Given Clause Loops

Jasmin Blanchette Qi Qiu Sophie Tourret

May 26, 2024

Abstract

This Isabelle/HOL formalization extends the `Saturation_Framework` and `Saturation_Framework_Extensions` entries of the *Archive of Formal Proofs* with the specification and verification of four semiabstract given clause procedures, or "loops": the DISCOUNT, Otter, iProver, and Zipperposition loops. For each loop, (dynamic) refutational completeness is proved under the assumption that the underlying calculus is (statically) refutationally complete and that the used queue data structures are fair.

The formalization is inspired by the proof sketches found in the article "A comprehensive framework for saturation theorem proving" by Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette (*Journal of Automated Reasoning* **66**(4): 499–539, 2022). A paper titled "Verified given clause procedures" about the present formalization is in the works.

Contents

1 Utilities for Given Clause Loops

This section contains various lemmas used by the rest of the formalization of given clause procedures.

theory *Given-Clause-Loops-Util*
imports
HOL−Library.FSet
HOL−Library.Multiset
Ordered-Resolution-Prover.Lazy-List-Chain
Weighted-Path-Order.Multiset-Extension-Pair
Lambda-Free-RPOs.Lambda-Free-Util
begin

hide-const (**open**) *Seq.chain*

hide-fact (**open**) *Abstract-Rewriting.chain-mono*

declare *fset-of-list.rep-eq* [*simp*]

instance *bool* :: *wellorder*
⟨*proof*⟩

lemma *finite-imp-set-eq*:
assumes *fin*: *finite A*
shows ∃ *xs*. *set xs* = *A*
⟨*proof*⟩

lemma *Union-Setcompr-member-mset-mono*:
assumes *sub*: *P* ⊆# *Q*
shows ⋃ {*f x* |*x*. *x* ∈# *P*} ⊆ ⋃ {*f x* |*x*. *x* ∈# *Q*}
⟨*proof*⟩

lemma *singletons-in-mult1*: (*x*, *y*) ∈ *R* ⟹ ({#*x*#}, {#*y*#}) ∈ *mult1 R*
⟨*proof*⟩

lemma *singletons-in-mult*: (*x*, *y*) ∈ *R* ⟹ ({#*x*#}, {#*y*#}) ∈ *mult R*
⟨*proof*⟩

lemma *multiset-union-diff-assoc*:
fixes *A B C* :: *'a multiset*
assumes *A* ∩# *C* = {#}
shows *A* + *B* − *C* = *A* + (*B* − *C*)

⟨*proof*⟩

lemma *Liminf-llist-subset*:
assumes
llength Xs = *llength Ys* **and**
∀ *i* < *llength Xs*. *lnth Xs i* ⊆ *lnth Ys i*
shows *Liminf-llist Xs* ⊆ *Liminf-llist Ys*
⟨*proof*⟩

lemma *countable-imp-lset*:
assumes *count*: *countable A*
shows ∃ *as*. *lset as* = *A*
⟨*proof*⟩

lemma *distinct-imp-notin-set-drop-Suc*:
assumes
distinct xs
i < *length xs*
xs ! *i* = *x*
shows *x* ∉ *set* (*drop* (*Suc i*) *xs*)
⟨*proof*⟩

lemma *distinct-set-drop-removeAll-hd*:
assumes
distinct xs
xs ≠ []
shows *set* (*drop n* (*removeAll* (*hd xs*) *xs*)) = *set* (*drop* (*Suc n*) *xs*)
⟨*proof*⟩

lemma *set-drop-removeAll*: *set* (*drop n* (*removeAll y xs*)) ⊆ *set* (*drop n xs*)
⟨*proof*⟩

lemma *set-drop-fold-removeAll*: *set* (*drop k* (*fold removeAll ys xs*)) ⊆ *set* (*drop k xs*)
⟨*proof*⟩

lemma *set-drop-append-subseteq*: *set* (*drop n* (*xs* @ *ys*)) ⊆ *set* (*drop n xs*) ∪ *set ys*
⟨*proof*⟩

lemma *distinct-fold-removeAll*:
assumes *dist*: *distinct xs*
shows *distinct* (*fold removeAll ys xs*)
⟨*proof*⟩

lemma *set-drop-append-cons*: *set* (*drop n* (*xs* @ *ys*)) ⊆ *set* (*drop n* (*xs* @ *y* # *ys*))
⟨*proof*⟩

lemma *chain-ltl*: *chain R sts* ⟹ ¬ *lnull* (*ltl sts*) ⟹ *chain R* (*ltl sts*)
⟨*proof*⟩

end

2 More Lemmas about Given Clause Architectures

This section proves lemmas about Tourret's formalization of the abstract given clause procedures GC and LGC.

theory *More-Given-Clause-Architectures*
  imports *Saturation-Framework.Given-Clause-Architectures*
begin

2.1 Inference System

context *inference-system*
begin

lemma *Inf-from-empty*: *Inf-from* $\{\} = \{\iota \in Inf.\ prems\text{-}of\ \iota = []\}$
  ⟨*proof*⟩

end

2.2 Given Clause Procedure Basis

context *given-clause-basis*
begin

lemma *no-labels-entails-mono-left*: $M \subseteq N \Longrightarrow M \models\!\cap\mathcal{G}\ P \Longrightarrow N \models\!\cap\mathcal{G}\ P$
  ⟨*proof*⟩

lemma *no-labels-Red-F-imp-Red-F*:
  assumes $C \in no\text{-}labels.Red\text{-}F\ (fst\ `\ \mathcal{N})$
  shows $(C,\ l) \in Red\text{-}F\ \mathcal{N}$
⟨*proof*⟩

lemma *succ-F-imp-Red-F*:
  assumes
    $C' \in fst\ `\ \mathcal{N}$ **and**
    $C' \prec\cdot C$
  shows $(C,\ l) \in Red\text{-}F\ \mathcal{N}$
⟨*proof*⟩

lemma *succ-L-imp-Red-F*:
  assumes
    $(C',\ l') \in \mathcal{N}$ **and**
    $C' \preceq\cdot C$ **and**
    $l' \sqsubset L\ l$
  shows $(C,\ l) \in Red\text{-}F\ \mathcal{N}$
⟨*proof*⟩

lemma *prj-fl-set-to-f-set-distr-union* [*simp*]: $fst\ `\ (\mathcal{M} \cup \mathcal{N}) = fst\ `\ \mathcal{M} \cup fst\ `\ \mathcal{N}$
  ⟨*proof*⟩

lemma *prj-labeledN-eq-N* [*simp*]: $fst\ `\ \{(C,\ l) \mid C.\ C \in N\} = N$
⟨*proof*⟩

end

2.3 Given Clause Procedure

context *given-clause*
begin

lemma *remove-redundant*:
assumes $(C, l) \in$ *Red-F* $\mathcal{N}$
shows $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow GC\ \mathcal{N}$
⟨*proof*⟩

lemma *remove-redundant-no-label*:
assumes $C \in$ *no-labels.Red-F* (*fst* ‘ $\mathcal{N}$)
shows $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow GC\ \mathcal{N}$
⟨*proof*⟩

lemma *add-inactive*:
assumes $l \neq$ *active*
shows $\mathcal{N} \rightsquigarrow GC\ \mathcal{N} \cup \{(C, l)\}$
⟨*proof*⟩

lemma *remove-succ-F*:
assumes
$(C', l') \in \mathcal{N}$ **and**
$C' \prec\cdot C$
shows $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow GC\ \mathcal{N}$
⟨*proof*⟩

lemma *remove-succ-L*:
assumes
$(C', l') \in \mathcal{N}$ **and**
$C' \preceq\cdot C$ **and**
$l' \sqsubset L\ l$
shows $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow GC\ \mathcal{N}$
⟨*proof*⟩

lemma *relabel-inactive*:
assumes
$l' \sqsubset L\ l$ **and**
$l' \neq$ *active*
shows $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow GC\ \mathcal{N} \cup \{(C, l')\}$
⟨*proof*⟩

end

2.4 Lazy Given Clause Procedure

context *lazy-given-clause*
begin

lemma *remove-redundant*:
assumes $(C, l) \in$ *Red-F* $\mathcal{N}$
shows $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow LGC\ (T, \mathcal{N})$
⟨*proof*⟩

lemma *remove-redundant-no-label*:
assumes $C \in$ *no-labels.Red-F* (*fst* ‘ $\mathcal{N}$)

shows $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow LGC\ (T, \mathcal{N})$
⟨*proof*⟩

lemma *add-inactive*:
assumes $l \neq active$
shows $(T, \mathcal{N}) \rightsquigarrow LGC\ (T, \mathcal{N} \cup \{(C, l)\})$
⟨*proof*⟩

lemma *remove-succ-F*:
assumes
$(C', l') \in \mathcal{N}$ **and**
$C' \prec\cdot C$
shows $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow LGC\ (T, \mathcal{N})$
⟨*proof*⟩

lemma *remove-succ-L*:
assumes
$(C', l') \in \mathcal{N}$ **and**
$C' \preceq\cdot C$ **and**
$l' \sqsubset L\ l$
shows $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow LGC\ (T, \mathcal{N})$
⟨*proof*⟩

lemma *relabel-inactive*:
assumes
$l' \sqsubset L\ l$ **and**
$l' \neq active$
shows $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow LGC\ (T, \mathcal{N} \cup \{(C, l')\})$
⟨*proof*⟩

end

end

3 DISCOUNT Loop

The DISCOUNT loop is one of the two best-known given clause procedures. It is formalized as an instance of the abstract procedure *LGC*.

theory *DISCOUNT-Loop*
imports
Given-Clause-Loops-Util
More-Given-Clause-Architectures
begin

3.1 Locale

datatype *DL-label* =
Passive | *YY* | *Active*

primrec *nat-of-DL-label* :: *DL-label* ⇒ *nat* **where**
nat-of-DL-label Passive = *2*
| *nat-of-DL-label YY* = *1*
| *nat-of-DL-label Active* = *0*

definition *DL-Prec-L* :: *DL-label* ⇒ *DL-label* ⇒ *bool* (**infix** ⊏*L* *50*) **where**
DL-Prec-L l l′ ⟷ *nat-of-DL-label l* < *nat-of-DL-label l′*

locale *discount-loop* = *labeled-lifting-intersection Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q*
{ι_{FL} :: ('*f* × '*l*) *inference*. *Infer* (*map fst* (*prems-of* ι_{FL})) (*fst* (*concl-of* ι_{FL})) ∈ *Inf-F*}
for
Bot-F :: '*f set*
and *Inf-F* :: '*f inference set*
and *Bot-G* :: '*g set*
and *Q* :: '*q set*
and *entails-q* :: '*q* ⇒ '*g set* ⇒ '*g set* ⇒ *bool*
and *Inf-G-q* :: ‹'*q* ⇒ '*g inference set*›
and *Red-I-q* :: '*q* ⇒ '*g set* ⇒ '*g inference set*
and *Red-F-q* :: '*q* ⇒ '*g set* ⇒ '*g set*
and $\mathcal{G}$*-F-q* :: '*q* ⇒ '*f* ⇒ '*g set*
and $\mathcal{G}$*-I-q* :: '*q* ⇒ '*f inference* ⇒ '*g inference set option*
+ **fixes**
Equiv-F :: '*f* ⇒ '*f* ⇒ *bool* (**infix** ≐ *50*) **and**
Prec-F :: '*f* ⇒ '*f* ⇒ *bool* (**infix** ≺· *50*)
assumes
equiv-equiv-F: *equivp* (≐) **and**
wf-prec-F: *minimal-element* (≺·) *UNIV* **and**
compat-equiv-prec: *C1* ≐ *D1* ⟹ *C2* ≐ *D2* ⟹ *C1* ≺· *C2* ⟹ *D1* ≺· *D2* **and**
equiv-F-grounding: *q* ∈ *Q* ⟹ *C1* ≐ *C2* ⟹ $\mathcal{G}$*-F-q q C1* ⊆ $\mathcal{G}$*-F-q q C2* **and**
prec-F-grounding: *q* ∈ *Q* ⟹ *C2* ≺· *C1* ⟹ $\mathcal{G}$*-F-q q C1* ⊆ $\mathcal{G}$*-F-q q C2* **and**
static-ref-comp: *statically-complete-calculus Bot-F Inf-F* (⊨∩$\mathcal{G}$)
no-labels.Red-I-$\mathcal{G}$ *no-labels.Red-F-*$\mathcal{G}$*-empty* **and**
inf-have-prems: ι*F* ∈ *Inf-F* ⟹ *prems-of* ι*F* ≠ []
begin

lemma *po-on-DL-Prec-L*: *po-on* (⊏*L*) *UNIV*
⟨*proof*⟩

lemma *wfp-on-DL-Prec-L*: *wfp-on* (⊏*L*) *UNIV*
⟨*proof*⟩

lemma *Active-minimal*: *l2* ≠ *Active* ⟹ *Active* ⊏*L l2*
⟨*proof*⟩

lemma *at-least-two-labels*: ∃ *l2*. *Active* ⊏*L l2*
⟨*proof*⟩

sublocale *lgc?*: *lazy-given-clause Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F DL-Prec-L Active*
⟨*proof*⟩

notation *lgc.step* (**infix** ⇝*LGC* *50*)

3.2 Basic Definitions and Lemmas

abbreviation *c-dot-succ* :: '*f* ⇒ '*f* ⇒ *bool* (**infix** ·≻ *50*) **where**
C ·≻ *C′* ≡ *C′* ≺· *C*
abbreviation *sqsupset* :: *DL-label* ⇒ *DL-label* ⇒ *bool* (**infix** ⊐*L* *50*) **where**
l ⊐*L l′* ≡ *l′* ⊏*L l*

fun *labeled-formulas-of* :: $'f\ set \times 'f\ set \times 'f\ set \Rightarrow ('f \times DL\text{-}label)\ set$ **where**
labeled-formulas-of $(P,\ Y,\ A) = \{(C,\ Passive) \mid C.\ C \in P\} \cup \{(C,\ YY) \mid C.\ C \in Y\} \cup$
$\{(C,\ Active) \mid C.\ C \in A\}$

lemma *labeled-formulas-of-alt-def*:
labeled-formulas-of $(P,\ Y,\ A) =$
$(\lambda C.\ (C,\ Passive))\ `\ P \cup (\lambda C.\ (C,\ YY))\ `\ Y \cup (\lambda C.\ (C,\ Active))\ `\ A$
⟨*proof*⟩

fun
state :: $'f\ inference\ set \times 'f\ set \times 'f\ set \times 'f\ set \Rightarrow 'f\ inference\ set \times ('f \times DL\text{-}label)\ set$
where
state $(T,\ P,\ Y,\ A) = (T,$ *labeled-formulas-of* $(P,\ Y,\ A))$

lemma *state-alt-def*:
state $(T,\ P,\ Y,\ A) = (T,\ (\lambda C.\ (C,\ Passive))\ `\ P \cup (\lambda C.\ (C,\ YY))\ `\ Y \cup (\lambda C.\ (C,\ Active))\ `\ A)$
⟨*proof*⟩

inductive
DL :: $'f\ inference\ set \times ('f \times DL\text{-}label)\ set \Rightarrow 'f\ inference\ set \times ('f \times DL\text{-}label)\ set \Rightarrow bool$
(**infix** $\rightsquigarrow DL\ 50$)
where
compute-infer: $\iota \in$ *no-labels.Red-I* $(A \cup \{C\}) \Longrightarrow$
state $(T \cup \{\iota\},\ P,\ \{\},\ A) \rightsquigarrow DL$ *state* $(T,\ P,\ \{C\},\ A)$
| *choose-p*: *state* $(T,\ P \cup \{C\},\ \{\},\ A) \rightsquigarrow DL$ *state* $(T,\ P,\ \{C\},\ A)$
| *delete-fwd*: $C \in$ *no-labels.Red-F* $A \vee (\exists\, C' \in A.\ C' \preceq\cdot\ C) \Longrightarrow$
state $(T,\ P,\ \{C\},\ A) \rightsquigarrow DL$ *state* $(T,\ P,\ \{\},\ A)$
| *simplify-fwd*: $C \in$ *no-labels.Red-F* $(A \cup \{C'\}) \Longrightarrow$
state $(T,\ P,\ \{C\},\ A) \rightsquigarrow DL$ *state* $(T,\ P,\ \{C'\},\ A)$
| *delete-bwd*: $C' \in$ *no-labels.Red-F* $\{C\} \vee C' \cdot\succ C \Longrightarrow$
state $(T,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow DL$ *state* $(T,\ P,\ \{C\},\ A)$
| *simplify-bwd*: $C' \in$ *no-labels.Red-F* $\{C,\ C''\} \Longrightarrow$
state $(T,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow DL$ *state* $(T,\ P \cup \{C''\},\ \{C\},\ A)$
| *schedule-infer*: $T' =$ *no-labels.Inf-between* $A\ \{C\} \Longrightarrow$
state $(T,\ P,\ \{C\},\ A) \rightsquigarrow DL$ *state* $(T \cup T',\ P,\ \{\},\ A \cup \{C\})$
| *delete-orphan-infers*: $T' \cap$ *no-labels.Inf-from* $A = \{\} \Longrightarrow$
state $(T \cup T',\ P,\ Y,\ A) \rightsquigarrow DL$ *state* $(T,\ P,\ Y,\ A)$

lemma *If-f-in-A-then-fl-in-PYA*: $C' \in A \Longrightarrow (C',\ Active) \in$ *labeled-formulas-of* $(P,\ Y,\ A)$
⟨*proof*⟩

lemma *PYA-add-passive-formula*[*simp*]:
labeled-formulas-of $(P,\ Y,\ A) \cup \{(C,\ Passive)\} =$ *labeled-formulas-of* $(P \cup \{C\},\ Y,\ A)$
⟨*proof*⟩

lemma *P0A-add-y-formula*[*simp*]:
labeled-formulas-of $(P,\ \{\},\ A) \cup \{(C,\ YY)\} =$ *labeled-formulas-of* $(P,\ \{C\},\ A)$
⟨*proof*⟩

lemma *PYA-add-active-formula*[*simp*]:
labeled-formulas-of $(P,\ Y,\ A) \cup \{(C',\ Active)\} =$ *labeled-formulas-of* $(P,\ Y,\ A \cup \{C'\})$
⟨*proof*⟩

lemma *prj-active-subset-of-state*: *fst* ` *active-subset* (*labeled-formulas-of* $(P,\ Y,\ A)) = A$
⟨*proof*⟩

lemma *active-subset-of-setOfFormulasWithLabelDiffActive*:
$l \neq Active \Longrightarrow$ *active-subset* $\{(C', l)\} = \{\}$
⟨*proof*⟩

