Extensions to the Comprehensive Framework for Saturation Theorem Proving

Jasmin Blanchette Sophie Tourret

March 19, 2025

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

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: ‹$\iota \in Inf \Longrightarrow set\ (prems\text{-}of\ \iota) \models \{concl\text{-}of\ \iota\}$›
begin

lemma *Inf-consist-preserving*:
assumes *n-cons*: $\neg\ N \models Bot$
shows $\neg\ N \cup concl\text{-}of\ `\ Inf\text{-}from\ N \models Bot$
⟨*proof*⟩

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*:
$chain\ (\rhd)\ Ns \Longrightarrow Red\text{-}F\ (Sup\text{-}llist\ Ns) \subseteq Red\text{-}F\ (Liminf\text{-}llist\ Ns)$
⟨*proof*⟩

lemma *Red-I-Sup-subset-Red-I-Liminf*:
$chain\ (\rhd)\ Ns \Longrightarrow Red\text{-}I\ (Sup\text{-}llist\ Ns) \subseteq Red\text{-}I\ (Liminf\text{-}llist\ Ns)$
⟨*proof*⟩

Proof idea due to Uwe Waldmann:

lemma *unsat-limit-iff*:
assumes
chain-red: $chain\ (\rhd)\ Ns$ **and**
chain-ent: $chain\ (\models)\ Ns$
shows $Liminf\text{-}llist\ Ns \models Bot \longleftrightarrow lhd\ Ns \models Bot$
⟨*proof*⟩

Some easy consequences:

lemma *Red-F-limit-Sup*: $chain\ (\rhd)\ Ns \Longrightarrow Red\text{-}F\ (Liminf\text{-}llist\ Ns) = Red\text{-}F\ (Sup\text{-}llist\ Ns)$
⟨*proof*⟩

lemma *Red-I-limit-Sup*: *chain* (▷) *Ns* ⟹ *Red-I* (*Liminf-llist Ns*) = *Red-I* (*Sup-llist Ns*)
⟨*proof*⟩

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* ⇒ *′f* **where**
main-prem-of ι ≡ *last* (*prems-of* ι)

abbreviation *side-prems-of* :: *′f inference* ⇒ *′f list* **where**
side-prems-of ι ≡ *butlast* (*prems-of* ι)

lemma *set-prems-of*:
set (*prems-of* ι) = (*if prems-of* ι = [] *then* {} *else* {*main-prem-of* ι} ∪ *set* (*side-prems-of* ι))
⟨*proof*⟩

locale *counterex-reducing-inference-system* = *inference-system Inf* + *consequence-relation*
for *Inf* :: *′f inference set* +
fixes
I-of :: *′f set* ⇒ *′f set* **and**
less :: *′f* ⇒ *′f* ⇒ *bool* (**infix** ‹≺› *50*)
assumes
wfp-less: *wfp* (≺) **and**
Inf-counterex-reducing:
$N \cap Bot = \{\} \Longrightarrow D \in N \Longrightarrow \neg\ I\text{-}of\ N \models \{D\} \Longrightarrow$
$(\bigwedge C.\ C \in N \Longrightarrow \neg\ I\text{-}of\ N \models \{C\} \Longrightarrow D \prec C \lor D = C) \Longrightarrow$
$\exists \iota \in Inf.\ prems\text{-}of\ \iota \neq [] \land main\text{-}prem\text{-}of\ \iota = D \land set\ (side\text{-}prems\text{-}of\ \iota) \subseteq N \land$
$I\text{-}of\ N \models set\ (side\text{-}prems\text{-}of\ \iota) \land \neg\ I\text{-}of\ N \models \{concl\text{-}of\ \iota\} \land concl\text{-}of\ \iota \prec D$

begin

lemma *ex-min-counterex*:
fixes *N* :: *′f set*
assumes ¬ *I* ⊨ *N*

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

end

Theorem 4.4 (generalizes Theorems 3.9 and 3.16):

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

theorem *saturated-model*:
assumes
satur: *saturated N* **and**
bot-ni-n: $N \cap Bot = \{\}$
shows *I-of N* $\models N$
⟨*proof*⟩

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: $N \models Bot$
shows $N \cap Bot \neq \{\}$
⟨*proof*⟩

end

2.2 Compactness

locale *concl-compact-consequence-relation* = *consequence-relation* +
assumes
entails-concl-compact: *finite EE* $\Longrightarrow CC \models EE \Longrightarrow \exists\, CC' \subseteq CC.$ *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 $\exists\, CC' \subseteq CC.$ *finite* $CC' \wedge CC' \cup DD \models EE$
⟨*proof*⟩

end

2.3 The Finitary Standard Redundancy Criterion

locale *finitary-standard-formula-redundancy* =
consequence-relation Bot (⊨)
for
Bot :: *'f set* **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ‹⊨› *50*) +
fixes
less :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺› *50*)

assumes
transp-less: *transp* $(\prec)$ **and**
wfp-less: *wfp* $(\prec)$
begin

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

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*: $N \subseteq N' \Longrightarrow$ *Red-F* $N \subseteq$ *Red-F* N'
⟨*proof*⟩

