0LessThan in-set-conv-nth insertCI lessThan-iff subrelCI} \)
moreover have \( \neg \) \( \) \( \text{set} \ (\text{prems-of} \ i) \subseteq \text{active-subset} \ N \cup \{(C0, \text{active})\} \) \( \) \text{using} \( C0\text{-prems-i by blast} \)

ultimately have \( \iota \in \text{with-labels}.\text{Inf-from2} \ (\text{active-subset} \ N) \{(\text{C0,active})\} \)
using \( \iota\text{-in-inf-fl prems-i-active unfolding with-labels.\text{Inf-from2-def with-labels.Inf-from-def}} \)
by blast
then have to-F \( \iota \in \text{no-labels}.\text{Non-ground}.\text{Inf-from2} \ (\text{fst} \circ (\text{active-subset} \ N)) \{ C0 \} \)

unfolding to-F-def with-labels.\text{Inf-from2-def with-labels.Inf-from-def}
o-no-labels.\text{Non-ground.\text{Inf-from2-def non-labels.\text{Non-ground.\text{Inf-from-def}}}}
using \( \text{Inf-FL-to-Inf-F by force} \)
then have \( \iota\text{-in-t2:} \) \( \) \( \text{to-F} \ i \in T2 \) using \( \) \text{tp-is t2-is by simp} \)

have \( j \in \{ 0..<m \} \Rightarrow (\forall k. \ k > n \Rightarrow enat \ k < \text{length} \ D \) \( \Rightarrow \) \( \text{prems-of} \ i \| j \in \text{active-subset} \ (\text{snd} \ (\text{lnth} \ D \ k)) \) \) for \( j \)
proof (cases \( j = j0 \))

case True
assume \( j = j0 \)
then show \( (\forall k. \ k > n \Rightarrow enat \ k < \text{length} \ D \) \)
(prems-of i)!j ∈ active-subset (snd (lnth D k))) using j0-allin by simp

next
  case False
  assume j-in: j ∈ {0..<m} and
  j ≠ j0
  obtain nj where nj-len: (Suc nj) < llength D and
    nj-prems: (prems-of i)!j ▷ active-subset (snd (lnth D nj)) and
    nj-greater: (∀ k. k > nj → enat k < llength D →
      (prems-of i)!j ∈ active-subset (snd (lnth D 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 D →
  (prems-of i)!j ∈ active-subset (snd (lnth D k))))
using nj-greater n-bigger by auto
qed
then have allj-allk: (∀ c ∈ set (prems-of i). (∀ k. k > n → enat k < llength D →
  c ∈ active-subset (snd (lnth D k))))
using m-def by (metis atLeast0LessThan in-set-conv-nth lessThan-Iff)
have ∀ c ∈ set (prems-of i). snd c = active using prems-i-active unfolding active-subset-def by auto
then have ex-n-i-in: ∃ n. enat (Suc n) < llength D ∧ to-F i ∈ fst (lnth D (Suc n)) ∧
  (∀ c ∈ set (prems-of i). snd c = active) ∧
  (∀ c ∈ set (prems-of i). (∀ k. k > n → enat k < llength D →
    c ∈ active-subset (snd (lnth D 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 D ∧ to-F i ∈ fst (lnth D n) ∧
  (∀ c ∈ set (prems-of i). snd c = active) ∧
  (∀ c ∈ set (prems-of i). (∀ k. k ≥ n →
    enat k < llength D →
    c ∈ active-subset (snd (lnth D k))))
by (rule ccontr)
}
ultimately obtain n T2 N2 where i-in-suc-n: to-F i ∈ fst (lnth D n) and
  all-prems-active-after: m > 0 ⇒ (∀ c ∈ set (prems-of i). (∀ k. k ≥ n → enat k < llength D →
    c ∈ active-subset (snd (lnth D k)))) and
  suc-n-length: enat n < llength D and suc-nth-d-is: lnth D n = (T2, N2)
by (metis less-antisym old.prod.exhaust zero-less-Suc)
then have i-in-t2: to-F i ∈ T2 by simp
have ∃ p≥n. enat (Suc p) < llength D ∧ to-F i ∈ (fst (lnth D p)) ∧ to-F i ▷ (fst (lnth D (Suc p)))
proof (rule ccontr)
  assume contra: ¬ (∃ p≥n. enat (Suc p) < llength D ∧ to-F i ∈ (fst (lnth D p)) ∧
    to-F i ▷ (fst (lnth D (Suc p))))
then have i-in-suc: p0 ≥ n ⇒ enat (Suc p0) < llength D ⇒ to-F i ∈ (fst (lnth D p0)) ⇒
  to-F i ∈ (fst (lnth D (Suc p0))) for p0
by blast
have p0 ≥ n ⇒ enat p0 < llength D ⇒ to-F i ∈ (fst (lnth D 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 D ⇒ to-F i ∈ fst (lnth D p0) and
def-suc-suc-smaller-d: enat (Suc p0) < llength D
have suc-p-bigger-n: n ≤ p0 using p-bigger-n by simp
have suc-smaller-d: enat p0 < length D
  using suc-suc-smaller-d suc-ile-eq dual-order.strict-implies-order by blast
then have to-F i ∈ fst (lnth D p0) using induct-hyp by blast
then show ?case using i-in-suc[OF suc-p-bigger-n suc-suc-smaller-d] by blast
qed