3.3 Refinement

lemma *dl-compute-infer-in-lgc*:
assumes $\iota \in$ *no-labels.Red-I-*$\mathcal{G}$ $(A \cup \{C\})$
shows *state* $(T \cup \{\iota\}, P, \{\}, A) \rightsquigarrow LGC$ *state* $(T, P, \{C\}, A)$
⟨*proof*⟩

lemma *dl-choose-p-in-lgc*: *state* $(T, P \cup \{C\}, \{\}, A) \rightsquigarrow LGC$ *state* $(T, P, \{C\}, A)$
⟨*proof*⟩

lemma *dl-delete-fwd-in-lgc*:
assumes $(C \in$ *no-labels.Red-F* $A) \vee (\exists\, C' \in A.\ C' \preceq\cdot\ C)$
shows *state* $(T, P, \{C\}, A) \rightsquigarrow LGC$ *state* $(T, P, \{\}, A)$
⟨*proof*⟩

lemma *dl-simplify-fwd-in-lgc*:
assumes $C \in$ *no-labels.Red-F-*$\mathcal{G}$ $(A \cup \{C'\})$
shows *state* $(T, P, \{C\}, A) \rightsquigarrow LGC$ *state* $(T, P, \{C'\}, A)$
⟨*proof*⟩

lemma *dl-delete-bwd-in-lgc*:
assumes $C' \in$ *no-labels.Red-F-*$\mathcal{G}$ $\{C\} \vee C' \cdot\succ C$
shows *state* $(T, P, \{C\}, A \cup \{C'\}) \rightsquigarrow LGC$ *state* $(T, P, \{C\}, A)$
⟨*proof*⟩

lemma *dl-simplify-bwd-in-lgc*:
assumes $C' \in$ *no-labels.Red-F-*$\mathcal{G}$ $\{C, C''\}$
shows *state* $(T, P, \{C\}, A \cup \{C'\}) \rightsquigarrow LGC$ *state* $(T, P \cup \{C''\}, \{C\}, A)$
⟨*proof*⟩

lemma *dl-schedule-infer-in-lgc*:
assumes $T' =$ *no-labels.Inf-between* A $\{C\}$
shows *state* $(T, P, \{C\}, A) \rightsquigarrow LGC$ *state* $(T \cup T', P, \{\}, A \cup \{C\})$
⟨*proof*⟩

lemma *dl-delete-orphan-infers-in-lgc*:
assumes $T' \cap$ *no-labels.Inf-from* $A = \{\}$
shows *state* $(T \cup T', P, Y, A) \rightsquigarrow LGC$ *state* (T, P, Y, A)
⟨*proof*⟩

theorem *DL-step-imp-LGC-step*: $T\mathcal{M} \rightsquigarrow DL\ T\mathcal{M}' \Longrightarrow T\mathcal{M} \rightsquigarrow LGC\ T\mathcal{M}'$
⟨*proof*⟩

3.4 Completeness

theorem
assumes
dl-chain: *chain* $(\rightsquigarrow DL)$ *Sts* **and**
act: *active-subset* (*snd* (*lhd Sts*)) = {} **and**
pas: *passive-subset* (*Liminf-llist* (*lmap snd Sts*)) = {} **and**
no-prems-init: $\forall \iota \in$ *Inf-F*. *prems-of* $\iota = [] \longrightarrow \iota \in$ *fst* (*lhd Sts*) **and**

final-sched: *Liminf-llist* (*lmap fst Sts*) = {}
shows
DL-Liminf-saturated: *saturated* (*Liminf-llist* (*lmap snd Sts*)) **and**
DL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *snd* (*lhd Sts*) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *Liminf-llist* (*lmap snd Sts*) **and**
DL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *snd* (*lhd Sts*) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists\, i.$ *enat* $i <$ *llength Sts* $\wedge$ ($\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *snd* (*lnth Sts* *i*))
⟨*proof*⟩

end

end

4 Prover Queues and Fairness

This section covers the passive set data structure that arises in different prover loops in the literature (e.g., DISCOUNT, Otter).

theory *Prover-Queue*
imports
Given-Clause-Loops-Util
Ordered-Resolution-Prover.Lazy-List-Chain
begin

4.1 Basic Lemmas

lemma *set-drop-fold-maybe-append-singleton*:
set (*drop k* (*fold* ($\lambda y\ xs.$ *if* $y \in$ *set xs then xs else xs* @ [*y*]) *ys xs*)) $\subseteq$ *set* (*drop k* (*xs* @ *ys*))
⟨*proof*⟩

lemma *fold-maybe-append-removeAll*:
assumes $y \in$ *set xs*
shows *fold* ($\lambda y\ xs.$ *if* $y \in$ *set xs then xs else xs* @ [*y*]) (*removeAll y ys*) *xs* =
fold ($\lambda y\ xs.$ *if* $y \in$ *set xs then xs else xs* @ [*y*]) *ys xs*
⟨*proof*⟩

4.2 More on Relational Chains over Lazy Lists

definition *finitely-often* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ '*a llist* $\Rightarrow$ *bool* **where**
finitely-often R xs $\longleftrightarrow$
($\exists\, i.\ \forall j.\ i \leq j \longrightarrow$ *enat* (*Suc j*) $<$ *llength xs* $\longrightarrow \neg$ *R* (*lnth xs j*) (*lnth xs* (*Suc j*)))

abbreviation *infinitely-often* :: ('*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*) $\Rightarrow$ '*a llist* $\Rightarrow$ *bool* **where**
infinitely-often R xs $\equiv \neg$ *finitely-often R xs*

lemma *infinitely-often-alt-def*:
infinitely-often R xs $\longleftrightarrow$
($\forall\, i.\ \exists j.\ i \leq j \wedge$ *enat* (*Suc j*) $<$ *llength xs* $\wedge$ *R* (*lnth xs j*) (*lnth xs* (*Suc j*)))
⟨*proof*⟩

lemma *infinitely-often-lifting*:
assumes
r-imp-s: $\forall\, x\ x'.$ *R* (*f x*) (*f x'*) $\longrightarrow$ *S* (*g x*) (*g x'*) **and**
inf-r: *infinitely-often R* (*lmap f xs*)
shows *infinitely-often S* (*lmap g xs*)
⟨*proof*⟩

4.3 Locales

The passive set of a given clause prover can be organized in different ways—e.g., as a priority queue or as a list of queues. This locale abstracts over the specific data structure.

locale *prover-queue* =
  fixes
    empty :: *'q* **and**
    select :: *'q* ⇒ *'e* **and**
    add :: *'e* ⇒ *'q* ⇒ *'q* **and**
    remove :: *'e* ⇒ *'q* ⇒ *'q* **and**
    felems :: *'q* ⇒ *'e fset*
  assumes
    felems-empty[*simp*]: *felems empty* = {||} **and**
    felems-not-empty: *Q* ≠ *empty* ⟹ *felems Q* ≠ {||} **and**
    select-in-felems[*simp*]: *Q* ≠ *empty* ⟹ *select Q* |∈| *felems Q* **and**
    felems-add[*simp*]: *felems* (*add e Q*) = {|*e*|} |∪| *felems Q* **and**
    felems-remove[*simp*]: *felems* (*remove e Q*) = *felems Q* |−| {|*e*|} **and**
    add-again: *e* |∈| *felems Q* ⟹ *add e Q* = *Q*
begin

abbreviation *elems* :: *'q* ⇒ *'e set* **where**
  elems Q ≡ *fset* (*felems Q*)

lemma *elems-empty*: *elems empty* = {}
  ⟨*proof*⟩

lemma *formula-not-empty*[*simp*]: *Q* ≠ *empty* ⟹ *elems Q* ≠ {}
  ⟨*proof*⟩

lemma
  elems-add: *elems* (*add e Q*) = {*e*} ∪ *elems Q* **and**
  elems-remove: *elems* (*remove e Q*) = *elems Q* − {*e*}
  ⟨*proof*⟩

lemma *elems-fold-add*[*simp*]: *elems* (*fold add es Q*) = *set es* ∪ *elems Q*
  ⟨*proof*⟩

lemma *elems-fold-remove*[*simp*]: *elems* (*fold remove es Q*) = *elems Q* − *set es*
  ⟨*proof*⟩

inductive *queue-step* :: *'q* ⇒ *'q* ⇒ *bool* **where**
  queue-step-fold-addI: *queue-step Q* (*fold add es Q*)
| *queue-step-fold-removeI*: *queue-step Q* (*fold remove es Q*)

lemma *queue-step-idleI*: *queue-step Q Q*
  ⟨*proof*⟩

lemma *queue-step-addI*: *queue-step Q* (*add e Q*)
  ⟨*proof*⟩

lemma *queue-step-removeI*: *queue-step Q* (*remove e Q*)
  ⟨*proof*⟩

inductive *select-queue-step* :: *'q* ⇒ *'q* ⇒ *bool* **where**
  select-queue-stepI: *Q* ≠ *empty* ⟹ *select-queue-step Q* (*remove* (*select Q*) *Q*)

end

locale *fair-prover-queue* = *prover-queue empty select add remove felems*
  for
    empty :: *'q* **and**
    select :: *'q* ⇒ *'e* **and**
    add :: *'e* ⇒ *'q* ⇒ *'q* **and**
    remove :: *'e* ⇒ *'q* ⇒ *'q* **and**
    felems :: *'q* ⇒ *'e fset* +
  assumes *fair*: *chain queue-step Qs* ⟹ *infinitely-often select-queue-step Qs* ⟹
    lhd Qs = *empty* ⟹ *Liminf-llist* (*lmap elems Qs*) = {}
begin
end

4.4 Instantiation with FIFO Queue

As a proof of concept, we show that a FIFO queue can serve as a fair prover queue.

locale *fifo-prover-queue*
begin

sublocale *prover-queue* [] *hd* λ*y xs*. *if y* ∈ *set xs then xs else xs* @ [*y*] *removeAll fset-of-list*
⟨*proof*⟩

lemma *queue-step-preserves-distinct*:
  assumes
    dist: *distinct Q* **and**
    step: *queue-step Q Q'*
  shows *distinct Q'*
  ⟨*proof*⟩

lemma *chain-queue-step-preserves-distinct*:
  assumes
    chain: *chain queue-step Qs* **and**
    dist-hd: *distinct* (*lhd Qs*) **and**
    i-lt: *enat i* < *llength Qs*
  shows *distinct* (*lnth Qs i*)
  ⟨*proof*⟩

sublocale *fair-prover-queue* [] *hd* λ*y xs*. *if y* ∈ *set xs then xs else xs* @ [*y*] *removeAll*
  fset-of-list
⟨*proof*⟩

end

end

5 Fair DISCOUNT Loop

The fair DISCOUNT loop assumes that the passive queue is fair and ensures (dynamic) refutational completeness under that assumption.

theory *Fair-DISCOUNT-Loop*
  imports

Given-Clause-Loops-Util
DISCOUNT-Loop
Prover-Queue
begin

5.1 Locale

type-synonym (*'p*, *'f*) *DLf-state* = *'p* × *'f option* × *'f fset*

datatype *'f passive-elem* =
is-passive-inference: *Passive-Inference* (*passive-inference*: *'f inference*)
| *is-passive-formula*: *Passive-Formula* (*passive-formula*: *'f*)

lemma *passive-inference-filter*:
passive-inference ‘ *Set.filter is-passive-inference N* = {*ι*. *Passive-Inference ι* ∈ *N*}
⟨*proof*⟩

lemma *passive-formula-filter*:
passive-formula ‘ *Set.filter is-passive-formula N* = {*C*. *Passive-Formula C* ∈ *N*}
⟨*proof*⟩

locale *fair-discount-loop* =
discount-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F* +
fair-prover-queue empty select add remove felems
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
entails-q :: *'q* ⇒ *'g set* ⇒ *'g set* ⇒ *bool* **and**
Inf-G-q :: *'q* ⇒ *'g inference set* **and**
Red-I-q :: *'q* ⇒ *'g set* ⇒ *'g inference set* **and**
Red-F-q :: *'q* ⇒ *'g set* ⇒ *'g set* **and**
$\mathcal{G}$*-F-q* :: *'q* ⇒ *'f* ⇒ *'g set* **and**
$\mathcal{G}$*-I-q* :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**
Equiv-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≐› *50*) **and**
Prec-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺·› *50*) **and**
empty :: *'p* **and**
select :: *'p* ⇒ *'f passive-elem* **and**
add :: *'f passive-elem* ⇒ *'p* ⇒ *'p* **and**
remove :: *'f passive-elem* ⇒ *'p* ⇒ *'p* **and**
felems :: *'p* ⇒ *'f passive-elem fset* +
fixes
Prec-S :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ≺*S 50*)
assumes
wf-Prec-S: *minimal-element* (≺*S*) *UNIV* **and**
transp-Prec-S: *transp* (≺*S*) **and**
finite-Inf-between: *finite A* ⟹ *finite* (*no-labels.Inf-between A* {*C*})
begin

lemma *trans-Prec-S*: *trans* {(*x*, *y*). *x* ≺*S y*}
⟨*proof*⟩

lemma *irreflp-Prec-S*: *irreflp* (≺*S*)
⟨*proof*⟩

lemma *irrefl-Prec-S*: *irrefl* $\{(x,\ y).\ x \prec S\ y\}$
⟨*proof*⟩

5.2 Basic Definitions and Lemmas

abbreviation *passive-of* :: $('p,\ 'f)$ *DLf-state* ⇒ $'p$ **where**
passive-of St ≡ *fst St*
abbreviation *yy-of* :: $('p,\ 'f)$ *DLf-state* ⇒ $'f$ *option* **where**
yy-of St ≡ *fst* (*snd St*)
abbreviation *active-of* :: $('p,\ 'f)$ *DLf-state* ⇒ $'f$ *fset* **where**
active-of St ≡ *snd* (*snd St*)

definition *passive-inferences-of* :: $'p$ ⇒ $'f$ *inference set* **where**
passive-inferences-of P = $\{\iota.$ *Passive-Inference* $\iota \in$ *elems P*$\}$
definition *passive-formulas-of* :: $'p$ ⇒ $'f$ *set* **where**
passive-formulas-of P = $\{C.$ *Passive-Formula* $C \in$ *elems P*$\}$

lemma *finite-passive-inferences-of*: *finite* (*passive-inferences-of P*)
⟨*proof*⟩

lemma *finite-passive-formulas-of*: *finite* (*passive-formulas-of P*)
⟨*proof*⟩

abbreviation *all-formulas-of* :: $('p,\ 'f)$ *DLf-state* ⇒ $'f$ *set* **where**
all-formulas-of St ≡ *passive-formulas-of* (*passive-of St*) ∪ *set-option* (*yy-of St*) ∪
fset (*active-of St*)

lemma *passive-inferences-of-empty*[*simp*]: *passive-inferences-of empty* = {}
⟨*proof*⟩

lemma *passive-inferences-of-add-Passive-Inference*[*simp*]:
passive-inferences-of (*add* (*Passive-Inference* ι) *P*) = $\{\iota\}$ ∪ *passive-inferences-of P*
⟨*proof*⟩

lemma *passive-inferences-of-add-Passive-Formula*[*simp*]:
passive-inferences-of (*add* (*Passive-Formula C*) *P*) = *passive-inferences-of P*
⟨*proof*⟩

lemma *passive-inferences-of-fold-add-Passive-Inference*[*simp*]:
passive-inferences-of (*fold* (*add* ∘ *Passive-Inference*) *ιs P*) = *passive-inferences-of P* ∪ *set ιs*
⟨*proof*⟩

lemma *passive-inferences-of-fold-add-Passive-Formula*[*simp*]:
passive-inferences-of (*fold* (*add* ∘ *Passive-Formula*) *Cs P*) = *passive-inferences-of P*
⟨*proof*⟩

lemma *passive-inferences-of-remove-Passive-Inference*[*simp*]:
passive-inferences-of (*remove* (*Passive-Inference* ι) *P*) = *passive-inferences-of P* − $\{\iota\}$
⟨*proof*⟩

lemma *passive-inferences-of-remove-Passive-Formula*[*simp*]:
passive-inferences-of (*remove* (*Passive-Formula C*) *P*) = *passive-inferences-of P*
⟨*proof*⟩

lemma *passive-inferences-of-fold-remove-Passive-Inference*[*simp*]:
passive-inferences-of (*fold* (*remove* ∘ *Passive-Inference*) *ιs P*) = *passive-inferences-of P* − *set ιs*