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* $N.\ \mathit{finite}\ DD \land DD \cup CC \models \{E\} \land (\forall\, D' \in DD.\ D' \prec D)$
⟨*proof*⟩

lemma *Red-F-imp-ex-non-Red-F*:
assumes *c-in*: $C \in$ *Red-F* N
shows $\exists\, CC \subseteq N -$ *Red-F* $N.\ \mathit{finite}\ CC \land CC \models \{C\} \land (\forall\, C' \in CC.\ C' \prec C)$
⟨*proof*⟩

lemma *Red-F-subs-Red-F-diff-Red-F*: *Red-F* $N \subseteq$ *Red-F* $(N -$ *Red-F* $N)$
⟨*proof*⟩

lemma *Red-F-eq-Red-F-diff-Red-F*: *Red-F* $N =$ *Red-F* $(N -$ *Red-F* $N)$
⟨*proof*⟩

The following results correspond to Lemma 4.6.

lemma *Red-F-of-Red-F-subset*: $N' \subseteq$ *Red-F* $N \Longrightarrow$ *Red-F* $N \subseteq$ *Red-F* $(N - N')$
⟨*proof*⟩

lemma *Red-F-model*: $M \models N -$ *Red-F* $N \Longrightarrow M \models N$
⟨*proof*⟩

lemma *Red-F-Bot*: $B \in Bot \Longrightarrow N \models \{B\} \Longrightarrow N -$ *Red-F* $N \models \{B\}$
⟨*proof*⟩

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* $\Rightarrow$ $'f$ *set* $\Rightarrow$ *bool* (**infix** ‹⊨› *50*) **and**
less :: $'f \Rightarrow 'f \Rightarrow$ *bool* (**infix** ‹≺› *50*) +
assumes
Inf-has-prem: $\iota \in Inf \Longrightarrow$ *prems-of* $\iota \neq []$ **and**
Inf-reductive: $\iota \in Inf \Longrightarrow$ *concl-of* $\iota \prec$ *main-prem-of* ι

begin

definition *redundant-infer* :: *'f set* ⇒ *'f inference* ⇒ *bool* **where**
redundant-infer N ι ⟷
(∃ *DD* ⊆ *N*. *finite DD* ∧ *DD* ∪ *set* (*side-prems-of* ι) ⊨ {*concl-of* ι} ∧ (∀ *D* ∈ *DD*. *D* ≺ *main-prem-of* ι))

definition *Red-I* :: *'f set* ⇒ *'f inference set* **where**
Red-I N = {ι ∈ *Inf*. *redundant-infer N* ι}

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

lemma *Red-I-of-subset*: *N* ⊆ *N'* ⟹ *Red-I N* ⊆ *Red-I N'*
⟨*proof*⟩

lemma *Red-I-subs-Red-I-diff-Red-F*: *Red-I N* ⊆ *Red-I* (*N* − *Red-F N*)
⟨*proof*⟩

lemma *Red-I-eq-Red-I-diff-Red-F*: *Red-I N* = *Red-I* (*N* − *Red-F N*)
⟨*proof*⟩

lemma *Red-I-to-Inf*: *Red-I N* ⊆ *Inf*
⟨*proof*⟩

lemma *Red-I-of-Red-F-subset*: *N'* ⊆ *Red-F N* ⟹ *Red-I N* ⊆ *Red-I* (*N* − *N'*)
⟨*proof*⟩

lemma *Red-I-of-Inf-to-N*:
assumes
in-ι: ι ∈ *Inf* **and**
concl-in: *concl-of* ι ∈ *N*
shows ι ∈ *Red-I N*
⟨*proof*⟩

The following corresponds to Theorems 4.7 and 4.8:

sublocale *calculus Bot Inf* (⊨) *Red-I Red-F*
⟨*proof*⟩

end

2.4 The Standard Redundancy Criterion

locale *standard-formula-redundancy* =
concl-compact-consequence-relation Bot (⊨)
for
Bot :: *'f set* **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ‹⊨› *50*) +
fixes
less :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺› *50*)
assumes
transp-less: *transp* (≺) **and**
wfp-less: *wfp* (≺)
begin

definition *Red-F* :: *'f set* ⇒ *'f set* **where**
Red-F N = {*C*. ∃ *DD* ⊆ *N*. *DD* ⊨ {*C*} ∧ (∀ *D* ∈ *DD*. *D* ≺ *C*)}

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

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

lemma *Red-F-conv*: *Red-F* = *fin-std-red-F.Red-F*
⟨*proof*⟩

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

The following results correspond to Lemma 4.5.

lemma *Red-F-of-subset*: $N \subseteq N' \Longrightarrow$ *Red-F* $N \subseteq$ *Red-F* N'
⟨*proof*⟩

lemma *Red-F-imp-ex-non-Red-F*: $C \in$ *Red-F* $N \Longrightarrow \exists\, CC \subseteq N -$ *Red-F* $N.\ CC \models \{C\} \wedge (\forall\, C' \in CC.\ C' \prec C)$
⟨*proof*⟩

lemma *Red-F-subs-Red-F-diff-Red-F*: *Red-F* $N \subseteq$ *Red-F* ($N -$ *Red-F* N)
⟨*proof*⟩

lemma *Red-F-eq-Red-F-diff-Red-F*: *Red-F* $N =$ *Red-F* ($N -$ *Red-F* N)
⟨*proof*⟩

