

Extensions to the Comprehensive Framework for Saturation Theorem Proving

Jasmin Blanchette Sophie Tourret

March 29, 2023

Abstract

This Isabelle/HOL formalization extends the `Saturation_Framework` entry of the *Archive of Formal Proofs* with the following contributions:

- an application of the framework to prove Bachmair and Ganzinger’s resolution prover RP refutationally complete, which was formalized in a more ad hoc fashion by Schlichtkrull et al. in the *AFP* entry `Ordered_Resultion_Prover`;
- generalizations of various basic concepts formalized by Schlichtkrull et al., which were needed to verify RP and could be useful to formalize other calculi, such as superposition;
- alternative proofs of fairness (and hence saturation and ultimately refutational completeness) for the eager and lazy given clause procedures (GC and LGC) based on invariance.

Contents

1	Soundness	1
2	Counterexample-Reducing Inference Systems and the Standard Redundancy Criterion	3
2.1	Counterexample-Reducing Inference Systems	3
2.2	Compactness	5
2.3	The Finitary Standard Redundancy Criterion	5
2.4	The Standard Redundancy Criterion	9
2.5	Refutational Completeness	12
3	Clausal Calculi	13
3.1	Setup	14
3.2	Consequence Relation	14
3.3	Counterexample-Reducing Inference Systems	16
3.4	Counterexample-Reducing Calculi Equipped with a Standard Redundancy Criterion	16
4	Application of the Saturation Framework to Bachmair and Ganzinger’s RP	17
4.1	Setup	17
4.2	Library	18
4.3	Ground Layer	18
4.4	First-Order Layer	19
4.5	Labeled First-Order or Given Clause Layer	22

4.6	Resolution Prover Layer	24
4.7	Alternative Derivation of Previous RP Results	31
5	New Fairness Proofs for the Given Clause Prover Architectures	35
5.1	Given Clause Procedure	35
5.2	Lazy Given Clause	38

1 Soundness

```
theory Soundness
  imports Saturation-Framework.Calculus
begin
```

Although consistency-preservation usually suffices, soundness is a more precise concept and is sometimes useful.

```
locale sound-inference-system = inference-system + consequence-relation +
  assumes
    sound:  $\iota \in \text{Inf} \implies \text{set}(\text{prems-of } \iota) \models \{\text{concl-of } \iota\}$ 
begin
```

```
lemma Inf-consist-preserving:
  assumes n-cons:  $\neg N \models \text{Bot}$ 
  shows  $\neg N \cup \text{concl-of } \iota \text{ Inf-from } N \models \text{Bot}$ 
proof -
  have  $N \models \text{concl-of } \iota \text{ Inf-from } N$ 
  using sound unfolding Inf-from-def image-def Bex-def mem-Collect-eq
  by (smt all-formulas-entailed entails-trans mem-Collect-eq subset-entailed)
  then show ?thesis
  using n-cons entails-trans-strong by blast
qed
```

end

The limit of a derivation based on a redundancy criterion is satisfiable if and only if the initial set is satisfiable. This material is partly based on Section 4.1 of Bachmair and Ganzinger's *Handbook* chapter, but adapted to the saturation framework of Waldmann et al.

```
context calculus
begin
```

The next three lemmas correspond to Lemma 4.2:

```
lemma Red-F-Sup-subset-Red-F-Liminf:
  chain ( $\triangleright$ )  $Ns \implies \text{Red-F } (\text{Sup-llist } Ns) \subseteq \text{Red-F } (\text{Liminf-llist } Ns)$ 
  by (metis Liminf-llist-subset-Sup-llist Red-in-Sup Un-absorb1 calculus.Red-F-of-Red-F-subset
    calculus-axioms double-diff sup-ge2)
```

```
lemma Red-I-Sup-subset-Red-I-Liminf:
  chain ( $\triangleright$ )  $Ns \implies \text{Red-I } (\text{Sup-llist } Ns) \subseteq \text{Red-I } (\text{Liminf-llist } Ns)$ 
  by (metis Liminf-llist-subset-Sup-llist Red-I-of-Red-F-subset Red-in-Sup double-diff subset-refl)
```

Proof idea due to Uwe Waldmann:

```
lemma unsat-limit-iff:
  assumes
    chain-red: chain ( $\triangleright$ )  $Ns$  and
```

chain-ent: chain (\models) Ns
shows *Liminf-llist Ns \models Bot \longleftrightarrow lhd Ns \models Bot*
proof
assume *Liminf-llist Ns \models Bot*
moreover have *Sup-llist Ns \models Liminf-llist Ns*
by (*simp add: Liminf-llist-subset-Sup-llist subset-entailed*)
moreover have *lhd Ns \models Sup-llist Ns*
proof –
have *lhd Ns \models lnth Ns i if i < llength Ns for i*
using *that*
proof (*induct i*)
case *0*
then show *?case*
using *chain-ent chain-not-lnull lhd-conv-lnth subset-entailed by fastforce*
next
case (*Suc i*)
then show *?case*
using *Suc-ile-eq chain-ent chain-lnth-rel entails-trans less-le by blast*
qed
thus *?thesis*
unfolding *Sup-llist-def using entail-unions by fastforce*
qed
ultimately show *lhd Ns \models Bot*
using *entails-trans by blast*
next
assume *lhd Ns \models Bot*
then have *Sup-llist Ns \models Bot*
by (*meson chain-ent chain-not-lnull entails-trans lhd-subset-Sup-llist subset-entailed*)
then have *Sup-llist Ns – Red-F (Sup-llist Ns) \models Bot*
using *Red-F-Bot entail-set-all-formulas by blast*
then have *Liminf-llist Ns – Red-F (Sup-llist Ns) \models Bot*
by (*smt Diff-idemp Diff-mono Diff-subset Liminf-llist-subset-Sup-llist*
Red-F-Sup-subset-Red-F-Liminf Red-F-of-subset Red-in-Sup antisym-conv chain-red double-diff
entail-set-all-formulas order-refl order-trans subset-antisym)
then show *Liminf-llist Ns \models Bot*
by (*meson Diff-subset entails-trans subset-entailed*)
qed

Some easy consequences:

lemma *Red-F-limit-Sup: chain (\triangleright) Ns \implies Red-F (Liminf-llist Ns) = Red-F (Sup-llist Ns)*
by (*metis Liminf-llist-subset-Sup-llist Red-F-of-Red-F-subset Red-F-of-subset Red-in-Sup*
double-diff order-refl subset-antisym)

lemma *Red-I-limit-Sup: chain (\triangleright) Ns \implies Red-I (Liminf-llist Ns) = Red-I (Sup-llist Ns)*
by (*metis Liminf-llist-subset-Sup-llist Red-I-of-Red-F-subset Red-I-of-subset Red-in-Sup*
double-diff order-refl subset-antisym)

end

end

2 Counterexample-Reducing Inference Systems and the Standard Redundancy Criterion

theory *Standard-Redundancy-Criterion*

imports

Saturation-Framework.Calculus

HOL-Library.Multiset-Order

begin

The standard redundancy criterion can be defined uniformly for all inference systems equipped with a compact consequence relation. The essence of the refutational completeness argument can be carried out abstractly for counterexample-reducing inference systems, which enjoy a “smallest counterexample” property. This material is partly based on Section 4.2 of Bachmair and Ganzinger’s *Handbook* chapter, but adapted to the saturation framework of Waldmann et al.

2.1 Counterexample-Reducing Inference Systems

abbreviation *main-prem-of* :: 'f inference \Rightarrow 'f **where**

main-prem-of $\iota \equiv \text{last}(\text{prems-of } \iota)$

abbreviation *side-prems-of* :: 'f inference \Rightarrow 'f list **where**

side-prems-of $\iota \equiv \text{butlast}(\text{prems-of } \iota)$

lemma *set-prems-of*:

set (*prems-of* ι) = (if *prems-of* $\iota = []$ then {} else {*main-prem-of* ι } \cup *set* (*side-prems-of* ι))

by *clarsimp* (*metis Un-insert-right append-Nil2 append-butlast-last-id list.set(2) set-append*)

locale *counterex-reducing-inference-system* = *inference-system* *Inf* + *consequence-relation*

for *Inf* :: 'f inference set +

fixes

I-of :: 'f set \Rightarrow 'f set **and**

less :: 'f \Rightarrow 'f \Rightarrow bool (**infix** \prec 50)

assumes

wfP-less: *wfP* (\prec) **and**

Inf-counterex-reducing:

$N \cap \text{Bot} = \{\} \implies D \in N \implies \neg I\text{-of } N \models \{D\} \implies$

$(\bigwedge C. C \in N \implies \neg I\text{-of } N \models \{C\} \implies D \prec C \vee D = C) \implies$

$\exists \iota \in \text{Inf}. \text{prems-of } \iota \neq [] \wedge \text{main-prem-of } \iota = D \wedge \text{set}(\text{side-prems-of } \iota) \subseteq N \wedge$

$I\text{-of } N \models \text{set}(\text{side-prems-of } \iota) \wedge \neg I\text{-of } N \models \{\text{concl-of } \iota\} \wedge \text{concl-of } \iota \prec D$

begin

lemma *ex-min-counterex*:

fixes *N* :: 'f set

assumes $\neg I \models N$

shows $\exists C \in N. \neg I \models \{C\} \wedge (\forall D \in N. D \prec C \longrightarrow I \models \{D\})$

proof –

obtain *C* **where**

$C \in N$ **and** $\neg I \models \{C\}$

using *assms* *all-formulas-entailed* **by** *blast*

then have *c-in*: $C \in \{C \in N. \neg I \models \{C\}\}$

by *blast*

show *?thesis*

using *wfP-eq-minimal*[*THEN iffD1, rule-format, OF wfP-less c-in*] **by** *blast*

qed

end

Theorem 4.4 (generalizes Theorems 3.9 and 3.16):

locale *countere x -reducing-inference-system-with-trivial-redundancy* =
 countere x -reducing-inference-system - - *Inf* + *calculus* - *Inf* - λ -. {} λ -. {}
 for *Inf* :: 'f *inference set* +
 assumes *less-total*: $\bigwedge C D. C \neq D \implies C \prec D \vee D \prec C$
begin

theorem *saturated-model*:

assumes

satur: *saturated N* **and**

bot-ni-n: $N \cap \text{Bot} = \{\}$

shows *I-of N* $\models N$

proof (*rule ccontr*)

assume \neg *I-of N* $\models N$

then obtain *D* :: 'f **where**

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

d-cex: \neg *I-of N* $\models \{D\}$ **and**

d-min: $\bigwedge C. C \in N \implies C \prec D \implies$ *I-of N* $\models \{C\}$

by (*meson ex-min-countere x*)

then obtain ι :: 'f *inference* **where**

ι -inf: $\iota \in$ *Inf* **and**

concl-cex: \neg *I-of N* $\models \{\text{concl-of } \iota\}$ **and**

concl-lt-d: *concl-of* $\iota \prec D$

using *Inf-countere x -reducing*[*OF bot-ni-n*] *less-total*

by force

have *concl-of* $\iota \in N$

using *ι -inf Red-I-of-Inf-to-N* **by blast**

then show *False*

using *concl-cex concl-lt-d d-min* **by blast**

qed

An abstract version of Corollary 3.10 does not hold without some conditions, according to Nitpick:

corollary *saturated-complete*:

assumes

satur: *saturated N* **and**

unsat: $N \models \text{Bot}$

shows $N \cap \text{Bot} \neq \{\}$

oops

end

2.2 Compactness

locale *concl-compact-consequence-relation* = *consequence-relation* +

assumes

entails-concl-compact: *finite EE* $\implies CC \models EE \implies \exists CC' \subseteq CC. \text{finite } CC' \wedge CC' \models EE$

begin

lemma *entails-concl-compact-union*:

assumes

fin-e: finite EE and
cd-ent: $CC \cup DD \models EE$
shows $\exists CC' \subseteq CC. \text{finite } CC' \wedge CC' \cup DD \models EE$
proof –
obtain $CCDD'$ **where**
cd1-fin: finite CCDD' and
cd1-sub: $CCDD' \subseteq CC \cup DD$ and
cd1-ent: $CCDD' \models EE$
using *entails-concl-compact[OF fin-e cd-ent]* **by** *blast*

define CC' **where**
 $CC' = CCDD' - DD$
have $CC' \subseteq CC$
unfolding CC' -*def* **using** *cd1-sub* **by** *blast*
moreover **have** *finite CC'*
unfolding CC' -*def* **using** *cd1-fin* **by** *blast*
moreover **have** $CC' \cup DD \models EE$
unfolding CC' -*def* **using** *cd1-ent*
by (*metis Un-Diff-cancel2 Un-upper1 entails-trans subset-entailed*)
ultimately show *?thesis*
by *blast*
qed

end

2.3 The Finitary Standard Redundancy Criterion

locale *finitary-standard-formula-redundancy* =
consequence-relation Bot (\models)
for
Bot :: 'f set and
entails :: 'f set \Rightarrow 'f set \Rightarrow bool (infix \models 50) +
fixes
less :: 'f \Rightarrow 'f \Rightarrow bool (infix \prec 50)
assumes
transp-less: transp (\prec) and
wfP-less: wfP (\prec)
begin

definition *Red-F* :: *'f set \Rightarrow 'f set where*
 $Red-F\ N = \{C. \exists DD \subseteq N. \text{finite } DD \wedge DD \models \{C\} \wedge (\forall D \in DD. D \prec C)\}$

The following results correspond to Lemma 4.5. The lemma *wlog-non-Red-F* generalizes the core of the argument.

lemma *Red-F-of-subset*: $N \subseteq N' \implies Red-F\ N \subseteq Red-F\ N'$
unfolding *Red-F-def* **by** *fast*

lemma *wlog-non-Red-F*:
assumes
dd0-fin: finite DD0 and
dd0-sub: $DD0 \subseteq N$ and
dd0-ent: $DD0 \cup CC \models \{E\}$ and
dd0-lt: $\forall D' \in DD0. D' \prec D$
shows $\exists DD \subseteq N - Red-F\ N. \text{finite } DD \wedge DD \cup CC \models \{E\} \wedge (\forall D' \in DD. D' \prec D)$
proof –

have *mset-DD0-in*: $mset\text{-}set\ DD0 \in$
 $\{DD. set\text{-}mset\ DD \subseteq N \wedge set\text{-}mset\ DD \cup CC \models \{E\} \wedge (\forall D' \in set\text{-}mset\ DD. D' \prec D)\}$
using *assms finite-set-mset-mset-set* **by** *simp*
obtain $DD :: 'f\ multiset$ **where**
dd-sub-n: $set\text{-}mset\ DD \subseteq N$ **and**
ddcc-ent-e: $set\text{-}mset\ DD \cup CC \models \{E\}$ **and**
dd-lt-d: $\forall D' \in \# DD. D' \prec D$ **and**
d-min: $\forall y. multp\ (\prec)\ y\ DD \longrightarrow$
 $y \notin \{DD. set\text{-}mset\ DD \subseteq N \wedge set\text{-}mset\ DD \cup CC \models \{E\} \wedge (\forall D' \in \# DD. D' \prec D)\}$
using *wfP-eq-minimal*[*THEN iffD1*, *rule-format*, *OF wfP-less*[*THEN wfP-multp*] *mset-DD0-in*]
by *blast*

have $\forall Da \in \# DD. Da \notin Red\text{-}F\ N$

proof *clarify*

fix $Da :: 'f$

assume

da-in-dd: $Da \in \# DD$ **and**

da-rf: $Da \in Red\text{-}F\ N$

obtain $DDa0 :: 'f\ set$ **where**

dda0-sub-n: $DDa0 \subseteq N$ **and**

dda0-fin: *finite* $DDa0$ **and**

dda0-ent-da: $DDa0 \models \{Da\}$ **and**

dda0-lt-da: $\forall D \in DDa0. D \prec Da$

using *da-rf* **unfolding** *Red-F-def mem-Collect-eq*

by *blast*

define $DDa :: 'f\ multiset$ **where**

$DDa = DD - \{\#Da\# \} + mset\text{-}set\ DDa0$

have $set\text{-}mset\ DDa \subseteq N$

unfolding *DDa-def* **using** *dd-sub-n dda0-sub-n finite-set-mset-mset-set*[*OF dda0-fin*]

by (*smt* (*verit*, *best*) *contra-subsetD in-diffD subsetI union-iff*)

moreover **have** $set\text{-}mset\ DDa \cup CC \models \{E\}$

proof (*rule entails-trans-strong*[*of* - $\{Da\}$])

show $set\text{-}mset\ DDa \cup CC \models \{Da\}$

unfolding *DDa-def set-mset-union finite-set-mset-mset-set*[*OF dda0-fin*]

by (*rule entails-trans*[*OF* - *dda0-ent-da*]) (*auto intro: subset-entailed*)

next

have $H: set\text{-}mset\ (DD - \{\#Da\# \} + mset\text{-}set\ DDa0) \cup CC \cup \{Da\} =$