⟨*proof*⟩

lemma *passive-inferences-of-fold-remove-Passive-Formula*[*simp*]:
passive-inferences-of (*fold* (*remove* ∘ *Passive-Formula*) *Cs P*) = *passive-inferences-of P*
⟨*proof*⟩

lemma *passive-formulas-of-empty*[*simp*]: *passive-formulas-of empty* = {}
⟨*proof*⟩

lemma *passive-formulas-of-add-Passive-Inference*[*simp*]:
passive-formulas-of (*add* (*Passive-Inference* ι) *P*) = *passive-formulas-of P*
⟨*proof*⟩

lemma *passive-formulas-of-add-Passive-Formula*[*simp*]:
passive-formulas-of (*add* (*Passive-Formula C*) *P*) = {*C*} ∪ *passive-formulas-of P*
⟨*proof*⟩

lemma *passive-formulas-of-fold-add-Passive-Inference*[*simp*]:
passive-formulas-of (*fold* (*add* ∘ *Passive-Inference*) ι*s P*) = *passive-formulas-of P*
⟨*proof*⟩

lemma *passive-formulas-of-fold-add-Passive-Formula*[*simp*]:
passive-formulas-of (*fold* (*add* ∘ *Passive-Formula*) *Cs P*) = *passive-formulas-of P* ∪ *set Cs*
⟨*proof*⟩

lemma *passive-formulas-of-remove-Passive-Inference*[*simp*]:
passive-formulas-of (*remove* (*Passive-Inference* ι) *P*) = *passive-formulas-of P*
⟨*proof*⟩

lemma *passive-formulas-of-remove-Passive-Formula*[*simp*]:
passive-formulas-of (*remove* (*Passive-Formula C*) *P*) = *passive-formulas-of P* − {*C*}
⟨*proof*⟩

lemma *passive-formulas-of-fold-remove-Passive-Inference*[*simp*]:
passive-formulas-of (*fold* (*remove* ∘ *Passive-Inference*) ι*s P*) = *passive-formulas-of P*
⟨*proof*⟩

lemma *passive-formulas-of-fold-remove-Passive-Formula*[*simp*]:
passive-formulas-of (*fold* (*remove* ∘ *Passive-Formula*) *Cs P*) = *passive-formulas-of P* − *set Cs*
⟨*proof*⟩

fun *fstate* :: ('*p*, '*f*) *DLf-state* ⇒ '*f inference set* × ('*f* × *DL-label*) *set* **where**
fstate (*P*, *Y*, *A*) = *state* (*passive-inferences-of P*, *passive-formulas-of P*, *set-option Y*, *fset A*)

lemma *fstate-alt-def*:
fstate St = *state* (*passive-inferences-of* (*fst St*), *passive-formulas-of* (*fst St*),
set-option (*fst* (*snd St*)), *fset* (*snd* (*snd St*)))
⟨*proof*⟩

definition *Liminf-fstate* :: ('*p*, '*f*) *DLf-state llist* ⇒ '*f set* × '*f set* × '*f set* **where**
Liminf-fstate Sts =
(*Liminf-llist* (*lmap* (*passive-formulas-of* ∘ *passive-of*) *Sts*),
Liminf-llist (*lmap* (*set-option* ∘ *yy-of*) *Sts*),
Liminf-llist (*lmap* (*fset* ∘ *active-of*) *Sts*))

lemma *Liminf-fstate-commute*:
Liminf-llist (*lmap* (*snd* ∘ *fstate*) *Sts*) = *labeled-formulas-of* (*Liminf-fstate Sts*)
⟨*proof*⟩

fun *formulas-union* :: *'f set* × *'f set* × *'f set* ⇒ *'f set* **where**
formulas-union (*P*, *Y*, *A*) = *P* ∪ *Y* ∪ *A*

inductive *fair-DL* :: (*'p*, *'f*) *DLf-state* ⇒ (*'p*, *'f*) *DLf-state* ⇒ *bool* (**infix** ⇝*DLf 50*) **where**
compute-infer: *P* ≠ *empty* ⟹ *select P* = *Passive-Inference ι* ⟹
ι ∈ *no-labels.Red-I* (*fset A* ∪ {*C*}) ⟹
(*P*, *None*, *A*) ⇝*DLf* (*remove* (*select P*) *P*, *Some C*, *A*)
| *choose-p*: *P* ≠ *empty* ⟹ *select P* = *Passive-Formula C* ⟹
(*P*, *None*, *A*) ⇝*DLf* (*remove* (*select P*) *P*, *Some C*, *A*)
| *delete-fwd*: *C* ∈ *no-labels.Red-F* (*fset A*) ∨ (∃ *C'* ∈ *fset A*. *C'* ⪯· *C*) ⟹
(*P*, *Some C*, *A*) ⇝*DLf* (*P*, *None*, *A*)
| *simplify-fwd*: *C'* ≺*S C* ⟹ *C* ∈ *no-labels.Red-F* (*fset A* ∪ {*C'*}) ⟹
(*P*, *Some C*, *A*) ⇝*DLf* (*P*, *Some C'*, *A*)
| *delete-bwd*: *C'* |∉| *A* ⟹ *C'* ∈ *no-labels.Red-F* {*C*} ∨ *C'* ·≻ *C* ⟹
(*P*, *Some C*, *A* |∪| {|*C'*|}) ⇝*DLf* (*P*, *Some C*, *A*)
| *simplify-bwd*: *C'* |∉| *A* ⟹ *C''* ≺*S C'* ⟹ *C'* ∈ *no-labels.Red-F* {*C*, *C''*} ⟹
(*P*, *Some C*, *A* |∪| {|*C'*|}) ⇝*DLf* (*add* (*Passive-Formula C''*) *P*, *Some C*, *A*)
| *schedule-infer*: *set ιs* = *no-labels.Inf-between* (*fset A*) {*C*} ⟹
(*P*, *Some C*, *A*) ⇝*DLf* (*fold* (*add* ∘ *Passive-Inference*) *ιs P*, *None*, *A* |∪| {|*C*|})
| *delete-orphan-infers*: *ιs* ≠ [] ⟹ *set ιs* ⊆ *passive-inferences-of P* ⟹
set ιs ∩ *no-labels.Inf-from* (*fset A*) = {} ⟹
(*P*, *Y*, *A*) ⇝*DLf* (*fold* (*remove* ∘ *Passive-Inference*) *ιs P*, *Y*, *A*)

5.3 Initial State and Invariant

inductive *is-initial-DLf-state* :: (*'p*, *'f*) *DLf-state* ⇒ *bool* **where**
is-initial-DLf-state (*empty*, *None*, {||})

inductive *DLf-invariant* :: (*'p*, *'f*) *DLf-state* ⇒ *bool* **where**
passive-inferences-of P ⊆ *Inf-F* ⟹ *DLf-invariant* (*P*, *Y*, *A*)

lemma *initial-DLf-invariant*: *is-initial-DLf-state St* ⟹ *DLf-invariant St*
⟨*proof*⟩

lemma *step-DLf-invariant*:
assumes
inv: *DLf-invariant St* **and**
step: *St* ⇝*DLf St'*
shows *DLf-invariant St'*
⟨*proof*⟩

lemma *chain-DLf-invariant-lnth*:
assumes
chain: *chain* (⇝*DLf*) *Sts* **and**
fair-hd: *DLf-invariant* (*lhd Sts*) **and**
i-lt: *enat i* < *llength Sts*
shows *DLf-invariant* (*lnth Sts i*)
⟨*proof*⟩

lemma *chain-DLf-invariant-llast*:
assumes
chain: *chain* (⇝*DLf*) *Sts* **and**

fair-hd: *DLf-invariant* (*lhd Sts*) **and**
fin: *lfinite Sts*
shows *DLf-invariant* (*llast Sts*)
⟨*proof*⟩

5.4 Final State

inductive *is-final-DLf-state* :: ('*p*, '*f*) *DLf-state* ⇒ *bool* **where**
is-final-DLf-state (*empty*, *None*, *A*)

lemma *is-final-DLf-state-iff-no-DLf-step*:
assumes *inv*: *DLf-invariant St*
shows *is-final-DLf-state St* ⟷ (∀ *St'*. ¬ *St* ⇝*DLf St'*)
⟨*proof*⟩

5.5 Refinement

lemma *fair-DL-step-imp-DL-step*:
assumes *dlf*: (*P*, *Y*, *A*) ⇝*DLf* (*P'*, *Y'*, *A'*)
shows *fstate* (*P*, *Y*, *A*) ⇝*DL fstate* (*P'*, *Y'*, *A'*)
⟨*proof*⟩

lemma *fair-DL-step-imp-GC-step*:
(*P*, *Y*, *A*) ⇝*DLf* (*P'*, *Y'*, *A'*) ⟹ *fstate* (*P*, *Y*, *A*) ⇝*LGC fstate* (*P'*, *Y'*, *A'*)
⟨*proof*⟩

5.6 Completeness

fun *mset-of-fstate* :: ('*p*, '*f*) *DLf-state* ⇒ '*f multiset* **where**
mset-of-fstate (*P*, *Y*, *A*) =
image-mset concl-of (*mset-set* (*passive-inferences-of P*)) + *mset-set* (*passive-formulas-of P*) +
mset-set (*set-option Y*) + *mset-set* (*fset A*)

abbreviation *Precprec-S* :: '*f multiset* ⇒ '*f multiset* ⇒ *bool* (**infix** ≺≺*S* *50*) **where**
(≺≺*S*) ≡ *multp* (≺*S*)

lemma *wfP-Precprec-S*: *wfP* (≺≺*S*)
⟨*proof*⟩

definition *Less-state* :: ('*p*, '*f*) *DLf-state* ⇒ ('*p*, '*f*) *DLf-state* ⇒ *bool* (**infix** ⊏ *50*) **where**
St' ⊏ *St* ⟷
(*yy-of St'* = *None* ∧ *yy-of St* ≠ *None*)
∨ ((*yy-of St'* = *None* ⟷ *yy-of St* = *None*) ∧ *mset-of-fstate St'* ≺≺*S mset-of-fstate St*)

lemma *wfP-Less-state*: *wfP* (⊏)
⟨*proof*⟩

lemma *non-compute-infer-choose-p-DLf-step-imp-Less-state*:
assumes
step: *St* ⇝*DLf St'* **and**
yy: *yy-of St* ≠ *None* ∨ *yy-of St'* = *None*
shows *St'* ⊏ *St*
⟨*proof*⟩

lemma *yy-nonempty-DLf-step-imp-Less-state*:
assumes

step: $St \rightsquigarrow_{DLf} St'$ **and**
yy: *yy-of* $St \neq None$ **and**
yy': *yy-of* $St' \neq None$
shows $St' \sqsubset St$
⟨*proof*⟩

lemma *fair-DL-Liminf-yy-empty*:
assumes
len: *llength* $Sts = \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{DLf})$ *Sts* **and**
inv: *DLf-invariant* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*set-option* ∘ *yy-of*) *Sts*) = {}
⟨*proof*⟩

lemma *DLf-step-imp-queue-step*:
assumes $St \rightsquigarrow_{DLf} St'$
shows *queue-step* (*passive-of St*) (*passive-of St'*)
⟨*proof*⟩

lemma *fair-DL-Liminf-passive-empty*:
assumes
len: *llength* $Sts = \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{DLf})$ *Sts* **and**
init: *is-initial-DLf-state* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*elems* ∘ *passive-of*) *Sts*) = {}
⟨*proof*⟩

lemma *fair-DL-Liminf-passive-formulas-empty*:
assumes
len: *llength* $Sts = \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{DLf})$ *Sts* **and**
init: *is-initial-DLf-state* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*passive-formulas-of* ∘ *passive-of*) *Sts*) = {}
⟨*proof*⟩

lemma *fair-DL-Liminf-passive-inferences-empty*:
assumes
len: *llength* $Sts = \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{DLf})$ *Sts* **and**
init: *is-initial-DLf-state* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*passive-inferences-of* ∘ *passive-of*) *Sts*) = {}
⟨*proof*⟩

theorem
assumes
full: *full-chain* $(\rightsquigarrow_{DLf})$ *Sts* **and**
init: *is-initial-DLf-state* (*lhd Sts*)
shows
fair-DL-Liminf-saturated: *saturated* (*labeled-formulas-of* (*Liminf-fstate Sts*)) **and**
fair-DL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *passive-formulas-of* (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, B' \in$ *Bot-F*. $B' \in$ *formulas-union* (*Liminf-fstate Sts*) **and**
fair-DL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *passive-formulas-of* (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, i$. *enat* $i <$ *llength Sts* $\wedge$ $(\exists\, B' \in$ *Bot-F*. $B' \in$ *all-formulas-of* (*lnth Sts i*))
⟨*proof*⟩

end

5.7 Specialization with FIFO Queue

As a proof of concept, we specialize the passive set to use a FIFO queue, thereby eliminating the locale assumptions about the passive set.

locale *fifo-discount-loop* =
  discount-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F*
  for
    Bot-F :: *'f set* **and**
    Inf-F :: *'f inference set* **and**
    Bot-G :: *'g set* **and**
    Q :: *'q set* **and**
    entails-q :: *'q* ⇒ *'g set* ⇒ *'g set* ⇒ *bool* **and**
    Inf-G-q :: *'q* ⇒ *'g inference set* **and**
    Red-I-q :: *'q* ⇒ *'g set* ⇒ *'g inference set* **and**
    Red-F-q :: *'q* ⇒ *'g set* ⇒ *'g set* **and**
    $\mathcal{G}$*-F-q* :: *'q* ⇒ *'f* ⇒ *'g set* **and**
    $\mathcal{G}$*-I-q* :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**
    Equiv-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≐› *50*) **and**
    Prec-F :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ‹≺·› *50*) +
  fixes
    Prec-S :: *'f* ⇒ *'f* ⇒ *bool* (**infix** ≺*S 50*)
  assumes
    wf-Prec-S: *minimal-element* (≺*S*) *UNIV* **and**
    transp-Prec-S: *transp* (≺*S*) **and**
    finite-Inf-between: *finite A* ⟹ *finite* (*no-labels.Inf-between A* {*C*})
begin

sublocale *fifo-prover-queue*
  ⟨*proof*⟩

sublocale *fair-discount-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q*
  Equiv-F Prec-F [] *hd* λ*y xs. if y* ∈ *set xs then xs else xs* @ [*y*] *removeAll fset-of-list Prec-S*
⟨*proof*⟩

end

end

6 Otter Loop

The Otter loop is one of the two best-known given clause procedures. It is formalized as an instance of the abstract procedure *GC*.

theory *Otter-Loop*
  imports
    More-Given-Clause-Architectures
    Given-Clause-Loops-Util
begin

datatype *OL-label* =
  New | *XX* | *Passive* | *YY* | *Active*

primrec *nat-of-OL-label* :: *OL-label* ⇒ *nat* **where**
nat-of-OL-label New = *4*
| *nat-of-OL-label XX* = *3*
| *nat-of-OL-label Passive* = *2*
| *nat-of-OL-label YY* = *1*
| *nat-of-OL-label Active* = *0*

definition *OL-Prec-L* :: *OL-label* ⇒ *OL-label* ⇒ *bool* (**infix** ⊏*L* *50*) **where**
OL-Prec-L l l′ ⟷ *nat-of-OL-label l* < *nat-of-OL-label l′*

locale *otter-loop* = *labeled-lifting-intersection Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* 𝒢*-F-q* 𝒢*-I-q*
{ι*FL* :: (′*f* × *OL-label*) *inference*. *Infer* (*map fst* (*prems-of* ι*FL*)) (*fst* (*concl-of* ι*FL*)) ∈ *Inf-F*}
for
Bot-F :: ′*f set*
and *Inf-F* :: ′*f inference set*
and *Bot-G* :: ′*g set*
and *Q* :: ′*q set*
and *entails-q* :: ′*q* ⇒ ′*g set* ⇒ ′*g set* ⇒ *bool*
and *Inf-G-q* :: ‹′*q* ⇒ ′*g inference set*›
and *Red-I-q* :: ′*q* ⇒ ′*g set* ⇒ ′*g inference set*
and *Red-F-q* :: ′*q* ⇒ ′*g set* ⇒ ′*g set*
and 𝒢*-F-q* :: ′*q* ⇒ ′*f* ⇒ ′*g set*
and 𝒢*-I-q* :: ′*q* ⇒ ′*f inference* ⇒ ′*g inference set option*
+ **fixes**
Equiv-F :: ′*f* ⇒ ′*f* ⇒ *bool* (**infix** ≐ *50*) **and**
Prec-F :: ′*f* ⇒ ′*f* ⇒ *bool* (**infix** ≺· *50*)
assumes
equiv-equiv-F: *equivp* (≐) **and**
wf-prec-F: *minimal-element* (≺·) *UNIV* **and**
compat-equiv-prec: *C1* ≐ *D1* ⟹ *C2* ≐ *D2* ⟹ *C1* ≺· *C2* ⟹ *D1* ≺· *D2* **and**
equiv-F-grounding: *q* ∈ *Q* ⟹ *C1* ≐ *C2* ⟹ 𝒢*-F-q q C1* ⊆ 𝒢*-F-q q C2* **and**
prec-F-grounding: *q* ∈ *Q* ⟹ *C2* ≺· *C1* ⟹ 𝒢*-F-q q C1* ⊆ 𝒢*-F-q q C2* **and**
static-ref-comp: *statically-complete-calculus Bot-F Inf-F* (⊨∩𝒢)
*no-labels.Red-I-*𝒢 *no-labels.Red-F-*𝒢*-empty* **and**
inf-have-prems: ι*F* ∈ *Inf-F* ⟹ *prems-of* ι*F* ≠ []
begin

lemma *po-on-OL-Prec-L*: *po-on* (⊏*L*) *UNIV*
⟨*proof*⟩

lemma *wfp-on-OL-Prec-L*: *wfp-on* (⊏*L*) *UNIV*
⟨*proof*⟩

lemma *Active-minimal*: *l2* ≠ *Active* ⟹ *Active* ⊏*L l2*
⟨*proof*⟩

lemma *at-least-two-labels*: ∃ *l2*. *Active* ⊏*L l2*
⟨*proof*⟩

sublocale *gc?*: *given-clause Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* 𝒢*-F-q* 𝒢*-I-q Equiv-F Prec-F OL-Prec-L Active*
⟨*proof*⟩

notation *gc.step* (**infix** ⇝*GC* *50*)