The following results correspond to Lemma 4.6.

lemma *Red-F-of-Red-F-subset*: $N' \subseteq$ *Red-F* $N \Longrightarrow$ *Red-F* $N \subseteq$ *Red-F* ($N - N'$)
⟨*proof*⟩

lemma *Red-F-model*: $M \models N -$ *Red-F* $N \Longrightarrow M \models N$
⟨*proof*⟩

lemma *Red-F-Bot*: $B \in$ *Bot* $\Longrightarrow N \models \{B\} \Longrightarrow N -$ *Red-F* $N \models \{B\}$
⟨*proof*⟩

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* ⇒ *'f set* ⇒ *bool* (**infix** ‹$\models$› *50*) **and**
less :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹$\prec$› *50*) +
assumes
Inf-has-prem: $\iota \in$ *Inf* $\Longrightarrow$ *prems-of* $\iota \neq []$ **and**
Inf-reductive: $\iota \in$ *Inf* $\Longrightarrow$ *concl-of* $\iota \prec$ *main-prem-of* ι
begin

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

definition *Red-I* :: *'f set* ⇒ *'f inference set* **where**
Red-I $N = \{\iota \in$ *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* (⊨)
⟨*proof*⟩

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

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

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*: $N \subseteq N' \Longrightarrow$ *Red-I* $N \subseteq$ *Red-I* N'
⟨*proof*⟩

lemma *Red-I-subs-Red-I-diff-Red-F*: *Red-I* $N \subseteq$ *Red-I* (N − *Red-F* N)
⟨*proof*⟩

lemma *Red-I-eq-Red-I-diff-Red-F*: *Red-I* N = *Red-I* (N − *Red-F* N)
⟨*proof*⟩

lemma *Red-I-to-Inf*: *Red-I* $N \subseteq$ *Inf*
⟨*proof*⟩

lemma *Red-I-of-Red-F-subset*: $N' \subseteq$ *Red-F* $N \Longrightarrow$ *Red-I* $N \subseteq$ *Red-I* ($N - N'$)
⟨*proof*⟩

lemma *Red-I-of-Inf-to-N*:
$\iota \in$ *Inf* $\Longrightarrow$ *concl-of* $\iota \in N \Longrightarrow \iota \in$ *Red-I* N
⟨*proof*⟩

The following corresponds to Theorems 4.7 and 4.8:

sublocale *calculus Bot Inf* (⊨) *Red-I Red-F*
⟨*proof*⟩

end

2.5 Refutational Completeness

locale *calculus-with-standard-inference-redundancy* = *calculus Bot Inf* (⊨) *Red-I Red-F*
for *Bot* :: *'f set* **and** *Inf* **and** *entails* (**infix** ‹⊨› *50*) **and** *Red-I* **and** *Red-F* +
fixes
less :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺› *50*)
assumes
Inf-has-prem: $\iota \in$ *Inf* $\Longrightarrow$ *prems-of* $\iota \neq$ [] **and**
Red-I-imp-redundant-infer: $\iota \in$ *Red-I* $N \Longrightarrow$
(∃ *DD*⊆N. *DD* ∪ *set* (*side-prems-of* ι) ⊨ {*concl-of* ι} ∧ (∀ *C*∈*DD*. *C* ≺ *main-prem-of* ι))

sublocale *calculus-with-finitary-standard-redundancy* ⊆
calculus-with-standard-inference-redundancy Bot Inf (⊨) *Red-I Red-F*
⟨*proof*⟩

sublocale *calculus-with-standard-redundancy* ⊆
calculus-with-standard-inference-redundancy Bot Inf (⊨) *Red-I Red-F*
⟨*proof*⟩

locale *counterex-reducing-calculus-with-standard-inferance-redundancy* =
calculus-with-standard-inference-redundancy Bot Inf (⊨) *Red-I Red-F* (≺) +
counterex-reducing-inference-system Bot (⊨) *Inf I-of* (≺)
for
Bot :: ′*f set* **and**
Inf :: ′*f inference set* **and**
entails :: ′*f set* ⇒ ′*f set* ⇒ *bool* (**infix** ‹⊨› *50*) **and**
Red-I :: ′*f set* ⇒ ′*f inference set* **and**
Red-F :: ′*f set* ⇒ ′*f set* **and**
I-of :: ′*f set* ⇒ ′*f set* **and**
less :: ′*f* ⇒ ′*f* ⇒ *bool* (**infix** ‹≺› *50*) +
assumes *less-total*: $\bigwedge C\ D.\ C \neq D \Longrightarrow C \prec D \vee D \prec C$
begin

The following result loosely corresponds to Theorem 4.9.

lemma *saturated-model*:
assumes
satur: *saturated N* **and**
bot-ni-n: $N \cap Bot = \{\}$
shows *I-of N* ⊨ *N*
⟨*proof*⟩

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* ⊨ *Bot*
shows $N \cap Bot \neq \{\}$
⟨*proof*⟩

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 ⊫ (with or without subscripts) for the "models" and entailment relations on clauses and ⊨ for the abstract concept of consequence.

abbreviation *true-lit-thick* :: ′*a interp* ⇒ ′*a literal* ⇒ *bool* (**infix** ‹⊫*l*› *50*) **where**