$set\text{-}mset\ (DD + mset\text{-}set\ DDa0) \cup CC$

by (*smt* (*verit*) *Un-insert-left Un-insert-right da-in-dd insert-DiffM*

set-mset-add-mset-insert set-mset-union sup-bot.right-neutral)

show $set\text{-}mset\ DDa \cup CC \cup \{Da\} \models \{E\}$

unfolding *DDa-def H*

by (*rule entails-trans*[*OF* - *ddcc-ent-e*]) (*auto intro: subset-entailed*)

qed

moreover **have** $\forall D' \in \# DDa. D' \prec D$

using *dd-lt-d dda0-lt-da da-in-dd* **unfolding** *DDa-def*

using *transp-less*[*THEN transpD*]

using *finite-set-mset-mset-set*[*OF dda0-fin*]

by (*metis insert-DiffM2 union-iff*)

moreover **have** $multp\ (\prec)\ DDa\ DD$

unfolding *DDa-def multp-eq-multp_{DM}*[*OF wfP-imp-asymp*[*OF wfP-less*] *transp-less*] *multp_{DM}-def*

using *finite-set-mset-mset-set*[*OF dda0-fin*]

by (metis da-in-dd dda0-lt-da mset-subset-eq-single multi-self-add-other-not-self
 union-single-eq-member)
 ultimately show *False*
 using *d-min* by (auto intro!: antisym)
 qed
 then show *?thesis*
 using *dd-sub-s-n ddc-ent-e dd-lt-d* by *blast*
 qed

lemma *Red-F-imp-ex-non-Red-F*:
 assumes *c-in*: $C \in \text{Red-F } N$
 shows $\exists CC \subseteq N - \text{Red-F } N. \text{finite } CC \wedge CC \models \{C\} \wedge (\forall C' \in CC. C' \prec C)$
proof –
 obtain *DD* :: '*f* set where
dd-fin: *finite* *DD* and
dd-sub: $DD \subseteq N$ and
dd-ent: $DD \models \{C\}$ and
dd-lt: $\forall D \in DD. D \prec C$
 using *c-in*[*unfolded Red-F-def*] by *fast*
 show *?thesis*
 by (rule *wlog-non-Red-F*[of *DD N* $\{C\}$ *C C*, *simplified*, OF *dd-fin dd-sub dd-ent dd-lt*])
 qed

lemma *Red-F-sub-s-Red-F-diff-Red-F*: $\text{Red-F } N \subseteq \text{Red-F } (N - \text{Red-F } N)$
proof
 fix *C*
 assume *c-rf*: $C \in \text{Red-F } N$
 then obtain *CC* :: '*f* set where
cc-sub-s: $CC \subseteq N - \text{Red-F } N$ and
cc-fin: *finite* *CC* and
cc-ent-c: $CC \models \{C\}$ and
cc-lt-c: $\forall C' \in CC. C' \prec C$
 using *Red-F-imp-ex-non-Red-F*[of *C N*] by *blast*
 have $\forall D \in CC. D \notin \text{Red-F } N$
 using *cc-sub-s* by *fast*
 then have *cc-nr*:
 $\forall C \in CC. \forall DD \subseteq N. \text{finite } DD \wedge DD \models \{C\} \longrightarrow (\exists D \in DD. \neg D \prec C)$
 unfolding *Red-F-def* by *simp*
 have $CC \subseteq N$
 using *cc-sub-s* by *auto*
 then have $CC \subseteq N - \{C. \exists DD \subseteq N. \text{finite } DD \wedge DD \models \{C\} \wedge (\forall D \in DD. D \prec C)\}$
 using *cc-nr* by *blast*
 then show $C \in \text{Red-F } (N - \text{Red-F } N)$
 using *cc-fin cc-ent-c cc-lt-c* unfolding *Red-F-def* by *blast*
 qed

lemma *Red-F-eq-Red-F-diff-Red-F*: $\text{Red-F } N = \text{Red-F } (N - \text{Red-F } N)$
 by (*simp add*: *Red-F-of-subset Red-F-sub-s-Red-F-diff-Red-F set-eq-subset*)

The following results correspond to Lemma 4.6.

lemma *Red-F-of-Red-F-subset*: $N' \subseteq \text{Red-F } N \implies \text{Red-F } N \subseteq \text{Red-F } (N - N')$
 by (metis *Diff-mono Red-F-eq-Red-F-diff-Red-F Red-F-of-subset order-refl*)

lemma *Red-F-model*: $M \models N - \text{Red-F } N \implies M \models N$
 by (metis (no-types) *DiffI Red-F-imp-ex-non-Red-F entail-set-all-formulas entails-trans*)

subset-entailed)

lemma *Red-F-Bot*: $B \in \text{Bot} \implies N \models \{B\} \implies N - \text{Red-F } N \models \{B\}$
using *Red-F-model entails-trans subset-entailed* **by** *blast*

end

locale *calculus-with-finitary-standard-redundancy* =
inference-system *Inf* + *finitary-standard-formula-redundancy* *Bot* (\models) (\prec)
for

Inf :: 'f inference set **and**
Bot :: 'f set **and**
entails :: 'f set \implies 'f set \implies bool (**infix** \models 50) **and**
less :: 'f \implies 'f \implies bool (**infix** \prec 50) +

assumes

Inf-has-prem: $\iota \in \text{Inf} \implies \text{prems-of } \iota \neq []$ **and**
Inf-reductive: $\iota \in \text{Inf} \implies \text{concl-of } \iota \prec \text{main-prem-of } \iota$

begin

definition *redundant-infer* :: 'f set \implies 'f inference \implies bool **where**

redundant-infer $N \ \iota \iff$
 $(\exists DD \subseteq N. \text{finite } DD \wedge DD \cup \text{set } (\text{side-prems-of } \iota) \models \{\text{concl-of } \iota\} \wedge (\forall D \in DD. D \prec \text{main-prem-of } \iota))$

definition *Red-I* :: 'f set \implies 'f inference set **where**

Red-I $N = \{\iota \in \text{Inf}. \text{redundant-infer } N \ \iota\}$

The following results correspond to Lemma 4.6. It also uses *wlog-non-Red-F*.

lemma *Red-I-of-subset*: $N \subseteq N' \implies \text{Red-I } N \subseteq \text{Red-I } N'$

unfolding *Red-I-def redundant-infer-def* **by** *auto*

lemma *Red-I-sub-Red-I-diff-Red-F*: $\text{Red-I } N \subseteq \text{Red-I } (N - \text{Red-F } N)$

proof

fix ι

assume $\iota\text{-ri}$: $\iota \in \text{Red-I } N$

define *CC* :: 'f set **where**

CC = set (side-prems-of ι)

define *D* :: 'f **where**

D = main-prem-of ι

define *E* :: 'f **where**

E = concl-of ι

obtain *DD* :: 'f set **where**

dd-fin: finite *DD* **and**

dd-sub: $DD \subseteq N$ **and**

dd-ent: $DD \cup CC \models \{E\}$ **and**

dd-lt-d: $\forall C \in DD. C \prec D$

using $\iota\text{-ri}$ **unfolding** *Red-I-def redundant-infer-def CC-def D-def E-def* **by** *blast*

obtain *DDa* :: 'f set **where**

$DDa \subseteq N - \text{Red-F } N$ **and** finite *DDa* **and** $DDa \cup CC \models \{E\}$ **and** $\forall D' \in DDa. D' \prec D$

using *wlog-non-Red-F*[*OF dd-fin dd-sub dd-ent dd-lt-d*] **by** *blast*

then show $\iota \in \text{Red-I } (N - \text{Red-F } N)$

using $\iota\text{-ri}$ **unfolding** *Red-I-def redundant-infer-def CC-def D-def E-def* **by** *blast*

qed

lemma *Red-I-eq-Red-I-diff-Red-F*: $\text{Red-I } N = \text{Red-I } (N - \text{Red-F } N)$

by (metis Diff-subset Red-I-of-subset Red-I-sub-Red-I-diff-Red-F subset-antisym)

lemma *Red-I-to-Inf*: $Red-I\ N \subseteq Inf$
unfolding *Red-I-def* **by** *blast*

lemma *Red-I-of-Red-F-subset*: $N' \subseteq Red-F\ N \implies Red-I\ N \subseteq Red-I\ (N - N')$
by (metis Diff-mono Red-I-eq-Red-I-diff-Red-F Red-I-of-subset order-refl)

lemma *Red-I-of-Inf-to-N*:

assumes

in-ι: $\iota \in Inf$ **and**

concl-in: *concl-of* $\iota \in N$

shows $\iota \in Red-I\ N$

proof –

have *redundant-infer* $N\ \iota$

unfolding *redundant-infer-def*

by (rule *exI*[**where** $x = \{\text{concl-of } \iota\}$])

(*simp add: Inf-reductive[OF in-ι] subset-entailed concl-in*)

then show $\iota \in Red-I\ N$

by (*simp add: Red-I-def in-ι*)

qed

The following corresponds to Theorems 4.7 and 4.8:

sublocale *calculus Bot Inf* (\models) *Red-I Red-F*

by (*unfold-locales, fact Red-I-to-Inf, fact Red-F-Bot, fact Red-F-of-subset,*
fact Red-I-of-subset, fact Red-F-of-Red-F-subset, fact Red-I-of-Red-F-subset,
fact Red-I-of-Inf-to-N)

end

2.4 The Standard Redundancy Criterion

locale *standard-formula-redundancy* =

concl-compact-consequence-relation Bot (\models)

for

Bot :: 'f set **and**

entails :: 'f set \Rightarrow 'f set \Rightarrow bool (**infix** \models 50) +

fixes

less :: 'f \Rightarrow 'f \Rightarrow bool (**infix** \prec 50)

assumes

transp-less: *transp* (\prec) **and**

wfP-less: *wfP* (\prec)

begin

definition *Red-F* :: 'f set \Rightarrow 'f set **where**

$Red-F\ N = \{C. \exists DD \subseteq N. DD \models \{C\} \wedge (\forall D \in DD. D \prec C)\}$

Compactness of (\models) implies that *Red-F* is equivalent to its finitary counterpart.

interpretation *fin-std-red-F*: *finitary-standard-formula-redundancy Bot* (\models) (\prec)

using *transp-less asymp-on-less wfP-less* **by** *unfold-locales*

lemma *Red-F-conv*: $Red-F = fin-std-red-F.Red-F$

proof (*intro ext*)

fix N

show $Red-F\ N = fin-std-red-F.Red-F\ N$

unfolding *Red-F-def fin-std-red-F.Red-F-def*
using *entails-concl-compact*
by (*smt (verit, best) Collect-cong finite.emptyI finite-insert subset-eq*)
qed

The results from *finitary-standard-formula-redundancy* can now be lifted.

The following results correspond to Lemma 4.5.

lemma *Red-F-of-subset: $N \subseteq N' \implies Red-F N \subseteq Red-F N'$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-of-subset*)

lemma *Red-F-imp-ex-non-Red-F: $C \in Red-F N \implies \exists CC \subseteq N - Red-F N. CC \models \{C\} \wedge (\forall C' \in CC. C' \prec C)$*
unfolding *Red-F-conv*
using *fin-std-red-F.Red-F-imp-ex-non-Red-F* **by** *meson*

lemma *Red-F-sub-Red-F-diff-Red-F: $Red-F N \subseteq Red-F (N - Red-F N)$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-sub-Red-F-diff-Red-F*)

lemma *Red-F-eq-Red-F-diff-Red-F: $Red-F N = Red-F (N - Red-F N)$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-eq-Red-F-diff-Red-F*)

The following results correspond to Lemma 4.6.

lemma *Red-F-of-Red-F-subset: $N' \subseteq Red-F N \implies Red-F N \subseteq Red-F (N - N')$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-of-Red-F-subset*)

lemma *Red-F-model: $M \models N - Red-F N \implies M \models N$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-model*)

lemma *Red-F-Bot: $B \in Bot \implies N \models \{B\} \implies N - Red-F N \models \{B\}$*
unfolding *Red-F-conv*
by (*rule fin-std-red-F.Red-F-Bot*)

end

locale *calculus-with-standard-redundancy =*
inference-system Inf + standard-formula-redundancy Bot (\models) (\prec)
for
Inf **::** *'f inference set and*
Bot **::** *'f set and*
entails **::** *'f set \Rightarrow 'f set \Rightarrow bool (infix \models 50) and*
less **::** *'f \Rightarrow 'f \Rightarrow bool (infix \prec 50) +*
assumes
Inf-has-prem: $\iota \in Inf \implies$ prems-of $\iota \neq []$ and
Inf-reductive: $\iota \in Inf \implies$ concl-of $\iota \prec$ main-prem-of ι
begin

definition *redundant-infer* **::** *'f set \Rightarrow 'f inference \Rightarrow bool where*
redundant-infer $N \iota \iff$
($\exists DD \subseteq N. DD \cup set (side-prems-of \iota) \models \{concl-of \iota\} \wedge (\forall D \in DD. D \prec main-prem-of \iota)$)

definition *Red-I* :: 'f set \Rightarrow 'f inference set **where**

Red-I $N = \{\iota \in \text{Inf. redundant-infer } N \ \iota\}$

Compactness of (\models) implies that *Red-I* is equivalent to its finitary counterpart.

interpretation *fin-std-red*: calculus-with-finitary-standard-redundancy *Inf Bot* (\models)
using *transp-less asymp-on-less wfp-less Inf-has-prem Inf-reductive* **by** *unfold-locales*

lemma *redundant-infer-conv*: *redundant-infer* = *fin-std-red.redundant-infer*

proof (*intro ext*)

fix $N \ \iota$

show *redundant-infer* $N \ \iota \longleftrightarrow$ *fin-std-red.redundant-infer* $N \ \iota$

unfolding *redundant-infer-def fin-std-red.redundant-infer-def*

using *entails-concl-compact-union*

by (*smt (verit, ccfv-threshold) finite.emptyI finite-insert subset-eq*)

qed

lemma *Red-I-conv*: *Red-I* = *fin-std-red.Red-I*

unfolding *Red-I-def fin-std-red.Red-I-def*

unfolding *redundant-infer-conv*

by (*rule refl*)

The results from *calculus-with-finitary-standard-redundancy* can now be lifted.

The following results correspond to Lemma 4.6.

lemma *Red-I-of-subset*: $N \subseteq N' \Longrightarrow \text{Red-I } N \subseteq \text{Red-I } N'$

unfolding *Red-I-conv*

by (*rule fin-std-red.Red-I-of-subset*)

lemma *Red-I-sub-Red-I-diff-Red-F*: $\text{Red-I } N \subseteq \text{Red-I } (N - \text{Red-F } N)$

unfolding *Red-F-conv Red-I-conv*

by (*rule fin-std-red.Red-I-sub-Red-I-diff-Red-F*)

lemma *Red-I-eq-Red-I-diff-Red-F*: $\text{Red-I } N = \text{Red-I } (N - \text{Red-F } N)$

unfolding *Red-F-conv Red-I-conv*

by (*rule fin-std-red.Red-I-eq-Red-I-diff-Red-F*)

lemma *Red-I-to-Inf*: $\text{Red-I } N \subseteq \text{Inf}$

unfolding *Red-I-conv*

by (*rule fin-std-red.Red-I-to-Inf*)

lemma *Red-I-of-Red-F-subset*: $N' \subseteq \text{Red-F } N \Longrightarrow \text{Red-I } N \subseteq \text{Red-I } (N - N')$

unfolding *Red-F-conv Red-I-conv*

by (*rule fin-std-red.Red-I-of-Red-F-subset*)

lemma *Red-I-of-Inf-to-N*:

$\iota \in \text{Inf} \Longrightarrow \text{concl-of } \iota \in N \Longrightarrow \iota \in \text{Red-I } N$

unfolding *Red-I-conv*

by (*rule fin-std-red.Red-I-of-Inf-to-N*)

The following corresponds to Theorems 4.7 and 4.8:

sublocale *calculus Bot Inf* (\models) *Red-I Red-F*

by (*unfold-locales, fact Red-I-to-Inf, fact Red-F-Bot, fact Red-F-of-subset, fact Red-I-of-subset, fact Red-F-of-Red-F-subset, fact Red-I-of-Red-F-subset, fact Red-I-of-Inf-to-N*)

end

2.5 Refutational Completeness

locale *calculus-with-standard-inference-redundancy* = *calculus Bot Inf* (\models) *Red-I Red-F*
for *Bot* :: 'f set **and** *Inf* **and** *entails* (**infix** \models 50) **and** *Red-I* **and** *Red-F* +
fixes
less :: 'f \Rightarrow 'f \Rightarrow bool (**infix** \prec 50)
assumes
Inf-has-prem: $\iota \in \text{Inf} \implies \text{prems-of } \iota \neq []$ **and**
Red-I-imp-redundant-infer: $\iota \in \text{Red-I } N \implies$
 $(\exists DD \subseteq N. DD \cup \text{set } (\text{side-prems-of } \iota) \models \{\text{concl-of } \iota\} \wedge (\forall C \in DD. C \prec \text{main-prem-of } \iota))$

sublocale *calculus-with-finitary-standard-redundancy* \subseteq
calculus-with-standard-inference-redundancy Bot Inf (\models) *Red-I Red-F*
using *Inf-has-prem*
by (*unfold-locales*) (*auto simp: Red-I-def redundant-infer-def*)

sublocale *calculus-with-standard-redundancy* \subseteq
calculus-with-standard-inference-redundancy Bot Inf (\models) *Red-I Red-F*
using *Inf-has-prem*
by (*unfold-locales*) (*simp-all add: Red-I-def redundant-infer-def*)

locale *counterex-reducing-calculus-with-standard-inference-redundancy* =
calculus-with-standard-inference-redundancy Bot Inf (\models) *Red-I Red-F* (\prec) +
counterex-reducing-inference-system Bot (\models) *Inf I-of* (\prec)
for
Bot :: 'f set **and**
Inf :: 'f inference set **and**
entails :: 'f set \Rightarrow 'f set \Rightarrow bool (**infix** \models 50) **and**
Red-I :: 'f set \Rightarrow 'f inference set **and**
Red-F :: 'f set \Rightarrow 'f set **and**
I-of :: 'f set \Rightarrow 'f set **and**
less :: 'f \Rightarrow 'f \Rightarrow bool (**infix** \prec 50) +
assumes *less-total*: $\bigwedge C D. C \neq D \implies C \prec D \vee D \prec C$
begin

The following result loosely corresponds to Theorem 4.9.

lemma *saturated-model*:

assumes
satur: *saturated* *N* **and**
bot-ni-n: $N \cap \text{Bot} = \{\}$
shows *I-of* *N* \models *N*
proof (*rule ccontr*)
assume $\neg I\text{-of } N \models N$
then obtain *D* :: 'f **where**
d-in-n: *D* \in *N* **and**
d-cex: $\neg I\text{-of } N \models \{D\}$ **and**
d-min: $\bigwedge C. C \in N \implies C \prec D \implies I\text{-of } N \models \{C\}$
using *ex-min-counterex* **by** *blast*
then obtain ι :: 'f inference **where**
 ι -in: $\iota \in \text{Inf}$ **and**
 ι -mprem: *D* = *main-prem-of* ι **and**
sprem-sub-n: *set* (*side-prems-of* ι) \subseteq *N* **and**
sprem-true: *I-of* *N* \models *set* (*side-prems-of* ι) **and**
concl-cex: $\neg I\text{-of } N \models \{\text{concl-of } \iota\}$ **and**
concl-lt-d: *concl-of* ι \prec *D*
using *Inf-counterex-reducing*[*OF bot-ni-n*] *less-total* **by** *metis*

```

have  $\iota \in \text{Red-}I\ N$ 
  by (rule subsetD[OF satur[unfolded saturated-def Inf-from-def]],
    simp add:  $\iota$ -in set-prems-of Inf-has-prem)
    (use  $\iota$ -mprem d-in-n sprem-subs-n in blast)
then have  $\iota \in \text{Red-}I\ N$ 
  using Red-I-without-red-F by blast
then obtain  $DD :: 'f\ set$  where
  dd-subs-n:  $DD \subseteq N$  and
  dd-cc-ent-d:  $DD \cup set (side-prems-of\ \iota) \models \{concl-of\ \iota\}$  and
  dd-lt-d:  $\forall C \in DD. C \prec D$ 
  unfolding  $\iota$ -mprem using Red-I-imp-redundant-infer by meson
from dd-subs-n dd-lt-d have  $I-of\ N \models DD$ 
  using d-min by (meson ex-min-counterex subset-iff)
then have  $I-of\ N \models \{concl-of\ \iota\}$ 
  using entails-trans dd-cc-ent-d entail-union sprem-true by blast
then show False
  using concl-cex by auto
qed

```

A more faithful abstract version of Theorem 4.9 does not hold without some conditions, according to Nitpick:

corollary *saturated-complete:*

```

assumes
  satur: saturated N and
  unsat:  $N \models Bot$ 
shows  $N \cap Bot \neq \{\}$ 
oops

```

end

end

3 Clausal Calculi

theory *Clausal-Calculus*

```

imports
  Ordered-Resolution-Prover.Unordered-Ground-Resolution
  Soundness
  Standard-Redundancy-Criterion
begin

```

Various results about consequence relations, counterexample-reducing inference systems, and the standard redundancy criteria are specialized and customized for clauses as opposed to arbitrary formulas.