6.1 Basic Definitions and Lemmas

fun *state* :: *'f set* × *'f set* × *'f set* × *'f set* × *'f set* ⇒ (*'f* × *OL-label*) *set* **where**
state (*N*, *X*, *P*, *Y*, *A*) =
{(*C*, *New*) | *C*. *C* ∈ *N*} ∪ {(*C*, *XX*) | *C*. *C* ∈ *X*} ∪ {(*C*, *Passive*) | *C*. *C* ∈ *P*} ∪
{(*C*, *YY*) | *C*. *C* ∈ *Y*} ∪ {(*C*, *Active*) | *C*. *C* ∈ *A*}

lemma *state-alt-def*:
state (*N*, *X*, *P*, *Y*, *A*) =
(λ*C*. (*C*, *New*)) ‘ *N* ∪ (λ*C*. (*C*, *XX*)) ‘ *X* ∪ (λ*C*. (*C*, *Passive*)) ‘ *P* ∪ (λ*C*. (*C*, *YY*)) ‘ *Y* ∪
(λ*C*. (*C*, *Active*)) ‘ *A*
⟨*proof*⟩

inductive *OL* :: (*'f* × *OL-label*) *set* ⇒ (*'f* × *OL-label*) *set* ⇒ *bool* (**infix** ⇝*OL* *50*) **where**
choose-n: *C* ∉ *N* ⟹ *state* (*N* ∪ {*C*}, {}, *P*, {}, *A*) ⇝*OL* *state* (*N*, {*C*}, *P*, {}, *A*)
| *delete-fwd*: *C* ∈ *no-labels.Red-F* (*P* ∪ *A*) ∨ (∃ *C'* ∈ *P* ∪ *A*. *C'* ⪯· *C*) ⟹
state (*N*, {*C*}, *P*, {}, *A*) ⇝*OL* *state* (*N*, {}, *P*, {}, *A*)
| *simplify-fwd*: *C* ∈ *no-labels.Red-F* (*P* ∪ *A* ∪ {*C'*}) ⟹
state (*N*, {*C*}, *P*, {}, *A*) ⇝*OL* *state* (*N*, {*C'*}, *P*, {}, *A*)
| *delete-bwd-p*: *C'* ∈ *no-labels.Red-F* {*C*} ∨ *C* ≺· *C'* ⟹
state (*N*, {*C*}, *P* ∪ {*C'*}, {}, *A*) ⇝*OL* *state* (*N*, {*C*}, *P*, {}, *A*)
| *simplify-bwd-p*: *C'* ∈ *no-labels.Red-F* {*C*, *C''*} ⟹
state (*N*, {*C*}, *P* ∪ {*C'*}, {}, *A*) ⇝*OL* *state* (*N* ∪ {*C''*}, {*C*}, *P*, {}, *A*)
| *delete-bwd-a*: *C'* ∈ *no-labels.Red-F* {*C*} ∨ *C* ≺· *C'* ⟹
state (*N*, {*C*}, *P*, {}, *A* ∪ {*C'*}) ⇝*OL* *state* (*N*, {*C*}, *P*, {}, *A*)
| *simplify-bwd-a*: *C'* ∈ *no-labels.Red-F* ({*C*, *C''*}) ⟹
state (*N*, {*C*}, *P*, {}, *A* ∪ {*C'*}) ⇝*OL* *state* (*N* ∪ {*C''*}, {*C*}, *P*, {}, *A*)
| *transfer*: *state* (*N*, {*C*}, *P*, {}, *A*) ⇝*OL* *state* (*N*, {}, *P* ∪ {*C*}, {}, *A*)
| *choose-p*: *C* ∉ *P* ⟹ *state* ({}, {}, *P* ∪ {*C*}, {}, *A*) ⇝*OL* *state* ({}, {}, *P*, {*C*}, *A*)
| *infer*: *no-labels.Inf-between* *A* {*C*} ⊆ *no-labels.Red-I* (*A* ∪ {*C*} ∪ *M*) ⟹
state ({}, {}, *P*, {*C*}, *A*) ⇝*OL* *state* (*M*, {}, *P*, {}, *A* ∪ {*C*})

lemma *prj-state-union-sets* [*simp*]: *fst* ‘ *state* (*N*, *X*, *P*, *Y*, *A*) = *N* ∪ *X* ∪ *P* ∪ *Y* ∪ *A*
⟨*proof*⟩

lemma *active-subset-of-setOfFormulasWithLabelDiffActive*:
l ≠ *Active* ⟹ *active-subset* {(*C'*, *l*)} = {}
⟨*proof*⟩

lemma *state-add-C-New*: *state* (*N*, *X*, *P*, *Y*, *A*) ∪ {(*C*, *New*)} = *state* (*N* ∪ {*C*}, *X*, *P*, *Y*, *A*)
⟨*proof*⟩

lemma *state-add-C-XX*: *state* (*N*, *X*, *P*, *Y*, *A*) ∪ {(*C*, *XX*)} = *state* (*N*, *X* ∪ {*C*}, *P*, *Y*, *A*)
⟨*proof*⟩

lemma *state-add-C-Passive*: *state* (*N*, *X*, *P*, *Y*, *A*) ∪ {(*C*, *Passive*)} = *state* (*N*, *X*, *P* ∪ {*C*}, *Y*, *A*)
⟨*proof*⟩

lemma *state-add-C-YY*: *state* (*N*, *X*, *P*, *Y*, *A*) ∪ {(*C*, *YY*)} = *state* (*N*, *X*, *P*, *Y* ∪ {*C*}, *A*)
⟨*proof*⟩

lemma *state-add-C-Active*: *state* (*N*, *X*, *P*, *Y*, *A*) ∪ {(*C*, *Active*)} = *state* (*N*, *X*, *P*, *Y*, *A* ∪ {*C*})
⟨*proof*⟩

lemma *prj-ActiveSubset-of-state*: *fst* ‘ *active-subset* (*state* (*N*, *X*, *P*, *Y*, *A*)) = *A*
⟨*proof*⟩

6.2 Refinement

lemma *chooseN-in-GC*: *state* $(N \cup \{C\}, \{\}, P, \{\}, A) \rightsquigarrow GC$ *state* $(N, \{C\}, P, \{\}, A)$
⟨*proof*⟩

lemma *deleteFwd-in-GC*:
assumes $C \in$ *no-labels.Red-F* $(P \cup A) \vee (\exists\, C' \in P \cup A.\ C' \preceq\cdot\ C)$
shows *state* $(N, \{C\}, P, \{\}, A) \rightsquigarrow GC$ *state* $(N, \{\}, P, \{\}, A)$
⟨*proof*⟩

lemma *simplifyFwd-in-GC*:
$C \in$ *no-labels.Red-F* $(P \cup A \cup \{C'\}) \Longrightarrow$
state $(N, \{C\}, P, \{\}, A) \rightsquigarrow GC$ *state* $(N, \{C'\}, P, \{\}, A)$
⟨*proof*⟩

lemma *deleteBwdP-in-GC*:
assumes $C' \in$ *no-labels.Red-F* $\{C\} \vee C \prec\cdot C'$
shows *state* $(N, \{C\}, P \cup \{C'\}, \{\}, A) \rightsquigarrow GC$ *state* $(N, \{C\}, P, \{\}, A)$
⟨*proof*⟩

lemma *simplifyBwdP-in-GC*:
assumes $C' \in$ *no-labels.Red-F* $\{C, C''\}$
shows *state* $(N, \{C\}, P \cup \{C'\}, \{\}, A) \rightsquigarrow GC$ *state* $(N \cup \{C''\}, \{C\}, P, \{\}, A)$
⟨*proof*⟩

lemma *deleteBwdA-in-GC*:
assumes $C' \in$ *no-labels.Red-F* $\{C\} \vee C \prec\cdot C'$
shows *state* $(N, \{C\}, P, \{\}, A \cup \{C'\}) \rightsquigarrow GC$ *state* $(N, \{C\}, P, \{\}, A)$
⟨*proof*⟩

lemma *simplifyBwdA-in-GC*:
assumes $C' \in$ *no-labels.Red-F* $\{C, C''\}$
shows *state* $(N, \{C\}, P, \{\}, A \cup \{C'\}) \rightsquigarrow GC$ *state* $(N \cup \{C''\}, \{C\}, P, \{\}, A)$
⟨*proof*⟩

lemma *transfer-in-GC*: *state* $(N, \{C\}, P, \{\}, A) \rightsquigarrow GC$ *state* $(N, \{\}, P \cup \{C\}, \{\}, A)$
⟨*proof*⟩

lemma *chooseP-in-GC*: *state* $(\{\}, \{\}, P \cup \{C\}, \{\}, A) \rightsquigarrow GC$ *state* $(\{\}, \{\}, P, \{C\}, A)$
⟨*proof*⟩

lemma *infer-in-GC*:
assumes *no-labels.Inf-between* $A\ \{C\} \subseteq$ *no-labels.Red-I* $(A \cup \{C\} \cup M)$
shows *state* $(\{\}, \{\}, P, \{C\}, A) \rightsquigarrow GC$ *state* $(M, \{\}, P, \{\}, A \cup \{C\})$
⟨*proof*⟩

theorem *OL-step-imp-GC-step*: $M \rightsquigarrow OL\ M' \Longrightarrow M \rightsquigarrow GC\ M'$
⟨*proof*⟩

6.3 Completeness

theorem
assumes
ol-chain: *chain* $(\rightsquigarrow OL)$ *Sts* **and**
act: *active-subset* $(lhd\ Sts) = \{\}$ **and**
pas: *passive-subset* $(Liminf\text{-}llist\ Sts) = \{\}$

shows
OL-Liminf-saturated: *saturated* (*Liminf-llist Sts*) **and**
OL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *lhd Sts* $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, BL \in$ *Bot-FL*. *BL* $\in$ *Liminf-llist Sts* **and**
OL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *lhd Sts* $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, i$. *enat* $i <$ *llength Sts* $\wedge$ $(\exists\, BL \in$ *Bot-FL*. *BL* $\in$ *lnth Sts* $i)$
⟨*proof*⟩

end

end

7 Definition of Fair Otter Loop

The fair Otter loop assumes that the passive queue is fair and ensures (dynamic) refutational completeness under that assumption. This section contains only the loop's definition.

theory *Fair-Otter-Loop-Def*
imports
Otter-Loop
Prover-Queue
begin

7.1 Locale

type-synonym $('p, 'f)$ *OLf-state* $=$ $'f$ *fset* $\times$ $'f$ *option* $\times$ $'p \times$ $'f$ *option* $\times$ $'f$ *fset*

locale *fair-otter-loop* =
otter-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F* +
fair-prover-queue empty select add remove felems
for
Bot-F :: $'f$ *set* **and**
Inf-F :: $'f$ *inference set* **and**
Bot-G :: $'g$ *set* **and**
Q :: $'q$ *set* **and**
entails-q :: $'q \Rightarrow$ $'g$ *set* $\Rightarrow$ $'g$ *set* $\Rightarrow$ *bool* **and**
Inf-G-q :: $'q \Rightarrow$ $'g$ *inference set* **and**
Red-I-q :: $'q \Rightarrow$ $'g$ *set* $\Rightarrow$ $'g$ *inference set* **and**
Red-F-q :: $'q \Rightarrow$ $'g$ *set* $\Rightarrow$ $'g$ *set* **and**
$\mathcal{G}$*-F-q* :: $'q \Rightarrow$ $'f \Rightarrow$ $'g$ *set* **and**
$\mathcal{G}$*-I-q* :: $'q \Rightarrow$ $'f$ *inference* $\Rightarrow$ $'g$ *inference set option* **and**
Equiv-F :: $'f \Rightarrow$ $'f \Rightarrow$ *bool* (**infix** $\langle\doteq\rangle$ *50*) **and**
Prec-F :: $'f \Rightarrow$ $'f \Rightarrow$ *bool* (**infix** $\langle\prec\cdot\rangle$ *50*) **and**
empty :: $'p$ **and**
select :: $'p \Rightarrow$ $'f$ **and**
add :: $'f \Rightarrow$ $'p \Rightarrow$ $'p$ **and**
remove :: $'f \Rightarrow$ $'p \Rightarrow$ $'p$ **and**
felems :: $'p \Rightarrow$ $'f$ *fset* +
fixes
Prec-S :: $'f \Rightarrow$ $'f \Rightarrow$ *bool* (**infix** $\prec S$ *50*)
assumes
wf-Prec-S: *minimal-element* $(\prec S)$ *UNIV* **and**
transp-Prec-S: *transp* $(\prec S)$ **and**
finite-Inf-between: *finite A* $\Longrightarrow$ *finite* (*no-labels.Inf-between A* $\{C\}$)
begin

lemma *trans-Prec-S*: *trans* $\{(x, y).\ x \prec S\ y\}$
⟨*proof*⟩

lemma *irreflp-Prec-S*: *irreflp* $(\prec S)$
⟨*proof*⟩

lemma *irrefl-Prec-S*: *irrefl* $\{(x, y).\ x \prec S\ y\}$
⟨*proof*⟩

7.2 Basic Definitions and Lemmas

abbreviation *new-of* :: $('p, 'f)$ *OLf-state* ⇒ $'f$ *fset* **where**
new-of St ≡ *fst St*
abbreviation *xx-of* :: $('p, 'f)$ *OLf-state* ⇒ $'f$ *option* **where**
xx-of St ≡ *fst* (*snd St*)
abbreviation *passive-of* :: $('p, 'f)$ *OLf-state* ⇒ $'p$ **where**
passive-of St ≡ *fst* (*snd* (*snd St*))
abbreviation *yy-of* :: $('p, 'f)$ *OLf-state* ⇒ $'f$ *option* **where**
yy-of St ≡ *fst* (*snd* (*snd* (*snd St*)))
abbreviation *active-of* :: $('p, 'f)$ *OLf-state* ⇒ $'f$ *fset* **where**
active-of St ≡ *snd* (*snd* (*snd* (*snd St*)))

abbreviation *all-formulas-of* :: $('p, 'f)$ *OLf-state* ⇒ $'f$ *set* **where**
all-formulas-of St ≡ *fset* (*new-of St*) ∪ *set-option* (*xx-of St*) ∪ *elems* (*passive-of St*) ∪
set-option (*yy-of St*) ∪ *fset* (*active-of St*)

fun *fstate* :: $'f$ *fset* × $'f$ *option* × $'p$ × $'f$ *option* × $'f$ *fset* ⇒ $('f$ × *OL-label*$)$ *set* **where**
fstate (N, X, P, Y, A) = *state* (*fset N*, *set-option X*, *elems P*, *set-option Y*, *fset A*)

lemma *fstate-alt-def*:
fstate St =
state (*fset* (*fst St*), *set-option* (*fst* (*snd St*)), *elems* (*fst* (*snd* (*snd St*))),
set-option (*fst* (*snd* (*snd* (*snd St*)))), *fset* (*snd* (*snd* (*snd* (*snd St*)))))
⟨*proof*⟩

definition
Liminf-fstate :: $('p, 'f)$ *OLf-state llist* ⇒ $'f$ *set* × $'f$ *set* × $'f$ *set* × $'f$ *set* × $'f$ *set*
where
Liminf-fstate Sts =
(*Liminf-llist* (*lmap* (*fset* ∘ *new-of*) *Sts*),
Liminf-llist (*lmap* (*set-option* ∘ *xx-of*) *Sts*),
Liminf-llist (*lmap* (*elems* ∘ *passive-of*) *Sts*),
Liminf-llist (*lmap* (*set-option* ∘ *yy-of*) *Sts*),
Liminf-llist (*lmap* (*fset* ∘ *active-of*) *Sts*))

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

fun *state-union* :: $'f$ *set* × $'f$ *set* × $'f$ *set* × $'f$ *set* × $'f$ *set* ⇒ $'f$ *set* **where**
state-union $(N, X, P, Y, A) = N \cup X \cup P \cup Y \cup A$

inductive *fair-OL* :: $('p, 'f)$ *OLf-state* ⇒ $('p, 'f)$ *OLf-state* ⇒ *bool* (**infix** $\rightsquigarrow$*OLf 50*) **where**
choose-n: $C \mid\notin\mid N \Longrightarrow (N \mid\cup\mid \{\mid C \mid\}, None, P, None, A) \rightsquigarrow OLf\ (N, Some\ C, P, None, A)$
| *delete-fwd*: $C \in$ *no-labels.Red-F* $(elems\ P \cup fset\ A) \vee (\exists\, C' \in elems\ P \cup fset\ A.\ C' \preceq\cdot C) \Longrightarrow$
$(N, Some\ C, P, None, A) \rightsquigarrow OLf\ (N, None, P, None, A)$

| *simplify-fwd*: $C' \prec S\ C \Longrightarrow C \in$ *no-labels.Red-F* (*elems* $P \cup$ *fset* $A \cup \{C'\}) \Longrightarrow$
  (N, *Some* C, P, *None*, A) $\rightsquigarrow$*OLf* (N, *Some* C', P, *None*, A)
| *delete-bwd-p*: $C' \in$ *elems* $P \Longrightarrow C' \in$ *no-labels.Red-F* $\{C\} \vee C \prec\cdot C' \Longrightarrow$
  (N, *Some* C, P, *None*, A) $\rightsquigarrow$*OLf* (N, *Some* C, *remove* C' P, *None*, A)
| *simplify-bwd-p*: $C'' \prec S\ C' \Longrightarrow C' \in$ *elems* $P \Longrightarrow C' \in$ *no-labels.Red-F* $\{C, C''\} \Longrightarrow$
  (N, *Some* C, P, *None*, A) $\rightsquigarrow$*OLf* (N |∪| $\{|C''|\}$, *Some* C, *remove* C' P, *None*, A)
| *delete-bwd-a*: $C' \,|\notin|\, A \Longrightarrow C' \in$ *no-labels.Red-F* $\{C\} \vee C \prec\cdot C' \Longrightarrow$
  (N, *Some* C, P, *None*, A |∪| $\{|C'|\}$) $\rightsquigarrow$*OLf* (N, *Some* C, P, *None*, A)
| *simplify-bwd-a*: $C'' \prec S\ C' \Longrightarrow C' \,|\notin|\, A \Longrightarrow C' \in$ *no-labels.Red-F* $\{C, C''\} \Longrightarrow$
  (N, *Some* C, P, *None*, A |∪| $\{|C'|\}$) $\rightsquigarrow$*OLf* (N |∪| $\{|C''|\}$, *Some* C, P, *None*, A)
| *transfer*: (N, *Some* C, P, *None*, A) $\rightsquigarrow$*OLf* (N, *None*, *add* C P, *None*, A)
| *choose-p*: $P \neq$ *empty* $\Longrightarrow$
  ($\{||\}$, *None*, P, *None*, A) $\rightsquigarrow$*OLf* ($\{||\}$, *None*, *remove* (*select* P) P, *Some* (*select* P), A)
| *infer*: *no-labels.Inf-between* (*fset* A) $\{C\} \subseteq$ *no-labels.Red-I* (*fset* $A \cup \{C\} \cup$ *fset* M) $\Longrightarrow$
  ($\{||\}$, *None*, P, *Some* C, A) $\rightsquigarrow$*OLf* (M, *None*, P, *None*, A |∪| $\{|C|\}$)