then have i-in-all-bigger-n: ∀ j. j ≥ n ∧ enat j < length D → to-F i ∈ (fst (lnth D j))
  by presburger
have length (lmap fst D) = length D by force
then have to-F i ∈ (∩ { j. n ≤ j ∧ enat j < length (lmap fst D)})
  using i-in-all-bigger-n using Suc-le-D by auto
then have to-F i ∈ Liminf-llist (lmap fst D)
unfolding Liminf-llist-def using suc-n-length by auto
then show False using final-schedule by fast

then obtain p where p-greater-n: p ≥ n and p-smaller-d: enat (Suc p) < length D and
  i-in-p: to-F i ∈ (fst (lnth D p)) and i-notin-suc-p: to-F i /∈ (fst (lnth D (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 D p → LGC lnth D (Suc p) using deriv p-smaller-d chain-lnth-rel by blast
then have ∃ T1 T2 i N2 N1 M. lnth D p = (T1, N1) ∧ lnth D (Suc p) = (T2, N2) ∧
  T1 = T2 ∪ { i } ∧ T2 ∩ { i } = {} ∧ N2 = N1 ∪ M ∧ active-subset M = { } ∧
  i ∈ no-labels.empty-ord-lifted-calc-u-red-crit-family.Red-Inf-Q (fst ‘ (N1 ∪ M))
  proof
  have ci-or-do: (∃ T1 T2 i. N2 N1 M. lnth D p = (T1, N1) ∧ lnth D (Suc p) = (T2, N2) ∧
    T1 = T2 ∪ { i } ∧ T2 ∩ { i } = {} ∧ N2 = N1 ∪ M ∧ active-subset M = { } ∧
    i ∈ no-labels.empty-ord-lifted-calc-u-red-crit-family.Red-Inf-Q (fst ‘ (N1 ∪ M)) ≡
    (∃ T1 T2 T' N. lnth D p = (T1, N) ∧ lnth D (Suc p) = (T2, N) ∧
    T1 = T2 ∪ T' ∧ T2 ∩ T' = {} ∧
    T' ∩ no-labels.Non-ground.Inf-from (fst ‘ active-subset N) = { })
    using Lazy-Given-Clause-step.simps[of lnth D p lnth D (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 i))!j ∈ (fst ‘ (active-subset (snd (lnth D p))))
    for j
    proof
      fix j
      assume
      m-pos: m > 0 and
      j-in: j ∈ {0..<m}
      then have (prems-of i)!j ∈ (active-subset (snd (lnth D 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 i))!j ∈ (fst ‘ (active-subset (snd (lnth D p))))
        by blast
      then show (prems-of (to-F i))!j ∈ (fst ‘ (active-subset (snd (lnth D p))))
        unfolding to-F-def using j-in m-def by simp
      qed
    then have prems-i-active-p: m > 0 ⇒
      to-F i ∈ no-labels.Non-ground.Inf-from (fst ‘ active-subset (snd (lnth D p)))
      using i-in-F unfolding no-labels.Non-ground.Inf-from-def
      by (smt atLeast0LessThan in-set-conv-nth lessThan iff m-def-F mem-Collect-eq subsetI)
  have m = 0 ⇒ (∃ T1 T2 i. N2 N1 M. lnth D p = (T1, N1) ∧ lnth D (Suc p) = (T2, N2) ∧
\[ T_1 = T_2 \cup \{i\} \land T_2 \cap \{i\} = \{\} \land N_2 = N_1 \cup M \land \text{active-subset } M = \{\} \land \\
i \in \text{no-labels.\emptyset\text{-ord-lifted-calc-w-red-crit-family.Rd-Inf-Q (fst ' (N_1 \cup M))}} \]

using ci-or-do premise-free-inf-always-from[of to-F i fst ' active-subset -. OF i-in-F]

\[ \text{m-def i-in-p i-notin-suc-p m-def-F by auto} \]

then show \( (\exists T_1 \ T_2 \ i \ N_2 \ N_1 \ M. \ \text{l nth D p = (T_1, N_1) \land l nth D (Suc p) = (T_2, N_2) \land} \)

\( T_1 = T_2 \cup \{i\} \land T_2 \cap \{i\} = \{\} \land N_2 = N_1 \cup M \land \text{active-subset } M = \{\} \land \\
i \in \text{no-labels.\emptyset\text{-ord-lifted-calc-w-red-crit-family.Rd-Inf-Q (fst ' (N_1 \cup M))}} \]

using ci-or-do i-in-p i-notin-suc-p prems-i-active unfolding active-subset-def by force

qed

then obtain \( T_1p \ T_2p \ N_1p \ N_2p \Mp \) where \( \text{l nth D p = (T_1p, N_1p) \ and} \)