3.1 Setup

To avoid confusion, we use the symbol \models (with or without subscripts) for the “models” and entailment relations on clauses and \models for the abstract concept of consequence.

abbreviation *true-lit-thick* :: *'a interp* \Rightarrow *'a literal* \Rightarrow *bool* (**infix** \models_l 50) **where**
 $I \models_l L \equiv I \models L$

abbreviation *true-cls-thick* :: *'a interp* \Rightarrow *'a clause* \Rightarrow *bool* (**infix** \models 50) **where**

$I \models C \equiv I \vDash C$

abbreviation *true-clss-thick* :: 'a interp \Rightarrow 'a clause set \Rightarrow bool (**infix** \models_s 50) **where**
 $I \models_s C \equiv I \vDash_s C$

abbreviation *true-cls-mset-thick* :: 'a interp \Rightarrow 'a clause multiset \Rightarrow bool (**infix** \models_m 50) **where**
 $I \models_m C \equiv I \vDash_m C$

no-notation *true-lit* (**infix** \models_l 50)

no-notation *true-cls* (**infix** \models 50)

no-notation *true-clss* (**infix** \models_s 50)

no-notation *true-cls-mset* (**infix** \models_m 50)

3.2 Consequence Relation

abbreviation *entails-clss* :: 'a clause set \Rightarrow 'a clause set \Rightarrow bool (**infix** \models_e 50) **where**
 $N1 \models_e N2 \equiv \forall I. I \models_s N1 \longrightarrow I \models_s N2$

lemma *entails-iff-unsatisfiable-single*:

$CC \models_e \{E\} \longleftrightarrow \neg \text{satisfiable} (CC \cup \{\{\#- L\# \mid L. L \in \# E\}\})$ (**is** $- \longleftrightarrow - (- \cup ?NegD)$)

proof

assume *c-ent-e*: $CC \models_e \{E\}$

have $\neg I \models_s CC \cup ?NegD$ **for** I

using *c-ent-e*[*rule-format*, of I]

unfolding *true-clss-def true-cls-def true-lit-def if-distribR if-bool-eq-conj*

by (*fastforce simp: ball-Un is-pos-neg-not-is-pos*)

then show $\neg \text{satisfiable} (CC \cup ?NegD)$

by *auto*

next

assume $\neg \text{satisfiable} (CC \cup ?NegD)$

then have $\neg I \models_s CC \cup ?NegD$ **for** I

by *auto*

then show $CC \models_e \{E\}$

unfolding *true-clss-def true-cls-def true-lit-def if-distribR if-bool-eq-conj*

by (*fastforce simp: ball-Un is-pos-neg-not-is-pos*)

qed

lemma *entails-iff-unsatisfiable*:

$CC \models_e EE \longleftrightarrow (\forall E \in EE. \neg \text{satisfiable} (CC \cup \{\{\#- L\# \mid L. L \in \# E\}\}))$ (**is** $?lhs = ?rhs$)

proof –

have $?lhs \longleftrightarrow (\forall E \in EE. CC \models_e \{E\})$

unfolding *true-clss-def* **by** *auto*

also have $\dots \longleftrightarrow ?rhs$

unfolding *entails-iff-unsatisfiable-single* **by** *auto*

finally show *?thesis*

qed

interpretation *consequence-relation* $\{\{\#\}\}$ (\models_e)

proof

fix $N2 N1$:: 'a clause set

assume $\forall C \in N2. N1 \models_e \{C\}$

then show $N1 \models_e N2$

unfolding *true-clss-singleton* **by** (*simp add: true-clss-def*)

qed (*auto intro: true-clss-mono*)

interpretation *concl-compact-consequence-relation* $\{\{\#\}\} :: ('a :: \text{wellorder}) \text{ clause set } (\models_e)$

proof

fix $CC\ EE :: 'a \text{ clause set}$

assume

fin-e: *finite* EE **and**

c-ent-e: $CC \models_e EE$

have $\forall E \in EE. \neg \text{satisfiable } (CC \cup \{\{\#\} - L\# \mid L. L \in\# E\})$

using *c-ent-e*[*unfolded entails-iff-unsatisfiable*].

then have $\forall E \in EE. \exists DD \subseteq CC \cup \{\{\#\} - L\# \mid L. L \in\# E\}. \text{finite } DD \wedge \neg \text{satisfiable } DD$

by (*subst* (*asm*) *clausal-logic-compact*)

then obtain $DD\text{-of}$ **where**

d-of: $\forall E \in EE. DD\text{-of } E \subseteq CC \cup \{\{\#\} - L\# \mid L. L \in\# E\} \wedge \text{finite } (DD\text{-of } E)$

$\wedge \neg \text{satisfiable } (DD\text{-of } E)$

by *moura*

define CC' **where**

$CC' = (\bigcup E \in EE. DD\text{-of } E - \{\{\#\} - L\# \mid L. L \in\# E\})$

have $CC' \subseteq CC$

unfolding $CC'\text{-def}$ **using** *d-of* **by** *auto*

moreover have *c'-fin*: *finite* CC'

unfolding $CC'\text{-def}$ **using** *d-of fin-e* **by** *blast*

moreover have $CC' \models_e EE$

unfolding *entails-iff-unsatisfiable*

proof

fix E

assume *e-in*: $E \in EE$

have $DD\text{-of } E \subseteq CC' \cup \{\{\#\} - L\# \mid L. L \in\# E\}$

using *e-in d-of* **unfolding** $CC'\text{-def}$ **by** *auto*

moreover have $\neg \text{satisfiable } (DD\text{-of } E)$

using *e-in d-of* **by** *auto*

ultimately show $\neg \text{satisfiable } (CC' \cup \{\{\#\} - L\# \mid L. L \in\# E\})$

by (*rule unsatisfiable-mono*[*of DD-of E*])

qed

ultimately show $\exists CC' \subseteq CC. \text{finite } CC' \wedge CC' \models_e EE$

by *blast*

qed

3.3 Counterexample-Reducing Inference Systems

definition *clss-of-interp* $:: 'a \text{ set} \Rightarrow 'a \text{ literal multiset set}$ **where**

clss-of-interp $I = \{\{\#\}(\text{if } A \in I \text{ then Pos else Neg}) A\# \mid A. \text{True}\}$

lemma *true-clss-of-interp-iff-equal*[*simp*]: $J \models_s \text{clss-of-interp } I \iff J = I$

unfolding *clss-of-interp-def* *true-clss-def* *true-cls-def* *true-lit-def* **by** *force*

lemma *entails-iff-models*[*simp*]: $\text{clss-of-interp } I \models_e CC \iff I \models_s CC$

by *simp*

locale *clausal-counterex-reducing-inference-system* = *inference-system* *Inf*

for $Inf :: ('a :: \text{wellorder}) \text{ clause inference set} +$

fixes $J\text{-of} :: 'a \text{ clause set} \Rightarrow 'a \text{ interp}$

assumes *clausal-Inf-counterex-reducing*:

$\{\#\} \notin N \implies D \in N \implies \neg J\text{-of } N \models D \implies (\bigwedge C. C \in N \implies \neg J\text{-of } N \models C \implies D \leq C) \implies$

$\exists \iota \in \text{Inf}. \text{prems-of } \iota \neq \square \wedge \text{main-prem-of } \iota = D \wedge \text{set } (\text{side-prems-of } \iota) \subseteq N \wedge$
 $J\text{-of } N \Vdash_s \text{set } (\text{side-prems-of } \iota) \wedge \neg J\text{-of } N \Vdash \text{concl-of } \iota \wedge \text{concl-of } \iota < D$

begin

abbreviation $I\text{-of} :: 'a \text{ clause set} \Rightarrow 'a \text{ clause set}$ **where**

$I\text{-of } N \equiv \text{cls-of-interp } (J\text{-of } N)$

lemma *Inf-counterex-reducing*:

assumes

$\text{bot-ni-n}: N \cap \{\#\} = \{\}$ **and**

$\text{d-in-n}: D \in N$ **and**

$\text{n-ent-d}: \neg I\text{-of } N \Vdash_e \{D\}$ **and**

$\text{d-min}: \bigwedge C. C \in N \Longrightarrow \neg I\text{-of } N \Vdash_e \{C\} \Longrightarrow D \leq C$

shows $\exists \iota \in \text{Inf}. \text{prems-of } \iota \neq \square \wedge \text{main-prem-of } \iota = D \wedge \text{set } (\text{side-prems-of } \iota) \subseteq N$

$\wedge I\text{-of } N \Vdash_e \text{set } (\text{side-prems-of } \iota) \wedge \neg I\text{-of } N \Vdash_e \{\text{concl-of } \iota\} \wedge \text{concl-of } \iota < D$

using *bot-ni-n clausal-Inf-counterex-reducing d-in-n d-min n-ent-d* **by** *auto*

sublocale *counterex-reducing-inference-system* $\{\#\}$ (\Vdash_e) *Inf* $I\text{-of}$

$(<) :: 'a \text{ clause} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$

using *Inf-counterex-reducing*

by *unfold-locales (simp-all add: less-eq-multiset-def)*

end

3.4 Counterexample-Reducing Calculi Equipped with a Standard Redundancy Criterion

locale *clausal-counterex-reducing-calculus-with-standard-redundancy* =

calculus-with-standard-redundancy *Inf* $\{\#\}$ (\Vdash_e) $(<) :: 'a \text{ clause} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$ +

clausal-counterex-reducing-inference-system *Inf* $J\text{-of}$

for

$\text{Inf} :: ('a :: \text{wellorder}) \text{ clause inference set}$ **and**

$\text{J-of} :: 'a \text{ clause set} \Rightarrow 'a \text{ set}$

begin

sublocale *counterex-reducing-calculus-with-standard-inference-redundancy* $\{\#\}$ *Inf* (\Vdash_e) *Red-I*

Red-F $I\text{-of } (<) :: 'a \text{ clause} \Rightarrow 'a \text{ clause} \Rightarrow \text{bool}$

proof *unfold-locales*

fix $C D :: 'a \text{ clause}$

show $C \neq D \Longrightarrow C < D \vee D < C$

by *fastforce*

qed

lemma *clausal-saturated-model*: $\text{saturated } N \Longrightarrow \{\#\} \notin N \Longrightarrow J\text{-of } N \Vdash_s N$

by *(simp add: saturated-model[simplified])*

corollary *clausal-saturated-complete*: $\text{saturated } N \Longrightarrow (\forall I. \neg I \Vdash_s N) \Longrightarrow \{\#\} \in N$

using *clausal-saturated-model* **by** *blast*

end

end

4 Application of the Saturation Framework to Bachmair and Ganzinger's RP

```

theory FO-Ordered-Resolution-Prover-Revisited
imports
  Ordered-Resolution-Prover.FO-Ordered-Resolution-Prover
  Saturation-Framework.Given-Clause-Architectures
  Clausal-Calculus
  Soundness
begin

```

The main results about Bachmair and Ganzinger's RP prover, as established in Section 4.3 of their *Handbook* chapter and formalized by Schlichtkrull et al., are re-proved here using the saturation framework of Waldmann et al.

4.1 Setup

```

no-notation true-lit (infix  $\models_l$  50)
no-notation true-cls (infix  $\models$  50)
no-notation true-clss (infix  $\models_s$  50)
no-notation true-cls-mset (infix  $\models_m$  50)

```

```

hide-type (open) Inference-System.inference

```

```

hide-const (open) Inference-System.Infer Inference-System.main-prem-of
  Inference-System.side-prems-of Inference-System.prems-of Inference-System.concl-of
  Inference-System.concls-of Inference-System.infer-from

```

```

type-synonym 'a old-inference = 'a Inference-System.inference

```

```

abbreviation old-Infer  $\equiv$  Inference-System.Infer
abbreviation old-side-prems-of  $\equiv$  Inference-System.side-prems-of
abbreviation old-main-prem-of  $\equiv$  Inference-System.main-prem-of
abbreviation old-concl-of  $\equiv$  Inference-System.concl-of
abbreviation old-prems-of  $\equiv$  Inference-System.prems-of
abbreviation old-concls-of  $\equiv$  Inference-System.concls-of
abbreviation old-infer-from  $\equiv$  Inference-System.infer-from

```

```

lemmas old-infer-from-def = Inference-System.infer-from-def

```

4.2 Library

```

lemma set-zip-replicate-right[simp]:
  set (zip xs (replicate (length xs) y)) = ( $\lambda x. (x, y)$ ) ` set xs
  by (induct xs) auto

```

4.3 Ground Layer

```

context FO-resolution-prover
begin

```

```

no-notation RP (infix  $\rightsquigarrow$  50)
notation RP (infix  $\rightsquigarrow_{RP}$  50)

```

```

interpretation gr: ground-resolution-with-selection S-M S M

```

using *selection-axioms* **by** *unfold-locales (fact S-M-selects-subseteq S-M-selects-neg-lits)+*

definition *G-Inf* :: 'a clause set \Rightarrow 'a clause inference set **where**

G-Inf *M* = {*Infer* (*CAs* @ [*DA*]) *E* | *CAs* *DA* *AAs* *As* *E*. *gr.ord-resolve* *M* *CAs* *DA* *AAs* *As* *E*}

lemma *G-Inf-have-prems*: $\iota \in G\text{-Inf } M \implies \text{prems-of } \iota \neq []$

unfolding *G-Inf-def* **by** *auto*

lemma *G-Inf-reductive*: $\iota \in G\text{-Inf } M \implies \text{concl-of } \iota < \text{main-prem-of } \iota$

unfolding *G-Inf-def* **by** (*auto* *dest*: *gr.ord-resolve-reductive*)

interpretation *G*: *sound-inference-system* *G-Inf* *M* {*{#}*} (\models_e)

proof

fix ι

assume *i-in*: $\iota \in G\text{-Inf } M$

moreover

{

fix *I*

assume *I-ent-prems*: $I \models_s \text{set } (\text{prems-of } \iota)$

obtain *CAs* *AAs* *As* **where**

the-inf: *gr.ord-resolve* *M* *CAs* (*main-prem-of* ι) *AAs* *As* (*concl-of* ι) **and**

CAs: *CAs* = *side-prems-of* ι

using *i-in* **unfolding** *G-Inf-def* **by** *auto*

then have $I \models \text{concl-of } \iota$

using *gr.ord-resolve-sound*[*of* *M* *CAs* *main-prem-of* ι *AAs* *As* *concl-of* ι *I*]

by (*metis* *I-ent-prems* *G-Inf-have-prems* *i-in* *insert-is-Un* *set-mset-mset* *set-prems-of* *true-clss-insert* *true-clss-set-mset*)

}

ultimately show $\text{set } (\text{inference.prems-of } \iota) \models_e \{\text{concl-of } \iota\}$

by *simp*

qed

interpretation *G*: *clausal-counterex-reducing-inference-system* *G-Inf* *M* *gr.INTERP* *M*

proof

fix *N* *D*

assume

$\{\#\} \notin N$ **and**

$D \in N$ **and**

$\neg \text{gr.INTERP } M N \models D$ **and**

$\bigwedge C. C \in N \implies \neg \text{gr.INTERP } M N \models C \implies D \leq C$

then obtain *CAs* *AAs* *As* *E* **where**

cas-in: $\text{set } CAs \subseteq N$ **and**

n-mod-cas: *gr.INTERP* *M* *N* $\models_m \text{mset } CAs$ **and**

ca-prod: $\bigwedge CA. CA \in \text{set } CAs \implies \text{gr.production } M N CA \neq \{\}$ **and**

e-res: *gr.ord-resolve* *M* *CAs* *D* *AAs* *As* *E* **and**

n-nmod-e: $\neg \text{gr.INTERP } M N \models E$ **and**

e-lt-d: $E < D$

using *gr.ord-resolve-counterex-reducing* **by** *blast*

define ι **where**

$\iota = \text{Infer } (CAs @ [D]) E$

have $\iota \in G\text{-Inf } M$

unfolding $\iota\text{-def}$ *G-Inf-def* **using** *e-res* **by** *auto*

moreover have $\text{prems-of } \iota \neq []$

unfolding $\iota\text{-def}$ **by** *simp*

moreover have *main-prem-of* $\iota = D$
unfolding ι -def **by** *simp*
moreover have *set (side-prems-of* ι) $\subseteq N$
unfolding ι -def **using** *cas-in* **by** *simp*
moreover have *gr.INTERP M N* \models_s *set (side-prems-of* ι)
unfolding ι -def **using** *n-mod-cas ca-prod* **by** (*simp add: gr.productive-imp-INTERP true-cls-def*)
moreover have \neg *gr.INTERP M N* \models *concl-of* ι
unfolding ι -def **using** *n-nmod-e* **by** *simp*
moreover have *concl-of* $\iota < D$
unfolding ι -def **using** *e-lt-d* **by** *simp*
ultimately show $\exists \iota \in G\text{-Inf } M. \text{prems-of } \iota \neq [] \wedge \text{main-prem-of } \iota = D \wedge \text{set (side-prems-of } \iota) \subseteq N$
 \wedge
gr.INTERP M N \models_s *set (side-prems-of* ι) $\wedge \neg$ *gr.INTERP M N* \models *concl-of* $\iota \wedge \text{concl-of } \iota < D$
by *blast*
qed

interpretation *G: clausal-counterex-reducing-calculus-with-standard-redundancy G-Inf M*
gr.INTERP M
using *G-Inf-have-prems G-Inf-reductive*
by (*unfold-locales*) *simp-all*

interpretation *G: statically-complete-calculus {{#}} G-Inf M (\models_e) G.Red-I M G.Red-F*
by *unfold-locales (use G.clausal-saturated-complete in blast)*