7.3 Initial State and Invariant

inductive *is-initial-OLf-state* :: ('*p*, '*f*) *OLf-state* ⇒ *bool* **where**
  is-initial-OLf-state (N, *None*, *empty*, *None*, $\{||\}$)

inductive *OLf-invariant* :: ('*p*, '*f*) *OLf-state* ⇒ *bool* **where**
  ($N = \{||\} \wedge X =$ *None*) $\vee$ $Y =$ *None* $\Longrightarrow$ *OLf-invariant* (N, X, P, Y, A)

lemma *initial-OLf-invariant*: *is-initial-OLf-state St* $\Longrightarrow$ *OLf-invariant St*
  ⟨*proof*⟩

lemma *step-OLf-invariant*:
  assumes *step*: *St* $\rightsquigarrow$*OLf St'*
  shows *OLf-invariant St'*
  ⟨*proof*⟩

lemma *chain-OLf-invariant-lnth*:
  assumes
    chain: *chain* ($\rightsquigarrow$*OLf*) *Sts* **and**
    fair-hd: *OLf-invariant* (*lhd Sts*) **and**
    i-lt: *enat i* < *llength Sts*
  shows *OLf-invariant* (*lnth Sts i*)
  ⟨*proof*⟩

lemma *chain-OLf-invariant-llast*:
  assumes
    chain: *chain* ($\rightsquigarrow$*OLf*) *Sts* **and**
    fair-hd: *OLf-invariant* (*lhd Sts*) **and**
    fin: *lfinite Sts*
  shows *OLf-invariant* (*llast Sts*)
⟨*proof*⟩

7.4 Final State

inductive *is-final-OLf-state* :: ('*p*, '*f*) *OLf-state* ⇒ *bool* **where**
  is-final-OLf-state ($\{||\}$, *None*, *empty*, *None*, A)

lemma *is-final-OLf-state-iff-no-OLf-step*:
  assumes *inv*: *OLf-invariant St*
  shows *is-final-OLf-state St* $\longleftrightarrow$ (∀ *St'*. ¬ *St* $\rightsquigarrow$*OLf St'*)

⟨*proof*⟩

7.5 Refinement

lemma *fair-OL-step-imp-OL-step*:
  assumes *olf*: $(N,\ X,\ P,\ Y,\ A) \rightsquigarrow OLf\ (N',\ X',\ P',\ Y',\ A')$
  shows *fstate* $(N,\ X,\ P,\ Y,\ A) \rightsquigarrow OL$ *fstate* $(N',\ X',\ P',\ Y',\ A')$
  ⟨*proof*⟩

lemma *fair-OL-step-imp-GC-step*:
  $(N,\ X,\ P,\ Y,\ A) \rightsquigarrow OLf\ (N',\ X',\ P',\ Y',\ A') \Longrightarrow$
   fstate $(N,\ X,\ P,\ Y,\ A) \rightsquigarrow GC$ *fstate* $(N',\ X',\ P',\ Y',\ A')$
  ⟨*proof*⟩

end

end

8 iProver Loop

The iProver loop is a variant of the Otter loop that supports the elimination of clauses that are made redundant by their own children.

theory *iProver-Loop*
  imports *Otter-Loop*
begin

context *otter-loop*
begin

8.1 Definition

inductive *IL* :: $('f \times$ *OL-label*$)$ *set* $\Rightarrow$ $('f \times$ *OL-label*$)$ *set* $\Rightarrow$ *bool* (**infix** $\rightsquigarrow IL\ 50$)
where
  ol: $St \rightsquigarrow OL\ St' \Longrightarrow St \rightsquigarrow IL\ St'$
| *red-by-children*: $C \in$ *no-labels.Red-F* $(A \cup M) \vee (M = \{C'\} \wedge C' \prec\cdot C) \Longrightarrow$
    state $(\{\},\ \{\},\ P,\ \{C\},\ A) \rightsquigarrow IL$ *state* $(M,\ \{\},\ P,\ \{\},\ A)$

8.2 Refinement

lemma *red-by-children-in-GC*:
  assumes $C \in$ *no-labels.Red-F* $(A \cup M) \vee (M = \{C'\} \wedge C' \prec\cdot C)$
  shows *state* $(\{\},\ \{\},\ P,\ \{C\},\ A) \rightsquigarrow GC$ *state* $(M,\ \{\},\ P,\ \{\},\ A)$
⟨*proof*⟩

theorem *IL-step-imp-GC-step*: $M \rightsquigarrow IL\ M' \Longrightarrow M \rightsquigarrow GC\ M'$
⟨*proof*⟩

8.3 Completeness

theorem
  assumes
    il-chain: *chain* $(\rightsquigarrow IL)$ *Sts* **and**
    act: *active-subset* (*lhd Sts*) $= \{\}$ **and**
    pas: *passive-subset* (*Liminf-llist Sts*) $= \{\}$
  shows

IL-Liminf-saturated: *saturated* (*Liminf-llist Sts*) **and**
IL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *lhd Sts* $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *Liminf-llist Sts* **and**
IL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *lhd Sts* $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists$ *i*. *enat i* $<$ *llength Sts* $\wedge$ ($\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *lnth Sts i*)
⟨*proof*⟩

end

end

9 Fair iProver Loop

The fair iProver loop assumes that the passive queue is fair and ensures (dynamic) refutational completeness under that assumption. From this completeness proof, we also easily derive (in a separate section) the completeness of the Otter loop.

theory *Fair-iProver-Loop*
imports
Given-Clause-Loops-Util
Fair-Otter-Loop-Def
iProver-Loop
begin

9.1 Locale

context *fair-otter-loop*
begin

9.2 Basic Definition

inductive *fair-IL* :: (*'p*, *'f*) *OLf-state* $\Rightarrow$ (*'p*, *'f*) *OLf-state* $\Rightarrow$ *bool* (**infix** $\rightsquigarrow$*ILf 50*) **where**
ol: *St* $\rightsquigarrow$*OLf St'* $\Longrightarrow$ *St* $\rightsquigarrow$*ILf St'*
| *red-by-children*: *C* $\in$ *no-labels.Red-F* (*fset A* $\cup$ *fset M*) $\vee$ *fset M* $= \{C'\} \wedge C' \prec\cdot C \Longrightarrow$
($\{||\}$, *None*, *P*, *Some C*, *A*) $\rightsquigarrow$*ILf* (*M*, *None*, *P*, *None*, *A*)

9.3 Initial State and Invariant

lemma *step-ILf-invariant*:
assumes *St* $\rightsquigarrow$*ILf St'*
shows *OLf-invariant St'*
⟨*proof*⟩

lemma *chain-ILf-invariant-lnth*:
assumes
chain: *chain* ($\rightsquigarrow$*ILf*) *Sts* **and**
fair-hd: *OLf-invariant* (*lhd Sts*) **and**
i-lt: *enat i* $<$ *llength Sts*
shows *OLf-invariant* (*lnth Sts i*)
⟨*proof*⟩

lemma *chain-ILf-invariant-llast*:
assumes
chain: *chain* ($\rightsquigarrow$*ILf*) *Sts* **and**
fair-hd: *OLf-invariant* (*lhd Sts*) **and**

fin: *lfinite Sts*
shows *OLf-invariant* (*llast Sts*)
⟨*proof*⟩

9.4 Final State

lemma *is-final-OLf-state-iff-no-ILf-step*:
assumes *inv*: *OLf-invariant St*
shows *is-final-OLf-state St* ⟷ (∀ *St'*. ¬ *St* ⇝*ILf St'*)
⟨*proof*⟩

9.5 Refinement

lemma *fair-IL-step-imp-IL-step*:
assumes *ilf*: (*N*, *X*, *P*, *Y*, *A*) ⇝*ILf* (*N'*, *X'*, *P'*, *Y'*, *A'*)
shows *fstate* (*N*, *X*, *P*, *Y*, *A*) ⇝*IL fstate* (*N'*, *X'*, *P'*, *Y'*, *A'*)
⟨*proof*⟩

lemma *fair-IL-step-imp-GC-step*:
(*N*, *X*, *P*, *Y*, *A*) ⇝*ILf* (*N'*, *X'*, *P'*, *Y'*, *A'*) ⟹
fstate (*N*, *X*, *P*, *Y*, *A*) ⇝*GC fstate* (*N'*, *X'*, *P'*, *Y'*, *A'*)
⟨*proof*⟩

9.6 Completeness

fun *mset-of-fstate* :: ('*p*, '*f*) *OLf-state* ⇒ '*f multiset* **where**
mset-of-fstate (*N*, *X*, *P*, *Y*, *A*) =
mset-set (*fset N*) + *mset-set* (*set-option X*) + *mset-set* (*elems P*) + *mset-set* (*set-option Y*) +
mset-set (*fset A*)

abbreviation *Precprec-S* :: '*f multiset* ⇒ '*f multiset* ⇒ *bool* (**infix** ≺≺*S* 50) **where**
(≺≺*S*) ≡ *multp* (≺*S*)

lemma *wfP-Precprec-S*: *wfP* (≺≺*S*)
⟨*proof*⟩

definition *Less1-state* :: ('*p*, '*f*) *OLf-state* ⇒ ('*p*, '*f*) *OLf-state* ⇒ *bool* (**infix** ⊏*1* 50) **where**
St' ⊏*1 St* ⟷
mset-of-fstate St' ≺≺*S mset-of-fstate St*
∨ (*mset-of-fstate St'* = *mset-of-fstate St*
∧ (*mset-set* (*fset* (*new-of St'*)) ≺≺*S mset-set* (*fset* (*new-of St*))
∨ (*mset-set* (*fset* (*new-of St'*)) = *mset-set* (*fset* (*new-of St*))
∧ *mset-set* (*set-option* (*xx-of St'*)) ≺≺*S mset-set* (*set-option* (*xx-of St*)))))

lemma *wfP-Less1-state*: *wfP* (⊏*1*)
⟨*proof*⟩

definition *Less2-state* :: ('*p*, '*f*) *OLf-state* ⇒ ('*p*, '*f*) *OLf-state* ⇒ *bool* (**infix** ⊏*2* 50) **where**
St' ⊏*2 St* ≡
mset-set (*set-option* (*yy-of St'*)) ≺≺*S mset-set* (*set-option* (*yy-of St*))
∨ (*mset-set* (*set-option* (*yy-of St'*)) = *mset-set* (*set-option* (*yy-of St*))
∧ *St'* ⊏*1 St*)

lemma *wfP-Less2-state*: *wfP* (⊏*2*)
⟨*proof*⟩

lemma *fair-IL-Liminf-yy-empty*:
assumes
full: *full-chain* (⇝*ILf*) *Sts* **and**
inv: *OLf-invariant* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*set-option* ∘ *yy-of*) *Sts*) = {}
⟨*proof*⟩

lemma *xx-nonempty-OLf-step-imp-Precprec-S*:
assumes
step: *St* ⇝*OLf St'* **and**
xx: *xx-of St* ≠ *None* **and**
xx': *xx-of St'* ≠ *None*
shows *mset-of-fstate St'* ≺≺*S mset-of-fstate St*
⟨*proof*⟩

lemma *xx-nonempty-ILf-step-imp-Precprec-S*:
assumes
step: *St* ⇝*ILf St'* **and**
xx: *xx-of St* ≠ *None* **and**
xx': *xx-of St'* ≠ *None*
shows *mset-of-fstate St'* ≺≺*S mset-of-fstate St*
⟨*proof*⟩

lemma *fair-IL-Liminf-xx-empty*:
assumes
len: *llength Sts* = ∞ **and**
full: *full-chain* (⇝*ILf*) *Sts* **and**
inv: *OLf-invariant* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*set-option* ∘ *xx-of*) *Sts*) = {}
⟨*proof*⟩

lemma *xx-nonempty-OLf-step-imp-Less1-state*:
assumes *step*: (*N*, *Some C*, *P*, *Y*, *A*) ⇝*OLf* (*N'*, *Some C'*, *P'*, *Y'*, *A'*) (**is** *?bef* ⇝*OLf ?aft*)
shows *?aft* ⊏*1 ?bef*
⟨*proof*⟩

lemma *yy-empty-OLf-step-imp-Less1-state*:
assumes
step: *St* ⇝*OLf St'* **and**
yy: *yy-of St* = *None* **and**
yy': *yy-of St'* = *None*
shows *St'* ⊏*1 St*
⟨*proof*⟩

lemma *yy-empty-ILf-step-imp-Less1-state*:
assumes
step: *St* ⇝*ILf St'* **and**
yy: *yy-of St* = *None* **and**
yy': *yy-of St'* = *None*
shows *St'* ⊏*1 St*
⟨*proof*⟩

lemma *fair-IL-Liminf-new-empty*:
assumes
len: *llength Sts* = ∞ **and**

full: *full-chain* ($\rightsquigarrow$*ILf*) *Sts* **and**
inv: *OLf-invariant* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*fset* $\circ$ *new-of*) *Sts*) = {}
⟨*proof*⟩

lemma *yy-empty-OLf-step-imp-Less2-state*:
assumes *step*: (*N*, *X*, *P*, *None*, *A*) $\rightsquigarrow$*OLf* (*N'*, *X'*, *P'*, *None*, *A'*) (**is** *?bef* $\rightsquigarrow$*OLf* *?aft*)
shows *?aft* $\sqsubset$*2* *?bef*
⟨*proof*⟩

lemma *non-choose-p-OLf-step-imp-Less2-state*:
assumes
step: *St* $\rightsquigarrow$*OLf St'* **and**
yy: *yy-of St'* = *None*
shows *St'* $\sqsubset$*2 St*
⟨*proof*⟩

lemma *non-choose-p-ILf-step-imp-Less2-state*:
assumes
step: *St* $\rightsquigarrow$*ILf St'* **and**
yy: *yy-of St'* = *None*
shows *St'* $\sqsubset$*2 St*
⟨*proof*⟩

lemma *OLf-step-imp-queue-step*:
assumes *St* $\rightsquigarrow$*OLf St'*
shows *queue-step* (*passive-of St*) (*passive-of St'*)
⟨*proof*⟩

lemma *ILf-step-imp-queue-step*:
assumes *step*: *St* $\rightsquigarrow$*ILf St'*
shows *queue-step* (*passive-of St*) (*passive-of St'*)
⟨*proof*⟩

lemma *fair-IL-Liminf-passive-empty*:
assumes
len: *llength Sts* = ∞ **and**
full: *full-chain* ($\rightsquigarrow$*ILf*) *Sts* **and**
init: *is-initial-OLf-state* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*elems* $\circ$ *passive-of*) *Sts*) = {}
⟨*proof*⟩

theorem
assumes
full: *full-chain* ($\rightsquigarrow$*ILf*) *Sts* **and**
init: *is-initial-OLf-state* (*lhd Sts*)
shows
fair-IL-Liminf-saturated: *saturated* (*state* (*Liminf-fstate Sts*)) **and**
fair-IL-complete-Liminf: *B* $\in$ *Bot-F* $\Longrightarrow$ *fset* (*new-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists$ *B'* $\in$ *Bot-F*. *B'* $\in$ *state-union* (*Liminf-fstate Sts*) **and**
fair-IL-complete: *B* $\in$ *Bot-F* $\Longrightarrow$ *fset* (*new-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists$ *i*. *enat i* < *llength Sts* $\wedge$ ($\exists$ *B'* $\in$ *Bot-F*. *B'* $\in$ *all-formulas-of* (*lnth Sts i*))
⟨*proof*⟩

end

end

10 Completeness of Fair Otter Loop

The Otter loop is a special case of the iProver loop, with fewer rules. We can therefore reuse the fair iProver loop's completeness result to derive the (dynamic) refutational completeness of the fair Otter loop.

theory *Fair-Otter-Loop-Complete*
imports *Fair-iProver-Loop*
begin

10.1 Completeness

context *fair-otter-loop*
begin

theorem
assumes
full: *full-chain* ($\rightsquigarrow$*OLf*) *Sts* **and**
init: *is-initial-OLf-state* (*lhd Sts*)
shows
fair-OL-Liminf-saturated: *saturated* (*state* (*Liminf-fstate Sts*)) **and**
fair-OL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *fset* (*new-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, B' \in$ *Bot-F*. $B' \in$ *state-union* (*Liminf-fstate Sts*) **and**
fair-OL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *fset* (*new-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
$\exists\, i$. *enat* $i <$ *llength Sts* $\wedge$ ($\exists\, B' \in$ *Bot-F*. $B' \in$ *all-formulas-of* (*lnth Sts* i))
⟨*proof*⟩

end

10.2 Specialization with FIFO Queue

As a proof of concept, we specialize the passive set to use a FIFO queue, thereby eliminating the locale assumptions about the passive set.