\( \text{suc-p-is: l nth D (Suc p) = (T_2p, N_2p) \ and} \ T_1p = T_2p \cup \{\text{to-F } i\} \and T_2p \cap \{\text{to-F } i\} = \{\} \and \\
\text{n2p-is: } N_2p = N_1p \cup \Mp \land \text{active-subset } Mp = \{\} \land \\
i \in \text{no-labels.\emptyset\text{-ord-lifted-calc-w-red-crit-family.Rd-Inf-Q (fst ' (N_1p \cup Mp))}} \]

using i-in-p i-notin-suc-p by fastforce

have to-F i \in\text{ no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q (fst ' (snd (l nth D (Suc p))))} \)

using i-in-red-inf suc-p-is n2p-is by fastforce

then have \( \forall q. (\text{G-Inf-q q (to-F i)} \neq \text{None} \land \\
\text{the (G-Inf-q q (to-F i)}) \subseteq \text{Red-Inf-q q (}\bigcup (G-F-q q ' (fst ' (snd (l nth D (Suc p)))))) \)

\( \lor (G-Inf-q q (to-F i) = \text{None} \land \\
G-F-q q (\text{concl-of (to-F i)}) \subseteq (\bigcup (G-F-q q ' (fst ' (snd (l nth D (Suc p)))))) \lor \\
\text{Red-Inf-q q (}\bigcup (G-F-q q ' (fst ' (snd (l nth D (Suc p)))))) \)

unfolding to-F-def no-labels.lifted-calc-w-red-crit-family.Red-Inf-Q-def

no-labels.Rd-Inf-G-q-def no-labels.G-set-q-def

by fastforce

then have \( i \in \text{ with-labels.Rd-Inf-Q (snd (l nth D (Suc p)))} \)


G-F-L-q-def using i-in-inf-fl by auto

then show \( i \in \text{labeled-ord-red-crit-fam.\emptyset\text{-ord-lifted-calc-w-red-crit-family.inter-red-crit-calculus.Sup-Red-Inf-llist (}\text{lmap snd D}) \)

unfolding

\( \text{labeled-ord-red-crit-fam.\emptyset\text{-ord-lifted-calc-w-red-crit-family.inter-red-crit-calculus.Sup-Red-Inf-llist-def using red-inf-equiv2 suc-n-length p-smaller-d by auto} \)

qed

qed

\[ \text{theorem lgc-complete: chain (\Rightarrow \text{LGC}) D \Rightarrow l length D > 0 \Rightarrow \text{active-subset (snd (l nth D 0)) = \{\}} \Rightarrow \\
\Rightarrow \text{non-active-subset (Liminf-llist (lmap snd D)) = \{\}} \Rightarrow \\
(\forall i. \in \text{Inf-F. length (prems-of i) = 0 } \Rightarrow i \in (\text{fst (l nth D 0 key)})) \Rightarrow \\
\text{Liminf-llist (lmap fst D) = \{\} \Rightarrow B \in \text{Bot-F \Rightarrow no-labels.\text{entails-G-Q (fst ' (snd (l nth D 0)) key)} \}} \{B\} \Rightarrow \\
(\exists i. \text{enat i < l length D } \lor (\exists BL \in \text{Bot-FL. BL \in (snd (l nth D i))}) \)

proof –

fix B

assume

derv: chain (\Rightarrow \text{LGC}) D and

not-empty-d: l length D > 0 and

init-state: active-subset (snd (l nth D 0)) = \{\} and

final-state: non-active-subset (Liminf-llist (lmap snd D)) = \{\} and

no-prems-init-active: \( \forall i. \in \text{Inf-F. length (prems-of i) = 0 } \Rightarrow i \in (\text{fst (l nth D 0))} \and

final-schedule: Liminf-llist (lmap fst D) = \{\} and

b-in: B \in \text{Bot-F and}
bot-entailed: no-labels.entails-G-Q (fst ' (snd (lnth D 0))) {B}

have labeled-b-in: (B,active) ∈ Bot-FL unfolding Bot-FL-def using b-in by simp

have not-empty-d2: ¬ lnull (lmap snd D) using not-empty-d by force

have simp-snd-lmap: lnth (lmap snd D) 0 = snd (lnth D 0)

using lnth-lmap[of 0 D snd] not-empty-d by (simp add: zero-enat-def)

have labeled-bot-entailed: entails-G-L-Q (snd (lnth D 0)) {((B,active))}

using labeled-entailment-lifting bot-entailed by fastforce

have fair (lmap snd D)

using lgc-fair[of deriv not-empty-d init-state final-state no-prems-init-active final-schedule].

then have ∃ i ∈ {i. enat i < llength D}. ∃ BL ∈ Bot-FL. BL ∈ (snd (lnth D i))

using labeled-ordered-dynamic-ref-comp labeled-b-in not-empty-d2 lgc-to-red[of deriv]

unfolding dynamic-refutational-complete-calculus-def

dynamic-refutational-complete-calculus-axioms-def

by (metis (mono-tags, lifting) llength-lmap lnth-lmap mem-Collect-eq)

then show ?thesis by blast

det

det