4.4 First-Order Layer

abbreviation $\mathcal{G}\text{-F} :: \langle 'a \text{ clause} \Rightarrow 'a \text{ clause set} \rangle$ **where**
 $\langle \mathcal{G}\text{-F} \equiv \text{grounding-of-cls} \rangle$

abbreviation $\mathcal{G}\text{-Fset} :: \langle 'a \text{ clause set} \Rightarrow 'a \text{ clause set} \rangle$ **where**
 $\langle \mathcal{G}\text{-Fset} \equiv \text{grounding-of-cls} \rangle$

lemmas $\mathcal{G}\text{-F-def} = \text{grounding-of-cls-def}$
lemmas $\mathcal{G}\text{-Fset-def} = \text{grounding-of-cls-def}$

definition $\mathcal{G}\text{-I} :: \langle 'a \text{ clause set} \Rightarrow 'a \text{ clause inference} \Rightarrow 'a \text{ clause inference set} \rangle$ **where**
 $\langle \mathcal{G}\text{-I } M \ \iota = \{ \text{Infer (prems-of } \iota \cdot \text{cl } \varrho_s) (\text{concl-of } \iota \cdot \varrho) \mid \varrho \ \varrho_s. \text{is-ground-subst-list } \varrho_s \wedge \text{is-ground-subst } \varrho \wedge \text{Infer (prems-of } \iota \cdot \text{cl } \varrho_s) (\text{concl-of } \iota \cdot \varrho) \in G\text{-Inf } M \} \rangle$

abbreviation
 $\mathcal{G}\text{-I-opt} :: \langle 'a \text{ clause set} \Rightarrow 'a \text{ clause inference} \Rightarrow 'a \text{ clause inference set option} \rangle$
where
 $\langle \mathcal{G}\text{-I-opt } M \ \iota \equiv \text{Some } (\mathcal{G}\text{-I } M \ \iota) \rangle$

definition $F\text{-Inf} :: 'a \text{ clause inference set}$ **where**
 $F\text{-Inf} = \{ \text{Infer (CAs @ [DA]) } E \mid \text{CAs DA AAs As } \sigma \ E. \text{ord-resolve-rename } S \ \text{CAs DA AAs As } \sigma \ E \}$

lemma $F\text{-Inf-have-prems}: \iota \in F\text{-Inf} \implies \text{prems-of } \iota \neq []$
unfolding $F\text{-Inf-def}$ **by** *force*

interpretation *F: lifting-intersection F-Inf {{#}} UNIV G-Inf $\lambda N. (\models_e) G.Red-I \lambda N. G.Red-F$*
 $\{\{\#\}\} \lambda N. \mathcal{G}\text{-F } \mathcal{G}\text{-I-opt } \lambda D \ C \ C'. \text{False}$
proof (*unfold-locales; (intro ball) ?*)
show $UNIV \neq \{\}$
by (*rule UNIV-not-empty*)

```

next
  show consequence-relation  $\{\{\#\}\}$  ( $\models_e$ )
    by (fact consequence-relation-axioms)
next
  show  $\wedge M. \text{tiebreaker-lifting } \{\{\#\}\} F\text{-Inf } \{\{\#\}\} (\models_e) (G\text{-Inf } M) (G\text{-Red-I } M) G\text{-Red-F } \mathcal{G}\text{-F } (G\text{-I-opt } M)$ 
    ( $\lambda D C C'. \text{False}$ )
  proof
    fix  $M \iota$ 
    show the  $(G\text{-I-opt } M \iota) \subseteq G\text{-Red-I } M (G\text{-F } (\text{concl-of } \iota))$ 
      unfolding option.sel
    proof
      fix  $\iota'$ 
      assume  $\iota' \in G\text{-I } M \iota$ 
      then obtain  $\varrho \varrho s$  where
         $\iota': \iota' = \text{Infer } (\text{prems-of } \iota \cdot \text{cl } \varrho s) (\text{concl-of } \iota \cdot \varrho)$  and
         $\varrho\text{-gr}: \text{is-ground-subst } \varrho$  and
         $\varrho\text{-infer}: \text{Infer } (\text{prems-of } \iota \cdot \text{cl } \varrho s) (\text{concl-of } \iota \cdot \varrho) \in G\text{-Inf } M$ 
      unfolding  $\mathcal{G}\text{-I-def}$  by blast

      show  $\iota' \in G\text{-Red-I } M (G\text{-F } (\text{concl-of } \iota))$ 
        unfolding  $G\text{-Red-I-def } G\text{-redundant-infer-def mem-Collect-eq}$  using  $\iota' \varrho\text{-gr } \varrho\text{-infer}$ 
        by (metis inference.sel(2)  $G\text{-Inf-reductive empty-iff ground-subst-ground-clss}$ 
           $\text{grounding-of-clss-ground insert-iff subst-clss-eq-grounding-of-clss-subset-eq}$ 
           $\text{true-clss-union}$ )
    qed
  qed (auto simp:  $\mathcal{G}\text{-F-def ex-ground-subst}$ )
qed

notation  $F.\text{entails-}\mathcal{G}$  (infix  $\models_{\mathcal{G}}$  50)

lemma  $F\text{-entails-}\mathcal{G}\text{-iff}: N1 \models_{\mathcal{G}} N2 \iff \bigcup (G\text{-F } ' N1) \models_e \bigcup (G\text{-F } ' N2)$ 
  unfolding  $F.\text{entails-}\mathcal{G}\text{-def}$  by simp

lemma  $\text{true-Union-grounding-of-clss-iff}$ :
   $I \models_s (\bigcup C \in N. \{C \cdot \sigma \mid \sigma. \text{is-ground-subst } \sigma\}) \iff (\forall \sigma. \text{is-ground-subst } \sigma \implies I \models_s N \cdot \text{cs } \sigma)$ 
  unfolding  $\text{true-clss-def subst-clss-def}$  by blast

interpretation  $F$ : sound-inference-system  $F\text{-Inf } \{\{\#\}\} (\models_{\mathcal{G}})$ 
proof
  fix  $\iota$ 
  assume  $i\text{-in}: \iota \in F\text{-Inf}$ 
  moreover
  {
    fix  $I \eta$ 
    assume
       $I\text{-entails-prems}: \forall \sigma. \text{is-ground-subst } \sigma \implies I \models_s \text{set } (\text{prems-of } \iota) \cdot \text{cs } \sigma$  and
       $\eta\text{-gr}: \text{is-ground-subst } \eta$ 
    obtain  $CAs AAs As \sigma$  where
       $\text{the-inf}: \text{ord-resolve-rename } S CAs (\text{main-prem-of } \iota) AAs As \sigma (\text{concl-of } \iota)$  and
       $CAs: CAs = \text{side-prems-of } \iota$ 
    using  $i\text{-in}$  unfolding  $F\text{-Inf-def}$  by auto
    have  $\text{prems}: \text{mset } (\text{prems-of } \iota) = \text{mset } (\text{side-prems-of } \iota) + \{\#\text{main-prem-of } \iota\}$ 
    by (metis (no-types)  $F\text{-Inf-have-prems}[OF i-in]$   $\text{add.right-neutral append-Cons append-Nil2}$ 
       $\text{append-butlast-last-id mset.simps}(2)$   $\text{mset-rev mset-single-iff-right rev-append}$ )
  }

```

rev-is-Nil-conv union-mset-add-mset-right
have $I \Vdash \text{concl-of } \iota \cdot \eta$
using *ord-resolve-rewrite-sound*[*OF the-inf, of I η, OF - η-gr*]
unfolding *CAs prems*[*symmetric*] **using** *I-entails-prems*
by (*metis set-mset-mset set-mset-subst-cls-mset-subst-cls true-cls-set-mset*)
}
ultimately show *set (inference.prems-of ι) $\Vdash_{\mathcal{G}e}$ {concl-of ι}*
unfolding *F.entails- \mathcal{G} -def \mathcal{G} -F-def true-Union-grounding-of-cls-iff* **by** *auto*
qed

lemma *G-Inf-overapprox-F-Inf: $\iota_0 \in G.\text{Inf-from } M (\bigcup (\mathcal{G}\text{-F } ' M)) \implies \exists \iota \in F.\text{Inf-from } M. \iota_0 \in \mathcal{G}\text{-I } M \iota$*

proof –

assume $\iota_0\text{-in: } \iota_0 \in G.\text{Inf-from } M (\bigcup (\mathcal{G}\text{-F } ' M))$
have $\text{prems-}\iota_0\text{-in: } \text{set } (\text{prems-of } \iota_0) \subseteq \bigcup (\mathcal{G}\text{-F } ' M)$
using $\iota_0\text{-in}$ **unfolding** *G-Inf-from-def* **by** *simp*
note $\iota_0\text{-G-Inf} = G.\text{Inf-if-Inf-from}$ [*OF $\iota_0\text{-in}$*]
then obtain *CAs DA AAs As E* **where**
 $\text{gr-res: } \langle \text{gr.ord-resolve } M \text{ CAs DA AAs As } E \rangle$ **and**
 $\iota_0\text{-is: } \langle \iota_0 = \text{Infer } (CAs @ [DA]) \ E \rangle$
unfolding *G-Inf-def* **by** *auto*

have $\text{CAs-in: } \langle \text{set } CAs \subseteq \text{set } (\text{prems-of } \iota_0) \rangle$
by (*simp add: $\iota_0\text{-is subsetI}$*)

then have $\text{ground-CAs: } \langle \text{is-ground-cls-list } CAs \rangle$

using $\text{prems-}\iota_0\text{-in}$ *union-grounding-of-cls-ground is-ground-cls-list-def is-ground-cls-def* **by** *auto*

have $\text{DA-in: } \langle DA \in \text{set } (\text{prems-of } \iota_0) \rangle$

using $\iota_0\text{-is}$ **by** *simp*

then have $\text{ground-DA: } \langle \text{is-ground-cls } DA \rangle$

using $\text{prems-}\iota_0\text{-in}$ *union-grounding-of-cls-ground is-ground-cls-def* **by** *auto*

obtain σ **where**

$\text{grounded-res: } \langle \text{ord-resolve } (S\text{-M } S\text{-M}) \text{ CAs DA AAs As } \sigma \ E \rangle$

using *ground-ord-resolve-imp-ord-resolve*[*OF ground-DA ground-CAs*
 $\text{gr.ground-resolution-with-selection-axioms gr-res}$] **by** *auto*

have $\text{prems-ground: } \langle \{DA\} \cup \text{set } CAs \subseteq \mathcal{G}\text{-Fset } M \rangle$

using $\text{prems-}\iota_0\text{-in}$ *CAs-in DA-in* **unfolding** *\mathcal{G} -Fset-def* **by** *fast*

obtain $\eta_s \eta \eta_2 \text{ CAs0 DA0 AAs0 As0 E0 } \tau$ **where**

$\text{ground-n: } \text{is-ground-subst } \eta$ **and**

$\text{ground-ns: } \text{is-ground-subst-list } \eta_s$ **and**

$\text{ground-n2: } \text{is-ground-subst } \eta_2$ **and**

$\text{ngr-res: } \text{ord-resolve-rewrite } S \text{ CAs0 DA0 AAs0 As0 } \tau \ E0$ **and**

$\text{CAs0-is: } \text{CAs0} \cdot \text{cl } \eta_s = \text{CAs}$ **and**

$\text{DA0-is: } \text{DA0} \cdot \eta = \text{DA}$ **and**

$\text{E0-is: } \text{E0} \cdot \eta_2 = \text{E}$ **and**

$\text{prems-in: } \{DA0\} \cup \text{set } \text{CAs0} \subseteq M$ **and**

$\text{len-CAs0: } \text{length } \text{CAs0} = \text{length } \text{CAs}$ **and**

$\text{len-ns: } \text{length } \eta_s = \text{length } \text{CAs}$

using *ord-resolve-rewrite-lifting*[*OF - grounded-res selection-axioms prems-ground*] *sel-stable*
by *smt*

have $\text{length } \text{CAs0} = \text{length } \eta_s$

using len-CAs0 len-ns **by** *simp*

then have $\iota_0\text{-is': } \iota_0 = \text{Infer } ((\text{CAs0} @ [\text{DA0}]) \cdot \text{cl } (\eta_s @ [\eta])) (\text{E0} \cdot \eta_2)$

unfolding $\iota_0\text{-is}$ **by** (*auto simp: CAs0-is[symmetric] DA0-is[symmetric] E0-is[symmetric]*)

define $\iota :: 'a \text{ clause inference where}$
 $\iota = \text{Infer } (CAs0 \ @ \ [DA0]) \ E0$

have $i\text{-}F\text{-}Inf: \langle \iota \in F\text{-}Inf \rangle$
unfolding $\iota\text{-}def \ F\text{-}Inf\text{-}def$ **using** $ngr\text{-}res$ **by** $auto$

have $\exists \varrho \ \varrho s. \iota_0 = \text{Infer } ((CAs0 \ @ \ [DA0]) \ \cdot\text{cl } \varrho s) \ (E0 \ \cdot \ \varrho) \wedge \text{is-ground-subst-list } \varrho s$
 $\wedge \text{is-ground-subst } \varrho \wedge \text{Infer } ((CAs0 \ @ \ [DA0]) \ \cdot\text{cl } \varrho s) \ (E0 \ \cdot \ \varrho) \in G\text{-}Inf \ M$
by $(\text{rule } exI[\text{of} - \eta 2], \text{rule } exI[\text{of} - \eta s \ @ \ [\eta]], \text{use } \text{ground-ns} \ \mathbf{in}$
 $\langle \text{auto intro: ground-n ground-n2 } \iota_0\text{-}G\text{-}Inf[\text{unfolded } \iota_0\text{-}is']$
 $\text{simp: } \iota_0\text{-}is' \ \text{is-ground-subst-list-def} \rangle)$

then have $\langle \iota_0 \in \mathcal{G}\text{-}I \ M \ \iota \rangle$
unfolding $\mathcal{G}\text{-}I\text{-}def \ \iota\text{-}def \ CAs0\text{-}is[\text{symmetric}] \ DA0\text{-}is[\text{symmetric}] \ E0\text{-}is[\text{symmetric}]$ **by** simp

moreover have $\langle \iota \in F\text{-}Inf\text{-}from \ M \rangle$
using $\text{prems-in } i\text{-}F\text{-}Inf$ **unfolding** $F\text{-}Inf\text{-}from\text{-}def \ \iota\text{-}def$ **by** simp

ultimately show $?thesis$
by blast

qed

interpretation $F: \text{statically-complete-calculus } \{\{\#\}\} \ F\text{-}Inf \ (\models_{\mathcal{G}} e) \ F\text{-}Red\text{-}I\text{-}\mathcal{G} \ F\text{-}Red\text{-}F\text{-}\mathcal{G}\text{-}empty$

proof $(\text{rule } F\text{-}stat\text{-}ref\text{-}comp\text{-}to\text{-}non\text{-}ground\text{-}fam\text{-}inter; \text{clarsimp}; \text{intro } exI?)$
show $\bigwedge M. \text{statically-complete-calculus } \{\{\#\}\} \ (G\text{-}Inf \ M) \ (\models e) \ (G\text{-}Red\text{-}I \ M) \ G\text{-}Red\text{-}F$
by $(\text{fact } G\text{-}statically\text{-}complete\text{-}calculus\text{-}axioms)$

next
fix N
assume $F\text{-}saturated \ N$
show $F\text{-}ground\text{-}Inf\text{-}from\text{-}q \ N \ (\bigcup (G\text{-}F \ ' \ N)) \subseteq \{\iota. \exists \iota' \in F\text{-}Inf\text{-}from \ N. \iota \in \mathcal{G}\text{-}I \ N \ \iota'\}$
 $\cup G\text{-}Red\text{-}I \ N \ (\bigcup (G\text{-}F \ ' \ N))$
using $G\text{-}Inf\text{-}overapprox\text{-}F\text{-}Inf$ **unfolding** $F\text{-}ground\text{-}Inf\text{-}from\text{-}q\text{-}def \ \mathcal{G}\text{-}I\text{-}def$ **by** fastforce

qed

4.5 Labeled First-Order or Given Clause Layer

datatype $label = New \ | \ Processed \ | \ Old$

abbreviation $F\text{-}Equiv :: 'a \ \text{clause} \Rightarrow 'a \ \text{clause} \Rightarrow bool \ (\text{infix } \doteq \ 50) \ \text{where}$
 $C \doteq D \equiv \text{generalizes } C \ D \wedge \text{generalizes } D \ C$

abbreviation $F\text{-}Prec :: 'a \ \text{clause} \Rightarrow 'a \ \text{clause} \Rightarrow bool \ (\text{infix } \prec \cdot \ 50) \ \text{where}$
 $C \prec \cdot D \equiv \text{strictly-generalizes } C \ D$

fun $L\text{-}Prec :: label \Rightarrow label \Rightarrow bool \ (\text{infix } \sqsubset l \ 50) \ \text{where}$
 $Old \ \sqsubset l \ \longleftrightarrow l \neq Old$
 $| \ Processed \ \sqsubset l \ \longleftrightarrow l = New$
 $| \ New \ \sqsubset l \ \longleftrightarrow False$

lemma $irrefl\text{-}L\text{-}Prec: \neg l \ \sqsubset l$
by $(\text{cases } l) \ \text{auto}$

lemma $trans\text{-}L\text{-}Prec: l1 \ \sqsubset l2 \ \Longrightarrow l2 \ \sqsubset l3 \ \Longrightarrow l1 \ \sqsubset l3$
by $(\text{cases } l1; \text{cases } l2; \text{cases } l3) \ \text{auto}$

lemma $wf\text{-}L\text{-}Prec: wfP \ (\sqsubset l)$
by $(\text{metis } L\text{-}Prec.\text{elims}(2) \ L\text{-}Prec.\text{simps}(3) \ \text{not-accp-down } wfP\text{-}accp\text{-}iff)$

interpretation $FL: \text{given-clause } \{\{\#\}\} \ F\text{-}Inf \ \{\{\#\}\} \ UNIV \ \lambda N. (\models e) \ G\text{-}Inf \ G\text{-}Red\text{-}I$