$I \Vdash l\ L \equiv I \models l\ L$

abbreviation *true-cls-thick* :: *'a interp* ⇒ *'a clause* ⇒ *bool* (**infix** ‹⊫› *50*) **where**
$I \Vdash C \equiv I \models C$

abbreviation *true-clss-thick* :: *'a interp* ⇒ *'a clause set* ⇒ *bool* (**infix** ‹⊫*s*› *50*) **where**
$I \Vdash s\ \mathcal{C} \equiv I \models s\ \mathcal{C}$

abbreviation *true-cls-mset-thick* :: *'a interp* ⇒ *'a clause multiset* ⇒ *bool* (**infix** ‹⊫*m*› *50*) **where**
$I \Vdash m\ \mathcal{C} \equiv I \models m\ \mathcal{C}$

no-notation *true-lit* (**infix** ‹⊨*l*› *50*)
no-notation *true-cls* (**infix** ‹⊨› *50*)
no-notation *true-clss* (**infix** ‹⊨*s*› *50*)
no-notation *true-cls-mset* (**infix** ‹⊨*m*› *50*)

3.2 Consequence Relation

abbreviation *entails-clss* :: *'a clause set* ⇒ *'a clause set* ⇒ *bool* (**infix** ‹⊫*e*› *50*) **where**
$N1 \Vdash e\ N2 \equiv \forall I.\ I \Vdash s\ N1 \longrightarrow I \Vdash s\ N2$

lemma *entails-iff-unsatisfiable-single*:
$CC \Vdash e\ \{E\} \longleftrightarrow \neg$ *satisfiable* $(CC \cup \{\{\#- L\#\}\ |L.\ L \in\#\ E\})$ (**is** - ⟷ - (- ∪ *?NegD*))
⟨*proof*⟩

lemma *entails-iff-unsatisfiable*:
$CC \Vdash e\ EE \longleftrightarrow (\forall E \in EE.\ \neg$ *satisfiable* $(CC \cup \{\{\#- L\#\}\ |L.\ L \in\#\ E\}))$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

interpretation *consequence-relation* {{#}} (⊫*e*)
⟨*proof*⟩

interpretation *concl-compact-consequence-relation* {{#}} :: (*'a* :: *wellorder*) *clause set* (⊫*e*)
⟨*proof*⟩

3.3 Counterexample-Reducing Inference Systems

definition *clss-of-interp* :: *'a set* ⇒ *'a literal multiset set* **where**
clss-of-interp $I = \{\{\#($*if* $A \in I$ *then Pos else Neg*$)\ A\#\}\ |A.$ *True*$\}$

lemma *true-clss-of-interp-iff-equal*[*simp*]: $J \Vdash s$ *clss-of-interp* $I \longleftrightarrow J = I$
⟨*proof*⟩

lemma *entails-iff-models*[*simp*]: *clss-of-interp* $I \Vdash e\ CC \longleftrightarrow I \Vdash s\ CC$
⟨*proof*⟩

locale *clausal-counterex-reducing-inference-system* = *inference-system Inf*
for *Inf* :: (*'a* :: *wellorder*) *clause inference set* +
fixes *J-of* :: *'a clause set* ⇒ *'a interp*
assumes *clausal-Inf-counterex-reducing*:
$\{\#\} \notin N \Longrightarrow D \in N \Longrightarrow \neg$ *J-of* $N \Vdash D \Longrightarrow (\bigwedge C.\ C \in N \Longrightarrow \neg$ *J-of* $N \Vdash C \Longrightarrow D \le C) \Longrightarrow$
$\exists \iota \in$ *Inf*. *prems-of* $\iota \neq [] \wedge$ *main-prem-of* $\iota = D \wedge$ *set* (*side-prems-of* $\iota) \subseteq N \wedge$
J-of $N \Vdash s$ *set* (*side-prems-of* $\iota) \wedge \neg$ *J-of* $N \Vdash$ *concl-of* $\iota \wedge$ *concl-of* $\iota < D$
begin

abbreviation *I-of* :: *'a clause set* ⇒ *'a clause set* **where**

I-of N ≡ *clss-of-interp* (*J-of N*)

lemma *Inf-counterex-reducing*:
assumes
bot-ni-n: $N \cap \{\{\#\}\} = \{\}$ **and**
d-in-n: $D \in N$ **and**
n-ent-d: ¬ *I-of N* ⊫*e* {*D*} **and**
d-min: $\bigwedge C.\ C \in N \Longrightarrow$ ¬ *I-of N* ⊫*e* {*C*} $\Longrightarrow D \leq C$
shows ∃ ι ∈ *Inf*. *prems-of* ι ≠ [] ∧ *main-prem-of* ι = *D* ∧ *set* (*side-prems-of* ι) ⊆ *N*
∧ *I-of N* ⊫*e set* (*side-prems-of* ι) ∧ ¬ *I-of N* ⊫*e* {*concl-of* ι} ∧ *concl-of* ι < *D*
⟨*proof*⟩

sublocale *counterex-reducing-inference-system* {{#}} (⊫*e*) *Inf I-of*
(<) :: ′*a clause* ⇒ ′*a clause* ⇒ *bool*
⟨*proof*⟩