locale *fifo-otter-loop* =
otter-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F*
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
entails-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* $\Rightarrow$ *bool* **and**
Inf-G-q :: *'q* $\Rightarrow$ *'g inference set* **and**
Red-I-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g inference set* **and**
Red-F-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-F-q* :: *'q* $\Rightarrow$ *'f* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-I-q* :: *'q* $\Rightarrow$ *'f inference* $\Rightarrow$ *'g inference set option* **and**
Equiv-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\langle\doteq\rangle$ *50*) **and**
Prec-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\langle\prec\cdot\rangle$ *50*) +
fixes
Prec-S :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\prec S$ *50*)
assumes

wf-Prec-S: *minimal-element* ($\prec S$) *UNIV* **and**
transp-Prec-S: *transp* ($\prec S$) **and**
finite-Inf-between: *finite A* $\Longrightarrow$ *finite* (*no-labels.Inf-between A* $\{C\}$)
begin

sublocale *fifo-prover-queue*
⟨*proof*⟩

sublocale *fair-otter-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q*
Equiv-F Prec-F [] *hd* $\lambda y\ xs.$ *if* $y \in$ *set xs then xs else xs* @ [*y*] *removeAll fset-of-list Prec-S*
⟨*proof*⟩

end

end

11 Zipperposition Loop with Ghost State

The Zipperposition loop is a variant of the DISCOUNT loop that can cope with inferences generating (countably) infinitely many conclusions. The version formalized here has an additional ghost component D in its state tuple, which is used in the refinement proof from the abstract procedure LGC.

theory *Zipperposition-Loop*
imports *DISCOUNT-Loop*
begin

context *discount-loop*
begin

11.1 Basic Definitions and Lemmas

fun *flat-inferences-of* :: *'f inference llist multiset* ⇒ *'f inference set* **where**
flat-inferences-of T = $\bigcup$ {*lset* ι |ι. $\iota \in\#$ *T*}

fun
zl-state :: *'f inference llist multiset* × *'f inference set* × *'f set* × *'f set* × *'f set* ⇒
'f inference set × (*'f* × *DL-label*) *set*
where
zl-state (*T*, *D*, *P*, *Y*, *A*) = (*flat-inferences-of T* − *D*, *labeled-formulas-of* (*P*, *Y*, *A*))

lemma *zl-state-alt-def*:
zl-state (*T*, *D*, *P*, *Y*, *A*) =
(*flat-inferences-of T* − *D*, (λ*C*. (*C*, *Passive*)) ‘ *P* ∪ (λ*C*. (*C*, *YY*)) ‘ *Y* ∪ (λ*C*. (*C*, *Active*)) ‘ *A*)
⟨*proof*⟩

inductive
ZL :: *'f inference set* × (*'f* × *DL-label*) *set* ⇒ *'f inference set* × (*'f* × *DL-label*) *set* ⇒ *bool*
(**infix** ⇝*ZL* *50*)
where
compute-infer: $\iota 0 \in$ *no-labels.Red-I* (*A* ∪ {*C*}) ⟹
zl-state (*T* + {#*LCons* $\iota 0$ ιs#}, *D*, *P*, {}, *A*) ⇝*ZL zl-state* (*T* + {#ιs#}, *D* ∪ {$\iota 0$}, *P* ∪ {*C*}, {}, *A*)
| *choose-p*: *zl-state* (*T*, *D*, *P* ∪ {*C*}, {}, *A*) ⇝*ZL zl-state* (*T*, *D*, *P*, {*C*}, *A*)
| *delete-fwd*: *C* ∈ *no-labels.Red-F A* ∨ (∃ *C*′ ∈ *A*. *C*′ ⪯· *C*) ⟹

zl-state $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow ZL$ *zl-state* $(T,\ D,\ P,\ \{\},\ A)$
| *simplify-fwd*: $C \in$ *no-labels.Red-F* $(A \cup \{C'\}) \Longrightarrow$
zl-state $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow ZL$ *zl-state* $(T,\ D,\ P,\ \{C'\},\ A)$
| *delete-bwd*: $C' \in$ *no-labels.Red-F* $\{C\} \vee C' \cdot\succ C \Longrightarrow$
zl-state $(T,\ D,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow ZL$ *zl-state* $(T,\ D,\ P,\ \{C\},\ A)$
| *simplify-bwd*: $C' \in$ *no-labels.Red-F* $\{C,\ C''\} \Longrightarrow$
zl-state $(T,\ D,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow ZL$ *zl-state* $(T,\ D,\ P \cup \{C''\},\ \{C\},\ A)$
| *schedule-infer*: *flat-inferences-of* $T' =$ *no-labels.Inf-between* $A\ \{C\} \Longrightarrow$
zl-state $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow ZL$ *zl-state* $(T + T',\ D -$ *flat-inferences-of* $T',\ P,\ \{\},\ A \cup \{C\})$
| *delete-orphan-infers*: *lset* $\iota s \cap$ *no-labels.Inf-from* $A = \{\} \Longrightarrow$
zl-state $(T + \{\#\iota s\#\},\ D,\ P,\ Y,\ A) \rightsquigarrow ZL$ *zl-state* $(T,\ D \cup$ *lset* $\iota s,\ P,\ Y,\ A)$

11.2 Refinement

lemma *zl-compute-infer-in-lgc*:
assumes $\iota 0 \in$ *no-labels.Red-I* $(A \cup \{C\})$
shows *zl-state* $(T + \{\#LCons\ \iota 0\ \iota s\#\},\ D,\ P,\ \{\},\ A) \rightsquigarrow LGC$
zl-state $(T + \{\#\iota s\#\},\ D \cup \{\iota 0\},\ P \cup \{C\},\ \{\},\ A)$
⟨*proof*⟩

lemma *zl-choose-p-in-lgc*: *zl-state* $(T,\ D,\ P \cup \{C\},\ \{\},\ A) \rightsquigarrow LGC$ *zl-state* $(T,\ D,\ P,\ \{C\},\ A)$
⟨*proof*⟩

lemma *zl-delete-fwd-in-lgc*:
assumes $C \in$ *no-labels.Red-F* $A \vee (\exists\, C' \in A.\ C' \preceq\cdot\ C)$
shows *zl-state* $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow LGC$ *zl-state* $(T,\ D,\ P,\ \{\},\ A)$
⟨*proof*⟩

lemma *zl-simplify-fwd-in-lgc*:
assumes $C \in$ *no-labels.Red-F-*$\mathcal{G}$ $(A \cup \{C'\})$
shows *zl-state* $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow LGC$ *zl-state* $(T,\ D,\ P,\ \{C'\},\ A)$
⟨*proof*⟩

lemma *zl-delete-bwd-in-lgc*:
assumes $C' \in$ *no-labels.Red-F-*$\mathcal{G}$ $\{C\} \vee C' \cdot\succ C$
shows *zl-state* $(T,\ D,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow LGC$ *zl-state* $(T,\ D,\ P,\ \{C\},\ A)$
⟨*proof*⟩

lemma *zl-simplify-bwd-in-lgc*:
assumes $C' \in$ *no-labels.Red-F-*$\mathcal{G}$ $\{C,\ C''\}$
shows *zl-state* $(T,\ D,\ P,\ \{C\},\ A \cup \{C'\}) \rightsquigarrow LGC$ *zl-state* $(T,\ D,\ P \cup \{C''\},\ \{C\},\ A)$
⟨*proof*⟩

lemma *zl-schedule-infer-in-lgc*:
assumes *flat-inferences-of* $T' =$ *no-labels.Inf-between* $A\ \{C\}$
shows *zl-state* $(T,\ D,\ P,\ \{C\},\ A) \rightsquigarrow LGC$
zl-state $(T + T',\ D -$ *flat-inferences-of* $T',\ P,\ \{\},\ A \cup \{C\})$
⟨*proof*⟩

lemma *zl-delete-orphan-infers-in-lgc*:
assumes *inter*: *lset* $\iota s \cap$ *no-labels.Inf-from* $A = \{\}$
shows *zl-state* $(T + \{\#\iota s\#\},\ D,\ P,\ Y,\ A) \rightsquigarrow LGC$ *zl-state* $(T,\ D \cup$ *lset* $\iota s,\ P,\ Y,\ A)$
⟨*proof*⟩

theorem *ZL-step-imp-LGC-step*: $St \rightsquigarrow ZL\ St' \Longrightarrow St \rightsquigarrow LGC\ St'$
⟨*proof*⟩

11.3 Completeness

theorem
  assumes
    zl-chain: *chain* ($\rightsquigarrow$*ZL*) *Sts* **and**
    act: *active-subset* (*snd* (*lhd Sts*)) = {} **and**
    pas: *passive-subset* (*Liminf-llist* (*lmap snd Sts*)) = {} **and**
    no-prems-init: $\forall \iota \in$ *Inf-F*. *prems-of* $\iota = [] \longrightarrow \iota \in$ *fst* (*lhd Sts*) **and**
    final-sched: *Liminf-llist* (*lmap fst Sts*) = {}
  shows
    ZL-Liminf-saturated: *saturated* (*Liminf-llist* (*lmap snd Sts*)) **and**
    ZL-complete-Liminf: *B* $\in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *snd* (*lhd Sts*) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
      $\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *Liminf-llist* (*lmap snd Sts*) **and**
    ZL-complete: *B* $\in$ *Bot-F* $\Longrightarrow$ *fst* ‘ *snd* (*lhd Sts*) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
      $\exists$ *i*. *enat i* < *llength Sts* $\wedge$ ($\exists$ *BL* $\in$ *Bot-FL*. *BL* $\in$ *snd* (*lnth Sts i*))
⟨*proof*⟩

end

end

12 Prover Lazy List Queues and Fairness

This section covers the to-do data structure that arises in the Zipperposition loop.

theory *Prover-Lazy-List-Queue*
  imports *Prover-Queue*
begin

12.1 Basic Lemmas

lemma *ne-and-in-set-take-imp-in-set-take-remove1*:
  assumes
    z $\neq$ *y* **and**
    z $\in$ *set* (*take m xs*)
  shows *z* $\in$ *set* (*take m* (*remove1 y xs*))
  ⟨*proof*⟩

12.2 Locales

locale *prover-lazy-list-queue* =
  fixes
    empty :: *'q* **and**
    add-llist :: *'e llist* $\Rightarrow$ *'q* $\Rightarrow$ *'q* **and**
    remove-llist :: *'e llist* $\Rightarrow$ *'q* $\Rightarrow$ *'q* **and**
    pick-elem :: *'q* $\Rightarrow$ *'e* $\times$ *'q* **and**
    llists :: *'q* $\Rightarrow$ *'e llist multiset*
  assumes
    llists-empty[*simp*]: *llists empty* = {#} **and**
    llists-not-empty: *Q* $\neq$ *empty* $\Longrightarrow$ *llists Q* $\neq$ {#} **and**
    llists-add[*simp*]: *llists* (*add-llist es Q*) = *llists Q* + {#*es*#} **and**
    llist-remove[*simp*]: *llists* (*remove-llist es Q*) = *llists Q* − {#*es*#} **and**
    llists-pick-elem: ($\exists$ *es* $\in$# *llists Q*. *es* $\neq$ *LNil*) $\Longrightarrow$
      $\exists$ *e es*. *LCons e es* $\in$# *llists Q* $\wedge$ *fst* (*pick-elem Q*) = *e*
        $\wedge$ *llists* (*snd* (*pick-elem Q*)) = *llists Q* − {#*LCons e es*#} + {#*es*#}
begin

abbreviation *has-elem* :: *'q ⇒ bool* **where**
has-elem Q ≡ ∃ es ∈# llists Q. es ≠ LNil

inductive *lqueue-step* :: *'q × 'e set ⇒ 'q × 'e set ⇒ bool* **where**
lqueue-step-fold-add-llistI:
lqueue-step (Q, D) (fold add-llist ess Q, D − ⋃ {lset es |es. es ∈ set ess})
| *lqueue-step-fold-remove-llistI*:
lqueue-step (Q, D) (fold remove-llist ess Q, D ∪ ⋃ {lset es |es. es ∈ set ess})
| *lqueue-step-pick-elemI*: *has-elem Q ⟹*
lqueue-step (Q, D) (snd (pick-elem Q), D ∪ {fst (pick-elem Q)})

lemma *lqueue-step-idleI*: *lqueue-step QD QD*
⟨*proof*⟩

lemma *lqueue-step-add-llistI*: *lqueue-step (Q, D) (add-llist es Q, D − lset es)*
⟨*proof*⟩

lemma *lqueue-step-remove-llistI*: *lqueue-step (Q, D) (remove-llist es Q, D ∪ lset es)*
⟨*proof*⟩

lemma *llists-fold-add-llist*[*simp*]: *llists (fold add-llist es Q) = mset es + llists Q*
⟨*proof*⟩

lemma *llists-fold-remove-llist*[*simp*]: *llists (fold remove-llist es Q) = llists Q − mset es*
⟨*proof*⟩

inductive *pick-lqueue-step-w-details* :: *'q × 'e set ⇒ 'e ⇒ 'e llist ⇒ 'q × 'e set ⇒ bool* **where**
pick-lqueue-step-w-detailsI: *LCons e es ∈# llists Q ⟹ fst (pick-elem Q) = e ⟹*
llists (snd (pick-elem Q)) = llists Q − {#LCons e es#} + {#es#} ⟹
pick-lqueue-step-w-details (Q, D) e es (snd (pick-elem Q), D ∪ {e})

inductive *pick-lqueue-step* :: *'q × 'e set ⇒ 'q × 'e set ⇒ bool* **where**
pick-lqueue-stepI: *pick-lqueue-step-w-details QD e es QD′ ⟹ pick-lqueue-step QD QD′*

inductive
remove-lqueue-step-w-details :: *'q × 'e set ⇒ 'e llist list ⇒ 'q × 'e set ⇒ bool*
where
remove-lqueue-step-w-detailsI:
remove-lqueue-step-w-details (Q, D) ess
(fold remove-llist ess Q, D ∪ ⋃ {lset es |es. es ∈ set ess})

end

locale *fair-prover-lazy-list-queue* =
prover-lazy-list-queue empty add-llist remove-llist pick-elem llists
for
empty :: *'q* **and**
add-llist :: *'e llist ⇒ 'q ⇒ 'q* **and**
remove-llist :: *'e llist ⇒ 'q ⇒ 'q* **and**
pick-elem :: *'q ⇒ 'e × 'q* **and**
llists :: *'q ⇒ 'e llist multiset* +
assumes *fair*: *chain lqueue-step QDs ⟹ infinitely-often pick-lqueue-step QDs ⟹*
LCons e es ∈# llists (fst (lnth QDs i)) ⟹
∃ j ≥ i. (∃ ess. LCons e es ∈ set ess

∧ *remove-lqueue-step-w-details* (*lnth QDs j*) *ess* (*lnth QDs* (*Suc j*)))
∨ *pick-lqueue-step-w-details* (*lnth QDs j*) *e es* (*lnth QDs* (*Suc j*))
begin

lemma *fair-strong*:
assumes
chain: *chain lqueue-step QDs* **and**
inf: *infinitely-often pick-lqueue-step QDs* **and**
es-in: *es* ∈# *llists* (*fst* (*lnth QDs i*)) **and**
k-lt: *enat k* < *llength es*
shows ∃ *j* ≥ *i*.
(∃ *k'* ≤ *k*. ∃ *ess*. *ldrop k' es* ∈ *set ess*
∧ *remove-lqueue-step-w-details* (*lnth QDs j*) *ess* (*lnth QDs* (*Suc j*)))
∨ *pick-lqueue-step-w-details* (*lnth QDs j*) (*lnth es k*) (*ldrop* (*Suc k*) *es*) (*lnth QDs* (*Suc j*))
⟨*proof*⟩

end

12.3 Instantiation with FIFO Queue

As a proof of concept, we show that a FIFO queue can serve as a fair prover lazy list queue.

type-synonym *'e fifo* = *nat* × (*'e* × *'e llist*) *list*

locale *fifo-prover-lazy-list-queue*
begin

definition *empty* :: *'e fifo* **where**
empty = (*0*, [])

fun *add-llist* :: *'e llist* ⇒ *'e fifo* ⇒ *'e fifo* **where**
add-llist LNil (*num-nils*, *ps*) = (*num-nils* + *1*, *ps*)
| *add-llist* (*LCons e es*) (*num-nils*, *ps*) = (*num-nils*, *ps* @ [(*e*, *es*)])

fun *remove-llist* :: *'e llist* ⇒ *'e fifo* ⇒ *'e fifo* **where**
remove-llist LNil (*num-nils*, *ps*) = (*num-nils* − *1*, *ps*)
| *remove-llist* (*LCons e es*) (*num-nils*, *ps*) = (*num-nils*, *remove1* (*e*, *es*) *ps*)

fun *pick-elem* :: *'e fifo* ⇒ *'e* × *'e fifo* **where**
pick-elem (-, []) = *undefined*
| *pick-elem* (*num-nils*, (*e*, *es*) # *ps*) =
(*e*,
(*case es of*
LNil ⇒ (*num-nils* + *1*, *ps*)
| *LCons e' es'* ⇒ (*num-nils*, *ps* @ [(*e'*, *es'*)])))

fun *llists* :: *'e fifo* ⇒ *'e llist multiset* **where**
llists (*num-nils*, *ps*) = *replicate-mset num-nils LNil* + *mset* (*map* (λ(*e*, *es*). *LCons e es*) *ps*)

sublocale *prover-lazy-list-queue empty add-llist remove-llist pick-elem llists*
⟨*proof*⟩

sublocale *fair-prover-lazy-list-queue empty add-llist remove-llist pick-elem llists*
⟨*proof*⟩

end

end

13 Fair Zipperposition Loop with Ghosts

theory *Fair-Zipperposition-Loop*
imports
Given-Clause-Loops-Util
Zipperposition-Loop
Prover-Lazy-List-Queue
begin

The fair Zipperposition loop makes assumptions about the scheduled inference queue and the passive clause queue and ensures (dynamic) refutational completeness under these assumptions. This version inherits the ghost state component from the "unfair" version of the loop.