$\lambda N. G.Red-F \lambda N. \mathcal{G}-F \mathcal{G}-I-opt (\doteq) (\prec \cdot) (\sqsubset l) Old$
proof (*unfold-locales; (intro ballI)?*)
show *equivp* (\doteq)
unfolding *equivp-def* **by** (*meson generalizes-refl generalizes-trans*)
next
show *po-on* ($\prec \cdot$) *UNIV*
unfolding *po-on-def irreflp-on-def transp-on-def*
using *strictly-generalizes-irrefl strictly-generalizes-trans* **by** *auto*
next
show *wfp-on* ($\prec \cdot$) *UNIV*
unfolding *wfp-on-UNIV* **by** (*metis wf-strictly-generalizes*)
next
show *po-on* ($\sqsubset l$) *UNIV*
unfolding *po-on-def irreflp-on-def transp-on-def* **using** *irrefl-L-Prec trans-L-Prec* **by** *blast*
next
show *wfp-on* ($\sqsubset l$) *UNIV*
unfolding *wfp-on-UNIV* **by** (*rule wf-L-Prec*)
next
fix *C1 D1 C2 D2*
assume
 $C1 \doteq D1$
 $C2 \doteq D2$
 $C1 \prec \cdot C2$
then show $D1 \prec \cdot D2$
by (*smt antisym size-mset-mono size-subst strictly-generalizes-def generalizes-def*
generalizes-trans)
next
fix *N C1 C2*
assume $C1 \doteq C2$
then show $\mathcal{G}-F C1 \subseteq \mathcal{G}-F C2$
unfolding *generalizes-def G-F-def* **by** *clarsimp (metis is-ground-comp-subst subst-cls-comp-subst)*
next
fix *N C2 C1*
assume $C2 \prec \cdot C1$
then show $\mathcal{G}-F C1 \subseteq \mathcal{G}-F C2$
unfolding *strictly-generalizes-def generalizes-def G-F-def*
by *clarsimp (metis is-ground-comp-subst subst-cls-comp-subst)*
next
show $\exists l. L-Prec Old l$
using *L-Prec.simps(1)* **by** *blast*
qed (*auto simp: F-Inf-have-prems*)

notation *FL.Prec-FL* (**infix** \sqsubset 50)
notation *FL.entails-G-L* (**infix** $\Vdash \mathcal{G}Le$ 50)
notation *FL.derive* (**infix** $\triangleright L$ 50)
notation *FL.step* (**infix** $\rightsquigarrow GC$ 50)

lemma *FL-Red-F-eq*:
 $FL.Red-F N =$
 $\{C. \forall D \in \mathcal{G}-F (fst C). D \in G.Red-F (\bigcup (\mathcal{G}-F 'fst' N)) \vee (\exists E \in N. E \sqsubset C \wedge D \in \mathcal{G}-F (fst E))\}$
unfolding *FL.Red-F-def FL.Red-F-G-q-def* **by** *auto*

lemma *mem-FL-Red-F-because-G-Red-F*:
 $(\forall D \in \mathcal{G}-F (fst Cl). D \in G.Red-F (\bigcup (\mathcal{G}-F 'fst' N))) \implies Cl \in FL.Red-F N$
unfolding *FL-Red-F-eq* **by** *auto*

lemma *mem-FL-Red-F-because-Prec-FL:*

$(\forall D \in \mathcal{G}\text{-}F \text{ (fst } Cl). \exists El \in N. El \sqsubset Cl \wedge D \in \mathcal{G}\text{-}F \text{ (fst } El)) \implies Cl \in FL\text{-}Red\text{-}F \ N$
unfolding *FL-Red-F-eq* **by** *auto*

4.6 Resolution Prover Layer

interpretation *sq: selection S-Q Sts*

unfolding *S-Q-def* **using** *S-M-selects-subseteq S-M-selects-neg-lits selection-axioms*
by *unfold-locales auto*

interpretation *gd: ground-resolution-with-selection S-Q Sts*

by *unfold-locales*

interpretation *src: standard-redundancy-criterion-counterex-reducing gd.ord-Γ Sts*

ground-resolution-with-selection.INTERP (S-Q Sts)

by *unfold-locales*

definition *lclss-of-state* :: 'a state \Rightarrow ('a clause \times label) set **where**

lclss-of-state St =

$(\lambda C. (C, \text{New})) \text{ ` } N\text{-of-state } St \cup (\lambda C. (C, \text{Processed})) \text{ ` } P\text{-of-state } St$
 $\cup (\lambda C. (C, \text{Old})) \text{ ` } Q\text{-of-state } St$

lemma *image-hd-lclss-of-state[simp]:* *fst ` lclss-of-state St = clss-of-state St*

unfolding *lclss-of-state-def* **by** *(auto simp: image-Un image-comp)*

lemma *insert-lclss-of-state[simp]:*

insert (C, New) (lclss-of-state (N, P, Q)) = lclss-of-state (N \cup {C}, P, Q)

insert (C, Processed) (lclss-of-state (N, P, Q)) = lclss-of-state (N, P \cup {C}, Q)

insert (C, Old) (lclss-of-state (N, P, Q)) = lclss-of-state (N, P, Q \cup {C})

unfolding *lclss-of-state-def image-def* **by** *auto*

lemma *union-lclss-of-state[simp]:*

lclss-of-state (N1, P1, Q1) \cup lclss-of-state (N2, P2, Q2) =

lclss-of-state (N1 \cup N2, P1 \cup P2, Q1 \cup Q2)

unfolding *lclss-of-state-def* **by** *auto*

lemma *mem-lclss-of-state[simp]:*

(C, New) \in lclss-of-state (N, P, Q) \iff C \in N

(C, Processed) \in lclss-of-state (N, P, Q) \iff C \in P

(C, Old) \in lclss-of-state (N, P, Q) \iff C \in Q

unfolding *lclss-of-state-def image-def* **by** *auto*

lemma *lclss-Liminf-commute:*

Liminf-llist (lmap lclss-of-state Sts) = lclss-of-state (Liminf-state Sts)

proof –

have \langle *Liminf-llist (lmap lclss-of-state Sts) =*

$(\lambda C. (C, \text{New})) \text{ ` } \text{Liminf-llist (lmap } N\text{-of-state Sts) } \cup$

$(\lambda C. (C, \text{Processed})) \text{ ` } \text{Liminf-llist (lmap } P\text{-of-state Sts) } \cup$

$(\lambda C. (C, \text{Old})) \text{ ` } \text{Liminf-llist (lmap } Q\text{-of-state Sts) } \rangle$

unfolding *lclss-of-state-def* **using** *Liminf-llist-lmap-union Liminf-llist-lmap-image*

by *(smt Pair-inject Un-iff disjoint-iff-not-equal imageE inj-onI label.simps(1,3,5)*

llist.map-cong)

then show *?thesis*

unfolding *lclss-of-state-def Liminf-state-def* **by** *auto*

qed

lemma *GC-tautology-step*:

assumes *tauto*: $Neg\ A \in\# \ C\ Pos\ A \in\# \ C$

shows $lclss\text{-of}\text{-state}\ (N \cup \{C\}, P, Q) \rightsquigarrow_{GC} lclss\text{-of}\text{-state}\ (N, P, Q)$

proof –

have *c ϑ -red*: $C\vartheta \in G.Red\text{-}F\ (\bigcup D \in N'.\ \mathcal{G}\text{-}F\ (fst\ D))$ **if** *in-g*: $C\vartheta \in \mathcal{G}\text{-}F\ C$

for $N' :: ('a\ clause \times label)\ set$ **and** $C\vartheta$

proof –

obtain ϑ **where**

$C\vartheta = C \cdot \vartheta$

using *in-g unfolding* $\mathcal{G}\text{-}F\text{-}def$ **by** *blast*

then have $Neg\ (A \cdot a\ \vartheta) \in\# \ C\vartheta$ **and** $Pos\ (A \cdot a\ \vartheta) \in\# \ C\vartheta$

using *tauto Neg-Melem-subst-atm-subst-cls Pos-Melem-subst-atm-subst-cls* **by** *auto*

then have $\{\} \models_e \{C\vartheta\}$

unfolding *true-cls-def true-cls-def true-lit-def if-distrib-fun*

by (*metis literal.disc literal.sel singletonD*)

then show *?thesis*

unfolding *G.Red-F-def* **by** *auto*

qed

show *?thesis*

proof (*rule FL.step.process[of - lclss-of-state (N, P, Q) {(C, New)} - {}]*)

show $\langle \{(C, New)\} \subseteq FL.Red\text{-}F\text{-}\mathcal{G}\ (lclss\text{-of}\text{-state}\ (N, P, Q) \cup \{\}) \rangle$

using *mem-FL-Red-F-because-G-Red-F c ϑ -red[of - lclss-of-state (N, P, Q)]*

unfolding *lclss-of-state-def* **by** *auto*

qed (*auto simp: lclss-of-state-def FL.active-subset-def*)

qed

lemma *GC-subsumption-step*:

assumes

d-in: $Dl \in N$ **and**

d-sub-c: $strictly\text{-subsumes}\ (fst\ Dl)\ (fst\ Cl) \vee subsumes\ (fst\ Dl)\ (fst\ Cl) \wedge snd\ Dl \sqsubseteq_l\ snd\ Cl$

shows $N \cup \{Cl\} \rightsquigarrow_{GC} N$

proof –

have *d-sub'-c*: $Cl \in FL.Red\text{-}F\ \{Dl\} \vee Dl \sqsubseteq Cl$

proof (*cases size (fst Dl) = size (fst Cl)*)

case *True*

assume *sizes-eq*: $\langle size\ (fst\ Dl) = size\ (fst\ Cl) \rangle$

have $\langle size\ (fst\ Dl) = size\ (fst\ Cl) \implies$

$strictly\text{-subsumes}\ (fst\ Dl)\ (fst\ Cl) \vee subsumes\ (fst\ Dl)\ (fst\ Cl) \wedge snd\ Dl \sqsubseteq_l\ snd\ Cl \implies$

$Dl \sqsubseteq Cl \rangle$

unfolding *FL.Prec-FL-def*

unfolding *generalizes-def strictly-generalizes-def strictly-subsumes-def subsumes-def*

by (*metis size-subst subset-mset.order-refl subseteq-mset-size-eql*)

then have $Dl \sqsubseteq Cl$

using *sizes-eq d-sub-c* **by** *auto*

then show *?thesis*

by (*rule disjI2*)

next

case *False*

then have *d-ssub-c*: $strictly\text{-subsumes}\ (fst\ Dl)\ (fst\ Cl)$

using *d-sub-c* **unfolding** *strictly-subsumes-def subsumes-def*

by (*metis size-subst strict-subset-subst-strictly-subsumes strictly-subsumes-antisym subset-mset.antisym-conv2*)

have $Cl \in FL.Red\text{-}F\ \{Dl\}$

proof (rule mem-FL-Red-F-because-G-Red-F)
show $\langle \forall D \in \mathcal{G}\text{-F} (fst\ Cl). D \in G.Red\text{-F} (\bigcup (\mathcal{G}\text{-F} \text{ 'fst' } \{Dl\})) \rangle$
using *d-ssub-c unfolding G.Red-F-def strictly-subsumes-def subsumes-def G-F-def*
proof *clarsimp*
fix $\sigma\ \sigma'$
assume
fst-not-in: $\langle \forall \sigma. \neg\ fst\ Cl \cdot \sigma \subseteq\#\ fst\ Dl \rangle$ **and**
fst-in: $\langle fst\ Dl \cdot \sigma \subseteq\#\ fst\ Cl \rangle$ **and**
gr-sig: $\langle is\text{-ground}\text{-subst}\ \sigma' \rangle$
have $\langle \{fst\ Dl \cdot \sigma \cdot \sigma'\} \subseteq \{fst\ Dl \cdot \sigma \mid \sigma. is\text{-ground}\text{-subst}\ \sigma \} \rangle$
using *gr-sig*
by (*metis (mono-tags, lifting) is-ground-comp-subst mem-Collect-eq singletonD subsetI subst-cls-comp-subst*)
moreover **have** $\langle \forall I. I \models_s \{fst\ Dl \cdot \sigma \cdot \sigma'\} \longrightarrow I \models fst\ Cl \cdot \sigma' \rangle$
using *fst-in*
by (*meson subst-cls-mono-mset true-cls-insert true-cls-subclause*)
moreover **have** $\langle \forall D \in \{fst\ Dl \cdot \sigma \cdot \sigma'\}. D < fst\ Cl \cdot \sigma' \rangle$
using *fst-not-in fst-in gr-sig*
proof *clarify*
show $\langle \forall \sigma. \neg\ fst\ Cl \cdot \sigma \subseteq\#\ fst\ Dl \implies fst\ Dl \cdot \sigma \subseteq\#\ fst\ Cl \implies is\text{-ground}\text{-subst}\ \sigma' \implies fst\ Dl \cdot \sigma \cdot \sigma' < fst\ Cl \cdot \sigma' \rangle$
by (*metis False size-subst subset-imp-less-mset subset-mset.le-less subst-subset-mono*)
qed
ultimately **show** $\langle \exists DD \subseteq \{fst\ Dl \cdot \sigma \mid \sigma. is\text{-ground}\text{-subst}\ \sigma \}.$
 $(\forall I. I \models_s DD \longrightarrow I \models fst\ Cl \cdot \sigma') \wedge (\forall D \in DD. D < fst\ Cl \cdot \sigma') \rangle$
by *blast*
qed
qed
then **show** *?thesis*
by (*rule disjI1*)
qed
show *?thesis*
proof (rule *FL.step.process[of - N {Cl} - {}]*, *simp+*)
show $\langle Cl \in FL.Red\text{-F}\mathcal{G}\ N \rangle$
using *d-sub'-c unfolding FL-Red-F-eq*
proof –
have $\langle \bigwedge D. D \in \mathcal{G}\text{-F} (fst\ Cl) \implies \forall E \in N. E \sqsubset Cl \longrightarrow D \notin \mathcal{G}\text{-F} (fst\ E) \implies \forall D \in \mathcal{G}\text{-F} (fst\ Cl). D \in G.Red\text{-F} (\mathcal{G}\text{-F} (fst\ Dl)) \vee Dl \sqsubset Cl \wedge D \in \mathcal{G}\text{-F} (fst\ Dl) \implies D \in G.Red\text{-F} (\bigcup a \in N. \mathcal{G}\text{-F} (fst\ a)) \rangle$
by (*metis (no-types, lifting) G.Red-F-of-subset SUP-upper d-in subset-iff*)
moreover **have** $\langle \bigwedge D. D \in \mathcal{G}\text{-F} (fst\ Cl) \implies \forall E \in N. E \sqsubset Cl \longrightarrow D \notin \mathcal{G}\text{-F} (fst\ E) \implies Dl \sqsubset Cl \implies D \in G.Red\text{-F} (\bigcup a \in N. \mathcal{G}\text{-F} (fst\ a)) \rangle$
by (*smt FL.Prec-FL-def FL.equiv-F-grounding FL.prec-F-grounding UNIV-witness d-in in-mono*)
ultimately **show** $\langle Cl \in \{C. \forall D \in \mathcal{G}\text{-F} (fst\ C). D \in G.Red\text{-F} (\bigcup (\mathcal{G}\text{-F} \text{ 'fst' } \{Dl\})) \vee (\exists E \in \{Dl\}. E \sqsubset C \wedge D \in \mathcal{G}\text{-F} (fst\ E)) \} \vee Dl \sqsubset Cl \implies Cl \in \{C. \forall D \in \mathcal{G}\text{-F} (fst\ C). D \in G.Red\text{-F} (\bigcup (\mathcal{G}\text{-F} \text{ 'fst' } N)) \vee (\exists E \in N. E \sqsubset C \wedge D \in \mathcal{G}\text{-F} (fst\ E)) \} \rangle$
by *auto*
qed
qed (*simp add: FL.active-subset-def*)
qed

lemma *GC-reduction-step*:
assumes

young: $\text{snd } Dl \neq \text{Old}$ **and**
d-sub-c: $\text{fst } Dl \subset\# \text{fst } Cl$
shows $N \cup \{Cl\} \rightsquigarrow GC N \cup \{Dl\}$
proof (*rule FL.step.process*[of - $N \{Cl\} - \{Dl\}$])
have $Cl \in FL.Red-F \{Dl\}$
proof (*rule mem-FL-Red-F-because-G-Red-F*)
show $\langle \forall D \in \mathcal{G}\text{-}F (\text{fst } Cl). D \in G.Red-F (\bigcup (\mathcal{G}\text{-}F \text{ 'fst ' } \{Dl\})) \rangle$
using *d-sub-c unfolding G.Red-F-def strictly-subsumes-def subsumes-def G-F-def*
proof *clarsimp*
fix σ
assume $\langle \text{is-ground-subst } \sigma \rangle$
then have $\langle \text{fst } Dl \cdot \sigma \subseteq \{\text{fst } Dl \cdot \sigma \mid \sigma. \text{is-ground-subst } \sigma\} \rangle$
by *blast*
moreover have $\langle \text{fst } Dl \cdot \sigma < \text{fst } Cl \cdot \sigma \rangle$
using *subst-subset-mono[OF d-sub-c, of σ] by (simp add: subset-imp-less-mset)*
moreover have $\langle \forall I. I \models \text{fst } Dl \cdot \sigma \longrightarrow I \models \text{fst } Cl \cdot \sigma \rangle$
using *subst-subset-mono[OF d-sub-c] true-clss-subclause by fast*
ultimately show $\langle \exists DD \subseteq \{\text{fst } Dl \cdot \sigma \mid \sigma. \text{is-ground-subst } \sigma\}. (\forall I. I \models_s DD \longrightarrow I \models \text{fst } Cl \cdot \sigma) \wedge (\forall D \in DD. D < \text{fst } Cl \cdot \sigma) \rangle$
by *blast*
qed
qed
then show $\{Cl\} \subseteq FL.Red-F (N \cup \{Dl\})$
using *FL.Red-F-of-subset by blast*
qed (*auto simp: FL.active-subset-def young*)

lemma *GC-processing-step*: $N \cup \{(C, \text{New})\} \rightsquigarrow GC N \cup \{(C, \text{Processed})\}$
proof (*rule FL.step.process*[of - $N \{(C, \text{New})\} - \{(C, \text{Processed})\}$])
have $(C, \text{New}) \in FL.Red-F \{(C, \text{Processed})\}$
by (*rule mem-FL-Red-F-because-Prec-FL*) (*simp add: FL.Prec-FL-def generalizes-refl*)
then show $\{(C, \text{New})\} \subseteq FL.Red-F (N \cup \{(C, \text{Processed})\})$
using *FL.Red-F-of-subset by blast*
qed (*auto simp: FL.active-subset-def*)

lemma *old-inferences-between-eq-new-inferences-between*:
old-concl-of 'inference-system.inferences-between (ord-FO- Γ S) N C =
concl-of 'F.Inf-between N {C} (is ?rp = ?f)
proof (*intro set-eqI iffI*)
fix E
assume $e\text{-in}: E \in \text{old-concl-of 'inference-system.inferences-between (ord-FO-}\Gamma \text{ S) N C}$

obtain $CAs \ DA \ AAs \ As \ \sigma$ **where**
e-res: *ord-resolve-rename S CAs DA AAs As σ E and*
cd-sub: *set CAs \cup {DA} \subseteq N \cup {C} and*
c-in: $C \in \text{set } CAs \cup \{DA\}$
using *e-in unfolding inference-system.inferences-between-def infer-from-def ord-FO- Γ -def by auto*

show $E \in \text{concl-of 'F.Inf-between N } \{C\}$
unfolding *F.Inf-between-alt F.Inf-from-def*
proof –
have $\langle \text{Infer } (CAs \ @ \ [DA]) \ E \in F\text{-Inf} \wedge \text{set } (\text{prems-of } (\text{Infer } (CAs \ @ \ [DA]) \ E)) \subseteq \text{insert } C \ N \wedge$
 $C \in \text{set } (\text{prems-of } (\text{Infer } (CAs \ @ \ [DA]) \ E)) \wedge E = \text{concl-of } (\text{Infer } (CAs \ @ \ [DA]) \ E) \rangle$
using *e-res cd-sub c-in F-Inf-def by auto*
then show $\langle E \in \text{concl-of '}\{\iota \in F\text{-Inf}. \iota \in \{\iota \in F\text{-Inf}. \text{set } (\text{prems-of } \iota) \subseteq N \cup \{C\}\} \wedge \text{set } (\text{prems-of } \iota) \cap \{C\} \neq \{\}\} \rangle$

by (smt Un-insert-right boolean-algebra-cancel.sup0 disjoint-insert mem-Collect-eq image-def)
 qed
 next
 fix E
 assume e-in: $E \in \text{concl-of } 'F.\text{Inf-between } N \{C\}$

 obtain CAs DA AAs As σ where
 e-res: ord-resolve-rename S CAs DA AAs As σ E and
 cd-sub: set CAs $\cup \{DA\} \subseteq N \cup \{C\}$ and
 c-in: $C \in \text{set CAs} \cup \{DA\}$
 using e-in unfolding F.Inf-between-alt F.Inf-from-def F-Inf-def inference-system.Inf-between-alt
 inference-system.Inf-from-def
 by (auto simp: image-def Bex-def)

 show $E \in \text{old-concl-of } ' \text{inference-system.infernces-between } (\text{ord-FO-}\Gamma S) N C$
 unfolding inference-system.infernces-between-def infer-from-def ord-FO- Γ -def
 using e-res cd-sub c-in
 by (clarsimp simp: image-def Bex-def, rule-tac $x = \text{old-Infer } (\text{mset CAs}) DA E$ in exI, auto)
 qed

lemma GC-inference-step:

assumes
 young: $l \neq \text{Old}$ and
 no-active: $FL.\text{active-subset } M = \{\}$ and
 m-sup: $\text{fst } 'M \supseteq \text{old-concl-of } ' \text{inference-system.infernces-between } (\text{ord-FO-}\Gamma S)$
 ($\text{fst } 'FL.\text{active-subset } N) C$
 shows $N \cup \{(C, l)\} \rightsquigarrow_{GC} N \cup \{(C, \text{Old})\} \cup M$
 proof (rule FL.step.infer[of - N C l - M])
 have m-sup': $\text{fst } 'M \supseteq \text{concl-of } ' F.\text{Inf-between } (\text{fst } 'FL.\text{active-subset } N) \{C\}$
 using m-sup unfolding old-infernces-between-eq-new-infernces-between .