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 {{#}} (⊫*e*) (<) :: ′*a clause* ⇒ ′*a clause* ⇒ *bool* +
clausal-counterex-reducing-inference-system Inf J-of
for
Inf :: (′*a* :: *wellorder*) *clause inference set* **and**
J-of :: ′*a clause set* ⇒ ′*a set*
begin

sublocale *counterex-reducing-calculus-with-standard-inferance-redundancy* {{#}} *Inf* (⊫*e*) *Red-I*
Red-F I-of (<) :: ′*a clause* ⇒ ′*a clause* ⇒ *bool*
⟨*proof*⟩

lemma *clausal-saturated-model*: *saturated N* ⟹ {#} ∉ *N* ⟹ *J-of N* ⊫*s N*
⟨*proof*⟩

corollary *clausal-saturated-complete*: *saturated N* ⟹ (∀ *I*. ¬ *I* ⊫*s N*) ⟹ {#} ∈ *N*
⟨*proof*⟩

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** ‹⊨l› *50*)
no-notation *true-cls* (**infix** ‹⊨› *50*)
no-notation *true-clss* (**infix** ‹⊨s› *50*)
no-notation *true-cls-mset* (**infix** ‹⊨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 ≡ Inference-System.Infer*
abbreviation *old-side-prems-of ≡ Inference-System.side-prems-of*
abbreviation *old-main-prem-of ≡ Inference-System.main-prem-of*
abbreviation *old-concl-of ≡ Inference-System.concl-of*
abbreviation *old-prems-of ≡ Inference-System.prems-of*
abbreviation *old-concls-of ≡ Inference-System.concls-of*
abbreviation *old-infer-from ≡ 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)) = (λx. (x, y)) ‘ set xs
  ⟨*proof*⟩

4.3 Ground Layer

context *FO-resolution-prover*
begin

no-notation *RP* (**infix** ‹⇝› *50*)
notation *RP* (**infix** ‹⇝RP› *50*)

interpretation *gr*: *ground-resolution-with-selection S-M S M*
  ⟨*proof*⟩

definition *G-Inf* :: *'a clause set ⇒ '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*: *ι ∈ G-Inf M ⟹ prems-of ι ≠ []*
  ⟨*proof*⟩

lemma *G-Inf-reductive*: *ι ∈ G-Inf M ⟹ concl-of ι < main-prem-of ι*
  ⟨*proof*⟩

interpretation *G*: *sound-inference-system G-Inf M {{#}} (⊫e)*

⟨*proof*⟩

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

interpretation *G*: *clausal-counterex-reducing-calculus-with-standard-redundancy G-Inf M gr.INTERP M*
⟨*proof*⟩

interpretation *G*: *statically-complete-calculus* $\{\{\#\}\}$ *G-Inf M* $(\models e)$ *G.Red-I M G.Red-F*
⟨*proof*⟩

4.4 First-Order Layer

abbreviation $\mathcal{G}$*-F* :: ‹*'a clause* ⇒ *'a clause set*› **where**
‹$\mathcal{G}$*-F* ≡ *grounding-of-cls*›

abbreviation $\mathcal{G}$*-Fset* :: ‹*'a clause set* ⇒ *'a clause set*› **where**
‹$\mathcal{G}$*-Fset* ≡ *grounding-of-clss*›

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

definition $\mathcal{G}$*-I* :: ‹*'a clause set* ⇒ *'a clause inference* ⇒ *'a clause inference set*› **where**
‹$\mathcal{G}$*-I M* ι = {*Infer* (*prems-of* ι ·*·cl* ϱs) (*concl-of* $\iota \cdot \varrho$) | ϱ ϱs.
is-ground-subst-list ϱs ∧ *is-ground-subst* ϱ
∧ *Infer* (*prems-of* ι ·*·cl* ϱs) (*concl-of* $\iota \cdot \varrho$) ∈ *G-Inf M*}›

abbreviation
$\mathcal{G}$*-I-opt* :: ‹*'a clause set* ⇒ *'a clause inference* ⇒ *'a clause inference set option*›
where
‹$\mathcal{G}$*-I-opt M* ι ≡ *Some* ($\mathcal{G}$*-I M* ι)›

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

lemma *F-Inf-have-prems*: $\iota \in$ *F-Inf* $\Longrightarrow$ *prems-of* $\iota \neq []$
⟨*proof*⟩

interpretation *F*: *lifting-intersection F-Inf* $\{\{\#\}\}$ *UNIV G-Inf* $\lambda N.$ $(\models e)$ *G.Red-I* $\lambda N.$ *G.Red-F* $\{\{\#\}\}$ $\lambda N.$ $\mathcal{G}$*-F* $\mathcal{G}$*-I-opt* λD C C'. *False*
⟨*proof*⟩

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

lemma *F-entails-*$\mathcal{G}$*-iff*: $N1 \models\mathcal{G}e\ N2 \longleftrightarrow \bigcup\ (\mathcal{G}\text{-}F\ `\ N1) \models e \bigcup\ (\mathcal{G}\text{-}F\ `\ N2)$
⟨*proof*⟩

lemma *true-Union-grounding-of-cls-iff*:
$I \models s\ (\bigcup C \in N.\ \{C \cdot \sigma\ |\sigma.\ \textit{is-ground-subst}\ \sigma\}) \longleftrightarrow (\forall \sigma.\ \textit{is-ground-subst}\ \sigma \longrightarrow I \models s\ N \cdot cs\ \sigma)$
⟨*proof*⟩

interpretation *F*: *sound-inference-system F-Inf* $\{\{\#\}\}$ $(\models\mathcal{G}e)$
⟨*proof*⟩