13.1 Locale

type-synonym $('t, 'p, 'f)$ *ZLf-state* $= 't \times 'f$ *inference set* $\times 'p \times 'f$ *option* $\times 'f$ *fset*

locale *fair-zipperposition-loop* =
discount-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F* +
todo: *fair-prover-lazy-list-queue t-empty t-add-llist t-remove-llist t-pick-elem t-llists* +
passive: *fair-prover-queue p-empty p-select p-add p-remove p-felems*
for
Bot-F :: $'f$ *set* **and**
Inf-F :: $'f$ *inference set* **and**
Bot-G :: $'g$ *set* **and**
Q :: $'q$ *set* **and**
entails-q :: $'q \Rightarrow 'g$ *set* $\Rightarrow 'g$ *set* $\Rightarrow$ *bool* **and**
Inf-G-q :: $'q \Rightarrow 'g$ *inference set* **and**
Red-I-q :: $'q \Rightarrow 'g$ *set* $\Rightarrow 'g$ *inference set* **and**
Red-F-q :: $'q \Rightarrow 'g$ *set* $\Rightarrow 'g$ *set* **and**
$\mathcal{G}$*-F-q* :: $'q \Rightarrow 'f \Rightarrow 'g$ *set* **and**
$\mathcal{G}$*-I-q* :: $'q \Rightarrow 'f$ *inference* $\Rightarrow 'g$ *inference set option* **and**
Equiv-F :: $'f \Rightarrow 'f \Rightarrow$ *bool* (**infix** $\doteq$ *50*) **and**
Prec-F :: $'f \Rightarrow 'f \Rightarrow$ *bool* (**infix** $\prec\cdot$ *50*) **and**
t-empty :: $'t$ **and**
t-add-llist :: $'f$ *inference llist* $\Rightarrow 't \Rightarrow 't$ **and**
t-remove-llist :: $'f$ *inference llist* $\Rightarrow 't \Rightarrow 't$ **and**
t-pick-elem :: $'t \Rightarrow 'f$ *inference* $\times 't$ **and**
t-llists :: $'t \Rightarrow 'f$ *inference llist multiset* **and**
p-empty :: $'p$ **and**
p-select :: $'p \Rightarrow 'f$ **and**
p-add :: $'f \Rightarrow 'p \Rightarrow 'p$ **and**
p-remove :: $'f \Rightarrow 'p \Rightarrow 'p$ **and**
p-felems :: $'p \Rightarrow 'f$ *fset* +
fixes
Prec-S :: $'f \Rightarrow 'f \Rightarrow$ *bool* (**infix** $\prec S$ *50*)
assumes
wf-Prec-S: *minimal-element* $(\prec S)$ *UNIV* **and**
transp-Prec-S: *transp* $(\prec S)$ **and**
countable-Inf-between: *finite* $A \Longrightarrow$ *countable* (*no-labels.Inf-between* A $\{C\}$)
begin

lemma *trans-Prec-S*: *trans* $\{(x, y).\ x \prec S\ y\}$
⟨*proof*⟩

lemma *irreflp-Prec-S*: *irreflp* $(\prec S)$
⟨*proof*⟩

lemma *irrefl-Prec-S*: *irrefl* $\{(x, y).\ x \prec S\ y\}$
⟨*proof*⟩

13.2 Basic Definitions and Lemmas

abbreviation *todo-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'t$ **where**
todo-of St ≡ *fst St*
abbreviation *done-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'f$ *inference set* **where**
done-of St ≡ *fst* (*snd St*)
abbreviation *passive-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'p$ **where**
passive-of St ≡ *fst* (*snd* (*snd St*))
abbreviation *yy-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'f$ *option* **where**
yy-of St ≡ *fst* (*snd* (*snd* (*snd St*)))
abbreviation *active-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'f$ *fset* **where**
active-of St ≡ *snd* (*snd* (*snd* (*snd St*)))

abbreviation *all-formulas-of* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'f$ *set* **where**
all-formulas-of St ≡ *passive.elems* (*passive-of St*) ∪ *set-option* (*yy-of St*) ∪ *fset* (*active-of St*)

fun *zl-fstate* :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $'f$ *inference set* × ($'f$ × *DL-label*) *set* **where**
zl-fstate (*T*, *D*, *P*, *Y*, *A*) = *zl-state* (*t-llists T*, *D*, *passive.elems P*, *set-option Y*, *fset A*)

lemma *zl-fstate-alt-def*:
zl-fstate St = *zl-state* (*t-llists* (*fst St*), *fst* (*snd St*), *passive.elems* (*fst* (*snd* (*snd St*))),
set-option (*fst* (*snd* (*snd* (*snd St*)))), *fset* (*snd* (*snd* (*snd* (*snd St*)))))
⟨*proof*⟩

definition
Liminf-zl-fstate :: $('t, 'p, 'f)$ *ZLf-state llist* ⇒ $'f$ *set* × $'f$ *set* × $'f$ *set*
where
Liminf-zl-fstate Sts =
(*Liminf-llist* (*lmap* (*passive.elems* ∘ *passive-of*) *Sts*),
Liminf-llist (*lmap* (*set-option* ∘ *yy-of*) *Sts*),
Liminf-llist (*lmap* (*fset* ∘ *active-of*) *Sts*))

lemma *Liminf-zl-fstate-commute*:
Liminf-llist (*lmap* (*snd* ∘ *zl-fstate*) *Sts*) = *labeled-formulas-of* (*Liminf-zl-fstate Sts*)
⟨*proof*⟩

fun *formulas-union* :: $'f$ *set* × $'f$ *set* × $'f$ *set* ⇒ $'f$ *set* **where**
formulas-union (*P*, *Y*, *A*) = *P* ∪ *Y* ∪ *A*

inductive
fair-ZL :: $('t, 'p, 'f)$ *ZLf-state* ⇒ $('t, 'p, 'f)$ *ZLf-state* ⇒ *bool* (**infix** ⇝*ZLf 50*)
where
compute-infer: (∃ι*s* ∈# *t-llists T*. ι*s* ≠ *LNil*) ⟹ *t-pick-elem T* = (ι*0*, *T'*) ⟹
ι*0* ∈ *no-labels.Red-I* (*fset A* ∪ {*C*}) ⟹
(*T*, *D*, *P*, *None*, *A*) ⇝*ZLf* (*T'*, *D* ∪ {ι*0*}, *p-add C P*, *None*, *A*)
| *choose-p*: *P* ≠ *p-empty* ⟹
(*T*, *D*, *P*, *None*, *A*) ⇝*ZLf* (*T*, *D*, *p-remove* (*p-select P*) *P*, *Some* (*p-select P*), *A*)

| *delete-fwd*: $C \in$ *no-labels.Red-F* (*fset A*) $\vee$ ($\exists\, C' \in$ *fset A*. $C' \preceq\cdot\ C$) $\Longrightarrow$
(T, D, P, *Some* C, A) $\rightsquigarrow$*ZLf* (T, D, P, *None*, A)
| *simplify-fwd*: $C' \prec$*S* $C \Longrightarrow C \in$ *no-labels.Red-F* (*fset A* $\cup$ {C'}) $\Longrightarrow$
(T, D, P, *Some* C, A) $\rightsquigarrow$*ZLf* (T, D, P, *Some* C', A)
| *delete-bwd*: $C' \,|\notin|\, A \Longrightarrow C' \in$ *no-labels.Red-F* {C} $\vee\ C' \cdot\succ C \Longrightarrow$
(T, D, P, *Some* C, A |$\cup$| {|C'|}) $\rightsquigarrow$*ZLf* (T, D, P, *Some* C, A)
| *simplify-bwd*: $C' \,|\notin|\, A \Longrightarrow C'' \prec$*S* $C' \Longrightarrow C' \in$ *no-labels.Red-F* {C, C''} $\Longrightarrow$
(T, D, P, *Some* C, A |$\cup$| {|C'|}) $\rightsquigarrow$*ZLf* (T, D, *p-add* C'' P, *Some* C, A)
| *schedule-infer*: *flat-inferences-of* (*mset* ιss) = *no-labels.Inf-between* (*fset A*) {C} $\Longrightarrow$
(T, D, P, *Some* C, A) $\rightsquigarrow$*ZLf*
(*fold t-add-llist* ιss T, D $-$ *flat-inferences-of* (*mset* ιss), P, *None*, A |$\cup$| {|C|})
| *delete-orphan-infers*: $\iota s \in\#$ *t-llists* $T \Longrightarrow$ *lset* $\iota s \cap$ *no-labels.Inf-from* (*fset A*) = {} $\Longrightarrow$
(T, D, P, Y, A) $\rightsquigarrow$*ZLf* (*t-remove-llist* ιs T, $D \cup$ *lset* ιs, P, Y, A)

inductive *compute-infer-step* :: ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ *bool* **where**
($\exists \iota s \in\#$ *t-llists* T. $\iota s \neq$ *LNil*) $\Longrightarrow$ *t-pick-elem* T = ($\iota 0$, T') $\Longrightarrow$
$\iota 0 \in$ *no-labels.Red-I* (*fset A* $\cup$ {C}) $\Longrightarrow$
compute-infer-step (T, D, P, *None*, A) (T', $D \cup$ {$\iota 0$}, *p-add* C P, *None*, A)

The step below is slightly more general than the corresponding step in ($\rightsquigarrow$*ZLf*), in the way it handles the D component. The extra generality simplifies an argument later, when we erase the D "ghost" component of the state.

inductive *choose-p-step* :: ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ *bool* **where**
$P \neq$ *p-empty* $\Longrightarrow$
choose-p-step (T, D, P, *None*, A) (T, D', *p-remove* (*p-select* P) P, *Some* (*p-select* P), A)

13.3 Initial State and Invariant

inductive *is-initial-ZLf-state* :: ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ *bool* **where**
flat-inferences-of (*mset* ιss) = *no-labels.Inf-from* {} $\Longrightarrow$
is-initial-ZLf-state (*fold t-add-llist* ιss *t-empty*, {}, *p-empty*, *None*, {||})

inductive *ZLf-invariant* :: ('*t*, '*p*, '*f*) *ZLf-state* $\Rightarrow$ *bool* **where**
flat-inferences-of (*t-llists* T) $\subseteq$ *Inf-F* $\Longrightarrow$ *ZLf-invariant* (T, D, P, Y, A)

lemma *initial-ZLf-invariant*:
assumes *is-initial-ZLf-state St*
shows *ZLf-invariant St*
⟨*proof*⟩

lemma *step-ZLf-invariant*:
assumes
inv: *ZLf-invariant St* **and**
step: *St* $\rightsquigarrow$*ZLf St'*
shows *ZLf-invariant St'*
⟨*proof*⟩

lemma *chain-ZLf-invariant-lnth*:
assumes
chain: *chain* ($\rightsquigarrow$*ZLf*) *Sts* **and**
fair-hd: *ZLf-invariant* (*lhd Sts*) **and**
i-lt: *enat i* < *llength Sts*
shows *ZLf-invariant* (*lnth Sts i*)
⟨*proof*⟩

lemma *chain-ZLf-invariant-llast*:
assumes
chain: *chain* (⇝*ZLf*) *Sts* **and**
fair-hd: *ZLf-invariant* (*lhd Sts*) **and**
fin: *lfinite Sts*
shows *ZLf-invariant* (*llast Sts*)
⟨*proof*⟩

13.4 Final State

inductive *is-final-ZLf-state* :: ('*t*, '*p*, '*f*) *ZLf-state* ⇒ *bool* **where**
is-final-ZLf-state (*t-empty*, *D*, *p-empty*, *None*, *A*)

lemma *is-final-ZLf-state-iff-no-ZLf-step*:
assumes *inv*: *ZLf-invariant St*
shows *is-final-ZLf-state St* ⟷ (∀ *St'*. ¬ *St* ⇝*ZLf St'*)
⟨*proof*⟩

13.5 Refinement

lemma *fair-ZL-step-imp-ZL-step*:
assumes *zlf*: (*T*, *D*, *P*, *Y*, *A*) ⇝*ZLf* (*T'*, *D'*, *P'*, *Y'*, *A'*)
shows *zl-fstate* (*T*, *D*, *P*, *Y*, *A*) ⇝*ZL zl-fstate* (*T'*, *D'*, *P'*, *Y'*, *A'*)
⟨*proof*⟩

lemma *fair-ZL-step-imp-GC-step*:
(*T*, *D*, *P*, *Y*, *A*) ⇝*ZLf* (*T'*, *D'*, *P'*, *Y'*, *A'*) ⟹
zl-fstate (*T*, *D*, *P*, *Y*, *A*) ⇝*LGC zl-fstate* (*T'*, *D'*, *P'*, *Y'*, *A'*)
⟨*proof*⟩

13.6 Completeness

fun *mset-of-zl-fstate* :: ('*t*, '*p*, '*f*) *ZLf-state* ⇒ '*f multiset* **where**
mset-of-zl-fstate (*T*, *D*, *P*, *Y*, *A*) =
mset-set (*passive.elems P*) + *mset-set* (*set-option Y*) + *mset-set* (*fset A*)

abbreviation *Precprec-S* :: '*f multiset* ⇒ '*f multiset* ⇒ *bool* (**infix** ≺≺*S* 50) **where**
(≺≺*S*) ≡ *multp* (≺*S*)

lemma *wfP-Precprec-S*: *wfP* (≺≺*S*)
⟨*proof*⟩

definition *Less-state* :: ('*t*, '*p*, '*f*) *ZLf-state* ⇒ ('*t*, '*p*, '*f*) *ZLf-state* ⇒ *bool* (**infix** ⊏ 50)
where
St' ⊏ *St* ⟷
mset-of-zl-fstate St' ≺≺*S mset-of-zl-fstate St*
∨ (*mset-of-zl-fstate St'* = *mset-of-zl-fstate St*
∧ (*mset-set* (*passive.elems* (*passive-of St'*)) ≺≺*S mset-set* (*passive.elems* (*passive-of St*))
∨ (*passive.elems* (*passive-of St'*) = *passive.elems* (*passive-of St*)
∧ (*mset-set* (*set-option* (*yy-of St'*)) ≺≺*S mset-set* (*set-option* (*yy-of St*))
∨ (*mset-set* (*set-option* (*yy-of St'*)) = *mset-set* (*set-option* (*yy-of St*))
∧ *size* (*t-llists* (*todo-of St'*)) < *size* (*t-llists* (*todo-of St*)))))))

lemma *wfP-Less-state*: *wfP* (⊏)
⟨*proof*⟩

lemma *non-compute-infer-ZLf-step-imp-Less-state*:
assumes
step: $St \rightsquigarrow_{ZLf} St'$ **and**
non-ci: $\neg$ *compute-infer-step St St'*
shows $St' \sqsubset St$
⟨*proof*⟩

lemma *yy-nonempty-ZLf-step-imp-Less-state*:
assumes
step: $St \rightsquigarrow_{ZLf} St'$ **and**
yy: *yy-of St* $\neq$ *None*
shows $St' \sqsubset St$
⟨*proof*⟩

lemma *fair-ZL-Liminf-yy-empty*:
assumes
len: *llength Sts* $= \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{ZLf})$ *Sts* **and**
inv: *ZLf-invariant* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (*set-option* $\circ$ *yy-of*) *Sts*) $= \{\}$
⟨*proof*⟩

lemma *ZLf-step-imp-passive-queue-step*:
assumes $St \rightsquigarrow_{ZLf} St'$
shows *passive.queue-step* (*passive-of St*) (*passive-of St'*)
⟨*proof*⟩

lemma *choose-p-step-imp-select-passive-queue-step*:
assumes *choose-p-step St St'*
shows *passive.select-queue-step* (*passive-of St*) (*passive-of St'*)
⟨*proof*⟩

lemma *fair-ZL-Liminf-passive-empty*:
assumes
len: *llength Sts* $= \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{ZLf})$ *Sts* **and**
init: *is-initial-ZLf-state* (*lhd Sts*) **and**
fair: *infinitely-often compute-infer-step Sts* $\longrightarrow$ *infinitely-often choose-p-step Sts*
shows *Liminf-llist* (*lmap* (*passive.elems* $\circ$ *passive-of*) *Sts*) $= \{\}$
⟨*proof*⟩

lemma *ZLf-step-imp-todo-queue-step*:
assumes $St \rightsquigarrow_{ZLf} St'$
shows *todo.lqueue-step* (*todo-of St*, *done-of St*) (*todo-of St'*, *done-of St'*)
⟨*proof*⟩

lemma *fair-ZL-Liminf-todo-empty*:
assumes
len: *llength Sts* $= \infty$ **and**
full: *full-chain* $(\rightsquigarrow_{ZLf})$ *Sts* **and**
init: *is-initial-ZLf-state* (*lhd Sts*)
shows *Liminf-llist* (*lmap* (λSt. *flat-inferences-of* (*t-llists* (*todo-of St*)) $-$ *done-of St*) *Sts*) $=$
$\{\}$
⟨*proof*⟩

theorem
  assumes
    full: *full-chain* ($\rightsquigarrow$*ZLf*) *Sts* **and**
    init: *is-initial-ZLf-state* (*lhd Sts*) **and**
    fair: *infinitely-often compute-infer-step Sts* $\longrightarrow$ *infinitely-often choose-p-step Sts*
  shows
    fair-ZL-Liminf-saturated: *saturated* (*labeled-formulas-of* (*Liminf-zl-fstate Sts*)) **and**
    fair-ZL-complete-Liminf: $B \in$ *Bot-F* $\Longrightarrow$ *passive.elems* (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
      $\exists\, B' \in$ *Bot-F*. $B' \in$ *formulas-union* (*Liminf-zl-fstate Sts*) **and**
    fair-ZL-complete: $B \in$ *Bot-F* $\Longrightarrow$ *passive.elems* (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ $\{B\} \Longrightarrow$
      $\exists\, i$. *enat* $i <$ *llength Sts* $\wedge$ ($\exists\, B' \in$ *Bot-F*. $B' \in$ *all-formulas-of* (*lnth Sts* i))
⟨*proof*⟩

end

end

14 Fair Zipperposition Loop without Ghosts

This version of the fair Zipperposition loop eliminates the ghost state component D, thus confirming that D is indeed a ghost.