 show $F.\text{Inf-between } (\text{fst } 'FL.\text{active-subset } N) \{C\} \subseteq F.\text{Red-I } (\text{fst } '(N \cup \{(C, \text{Old})\} \cup M))$
 proof
 fix ι
 assume $\iota\text{-in-if2}$: $\iota \in F.\text{Inf-between } (\text{fst } 'FL.\text{active-subset } N) \{C\}$
 note $\iota\text{-in} = F.\text{Inf-if-Inf-between}[OF \iota\text{-in-if2}]$
 have $\text{concl-of } \iota \in \text{fst } 'M$
 using m-sup' $\iota\text{-in-if2}$ m-sup' by (auto simp: image-def Collect-mono-iff F-Inf-between-alt)
 then have $\text{concl-of } \iota \in \text{fst } '(N \cup \{(C, \text{Old})\} \cup M)$
 by auto
 then show $\iota \in F.\text{Red-I-}\mathcal{G} (\text{fst } '(N \cup \{(C, \text{Old})\} \cup M))$
 by (rule F.Red-I-of-Inf-to-N[OF $\iota\text{-in}$])
 qed
 qed (use young no-active in auto)

lemma RP-step-imp-GC-step: $St \rightsquigarrow_{RP} St' \implies \text{lclss-of-state } St \rightsquigarrow_{GC} \text{lclss-of-state } St'$

proof (induction rule: RP.induct)
 case (tautology-deletion A C N P Q)
 then show ?case
 by (rule GC-tautology-step)
 next
 case (forward-subsumption D P Q C N)
 note d-in = this(1) and d-sub-c = this(2)
 show ?case
 proof (cases $D \in P$)

```

    case True
  then show ?thesis
    using GC-subsumption-step[of (D, Processed) lclss-of-state (N, P, Q) (C, New)] d-sub-c
    by auto
next
  case False
  then have D ∈ Q
    using d-in by simp
  then show ?thesis
    using GC-subsumption-step[of (D, Old) lclss-of-state (N, P, Q) (C, New)] d-sub-c by auto
qed
next
  case (backward-subsumption-P D N C P Q)
  note d-in = this(1) and d-ssub-c = this(2)
  then show ?case
    using GC-subsumption-step[of (D, New) lclss-of-state (N, P, Q) (C, Processed)] d-ssub-c
    by auto
next
  case (backward-subsumption-Q D N C P Q)
  note d-in = this(1) and d-ssub-c = this(2)
  then show ?case
    using GC-subsumption-step[of (D, New) lclss-of-state (N, P, Q) (C, Old)] d-ssub-c by auto
next
  case (forward-reduction D L' P Q L σ C N)
  show ?case
    using GC-reduction-step[of (C, New) (C + {#L#}, New) lclss-of-state (N, P, Q)] by auto
next
  case (backward-reduction-P D L' N L σ C P Q)
  show ?case
    using GC-reduction-step[of (C, Processed) (C + {#L#}, Processed) lclss-of-state (N, P, Q)]
    by auto
next
  case (backward-reduction-Q D L' N L σ C P Q)
  show ?case
    using GC-reduction-step[of (C, Processed) (C + {#L#}, Old) lclss-of-state (N, P, Q)]
    by auto
next
  case (clause-processing N C P Q)
  show ?case
    using GC-processing-step[of lclss-of-state (N, P, Q) C] by auto
next
  case (inference-computation N Q C P)
  note n = this(1)
  show ?case
  proof –
    have ⟨FL.active-subset (lclss-of-state (N, {}, {})) = {}⟩
      unfolding n by (auto simp: FL.active-subset-def)
    moreover have ⟨old-concls-of (inference-system.infernces-between (ord-FO-Γ S)
      (fst ' FL.active-subset (lclss-of-state (N, P, Q))) C) ⊆ N⟩
      unfolding n inference-system.infernces-between-def image-def mem-Collect-eq
      lclss-of-state-def infer-from-def
      by (auto simp: FL.active-subset-def)
    ultimately have ⟨lclss-of-state (N, insert C P, Q)  $\rightsquigarrow$  GC lclss-of-state (N, P, insert C Q)⟩
      using GC-inference-step[of Processed lclss-of-state (N, {}, {})]
      lclss-of-state (N, P, Q) C, simplified] by blast
  end

```

then show *?case*
by (*auto simp: FL.active-subset-def*)
qed
qed

lemma *RP-derivation-imp-GC-derivation*: $\text{chain } (\rightsquigarrow RP) \text{ Sts} \implies \text{chain } (\rightsquigarrow GC) (\text{lmap } \text{lclss-of-state } \text{Sts})$
using *chain-lmap RP-step-imp-GC-step* **by** *blast*

lemma *RP-step-imp-derive-step*: $\text{St} \rightsquigarrow RP \text{ St}' \implies \text{lclss-of-state } \text{St} \triangleright L \text{ lclss-of-state } \text{St}'$
by (*rule FL.one-step-equiv*) (*rule RP-step-imp-GC-step*)

lemma *RP-derivation-imp-derive-derivation*:
 $\text{chain } (\rightsquigarrow RP) \text{ Sts} \implies \text{chain } (\triangleright L) (\text{lmap } \text{lclss-of-state } \text{Sts})$
using *chain-lmap RP-step-imp-derive-step* **by** *blast*

theorem *RP-sound-new-statement*:

assumes
deriv: $\text{chain } (\rightsquigarrow RP) \text{ Sts}$ **and**
bot-in: $\{\#\} \in \text{clss-of-state } (\text{Liminf-state } \text{Sts})$
shows $\text{clss-of-state } (\text{lhd } \text{Sts}) \models_{\mathcal{G}e} \{\#\}$

proof –

have $\text{clss-of-state } (\text{Liminf-state } \text{Sts}) \models_{\mathcal{G}e} \{\#\}$
using *F.subset-entailed bot-in* **by** *auto*
then have *fst* ‘ $\text{Liminf-llist } (\text{lmap } \text{lclss-of-state } \text{Sts}) \models_{\mathcal{G}e} \{\#\}$ ’
by (*metis image-hd-lclss-of-state lclss-Liminf-commute*)
then have $\text{Liminf-llist } (\text{lmap } \text{lclss-of-state } \text{Sts}) \models_{\mathcal{G}Le} \text{FL.Bot-FL}$
using *FL.labeled-entailment-lifting* **by** *simp*
then have $\text{lhd } (\text{lmap } \text{lclss-of-state } \text{Sts}) \models_{\mathcal{G}Le} \text{FL.Bot-FL}$

proof –

assume $\langle \text{FL.entails-}\mathcal{G} (\text{Liminf-llist } (\text{lmap } \text{lclss-of-state } \text{Sts})) (\{\#\} \times \text{UNIV}) \rangle$
moreover have $\langle \text{chain } (\triangleright L) (\text{lmap } \text{lclss-of-state } \text{Sts}) \rangle$
using *deriv RP-derivation-imp-derive-derivation* **by** *simp*
moreover have $\langle \text{chain } \text{FL.entails-}\mathcal{G} (\text{lmap } \text{lclss-of-state } \text{Sts}) \rangle$
by (*smt F-entails-}\mathcal{G}-iff FL.labeled-entailment-lifting RP-model chain-lmap deriv }mathcal{G}-Fset-def image-hd-lclss-of-state*)
ultimately show $\langle \text{FL.entails-}\mathcal{G} (\text{lhd } (\text{lmap } \text{lclss-of-state } \text{Sts})) (\{\#\} \times \text{UNIV}) \rangle$
using *FL.unsat-limit-iff* **by** *blast*

qed

then have $\text{lclss-of-state } (\text{lhd } \text{Sts}) \models_{\mathcal{G}Le} \text{FL.Bot-FL}$
using *chain-not-lnull deriv* **by** *fastforce*

then show *?thesis*

unfolding *FL.entails-}\mathcal{G}-L-def F.entails-}\mathcal{G}-def lclss-of-state-def* **by** *auto*

qed

theorem *RP-saturated-if-fair-new-statement*:

assumes
deriv: $\text{chain } (\rightsquigarrow RP) \text{ Sts}$ **and**
init: $\text{FL.active-subset } (\text{lclss-of-state } (\text{lhd } \text{Sts})) = \{\}$ **and**
final: $\text{FL.passive-subset } (\text{Liminf-llist } (\text{lmap } \text{lclss-of-state } \text{Sts})) = \{\}$
shows $\text{FL.saturated } (\text{Liminf-llist } (\text{lmap } \text{lclss-of-state } \text{Sts}))$

proof –

note *nnil* = *chain-not-lnull[OF deriv]*
have *gc-deriv*: $\text{chain } (\rightsquigarrow GC) (\text{lmap } \text{lclss-of-state } \text{Sts})$
by (*rule RP-derivation-imp-GC-derivation[OF deriv]*)
show *?thesis*

using *nnil* *init* *final*
FL.fair-implies-Liminf-saturated[*OF FL.gc-to-red*[*OF gc-deriv*] *FL.gc-fair*[*OF gc-deriv*]] **by** *simp*
qed

corollary *RP-complete-if-fair-new-statement*:

assumes
deriv: *chain* (\sim *RP*) *Sts* **and**
init: *FL.active-subset* (*lclss-of-state* (*lhd Sts*)) = {} **and**
final: *FL.passive-subset* (*Liminf-llist* (*lmap lclss-of-state Sts*)) = {} **and**
unsat: *grounding-of-state* (*lhd Sts*) \models_e {{#}}
shows {{#}} \in *Q-of-state* (*Liminf-state Sts*)
proof –
note *nnil* = *chain-not-lnull*[*OF deriv*]
note *len* = *chain-length-pos*[*OF deriv*]
have *gc-deriv*: *chain* (\sim *GC*) (*lmap lclss-of-state Sts*)
by (*rule RP-derivation-imp-GC-derivation*[*OF deriv*])

have *hd-lcls*: *fst* ‘ *lhd* (*lmap lclss-of-state Sts*) = *lhd* (*lmap clss-of-state Sts*)
using *len zero-enat-def* **by** *auto*
have *hd-unsat*: *fst* ‘ *lhd* (*lmap lclss-of-state Sts*) $\models_{\mathcal{G}e}$ {{#}}
unfolding *hd-lcls F-entails-G-iff* **unfolding** *true-clss-def* **using** *unsat* **unfolding** *G-Fset-def*
by (*metis* (*no-types*, *lifting*) *chain-length-pos gc-deriv gr.ex-min-counterex i0-less*
length-eq-0 llength-lmap llength-lmap llist.map-sel(1) true-clss-empty true-clss-singleton)
have $\exists BL \in \{\{\#\}\} \times UNIV$. $BL \in$ *Liminf-llist* (*lmap lclss-of-state Sts*)
by (*rule FL.gc-complete-Liminf*[*OF gc-deriv*, *of* {{#}}])
(use final hd-unsat in <auto simp: init nnil>)
then show *?thesis*
unfolding *Liminf-state-def lclss-Liminf-commute*
using *final*[*unfolded FL.passive-subset-def*] *Liminf-state-def lclss-Liminf-commute* **by** *fastforce*
qed

4.7 Alternative Derivation of Previous RP Results

lemma *old-fair-imp-new-fair*:

assumes
nnul: \neg *lnull Sts* **and**
fair: *fair-state-seq Sts* **and**
empty-Q0: *Q-of-state* (*lhd Sts*) = {}
shows
FL.active-subset (*lclss-of-state* (*lhd Sts*)) = {} **and**
FL.passive-subset (*Liminf-llist* (*lmap lclss-of-state Sts*)) = {}
proof –
show *FL.active-subset* (*lclss-of-state* (*lhd Sts*)) = {}
using *nnul empty-Q0* **unfolding** *FL.active-subset-def* **by** (*cases Sts*) *auto*
next
show *FL.passive-subset* (*Liminf-llist* (*lmap lclss-of-state Sts*)) = {}
using *fair*
unfolding *fair-state-seq-def FL.passive-subset-def lclss-Liminf-commute lclss-of-state-def*
by *auto*
qed

lemma *old-redundant-infer-iff*:

src.redundant-infer $N \gamma \iff$
 $(\exists DD. DD \subseteq N \wedge DD \cup \text{set-mset}(\text{old-side-prems-of } \gamma) \models_e \{\text{old-concl-of } \gamma\})$
 $\wedge (\forall D \in DD. D < \text{old-main-prem-of } \gamma)$
(is ?lhs \iff ?rhs)

proof

assume *?rhs*

then obtain *DD0* **where**

DD0 \subseteq *N* **and**

DD0 \cup *set-mset* (*old-side-prems-of* γ) \models_e {*old-concl-of* γ } **and**

$\forall D \in DD0. D <$ *old-main-prem-of* γ

by *blast*

then obtain *DD* **where**

fin-dd: *finite* *DD* **and**

dd-in: *DD* \subseteq *N* **and**

dd-un: *DD* \cup *set-mset* (*old-side-prems-of* γ) \models_e {*old-concl-of* γ } **and**

all-dd: $\forall D \in DD. D <$ *old-main-prem-of* γ

using *entails-concl-compact-union*[of {*old-concl-of* γ } *DD0 set-mset* (*old-side-prems-of* γ)]

by *fast*

show *?lhs*

unfolding *src.redundant-infer-def* **using** *fin-dd dd-in dd-un all-dd*

by *simp* (*metis finite-set-mset-mset-set true-cls-set-mset*)

qed (*auto simp: src.redundant-infer-def*)

definition *old-infer-of* :: '*a* clause inference \Rightarrow '*a* old-inference **where**

old-infer-of $\iota =$ *old-Infer* (*mset* (*side-prems-of* ι)) (*main-prem-of* ι) (*concl-of* ι)

lemma *new-redundant-infer-imp-old-redundant-infer*:

G.redundant-infer *N* $\iota \Longrightarrow$ *src.redundant-infer* *N* (*old-infer-of* ι)

unfolding *old-redundant-infer-iff G.redundant-infer-def old-infer-of-def* **by** *simp*

lemma *saturated-imp-saturated-RP*:

assumes

satur: *FL.saturated* (*Liminf-llist* (*lmap* *lclss-of-state* *Sts*)) **and**

no-passive: *FL.passive-subset* (*Liminf-llist* (*lmap* *lclss-of-state* *Sts*)) = {}

shows *src.saturated-upto* *Sts* (*grounding-of-state* (*Liminf-state* *Sts*))

proof –

define *Q* **where**

Q = *Liminf-llist* (*lmap* *Q-of-state* *Sts*)

define *Ql* **where**

Ql = ($\lambda C. (C, Old)$) ‘ *Q*

define *G* **where**

G = \bigcup (*G-F* ‘ *Q*)

have *n-empty*: *N-of-state* (*Liminf-state* *Sts*) = {} **and**

p-empty: *P-of-state* (*Liminf-state* *Sts*) = {}

using *no-passive*[*unfolded FL.passive-subset-def*] *Liminf-state-def lclss-Liminf-commute*

by *fastforce+*

then have *limuls-eq*: *Liminf-llist* (*lmap* *lclss-of-state* *Sts*) = *Ql*

unfolding *Ql-def Q-def* **using** *Liminf-state-def lclss-Liminf-commute lclss-of-state-def* **by** *auto*

have *clst-eq*: *clss-of-state* (*Liminf-state* *Sts*) = *Q*

unfolding *n-empty p-empty Q-def* **by** (*auto simp: Liminf-state-def*)

have *gflimuls-eq*: ($\bigcup Cl \in Ql. G-F$ (*fst* *Cl*)) = *G*

unfolding *Ql-def G-def* **by** *auto*

have *gd.inferences-from* *Sts* *G* \subseteq *src.Ri* *Sts* *G*

proof

fix γ

assume γ -*inf*: $\gamma \in$ *gd.inferences-from* *Sts* *G*

obtain ι **where**
 ι -inff: $\iota \in G.$ Inf-from Q G **and**
 γ : $\gamma = \text{old-infer-of } \iota$
using γ -inf
unfolding $gd.inferences\text{-from-def}$ $old\text{-infer-from-def}$ $G.$ Inf-from-def $old\text{-infer-of-def}$
proof (*atomize-elim, clarify*)
assume
 g -is: $\langle \gamma \in gd.\text{ord-}\Gamma$ $Sts \rangle$ **and**
 $prems$ -in: $\langle \text{set-mset} (old\text{-side-prems-of } \gamma + \{\#\text{old-main-prem-of } \gamma\#\}) \subseteq G \rangle$
obtain CAs DA AAs As E **where** $main$ -in: $\langle DA \in G \rangle$ **and** $side$ -in: $\langle \text{set } CAs \subseteq G \rangle$ **and**
 g -is2: $\langle \gamma = \text{old-Infer} (mset\ CAs)\ DA\ E \rangle$ **and**
 ord -res: $\langle gd.\text{ord-resolve } Sts\ CAs\ DA\ AAs\ As\ E \rangle$
using g -is $prems$ -in **unfolding** $gd.\text{ord-}\Gamma$ -def **by** *auto*
define ι - γ **where** ι - $\gamma = \text{Infer} (CAs\ @\ [DA])\ E$
then have $\langle \iota$ - $\gamma \in G$ -Inf $Q \rangle$ **using** Q -of-state.simps g -is g -is2 ord -res $Liminf$ -state-def S - Q -def
unfolding $gd.\text{ord-}\Gamma$ -def G -Inf-def Q -def **by** *auto*
moreover have $\langle \text{set} (prems\text{-of } \iota$ - $\gamma) \subseteq G \rangle$
using g -is2 $prems$ -in **unfolding** ι - γ -def **by** *simp*
moreover have $\langle \gamma = \text{old-Infer} (mset (side\text{-prems-of } \iota$ - $\gamma)) (main\text{-prem-of } \iota$ - $\gamma) (concl\text{-of } \iota$ - $\gamma) \rangle$
using g -is2 **unfolding** ι - γ -def **by** *simp*
ultimately show
 $\langle \exists \iota. \iota \in \{ \iota \in G$ -Inf $Q. \text{set} (prems\text{-of } \iota) \subseteq G \} \wedge \gamma = \text{old-Infer} (mset (side\text{-prems-of } \iota) (main\text{-prem-of } \iota) (concl\text{-of } \iota)) \rangle$
by *blast*
qed
obtain ι' **where**
 ι' -inff: $\iota' \in F.$ Inf-from Q **and**
 ι -in- ι' : $\iota \in G$ -I Q ι'
using G -Inf-overapprox- F -Inf ι -inff **unfolding** G -def **by** *blast*