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

M ι
⟨*proof*⟩

interpretation *F*: *statically-complete-calculus* {{#}} *F-Inf* (⊫𝒢*e*) *F.Red-I-𝒢* *F.Red-F-𝒢-empty*
⟨*proof*⟩

4.5 Labeled First-Order or Given Clause Layer

datatype *label* = *New* | *Processed* | *Old*

abbreviation *F-Equiv* :: *'a clause* ⇒ *'a clause* ⇒ *bool* (**infix** ‹≐› *50*) **where**
$C \doteq D \equiv$ *generalizes C D* ∧ *generalizes D C*

abbreviation *F-Prec* :: *'a clause* ⇒ *'a clause* ⇒ *bool* (**infix** ‹≺·› *50*) **where**
C ≺· *D* ≡ *strictly-generalizes C D*

fun *L-Prec* :: *label* ⇒ *label* ⇒ *bool* (**infix** ‹⊏*l*› *50*) **where**
Old ⊏*l* *l* ⟷ *l* ≠ *Old*
| *Processed* ⊏*l* *l* ⟷ *l* = *New*
| *New* ⊏*l* *l* ⟷ *False*

lemma *irrefl-L-Prec*: ¬ *l* ⊏*l* *l*
⟨*proof*⟩

lemma *trans-L-Prec*: *l1* ⊏*l* *l2* ⟹ *l2* ⊏*l* *l3* ⟹ *l1* ⊏*l* *l3*
⟨*proof*⟩

lemma *wf-L-Prec*: *wfP* (⊏*l*)
⟨*proof*⟩

interpretation *FL*: *given-clause* {{#}} *F-Inf* {{#}} *UNIV* λ*N*. (⊫*e*) *G-Inf G.Red-I*
λ*N*. *G.Red-F* λ*N*. *𝒢-F* *𝒢-I-opt* (≐) (≺·) (⊏*l*) *Old*
⟨*proof*⟩

notation *FL.Prec-FL* (**infix** ‹⊏› *50*)
notation *FL.entails-𝒢-L* (**infix** ‹⊫𝒢*Le*› *50*)
notation *FL.derive* (**infix** ‹▷*L*› *50*)
notation *FL.step* (**infix** ‹⇝*GC*› *50*)

lemma *FL-Red-F-eq*:
FL.Red-F N =
{*C*. ∀ *D* ∈ *𝒢-F* (*fst C*). *D* ∈ *G.Red-F* (⋃ (*𝒢-F* ‘ *fst* ‘ *N*)) ∨ (∃ *E* ∈ *N*. *E* ⊏ *C* ∧ *D* ∈ *𝒢-F* (*fst E*))}
⟨*proof*⟩

lemma *mem-FL-Red-F-because-G-Red-F*:
(∀ *D* ∈ *𝒢-F* (*fst Cl*). *D* ∈ *G.Red-F* (⋃ (*𝒢-F* ‘ *fst* ‘ *N*))) ⟹ *Cl* ∈ *FL.Red-F N*
⟨*proof*⟩

lemma *mem-FL-Red-F-because-Prec-FL*:
(∀ *D* ∈ *𝒢-F* (*fst Cl*). ∃ *El* ∈ *N*. *El* ⊏ *Cl* ∧ *D* ∈ *𝒢-F* (*fst El*)) ⟹ *Cl* ∈ *FL.Red-F N*
⟨*proof*⟩

4.6 Resolution Prover Layer

interpretation *sq*: *selection S-Q Sts*
⟨*proof*⟩

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

interpretation *src*: *standard-redundancy-criterion-counterex-reducing gd.ord-*Γ *Sts*
ground-resolution-with-selection.INTERP (*S-Q Sts*)
⟨*proof*⟩

definition *lclss-of-state* :: *'a state* ⇒ (*'a clause* × *label*) *set* **where**
lclss-of-state St =
(λ*C*. (*C*, *New*)) ' *N-of-state St* ∪ (λ*C*. (*C*, *Processed*)) ' *P-of-state St*
∪ (λ*C*. (*C*, *Old*)) ' *Q-of-state St*

lemma *image-hd-lclss-of-state*[*simp*]: *fst* ' *lclss-of-state St* = *clss-of-state St*
⟨*proof*⟩

lemma *insert-lclss-of-state*[*simp*]:
insert (*C*, *New*) (*lclss-of-state* (*N*, *P*, *Q*)) = *lclss-of-state* (*N* ∪ {*C*}, *P*, *Q*)
insert (*C*, *Processed*) (*lclss-of-state* (*N*, *P*, *Q*)) = *lclss-of-state* (*N*, *P* ∪ {*C*}, *Q*)
insert (*C*, *Old*) (*lclss-of-state* (*N*, *P*, *Q*)) = *lclss-of-state* (*N*, *P*, *Q* ∪ {*C*})
⟨*proof*⟩