theory *Fair-Zipperposition-Loop-without-Ghosts*
  imports *Fair-Zipperposition-Loop*
begin

14.1 Locale

type-synonym ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* = '*t* × '*p* × '*f option* × '*f fset*

locale *fair-zipperposition-loop-wo-ghosts* =
  w-ghosts?: *fair-zipperposition-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q*
    $\mathcal{G}$*-I-q Equiv-F Prec-F t-empty t-add-llist t-remove-llist t-pick-elem t-llists p-empty p-select*
    p-add p-remove p-felems Prec-S
  for
    Bot-F :: '*f set* **and**
    Inf-F :: '*f inference set* **and**
    Bot-G :: '*g set* **and**
    Q :: '*q set* **and**
    entails-q :: '*q* ⇒ '*g set* ⇒ '*g set* ⇒ *bool* **and**
    Inf-G-q :: '*q* ⇒ '*g inference set* **and**
    Red-I-q :: '*q* ⇒ '*g set* ⇒ '*g inference set* **and**
    Red-F-q :: '*q* ⇒ '*g set* ⇒ '*g set* **and**
    $\mathcal{G}$*-F-q* :: '*q* ⇒ '*f* ⇒ '*g set* **and**
    $\mathcal{G}$*-I-q* :: '*q* ⇒ '*f inference* ⇒ '*g inference set option* **and**
    Equiv-F :: '*f* ⇒ '*f* ⇒ *bool* (**infix** $\doteq$ *50*) **and**
    Prec-F :: '*f* ⇒ '*f* ⇒ *bool* (**infix** $\prec\cdot$ *50*) **and**
    t-empty :: '*t* **and**
    t-add-llist :: '*f inference llist* ⇒ '*t* ⇒ '*t* **and**
    t-remove-llist :: '*f inference llist* ⇒ '*t* ⇒ '*t* **and**
    t-pick-elem :: '*t* ⇒ '*f inference* × '*t* **and**
    t-llists :: '*t* ⇒ '*f inference llist multiset* **and**
    p-empty :: '*p* **and**
    p-select :: '*p* ⇒ '*f* **and**
    p-add :: '*f* ⇒ '*p* ⇒ '*p* **and**

p-remove :: $'f \Rightarrow 'p \Rightarrow 'p$ **and**
p-felems :: $'p \Rightarrow 'f\ fset$ **and**
Prec-S :: $'f \Rightarrow 'f \Rightarrow bool$ (**infix** $\prec S\ 50$)
begin

fun *wo-ghosts-of* :: $('t, 'p, 'f)$ *ZLf-state* $\Rightarrow$ $('t, 'p, 'f)$ *ZLf-wo-ghosts-state* **where**
wo-ghosts-of $(T, D, P, Y, A) = (T, P, Y, A)$

inductive
fair-ZL-wo-ghosts ::
$('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ $('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ *bool*
(**infix** $\rightsquigarrow$*ZLfw 50*)
where
compute-infer: $(\exists \iota s \in\# t\text{-}llists\ T.\ \iota s \neq LNil) \Longrightarrow t\text{-}pick\text{-}elem\ T = (\iota 0, T') \Longrightarrow$
$\iota 0 \in no\text{-}labels.Red\text{-}I\ (fset\ A \cup \{C\}) \Longrightarrow$
$(T, P, None, A) \rightsquigarrow ZLfw\ (T', p\text{-}add\ C\ P, None, A)$
| *choose-p*: $P \neq p\text{-}empty \Longrightarrow$
$(T, P, None, A) \rightsquigarrow ZLfw\ (T, p\text{-}remove\ (p\text{-}select\ P)\ P, Some\ (p\text{-}select\ P), A)$
| *delete-fwd*: $C \in no\text{-}labels.Red\text{-}F\ (fset\ A) \vee (\exists C' \in fset\ A.\ C' \preceq\cdot\ C) \Longrightarrow$
$(T, P, Some\ C, A) \rightsquigarrow ZLfw\ (T, P, None, A)$
| *simplify-fwd*: $C' \prec S\ C \Longrightarrow C \in no\text{-}labels.Red\text{-}F\ (fset\ A \cup \{C'\}) \Longrightarrow$
$(T, P, Some\ C, A) \rightsquigarrow ZLfw\ (T, P, Some\ C', A)$
| *delete-bwd*: $C' \mid\notin\mid A \Longrightarrow C' \in no\text{-}labels.Red\text{-}F\ \{C\} \vee C' \cdot\succ C \Longrightarrow$
$(T, P, Some\ C, A \mid\cup\mid \{\mid C'\mid\}) \rightsquigarrow ZLfw\ (T, P, Some\ C, A)$
| *simplify-bwd*: $C' \mid\notin\mid A \Longrightarrow C'' \prec S\ C' \Longrightarrow C' \in no\text{-}labels.Red\text{-}F\ \{C, C''\} \Longrightarrow$
$(T, P, Some\ C, A \mid\cup\mid \{\mid C'\mid\}) \rightsquigarrow ZLfw\ (T, p\text{-}add\ C''\ P, Some\ C, A)$
| *schedule-infer*: $flat\text{-}inferences\text{-}of\ (mset\ \iota ss) = no\text{-}labels.Inf\text{-}between\ (fset\ A)\ \{C\} \Longrightarrow$
$(T, P, Some\ C, A) \rightsquigarrow ZLfw\ (fold\ t\text{-}add\text{-}llist\ \iota ss\ T, P, None, A \mid\cup\mid \{\mid C\mid\})$
| *delete-orphan-infers*: $\iota s \in\# t\text{-}llists\ T \Longrightarrow lset\ \iota s \cap no\text{-}labels.Inf\text{-}from\ (fset\ A) = \{\} \Longrightarrow$
$(T, P, Y, A) \rightsquigarrow ZLfw\ (t\text{-}remove\text{-}llist\ \iota s\ T, P, Y, A)$

inductive
compute-infer-step ::
$('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ $('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ *bool*
where
$(\exists \iota s \in\# t\text{-}llists\ T.\ \iota s \neq LNil) \Longrightarrow t\text{-}pick\text{-}elem\ T = (\iota 0, T') \Longrightarrow$
$\iota 0 \in no\text{-}labels.Red\text{-}I\ (fset\ A \cup \{C\}) \Longrightarrow$
compute-infer-step $(T, P, None, A)\ (T', p\text{-}add\ C\ P, None, A)$

inductive
choose-p-step :: $('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ $('t, 'p, 'f)$ *ZLf-wo-ghosts-state* $\Rightarrow$ *bool*
where
$P \neq p\text{-}empty \Longrightarrow$
choose-p-step $(T, P, None, A)\ (T, p\text{-}remove\ (p\text{-}select\ P)\ P, Some\ (p\text{-}select\ P), A)$

lemma *w-ghosts-compute-infer-step-imp-compute-infer-step*:
assumes *w-ghosts.compute-infer-step St St'*
shows *compute-infer-step* (*wo-ghosts-of St*) (*wo-ghosts-of St'*)
⟨*proof*⟩

lemma *choose-p-step-imp-w-ghosts-choose-p-step*:
assumes *choose-p-step* (*wo-ghosts-of St*) (*wo-ghosts-of St'*)
shows *w-ghosts.choose-p-step St St'*
⟨*proof*⟩

14.2 Basic Definitions and Lemmas

abbreviation *todo-of* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ '*t* **where**
todo-of St ≡ *fst St*
abbreviation *passive-of* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ '*p* **where**
passive-of St ≡ *fst* (*snd St*)
abbreviation *yy-of* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ '*f option* **where**
yy-of St ≡ *fst* (*snd* (*snd St*))
abbreviation *active-of* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ '*f fset* **where**
active-of St ≡ *snd* (*snd* (*snd St*))

abbreviation *all-formulas-of* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ '*f set* **where**
all-formulas-of St ≡ *passive.elems* (*passive-of St*) ∪ *set-option* (*yy-of St*) ∪ *fset* (*active-of St*)

definition
Liminf-zl-fstate :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state llist* ⇒ '*f set* × '*f set* × '*f set*
where
Liminf-zl-fstate Sts =
(*Liminf-llist* (*lmap* (*passive.elems* ∘ *passive-of*) *Sts*),
Liminf-llist (*lmap* (*set-option* ∘ *yy-of*) *Sts*),
Liminf-llist (*lmap* (*fset* ∘ *active-of*) *Sts*))

14.3 Initial States and Invariants

inductive *is-initial-ZLf-wo-ghosts-state* :: ('*t*, '*p*, '*f*) *ZLf-wo-ghosts-state* ⇒ *bool* **where**
flat-inferences-of (*mset ιss*) = *no-labels.Inf-from* {} ⟹
is-initial-ZLf-wo-ghosts-state (*fold t-add-llist ιss t-empty*, *p-empty*, *None*, {||})

lemma *is-initial-ZLf-state-imp-is-initial-ZLf-wo-ghosts-state*:
assumes *is-initial-ZLf-state St*
shows *is-initial-ZLf-wo-ghosts-state* (*wo-ghosts-of St*)
⟨*proof*⟩

lemma *is-initial-ZLf-wo-ghosts-state-imp-is-initial-ZLf-state*:
assumes
init: *is-initial-ZLf-wo-ghosts-state* (*wo-ghosts-of St*) **and**
don: *done-of St* = {}
shows *is-initial-ZLf-state St*
⟨*proof*⟩

end

14.4 Abstract Nonsense for Ghost–Ghostless Conversion

This subsection was originally contributed by Andrei Popescu.

locale *bisim* =
fixes *erase* :: '*state0* ⇒ '*state*
and *R* :: '*state* ⇒ '*state* ⇒ *bool* (**infix** ⇝ *60*)
and *R0* :: '*state0* ⇒ '*state0* ⇒ *bool* (**infix** ⇝*0 60*)
assumes *simul*: ⋀*St0 St'*. *erase St0* ⇝ *St'* ⟹ ∃*St0'*. *erase St0'* = *St'* ∧ *St0* ⇝*0 St0'*
begin

definition *lift* :: '*state0* ⇒ '*state* ⇒ '*state0* **where**
lift St0 St' = (*SOME St0'*. *erase St0'* = *St'* ∧ *St0* ⇝*0 St0'*)

lemma *lift*: *erase St0* ⇝ *St'* ⟹ *erase* (*lift St0 St'*) = *St'* ∧ *St0* ⇝*0 lift St0 St'*

⟨*proof*⟩

lemmas *erase-lift* = *lift*[*THEN conjunct1*]
lemmas *R0-lift* = *lift*[*THEN conjunct2*]

primcorec *theSts0* :: *'state0* ⇒ *'state llist* ⇒ *'state0 llist* **where**
theSts0 St0 Sts =
(*case Sts of*
LNil ⇒ *LCons St0 LNil*
| *LCons St Sts'* ⇒ *LCons St0* (*theSts0* (*lift St0 St*) *Sts'*))

lemma *theSts0-LNil*[*simp*]: *theSts0 St0 LNil* = *LCons St0 LNil*
⟨*proof*⟩

lemma *theSts0-LCons*[*simp*]: *theSts0 St0* (*LCons St Sts'*) = *LCons St0* (*theSts0* (*lift St0 St*) *Sts'*)
⟨*proof*⟩

lemma *simul-chain0*:
assumes *chain*: *lnull Sts* ∨ (*chain* (⇝) *Sts* ∧ *erase St0* ⇝ *lhd Sts*)
shows ∃ *Sts0*. *lhd Sts0* = *St0* ∧ *lmap erase* (*ltl Sts0*) = *Sts* ∧ *chain* (⇝*0*) *Sts0*
⟨*proof*⟩

lemma *simul-chain*:
assumes
chain: *chain* (⇝) *Sts* **and**
hd: *lhd Sts* = *erase St0*
shows ∃ *Sts0*. *lhd Sts0* = *St0* ∧ *lmap erase Sts0* = *Sts* ∧ *chain* (⇝*0*) *Sts0*
⟨*proof*⟩

end

14.5 Ghost–Ghostless Conversions, the Concrete Version

context *fair-zipperposition-loop-wo-ghosts*
begin

lemma
todo-of-wo-ghosts-of[*simp*]: *todo-of* (*wo-ghosts-of St*) = *w-ghosts.todo-of St* **and**
passive-of-wo-ghosts-of[*simp*]: *passive-of* (*wo-ghosts-of St*) = *w-ghosts.passive-of St* **and**
yy-of-wo-ghosts-of[*simp*]: *yy-of* (*wo-ghosts-of St*) = *w-ghosts.yy-of St* **and**
active-of-wo-ghosts-of[*simp*]: *active-of* (*wo-ghosts-of St*) = *w-ghosts.active-of St*
⟨*proof*⟩

lemma *fair-ZL-step-imp-fair-ZL-wo-ghosts-step*:
assumes *St* ⇝*ZLf St'*
shows *wo-ghosts-of St* ⇝*ZLfw wo-ghosts-of St'*
⟨*proof*⟩

lemma *fair-ZL-wo-ghosts-step-imp-fair-ZL-step*:
assumes *wo-ghosts-of St0* ⇝*ZLfw St'*
shows ∃ *St0'*. *wo-ghosts-of St0'* = *St'* ∧ *St0* ⇝*ZLf St0'*
⟨*proof*⟩

interpretation *bisim*: *bisim wo-ghosts-of* (⇝*ZLfw*) (⇝*ZLf*)
⟨*proof*⟩

lemma *chain-fair-ZL-step-wo-ghosts-imp-chain-fair-ZL-step*:
assumes *chain*: *chain* ($\rightsquigarrow$*ZLfw*) *Sts*
shows $\exists$ *Sts0*. *lmap wo-ghosts-of Sts0* = *Sts* $\wedge$ *chain* ($\rightsquigarrow$*ZLf*) *Sts0* $\wedge$ *done-of* (*lhd Sts0*) = {}
⟨*proof*⟩

lemma *full-chain-fair-ZL-step-wo-ghosts-imp-full-chain-fair-ZL-step*:
assumes *full-chain* ($\rightsquigarrow$*ZLfw*) *Sts*
shows $\exists$ *Sts0*. *Sts* = *lmap wo-ghosts-of Sts0* $\wedge$ *full-chain* ($\rightsquigarrow$*ZLf*) *Sts0* $\wedge$ *done-of* (*lhd Sts0*) = {}
⟨*proof*⟩

14.6 Completeness

theorem
assumes
full: *full-chain* ($\rightsquigarrow$*ZLfw*) *Sts* **and**
init: *is-initial-ZLf-wo-ghosts-state* (*lhd Sts*) **and**
fair: *infinitely-often compute-infer-step Sts* $\longrightarrow$ *infinitely-often choose-p-step Sts*
shows
fair-ZL-wo-ghosts-Liminf-saturated: *saturated* (*labeled-formulas-of* (*Liminf-zl-fstate Sts*)) **and**
fair-ZL-wo-ghosts-complete-Liminf: *B* $\in$ *Bot-F* $\Longrightarrow$
passive.elems (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists$ *B'* $\in$ *Bot-F*. *B'* $\in$ *formulas-union* (*Liminf-zl-fstate Sts*) **and**
fair-ZL-wo-ghosts-complete: *B* $\in$ *Bot-F* $\Longrightarrow$ *passive.elems* (*passive-of* (*lhd Sts*)) $\models\cap\mathcal{G}$ {*B*} $\Longrightarrow$
$\exists$ *i*. *enat i* < *llength Sts* $\wedge$ ($\exists$ *B* $\in$ *Bot-F*. *B* $\in$ *all-formulas-of* (*lnth Sts i*))
⟨*proof*⟩

end

14.7 Specialization with FIFO Queue

As a proof of concept, we specialize the passive set to use a FIFO queue, thereby eliminating the locale assumptions about the passive set.

locale *fifo-zipperposition-loop* =
discount-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q Equiv-F Prec-F*
for
Bot-F :: *'f set* **and**
Inf-F :: *'f inference set* **and**
Bot-G :: *'g set* **and**
Q :: *'q set* **and**
entails-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* $\Rightarrow$ *bool* **and**
Inf-G-q :: *'q* $\Rightarrow$ *'g inference set* **and**
Red-I-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g inference set* **and**
Red-F-q :: *'q* $\Rightarrow$ *'g set* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-F-q* :: *'q* $\Rightarrow$ *'f* $\Rightarrow$ *'g set* **and**
$\mathcal{G}$*-I-q* :: *'q* $\Rightarrow$ *'f inference* $\Rightarrow$ *'g inference set option* **and**
Equiv-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** ‹$\doteq$› *50*) **and**
Prec-F :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** ‹$\prec\cdot$› *50*) +
fixes
Prec-S :: *'f* $\Rightarrow$ *'f* $\Rightarrow$ *bool* (**infix** $\prec S$ *50*)
assumes
wf-Prec-S: *minimal-element* ($\prec S$) *UNIV* **and**
transp-Prec-S: *transp* ($\prec S$) **and**
countable-Inf-between: *finite A* $\Longrightarrow$ *countable* (*no-labels.Inf-between A* {*C*})
begin

sublocale *fifo-prover-queue*
⟨*proof*⟩

sublocale *fifo-prover-lazy-list-queue*
⟨*proof*⟩

sublocale *fair-zipperposition-loop Bot-F Inf-F Bot-G Q entails-q Inf-G-q Red-I-q Red-F-q* $\mathcal{G}$*-F-q* $\mathcal{G}$*-I-q*
Equiv-F Prec-F empty add-llist remove-llist pick-elem llists [] *hd*
λy xs. if y ∈ *set xs then xs else xs* @ [*y*] *removeAll fset-of-list Prec-S*
⟨*proof*⟩

end

end

15 Given Clause Loops

This section imports all the theory files of the given clause procedure formalization.

theory *Given-Clause-Loops*
imports
Fair-DISCOUNT-Loop
Fair-Otter-Loop-Complete
Fair-Zipperposition-Loop-without-Ghosts
begin
end