note ι' -inf = $F.$ Inf-if-Inf-from[OF ι' -inff]

let $?olds = \text{replicate} (\text{length} (prems\text{-of } \iota'))\ Old$

obtain ι'' **and** $l0$ **where**
 ι'' : $\iota'' = \text{Infer} (zip (prems\text{-of } \iota')\ ?olds) (concl\text{-of } \iota',\ l0)$ **and**
 ι'' -inf: $\iota'' \in FL.$ Inf-FL
using $FL.$ Inf-F-to-Inf-FL[OF ι' -inf, of $?olds$, *simplified*] **by** *simp*

have $\text{set} (prems\text{-of } \iota'') \subseteq Ql$
using ι' -inff[*unfolded* $F.$ Inf-from-def, *simplified*] **unfolding** ι'' Ql -def **by** *auto*
then have $\iota'' \in FL.$ Inf-from Ql
unfolding $FL.$ Inf-from-def **using** ι'' -inf **by** *simp*
moreover have $\iota' = FL.$ to- F ι''
unfolding ι'' **unfolding** $FL.$ to- F -def **by** *simp*
ultimately have $\iota \in G.$ Red-I Q G
using ι -in- ι'
 $FL.$ sat-inf-imp-ground-red-fam-inter[OF *satur, unfolded limuls-eq gflimuls-eq, simplified*]
by *blast*
then have $G.$ redundant-infer G ι
unfolding $G.$ Red-I-def **by** *auto*
then have γ -red: $src.$ redundant-infer G γ
unfolding γ **by** (*rule new-redundant-infer-imp-old-redundant-infer*)
moreover have $\gamma \in gd.\text{ord-}\Gamma$ Sts
using γ -inf $gd.inferences\text{-from-def}$ **by** *blast*

```

ultimately show  $\gamma \in \text{src.Ri Sts } G$ 
  unfolding src.Ri-def by auto
qed
then show ?thesis
  unfolding G-def clst-eq src.saturated-upto-def
  by clarsimp (smt Diff-subset gd.inferences-from-mono subset-eq G-Fset-def)
qed

```

theorem *RP-sound-old-statement:*

```

assumes
  deriv: chain ( $\rightsquigarrow$ RP) Sts and
  bot-in:  $\{\#\} \in \text{class-of-state (Liminf-state Sts)}$ 
shows  $\neg$  satisfiable (grounding-of-state (lhd Sts))
using RP-sound-new-statement[OF deriv bot-in] unfolding F-entails-G-iff G-Fset-def by simp

```

The theorem below is stated differently than the original theorem in RP: The grounding of the limit might be a strict subset of the limit of the groundings. Because saturation is neither monotone nor antimonotone, the two results are incomparable. See also *grounding-of-state-Liminf-state-subseteq*.

theorem *RP-saturated-if-fair-old-statement-altered:*

```

assumes
  deriv: chain ( $\rightsquigarrow$ RP) Sts and
  fair: fair-state-seq Sts and
  empty-Q0:  $Q\text{-of-state (lhd Sts)} = \{\}$ 
shows src.saturated-upto Sts (grounding-of-state (Liminf-state Sts))
proof -
  note fair' = old-fair-imp-new-fair[OF chain-not-lnull[OF deriv] fair empty-Q0]
  show ?thesis
    by (rule saturated-imp-saturated-RP[OF - fair'(2)], rule RP-saturated-if-fair-new-statement)
      (rule deriv fair')+
qed

```

corollary *RP-complete-if-fair-old-statement:*

```

assumes
  deriv: chain ( $\rightsquigarrow$ RP) Sts and
  fair: fair-state-seq Sts and
  empty-Q0:  $Q\text{-of-state (lhd Sts)} = \{\}$  and
  unsat:  $\neg$  satisfiable (grounding-of-state (lhd Sts))
shows  $\{\#\} \in Q\text{-of-state (Liminf-state Sts)}$ 
proof (rule RP-complete-if-fair-new-statement)
  show  $\langle \mathcal{G}\text{-Fset (N-of-state (lhd Sts)} \cup P\text{-of-state (lhd Sts)} \cup Q\text{-of-state (lhd Sts))} \parallel_e \{\#\} \rangle$ 
    using unsat unfolding F-entails-G-iff by auto
  qed (rule deriv old-fair-imp-new-fair[OF chain-not-lnull[OF deriv] fair empty-Q0])+
end
end

```

5 New Fairness Proofs for the Given Clause Prover Architectures

```

theory Given-Clause-Architectures-Revisited
  imports Saturation-Framework.Given-Clause-Architectures
begin

```

The given clause and lazy given clause procedures satisfy key invariants. This provides an

alternative way to prove fairness and hence saturation of the limit.

5.1 Given Clause Procedure

context *given-clause*
begin

definition *gc-invar* :: (*f* × *l*) set llist ⇒ enat ⇒ bool **where**
gc-invar *Ns* *i* ←→
Inf-from (active-subset (Liminf-upto-llist *Ns* *i*)) ⊆ *Sup-upto-llist* (lmap *Red-I-G* *Ns*) *i*

lemma *gc-invar-infinity*:

assumes

nnil: ¬ *lnull* *Ns* **and**

invar: ∀ *i*. enat *i* < llength *Ns* → *gc-invar* *Ns* (enat *i*)

shows *gc-invar* *Ns* ∞

unfolding *gc-invar-def*

proof (intro subsetI, unfold Liminf-upto-llist-infinity Sup-upto-llist-infinity)

fix *ι*

assume *ι-inff*: *ι* ∈ *Inf-from* (active-subset (Liminf-llist *Ns*))

define *As* **where**

As = lmap active-subset *Ns*

have *act-ns*: active-subset (Liminf-llist *Ns*) = Liminf-llist *As*

unfolding *As-def* active-subset-def Liminf-set-filter-commute[symmetric] ..

note *ι-inf* = *Inf-if-Inf-from*[OF *ι-inff*]

note *ι-inff'* = *ι-inff*[unfolded *act-ns*]

have ¬ *lnull* *As*

unfolding *As-def* **using** *nnil* **by** auto

moreover **have** set (prems-of *ι*) ⊆ Liminf-llist *As*

using *ι-inff'*[unfolded *Inf-from-def*] **by** simp

ultimately obtain *i* **where**

i-lt-as: enat *i* < llength *As* **and**

prems-sub-ge-i: set (prems-of *ι*) ⊆ (∩ *j* ∈ {*j*. *i* ≤ *j* ∧ enat *j* < llength *As*}. lnth *As* *j*)

using finite-subset-Liminf-llist-imp-exists-index **by** blast

note *i-lt-ns* = *i-lt-as*[unfolded *As-def*, simplified]

have set (prems-of *ι*) ⊆ lnth *As* *i*

using *prems-sub-ge-i* *i-lt-as* **by** auto

then have *ι* ∈ *Inf-from* (active-subset (lnth *Ns* *i*))

using *i-lt-as* *ι-inf* **unfolding** *Inf-from-def* *As-def* **by** auto

then have *ι* ∈ *Sup-upto-llist* (lmap *Red-I-G* *Ns*) (enat *i*)

using *nnil* *i-lt-ns* *invar*[unfolded *gc-invar-def*] **by** auto

then show *ι* ∈ *Sup-llist* (lmap *Red-I-G* *Ns*)

using *Sup-upto-llist-subset-Sup-llist* **by** fastforce

qed

lemma *gc-invar-gc-init*:

assumes

¬ *lnull* *Ns* **and**

active-subset (lhd *Ns*) = {}

shows $gc\text{-invar } Ns\ 0$
using $assms\ labeled\text{-inf}\text{-have}\text{-prems}\ \mathbf{unfolding}\ gc\text{-invar}\text{-def}\ Inf\text{-from}\text{-def}\ \mathbf{by}\ auto$

lemma $gc\text{-invar}\text{-gc}\text{-step}$:

assumes

$Si\text{-lt}$: $enat\ (Suc\ i) < llength\ Ns$ **and**

$invar$: $gc\text{-invar } Ns\ i$ **and**

$step$: $lnth\ Ns\ i \rightsquigarrow_{GC}\ lnth\ Ns\ (Suc\ i)$

shows $gc\text{-invar } Ns\ (Suc\ i)$

using $step\ Si\text{-lt}\ invar$

proof $cases$

have $i\text{-lt}$: $enat\ i < llength\ Ns$

using $Si\text{-lt}\ Suc\text{-ile}\text{-eq}\ order.\text{strict}\text{-implies}\text{-order}\ \mathbf{by}\ blast$

have $lim\text{-i}$: $Liminf\text{-upto}\text{-llist } Ns\ (enat\ i) = lnth\ Ns\ i$

using $i\text{-lt}\ \mathbf{by}\ auto$

have $lim\text{-Si}$: $Liminf\text{-upto}\text{-llist } Ns\ (enat\ (Suc\ i)) = lnth\ Ns\ (Suc\ i)$

using $Si\text{-lt}\ \mathbf{by}\ auto$

{

case $(process\ N\ M\ M')$

note $ni = this(1)$ **and** $nSi = this(2)$ **and** $m'\text{-pas} = this(4)$

have $Inf\text{-from}\ (active\text{-subset}\ (N\ \cup\ M')) \subseteq Inf\text{-from}\ (active\text{-subset}\ (N\ \cup\ M))$

using $m'\text{-pas}\ \mathbf{by}\ (simp\ add:\ Inf\text{-from}\text{-mono})$

also have $\dots \subseteq Sup\text{-upto}\text{-llist}\ (lmap\ Red\text{-I}\text{-G}\ Ns)\ (enat\ i)$

using $invar\ \mathbf{unfolding}\ gc\text{-invar}\text{-def}\ lim\text{-i}\ ni\ \mathbf{by}\ auto$

also have $\dots \subseteq Sup\text{-upto}\text{-llist}\ (lmap\ Red\text{-I}\text{-G}\ Ns)\ (enat\ (Suc\ i))$

by $simp$

finally show $?thesis$

unfolding $gc\text{-invar}\text{-def}\ lim\text{-Si}\ nSi\ .$

next

case $(infer\ N\ C\ L\ M)$

note $ni = this(1)$ **and** $nSi = this(2)$ **and** $l\text{-pas} = this(3)$ **and** $m\text{-pas} = this(4)$ **and**

$inff\text{-red} = this(5)$

{

fix ι

assume $\iota\text{-inff}$: $\iota \in Inf\text{-from}\ (active\text{-subset}\ (N\ \cup\ \{(C,\ active)\}\ \cup\ M))$

have $\iota\text{-inf}$: $\iota \in Inf\text{-FL}$

using $\iota\text{-inff}\ \mathbf{unfolding}\ Inf\text{-from}\text{-def}\ \mathbf{by}\ auto$

then have $F\iota\text{-inf}$: $to\text{-F}\ \iota \in Inf\text{-F}$

using $in\text{-Inf}\text{-FL}\text{-imp}\text{-to}\text{-F}\text{-in}\text{-Inf}\text{-F}\ \mathbf{by}\ blast$

have $\iota \in Inf\text{-from}\ (active\text{-subset}\ N\ \cup\ \{(C,\ active)\})$

using $\iota\text{-inff}\ m\text{-pas}\ \mathbf{by}\ simp$

then have $F\iota\text{-inff}$:

$to\text{-F}\ \iota \in no\text{-labels.}\ Inf\text{-from}\ (fst\ ' (active\text{-subset}\ N\ \cup\ \{(C,\ active)\}))$

using $F\iota\text{-inf}\ \mathbf{unfolding}\ to\text{-F}\text{-def}\ Inf\text{-from}\text{-def}\ no\text{-labels.}\ Inf\text{-from}\text{-def}\ \mathbf{by}\ auto$

have $\iota \in Sup\text{-upto}\text{-llist}\ (lmap\ Red\text{-I}\text{-G}\ Ns)\ (enat\ (Suc\ i))$

proof $(cases\ to\text{-F}\ \iota \in no\text{-labels.}\ Inf\text{-between}\ (fst\ ' active\text{-subset}\ N)\ \{C\})$

case $True$

then have $to\text{-F}\ \iota \in no\text{-labels.}\ Red\text{-I}\text{-G}\ (fst\ ' (N\ \cup\ \{(C,\ active)\}\ \cup\ M))$

using $inff\text{-red}\ \mathbf{by}\ auto$

```

then have  $\iota \in \text{Red-I-}\mathcal{G} (N \cup \{(C, \text{active})\} \cup M)$ 
  by (subst labeled-red-inf-eq-red-inf[OF  $\iota$ -inf])
then show ?thesis
  using Si-lt using nSi by auto
next
case False
then have  $\text{to-}F \ \iota \in \text{no-labels.Inf-from (fst `active-subset } N)$ 
  using F $\iota$ -inff unfolding no-labels.Inf-from-def no-labels.Inf-between-def by auto
then have  $\iota \in \text{Inf-from (active-subset } N)$ 
  using  $\iota$ -inf l-pas unfolding to-F-def Inf-from-def no-labels.Inf-from-def
  by clarsimp (smt  $\iota$ -inff[unfolded Inf-from-def] active-subset-def imageE image-subset-iff
    in-mono mem-Collect-eq prod.collapse)
then show ?thesis
  using invar l-pas unfolding gc-invar-def lim-i ni by auto
qed
}
then show ?thesis
  unfolding gc-invar-def lim-Si nSi by blast
}
qed

```

lemma *gc-invar-gc*:

```

assumes
  gc: chain ( $\rightsquigarrow GC$ ) Ns and
  init: active-subset (lhd Ns) = {} and
  i-lt:  $i < \text{llength } Ns$ 
shows gc-invar Ns i
using i-lt
proof (induct i)
case (enat i)
then show ?case
proof (induct i)
  case 0
  then show ?case
    using gc-invar-gc-init[OF chain-not-lnull[OF gc] init] by (simp add: enat-0)
next
case (Suc i)
note  $ih = \text{this}(1)$  and  $Si-lt = \text{this}(2)$ 
have i-lt:  $\text{enat } i < \text{llength } Ns$ 
  using Si-lt Suc-ile-eq less-le by blast
show ?case
  by (rule gc-invar-gc-step[OF Si-lt ih[OF i-lt] chain-lnth-rel[OF gc Si-lt]])
qed
qed simp

```

lemma *gc-fair-new-proof*:

```

assumes
  gc: chain ( $\rightsquigarrow GC$ ) Ns and
  init: active-subset (lhd Ns) = {} and
  lim: passive-subset (Liminf-list Ns) = {}
shows fair Ns
unfolding fair-def
proof –
have  $\text{Inf-from (Liminf-list } Ns) \subseteq \text{Inf-from (active-subset (Liminf-list } Ns))$  (is ?lhs  $\subseteq$  -)
  using lim unfolding active-subset-def passive-subset-def

```

by (*metis* (*no-types*, *lifting*) *Inf-from-mono empty-Collect-eq mem-Collect-eq subsetI*)
 also have ... \subseteq *Sup-llist* (*lmap Red-I-G Ns*) (**is** - \subseteq ?*rhs*)
 using *gc-invar-infinity*[*OF chain-not-lnull*[*OF gc*]] *gc-invar-gc*[*OF gc init*]
 unfolding *gc-invar-def* by *fastforce*
 finally show ?*lhs* \subseteq ?*rhs*
 .
 qed
 end

5.2 Lazy Given Clause

context *lazy-given-clause*
begin

definition *from-F* :: '*f* inference \Rightarrow ('*f* \times '*l*) inference set **where**
from-F $\iota = \{\iota' \in \text{Inf-FL}. \text{to-F } \iota' = \iota\}$

definition *lgc-invar* :: ('*f* inference set \times ('*f* \times '*l*) set) *llist* \Rightarrow *enat* \Rightarrow *bool* **where**
lgc-invar *TNs* *i* \longleftrightarrow
Inf-from (*active-subset* (*Liminf-upto-llist* (*lmap snd TNs*) *i*))
 $\subseteq \bigcup$ (*from-F* ' *Liminf-upto-llist* (*lmap fst TNs*) *i*) \cup *Sup-upto-llist* (*lmap* (*Red-I-G* \circ *snd*) *TNs*) *i*

lemma *lgc-invar-infinity*:

assumes
nnil: $\neg \text{lnull } \text{TNs}$ **and**
invar: $\forall i. \text{enat } i < \text{llength } \text{TNs} \longrightarrow \text{lgc-invar } \text{TNs } (\text{enat } i)$
shows *lgc-invar* *TNs* ∞
unfolding *lgc-invar-def*
proof (*intro subsetI*, *unfold Liminf-upto-llist-infinity Sup-upto-llist-infinity*)
fix ι
assume $\iota\text{-inff}$: $\iota \in \text{Inf-from } (\text{active-subset } (\text{Liminf-llist } (\text{lmap snd } \text{TNs})))$

define *As* **where**

As = *lmap* (*active-subset* \circ *snd*) *TNs*

have *act-ns*: *active-subset* (*Liminf-llist* (*lmap snd TNs*)) = *Liminf-llist* *As*
unfolding *As-def active-subset-def Liminf-set-filter-commute*[*symmetric*] *llist.map-comp* ..