lemma *union-lclss-of-state*[*simp*]:
lclss-of-state (*N1*, *P1*, *Q1*) ∪ *lclss-of-state* (*N2*, *P2*, *Q2*) =
lclss-of-state (*N1* ∪ *N2*, *P1* ∪ *P2*, *Q1* ∪ *Q2*)
⟨*proof*⟩

lemma *mem-lclss-of-state*[*simp*]:
(*C*, *New*) ∈ *lclss-of-state* (*N*, *P*, *Q*) ⟷ *C* ∈ *N*
(*C*, *Processed*) ∈ *lclss-of-state* (*N*, *P*, *Q*) ⟷ *C* ∈ *P*
(*C*, *Old*) ∈ *lclss-of-state* (*N*, *P*, *Q*) ⟷ *C* ∈ *Q*
⟨*proof*⟩

lemma *lclss-Liminf-commute*:
Liminf-llist (*lmap lclss-of-state Sts*) = *lclss-of-state* (*Liminf-state Sts*)
⟨*proof*⟩

lemma *GC-tautology-step*:
assumes *tauto*: *Neg A* ∈# *C Pos A* ∈# *C*
shows *lclss-of-state* (*N* ∪ {*C*}, *P*, *Q*) ⇝*GC lclss-of-state* (*N*, *P*, *Q*)
⟨*proof*⟩

lemma *GC-subsumption-step*:
assumes
d-in: *Dl* ∈ *N* **and**
d-sub-c: *strictly-subsumes* (*fst Dl*) (*fst Cl*) ∨ *subsumes* (*fst Dl*) (*fst Cl*) ∧ *snd Dl* ⊏*l snd Cl*
shows *N* ∪ {*Cl*} ⇝*GC N*
⟨*proof*⟩

lemma *GC-reduction-step*:
assumes
young: *snd Dl* ≠ *Old* **and**
d-sub-c: *fst Dl* ⊂# *fst Cl*
shows *N* ∪ {*Cl*} ⇝*GC N* ∪ {*Dl*}
⟨*proof*⟩

lemma *GC-processing-step*: $N \cup \{(C, New)\} \leadsto GC\ N \cup \{(C, Processed)\}$
⟨*proof*⟩

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*⟩

lemma *GC-inference-step*:
assumes
young: $l \neq Old$ **and**
no-active: *FL.active-subset M* = {} **and**
m-sup: *fst* ‘ *M* ⊇ *old-concl-of* ‘ *inference-system.inferences-between* (*ord-FO-*Γ *S*)
(*fst* ‘ *FL.active-subset N*) *C*
shows $N \cup \{(C, l)\} \leadsto GC\ N \cup \{(C, Old)\} \cup M$
⟨*proof*⟩

lemma *RP-step-imp-GC-step*: $St \leadsto RP\ St' \Longrightarrow$ *lclss-of-state St* $\leadsto GC$ *lclss-of-state St′*
⟨*proof*⟩

lemma *RP-derivation-imp-GC-derivation*: *chain* ($\leadsto$*RP*) *Sts* $\Longrightarrow$ *chain* ($\leadsto$*GC*) (*lmap lclss-of-state Sts*)
⟨*proof*⟩

lemma *RP-step-imp-derive-step*: $St \leadsto RP\ St' \Longrightarrow$ *lclss-of-state St* ▷*L* *lclss-of-state St′*
⟨*proof*⟩

lemma *RP-derivation-imp-derive-derivation*:
chain ($\leadsto$*RP*) *Sts* $\Longrightarrow$ *chain* (▷*L*) (*lmap lclss-of-state Sts*)
⟨*proof*⟩

theorem *RP-sound-new-statement*:
assumes
deriv: *chain* ($\leadsto$*RP*) *Sts* **and**
bot-in: {#} ∈ *clss-of-state* (*Liminf-state Sts*)
shows *clss-of-state* (*lhd Sts*) $\models\mathcal{G}e$ {{#}}
⟨*proof*⟩

theorem *RP-saturated-if-fair-new-statement*:
assumes
deriv: *chain* ($\leadsto$*RP*) *Sts* **and**
init: *FL.active-subset* (*lclss-of-state* (*lhd Sts*)) = {} **and**
final: *FL.passive-subset* (*Liminf-llist* (*lmap lclss-of-state Sts*)) = {}
shows *FL.saturated* (*Liminf-llist* (*lmap lclss-of-state Sts*))
⟨*proof*⟩

corollary *RP-complete-if-fair-new-statement*:
assumes
deriv: *chain* ($\leadsto$*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 {#} ∈ *Q-of-state* (*Liminf-state Sts*)
⟨*proof*⟩

4.7 Alternative Derivation of Previous RP Results

lemma *old-fair-imp-new-fair*:
  assumes
    nnul: ¬ *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*⟩

lemma *old-redundant-infer-iff*:
  src.redundant-infer N γ ⟷
    (∃ *DD*. *DD* ⊆ *N* ∧ *DD* ∪ *set-mset* (*old-side-prems-of γ*) ⊫*e* {*old-concl-of γ*}
      ∧ (∀ *D* ∈ *DD*. *D* < *old-main-prem-of γ*))
  (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

definition *old-infer-of* :: *'a clause inference* ⇒ *'a old-inference* **where**
  old-infer-of ι = *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 ι ⟹ *src.redundant-infer N* (*old-infer-of ι*)
  ⟨*proof*⟩