note $\iota\text{-inf} = \text{Inf-if-Inf-from}$ [*OF* $\iota\text{-inff}$]

note $\iota\text{-inff}' = \iota\text{-inff}$ [*unfolded act-ns*]

show $\iota \in \bigcup$ (*from-F* ' *Liminf-llist* (*lmap fst TNs*)) \cup *Sup-llist* (*lmap* (*Red-I-G* \circ *snd*) *TNs*)

proof –

{
assume $\iota\text{-ni-lim}$: $\iota \notin \bigcup$ (*from-F* ' *Liminf-llist* (*lmap fst TNs*))

have $\neg \text{lnull } \text{As}$

unfolding *As-def* **using** *nnil* **by** *auto*

moreover have *set* (*prems-of* ι) \subseteq *Liminf-llist* *As*

using $\iota\text{-inff}'$ [*unfolded Inf-from-def*] **by** *simp*

ultimately obtain *i* **where**

i-lt-as: *enat* *i* < *llength* *As* **and**

prems-sub-ge-i: *set* (*prems-of* ι) \subseteq ($\bigcap j \in \{j. i \leq j \wedge \text{enat } j < \text{llength } \text{As}\}. \text{lnth } \text{As } j$)

using *finite-subset-Liminf-llist-imp-exists-index* **by** *blast*

have $ts\text{-}nnil$: $\neg lnull$ ($lmap$ fst TNs)
using $As\text{-}def$ $nnil$ **by** $simp$

have $F\iota\text{-}ni\text{-}lim$: $to\text{-}F \iota \notin Liminf\text{-}llist$ ($lmap$ fst TNs)
using $\iota\text{-}inf$ $\iota\text{-}ni\text{-}lim$ **unfolding** $from\text{-}F\text{-}def$ **by** $auto$

obtain i' **where**
 $i\text{-}le\text{-}i'$: $i \leq i'$ **and**
 $i'\text{-}lt\text{-}as$: $enat\ i' < llength\ As$ **and**
 $F\iota\text{-}ni\text{-}i'$: $to\text{-}F \iota \notin lnth$ ($lmap$ fst TNs) i'
using $i\text{-}lt\text{-}as$ $not\text{-}Liminf\text{-}llist\text{-}imp\text{-}exists\text{-}index$ [$OF\ ts\text{-}nnil\ F\iota\text{-}ni\text{-}lim$, of i] **unfolding** $As\text{-}def$
by $auto$

have $\iota\text{-}ni\text{-}i'$: $\iota \notin \bigcup$ ($from\text{-}F$ ' fst ($lnth\ TNs\ i'$))
using $F\iota\text{-}ni\text{-}i'$ $i'\text{-}lt\text{-}as$ [$unfolded\ As\text{-}def$] **unfolding** $from\text{-}F\text{-}def$ **by** $auto$

have set ($prems\text{-}of\ \iota$) \subseteq ($\bigcap j \in \{j. i' \leq j \wedge enat\ j < llength\ As\}$. $lnth\ As\ j$)
using $prems\text{-}sub\text{-}ge\text{-}i\ i\text{-}le\text{-}i'$ **by** $auto$
then **have** set ($prems\text{-}of\ \iota$) $\subseteq lnth\ As\ i'$
using $i'\text{-}lt\text{-}as$ **by** $auto$
then **have** $\iota \in Inf\text{-}from$ ($active\text{-}subset$ (snd ($lnth\ TNs\ i'$)))
using $i'\text{-}lt\text{-}as$ $\iota\text{-}inf$ **unfolding** $Inf\text{-}from\text{-}def$ $As\text{-}def$ **by** $auto$
then **have** $\iota\text{-}in\text{-}i'$: $\iota \in Sup\text{-}upto\text{-}llist$ ($lmap$ ($Red\text{-}I\text{-}\mathcal{G} \circ snd$) TNs) ($enat\ i'$)
using $\iota\text{-}ni\text{-}i'$ $i'\text{-}lt\text{-}as$ [$unfolded\ As\text{-}def$] $invar$ [$unfolded\ lgc\text{-}invar\text{-}def$] **by** $auto$
then **have** $\iota \in Sup\text{-}llist$ ($lmap$ ($Red\text{-}I\text{-}\mathcal{G} \circ snd$) TNs)
using $Sup\text{-}upto\text{-}llist\text{-}subset\text{-}Sup\text{-}llist$ **by** $fastforce$

}
then **show** $?thesis$
by $blast$

qed
qed

lemma $lgc\text{-}invar\text{-}lgc\text{-}init$:

assumes

$nnil$: $\neg lnull\ TNs$ **and**

$n\text{-}init$: $active\text{-}subset$ (snd ($lhd\ TNs$)) = $\{\}$ **and**

$t\text{-}init$: $\forall \iota \in Inf\text{-}F. prems\text{-}of\ \iota = [] \longrightarrow \iota \in fst$ ($lhd\ TNs$)

shows $lgc\text{-}invar\ TNs\ 0$

unfolding $lgc\text{-}invar\text{-}def$

proof –

have $Inf\text{-}from$ ($active\text{-}subset$ ($Liminf\text{-}upto\text{-}llist$ ($lmap\ snd\ TNs$) 0)) =

$Inf\text{-}from\ \{\}$ (**is** $?lhs = -$)

using $nnil\ n\text{-}init$ **by** $auto$

also **have** $\dots \subseteq \bigcup$ ($from\text{-}F$ ' fst ($lhd\ TNs$))

using $t\text{-}init$ $Inf\text{-}FL\text{-}to\text{-}Inf\text{-}F$ **unfolding** $Inf\text{-}from\text{-}def$ $from\text{-}F\text{-}def$ $to\text{-}F\text{-}def$ **by** $force$

also **have** $\dots \subseteq \bigcup$ ($from\text{-}F$ ' fst ($lhd\ TNs$)) $\cup Red\text{-}I\text{-}\mathcal{G}$ (snd ($lhd\ TNs$))

by $fast$

also **have** $\dots = \bigcup$ ($from\text{-}F$ ' $Liminf\text{-}upto\text{-}llist$ ($lmap\ fst\ TNs$) 0)

$\cup Sup\text{-}upto\text{-}llist$ ($lmap$ ($Red\text{-}I\text{-}\mathcal{G} \circ snd$) TNs) 0 (**is** $- = ?rhs$)

using $nnil$ **by** $auto$

finally **show** $?lhs \subseteq ?rhs$

qed

lemma $lgc\text{-}invar\text{-}lgc\text{-}step$:

assumes

Si-lt: $\text{enat } (\text{Suc } i) < \text{llength } \text{TNs}$ **and**
invar: $\text{lgc-invar } \text{TNs } i$ **and**
step: $\text{lnth } \text{TNs } i \rightsquigarrow \text{LGC } \text{lnth } \text{TNs } (\text{Suc } i)$
shows $\text{lgc-invar } \text{TNs } (\text{Suc } i)$
using *step Si-lt invar*

proof cases

let $?Sup\text{-Red-}i = \text{Sup-upto-llist } (\text{lmap } (\text{Red-I-}\mathcal{G} \circ \text{snd}) \text{TNs}) (\text{enat } i)$
let $?Sup\text{-Red-}Si = \text{Sup-upto-llist } (\text{lmap } (\text{Red-I-}\mathcal{G} \circ \text{snd}) \text{TNs}) (\text{enat } (\text{Suc } i))$

have *i-lt*: $\text{enat } i < \text{llength } \text{TNs}$
using *Si-lt Suc-ile-eq order.strict-implies-order* **by** *blast*

have *lim-i*:
 $\text{Liminf-upto-llist } (\text{lmap } \text{fst } \text{TNs}) (\text{enat } i) = \text{lnth } (\text{lmap } \text{fst } \text{TNs}) i$
 $\text{Liminf-upto-llist } (\text{lmap } \text{snd } \text{TNs}) (\text{enat } i) = \text{lnth } (\text{lmap } \text{snd } \text{TNs}) i$
using *i-lt by auto*

have *lim-Si*:
 $\text{Liminf-upto-llist } (\text{lmap } \text{fst } \text{TNs}) (\text{enat } (\text{Suc } i)) = \text{lnth } (\text{lmap } \text{fst } \text{TNs}) (\text{Suc } i)$
 $\text{Liminf-upto-llist } (\text{lmap } \text{snd } \text{TNs}) (\text{enat } (\text{Suc } i)) = \text{lnth } (\text{lmap } \text{snd } \text{TNs}) (\text{Suc } i)$
using *Si-lt by auto*

{
case (*process N1 N M N2 M' T*)
note $tni = \text{this}(1)$ **and** $tnSi = \text{this}(2)$ **and** $n1 = \text{this}(3)$ **and** $n2 = \text{this}(4)$ **and** $m\text{-red} = \text{this}(5)$
and
 $m'\text{-pas} = \text{this}(6)$

have *ni*: $\text{lnth } (\text{lmap } \text{snd } \text{TNs}) i = N \cup M$
by (*simp add: i-lt n1 tni*)
have *nSi*: $\text{lnth } (\text{lmap } \text{snd } \text{TNs}) (\text{Suc } i) = N \cup M'$
by (*simp add: Si-lt n2 tnSi*)
have *ti*: $\text{lnth } (\text{lmap } \text{fst } \text{TNs}) i = T$
by (*simp add: i-lt tni*)
have *tSi*: $\text{lnth } (\text{lmap } \text{fst } \text{TNs}) (\text{Suc } i) = T$
by (*simp add: Si-lt tnSi*)

have $\text{Inf-from } (\text{active-subset } (N \cup M')) \subseteq \text{Inf-from } (\text{active-subset } (N \cup M))$
using *m'-pas* **by** (*simp add: Inf-from-mono*)
also have $\dots \subseteq \bigcup (\text{from-}F \text{ ' } T) \cup ?Sup\text{-Red-}i$
using *invar unfolding lgc-invar-def lim-i ni ti* .
also have $\dots \subseteq \bigcup (\text{from-}F \text{ ' } T) \cup ?Sup\text{-Red-}Si$
using *Sup-upto-llist-mono* **by** *auto*
finally show *?thesis*
unfolding *lgc-invar-def lim-Si nSi tSi* .

next

case (*schedule-infer T2 T1 T' N1 N C L N2*)
note $tni = \text{this}(1)$ **and** $tnSi = \text{this}(2)$ **and** $t2 = \text{this}(3)$ **and** $n1 = \text{this}(4)$ **and** $n2 = \text{this}(5)$ **and**
 $l\text{-pas} = \text{this}(6)$ **and** $t' = \text{this}(7)$

have *ni*: $\text{lnth } (\text{lmap } \text{snd } \text{TNs}) i = N \cup \{(C, L)\}$
by (*simp add: i-lt n1 tni*)
have *nSi*: $\text{lnth } (\text{lmap } \text{snd } \text{TNs}) (\text{Suc } i) = N \cup \{(C, \text{active})\}$
by (*simp add: Si-lt n2 tnSi*)
have *ti*: $\text{lnth } (\text{lmap } \text{fst } \text{TNs}) i = T1$
by (*simp add: i-lt tni*)

```

have tSi: lnth (lmap fst TNs) (Suc i) = T1 ∪ T'
  by (simp add: Si-lt t2 tnSi)

{
  fix ι
  assume ι-inff: ι ∈ Inf-from (active-subset (N ∪ {(C, active)}))

  have ι-inf: ι ∈ Inf-FL
    using ι-inff unfolding Inf-from-def by auto
  then have Fι-inf: to-F ι ∈ Inf-F
    using in-Inf-FL-imp-to-F-in-Inf-F by blast

  have ι ∈ ∪ (from-F ' (T1 ∪ T')) ∪ ?Sup-Red-Si
  proof (cases to-F ι ∈ no-labels.Inf-between (fst ' active-subset N) {C})
    case True
      then have ι ∈ ∪ (from-F ' (T1 ∪ T'))
        unfolding t' from-F-def using ι-inf by auto
      then show ?thesis
        by blast
    next
      case False
        moreover have to-F ι ∈ no-labels.Inf-from (fst ' (active-subset N ∪ {(C, active)}))
          using ι-inff Fι-inf unfolding to-F-def Inf-from-def no-labels.Inf-from-def by auto
        ultimately have to-F ι ∈ no-labels.Inf-from (fst ' active-subset N)
          unfolding no-labels.Inf-from-def no-labels.Inf-between-def by auto
        then have ι ∈ Inf-from (active-subset N)
          using ι-inf unfolding to-F-def no-labels.Inf-from-def
          by clarsimp (smt Inf-from-def Un-insert-right ι-inff active-subset-def
            boolean-algebra-cancel.sup0 imageE image-subset-iff insert-iff mem-Collect-eq
            prod.collapse snd-conv subset-iff)
        then have ι ∈ ∪ (from-F ' (T1 ∪ T')) ∪ ?Sup-Red-i
          using invar[unfolded lgc-invar-def] l-pas unfolding lim-i ni ti by auto
        then show ?thesis
          using Sup-upto-llist-mono by force
      qed
  }
  then show ?thesis
    unfolding lgc-invar-def lim-i lim-Si nSi tSi by fast
  next
    case (compute-infer T1 T2 ι' N2 N1 M)
    note tni = this(1) and tnSi = this(2) and t1 = this(3) and n2 = this(4) and m-pas = this(5)
  and
    ι'-red = this(6)

  have ni: lnth (lmap snd TNs) i = N1
    by (simp add: i-lt tni)
  have nSi: lnth (lmap snd TNs) (Suc i) = N1 ∪ M
    by (simp add: Si-lt n2 tnSi)
  have ti: lnth (lmap fst TNs) i = T2 ∪ {ι'}
    by (simp add: i-lt t1 tni)
  have tSi: lnth (lmap fst TNs) (Suc i) = T2
    by (simp add: Si-lt tnSi)

  {
    fix ι
  }

```

```

assume  $\iota$ -inff:  $\iota \in \text{Inf-from } (\text{active-subset } (N1 \cup M))$ 

have  $\iota$ -bef:  $\iota \in \bigcup (\text{from-F } \iota (T2 \cup \{\iota'\})) \cup \text{?Sup-Red-i}$ 
  using  $\iota$ -inff invar[unfolded lgc-invar-def lim-i ti ni] m-pas by auto
have  $\iota \in \bigcup (\text{from-F } \iota T2) \cup \text{?Sup-Red-Si}$ 
proof (cases  $\iota \in \text{from-F } \iota'$ )
  case  $\iota$ -in- $\iota'$ : True
  then have  $\iota \in \text{Red-I-}\mathcal{G} (N1 \cup M)$ 
  using  $\iota'$ -red from-F-def labeled-red-inf-eq-red-inf by auto
  then have  $\iota \in \text{?Sup-Red-Si}$ 
  using Si-lt by (simp add: n2 tnSi)
  then show ?thesis
  by auto
next
  case False
  then show ?thesis
  using  $\iota$ -bef by auto
qed
}
then show ?thesis
  unfolding lgc-invar-def lim-Si tSi nSi by blast
next
case (delete-orphan-infers T1 T2 T' N)
note tni = this(1) and tnSi = this(2) and t1 = this(3) and t'-orph = this(4)

have ni: lnth (lmap snd TNs) i = N
  by (simp add: i-lt tni)
have nSi: lnth (lmap snd TNs) (Suc i) = N
  by (simp add: Si-lt tnSi)
have ti: lnth (lmap fst TNs) i = T2  $\cup$  T'
  by (simp add: i-lt t1 tni)
have tSi: lnth (lmap fst TNs) (Suc i) = T2
  by (simp add: Si-lt tnSi)

{
  fix  $\iota$ 
  assume  $\iota$ -inff:  $\iota \in \text{Inf-from } (\text{active-subset } N)$ 

  have to-F  $\iota \notin T'$ 
    using t'-orph  $\iota$ -inff in-Inf-from-imp-to-F-in-Inf-from by blast
  hence  $\iota \notin \bigcup (\text{from-F } \iota T')$ 
    unfolding from-F-def by auto
  then have  $\iota \in \bigcup (\text{from-F } \iota T2) \cup \text{?Sup-Red-Si}$ 
    using  $\iota$ -inff invar unfolding lgc-invar-def lim-i ni ti by auto
}
then show ?thesis
  unfolding lgc-invar-def lim-Si tSi nSi by blast
}
qed

```

lemma lgc-invar-lgc:

assumes

lgc: chain (\rightsquigarrow LGC) TNs **and**

n-init: active-subset (snd (lhd TNs)) = {} **and**

t-init: $\forall \iota \in \text{Inf-F. } \text{prems-of } \iota = [] \longrightarrow \iota \in \text{fst } (\text{lhd TNs})$ **and**

```

    i-lt:  $i < \text{llength } TNs$ 
  shows lgc-invar  $TNs$  i
  using i-lt
proof (induct i)
  case (enat i)
  then show ?case
  proof (induct i)
    case 0
    then show ?case
      using lgc-invar-lgc-init[OF chain-not-lnull[OF lgc] n-init t-init] by (simp add: enat-0)
  next
  case (Suc i)
  note ih = this(1) and Si-lt = this(2)
  have i-lt:  $\text{enat } i < \text{llength } TNs$ 
    using Si-lt Suc-ile-eq less-le by blast
  show ?case
    by (rule lgc-invar-lgc-step[OF Si-lt ih[OF i-lt] chain-lnth-rel[OF lgc Si-lt]])
  qed
qed simp

```

lemma *lgc-fair-new-proof*:

```

  assumes
    lgc:  $\text{chain } (\rightsquigarrow LGC) TNs$  and
    n-init:  $\text{active-subset } (\text{snd } (\text{lhd } TNs)) = \{\}$  and
    n-lim:  $\text{passive-subset } (\text{Liminf-llist } (\text{lmap } \text{snd } TNs)) = \{\}$  and
    t-init:  $\forall \iota \in \text{Inf-}F. \text{prems-of } \iota = [] \longrightarrow \iota \in \text{fst } (\text{lhd } TNs)$  and
    t-lim:  $\text{Liminf-llist } (\text{lmap } \text{fst } TNs) = \{\}$ 
  shows fair ( $\text{lmap } \text{snd } TNs$ )
  unfolding fair-def llist.map-comp
proof -
  have Inf-from ( $\text{Liminf-llist } (\text{lmap } \text{snd } TNs)$ )
     $\subseteq \text{Inf-from } (\text{active-subset } (\text{Liminf-llist } (\text{lmap } \text{snd } TNs)))$  (is ?lhs  $\subseteq$  -)
    by (rule Inf-from-mono) (use n-lim passive-subset-def active-subset-def in blast)
  also have ...  $\subseteq \bigcup (\text{from-}F \text{ ' } \text{Liminf-upto-llist } (\text{lmap } \text{fst } TNs) \infty)$ 
     $\cup \text{Sup-llist } (\text{lmap } (\text{Red-IG} \circ \text{snd}) TNs)$ 
    using lgc-invar-infinity[OF chain-not-lnull[OF lgc]] lgc-invar-lgc[OF lgc n-init t-init]
    unfolding lgc-invar-def by fastforce
  also have ...  $\subseteq \text{Sup-llist } (\text{lmap } (\text{Red-IG} \circ \text{snd}) TNs)$  (is -  $\subseteq$  ?rhs)
    using t-lim by auto
  finally show ?lhs  $\subseteq$  ?rhs

```

qed

end

end