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*⟩

theorem *RP-sound-old-statement*:
  assumes
    deriv: *chain* (↝*RP*) *Sts* **and**
    bot-in: {#} ∈ *clss-of-state* (*Liminf-state Sts*)
  shows ¬ *satisfiable* (*grounding-of-state* (*lhd Sts*))
  ⟨*proof*⟩

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* (↝*RP*) *Sts* **and**
    fair: *fair-state-seq Sts* **and**
    empty-Q0: *Q-of-state* (*lhd Sts*) = {}
  shows *src.saturated-upto Sts* (*grounding-of-state* (*Liminf-state Sts*))
⟨*proof*⟩

corollary *RP-complete-if-fair-old-statement*:
  assumes
    deriv: *chain* (↝*RP*) *Sts* **and**
    fair: *fair-state-seq Sts* **and**
    empty-Q0: *Q-of-state* (*lhd Sts*) = {} **and**

unsat: ¬ *satisfiable* (*grounding-of-state* (*lhd Sts*))
shows {#} ∈ *Q-of-state* (*Liminf-state Sts*)
⟨*proof*⟩

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* ∞
⟨*proof*⟩

lemma *gc-invar-gc-init*:
assumes
¬ *lnull Ns* **and**
active-subset (*lhd Ns*) = {}
shows *gc-invar Ns 0*
⟨*proof*⟩

lemma *gc-invar-gc-step*:
assumes
Si-lt: *enat* (*Suc i*) < *llength Ns* **and**
invar: *gc-invar Ns i* **and**
step: *lnth Ns i* ⇝*GC lnth Ns* (*Suc i*)
shows *gc-invar Ns* (*Suc i*)
⟨*proof*⟩

lemma *gc-invar-gc*:
assumes
gc: *chain* (⇝*GC*) *Ns* **and**
init: *active-subset* (*lhd Ns*) = {} **and**
i-lt: *i* < *llength Ns*

shows *gc-invar Ns i*
⟨*proof*⟩

lemma *gc-fair-new-proof*:
assumes
gc: *chain* (↝*GC*) *Ns* **and**
init: *active-subset* (*lhd Ns*) = {} **and**
lim: *passive-subset* (*Liminf-llist Ns*) = {}
shows *fair Ns*
⟨*proof*⟩

end

5.2 Lazy Given Clause

context *lazy-given-clause*
begin

definition *from-F* :: *'f inference* ⇒ (*'f* × *'l*) *inference set* **where**
from-F ι = {ι' ∈ *Inf-FL*. *to-F* ι' = ι}

definition *lgc-invar* :: (*'f inference set* × (*'f* × *'l*) *set*) *llist* ⇒ *enat* ⇒ *bool* **where**
lgc-invar TNs i ⟷
Inf-from (*active-subset* (*Liminf-upto-llist* (*lmap snd TNs*) *i*))
⊆ ⋃ (*from-F* ‘ *Liminf-upto-llist* (*lmap fst TNs*) *i*) ∪ *Sup-upto-llist* (*lmap* (*Red-I-*$\mathcal{G}$ ∘ *snd*) *TNs*) *i*

lemma *lgc-invar-infinity*:
assumes
nnil: ¬ *lnull TNs* **and**
invar: ∀ *i*. *enat i* < *llength TNs* ⟶ *lgc-invar TNs* (*enat i*)
shows *lgc-invar TNs* ∞
⟨*proof*⟩

lemma *lgc-invar-lgc-init*:
assumes
nnil: ¬ *lnull TNs* **and**
n-init: *active-subset* (*snd* (*lhd TNs*)) = {} **and**
t-init: ∀ ι ∈ *Inf-F*. *prems-of* ι = [] ⟶ ι ∈ *fst* (*lhd TNs*)
shows *lgc-invar TNs 0*
⟨*proof*⟩

lemma *lgc-invar-lgc-step*:
assumes
Si-lt: *enat* (*Suc i*) < *llength TNs* **and**
invar: *lgc-invar TNs i* **and**
step: *lnth TNs i* ↝*LGC lnth TNs* (*Suc i*)
shows *lgc-invar TNs* (*Suc i*)
⟨*proof*⟩

lemma *lgc-invar-lgc*:
assumes
lgc: *chain* (↝*LGC*) *TNs* **and**
n-init: *active-subset* (*snd* (*lhd TNs*)) = {} **and**
t-init: ∀ ι ∈ *Inf-F*. *prems-of* ι = [] ⟶ ι ∈ *fst* (*lhd TNs*) **and**
i-lt: *i* < *llength TNs*
shows *lgc-invar TNs i*

⟨*proof*⟩

lemma *lgc-fair-new-proof*:
assumes
lgc: *chain* (⇝*LGC*) *TNs* **and**
n-init: *active-subset* (*snd* (*lhd TNs*)) = {} **and**
n-lim: *passive-subset* (*Liminf-llist* (*lmap snd TNs*)) = {} **and**
t-init: $\forall \iota \in$ *Inf-F*. *prems-of* $\iota = [] \longrightarrow \iota \in$ *fst* (*lhd TNs*) **and**
t-lim: *Liminf-llist* (*lmap fst TNs*) = {}
shows *fair* (*lmap snd TNs*)
⟨*proof*⟩

end

end