

# Given Clause Loops

Jasmin Blanchette

Qi Qiu

Sophie Tourret

March 17, 2025

## 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

<b>1</b>	<b>Utilities for Given Clause Loops</b>	<b>1</b>
<b>2</b>	<b>More Lemmas about Given Clause Architectures</b>	<b>3</b>
2.1	Inference System . . . . .	3
2.2	Given Clause Procedure Basis . . . . .	3
2.3	Given Clause Procedure . . . . .	4
2.4	Lazy Given Clause Procedure . . . . .	5
<b>3</b>	<b>DISCOUNT Loop</b>	<b>5</b>
3.1	Locale . . . . .	6
3.2	Basic Definitions and Lemmas . . . . .	7
3.3	Refinement . . . . .	8
3.4	Completeness . . . . .	9
<b>4</b>	<b>Prover Queues and Fairness</b>	<b>9</b>
4.1	Basic Lemmas . . . . .	9
4.2	More on Relational Chains over Lazy Lists . . . . .	9
4.3	Locales . . . . .	10
4.4	Instantiation with FIFO Queue . . . . .	11
<b>5</b>	<b>Fair DISCOUNT Loop</b>	<b>12</b>
5.1	Locale . . . . .	12
5.2	Basic Definitions and Lemmas . . . . .	13
5.3	Initial State and Invariant . . . . .	15
5.4	Final State . . . . .	16

5.5	Refinement	16
5.6	Completeness	16
5.7	Specialization with FIFO Queue	18
<b>6</b>	<b>Otter Loop</b>	<b>18</b>
6.1	Basic Definitions and Lemmas	20
6.2	Refinement	21
6.3	Completeness	22
<b>7</b>	<b>Definition of Fair Otter Loop</b>	<b>22</b>
7.1	Locale	22
7.2	Basic Definitions and Lemmas	23
7.3	Initial State and Invariant	24
7.4	Final State	25
7.5	Refinement	25
<b>8</b>	<b>iProver Loop</b>	<b>25</b>
8.1	Definition	25
8.2	Refinement	25
8.3	Completeness	26
<b>9</b>	<b>Fair iProver Loop</b>	<b>26</b>
9.1	Locale	26
9.2	Basic Definition	26
9.3	Initial State and Invariant	26
9.4	Final State	27
9.5	Refinement	27
9.6	Completeness	27
<b>10</b>	<b>Completeness of Fair Otter Loop</b>	<b>30</b>
10.1	Completeness	30
10.2	Specialization with FIFO Queue	30
<b>11</b>	<b>Zipperposition Loop with Ghost State</b>	<b>31</b>
11.1	Basic Definitions and Lemmas	31
11.2	Refinement	32
11.3	Completeness	33
<b>12</b>	<b>Prover Lazy List Queues and Fairness</b>	<b>33</b>
12.1	Basic Lemmas	33
12.2	Locales	33
12.3	Instantiation with FIFO Queue	35
<b>13</b>	<b>Fair Zipperposition Loop with Ghosts</b>	<b>36</b>
13.1	Locale	36
13.2	Basic Definitions and Lemmas	37
13.3	Initial State and Invariant	38
13.4	Final State	39
13.5	Refinement	39
13.6	Completeness	39

<b>14 Fair Zipperposition Loop without Ghosts</b>	<b>41</b>
14.1 Locale . . . . .	41
14.2 Basic Definitions and Lemmas . . . . .	43
14.3 Initial States and Invariants . . . . .	43
14.4 Abstract Nonsense for Ghost–Ghostless Conversion . . . . .	43
14.5 Ghost–Ghostless Conversions, the Concrete Version . . . . .	44
14.6 Completeness . . . . .	45
14.7 Specialization with FIFO Queue . . . . .	45
<b>15 Given Clause Loops</b>	<b>46</b>

## 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  $\exists xs. \text{set } xs = A$ 
  ⟨proof⟩

lemma Union-Setcompr-member-mset-mono:
  assumes sub:  $P \subseteq\# Q$ 
  shows  $\bigcup \{f\ x \mid x. x \in\# P\} \subseteq \bigcup \{f\ x \mid x. x \in\# Q\}$ 
  ⟨proof⟩

lemma singletons-in-mult1:  $(x, y) \in R \implies (\{x\}, \{y\}) \in \text{mult1 } R$ 
  ⟨proof⟩

lemma singletons-in-mult:  $(x, y) \in R \implies (\{x\}, \{y\}) \in \text{mult } R$ 
  ⟨proof⟩

lemma multiset-union-diff-assoc:
  fixes A B C :: 'a multiset
  assumes  $A \cap\# C = \{\#\}$ 
  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 {} = { $\iota \in \text{Inf. prems-of } \iota = []$ }
   $\langle \text{proof} \rangle$ 
```

```
end
```

### 2.2 Given Clause Procedure Basis

```
context given-clause-basis
begin
```

```
lemma no-labels-entails-mono-left:  $M \subseteq N \implies M \models_{\text{NG}} P \implies N \models_{\text{NG}} P$ 
   $\langle \text{proof} \rangle$ 
```

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

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

```
lemma succ-L-imp-Red-F:
  assumes
     $(C', l') \in \mathcal{N}$  and
     $C' \preceq \cdot C$  and
     $l' \sqsubset_L l$ 
  shows  $(C, l) \in \text{Red-F } \mathcal{N}$ 
   $\langle \text{proof} \rangle$ 
```

```
lemma prj-fl-set-to-f-set-distr-union [simp]:  $\text{fst } ' (\mathcal{M} \cup \mathcal{N}) = \text{fst } ' \mathcal{M} \cup \text{fst } ' \mathcal{N}$ 
   $\langle \text{proof} \rangle$ 
```

```
lemma prj-labeledN-eq-N [simp]:  $\text{fst } ' \{(C, l) \mid C. C \in \mathcal{N}\} = \mathcal{N}$ 
   $\langle \text{proof} \rangle$ 
```

```
end
```

## 2.3 Given Clause Procedure

**context** *given-clause*

**begin**

**lemma** *remove-redundant*:

**assumes**  $(C, l) \in \text{Red-F } \mathcal{N}$

**shows**  $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow_{GC} \mathcal{N}$

*<proof>*

**lemma** *remove-redundant-no-label*:

**assumes**  $C \in \text{no-labels.Red-F } (\text{fst } ' \mathcal{N})$

**shows**  $\mathcal{N} \cup \{(C, l)\} \rightsquigarrow_{GC} \mathcal{N}$

*<proof>*

**lemma** *add-inactive*:

**assumes**  $l \neq \text{active}$

**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**

$l' \neq \text{active}$

**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**  $(C, l) \in \text{Red-F } \mathcal{N}$

**shows**  $(T, \mathcal{N} \cup \{(C, l)\}) \rightsquigarrow_{LGC} (T, \mathcal{N})$

*<proof>*

**lemma** *remove-redundant-no-label*:

**assumes**  $C \in \text{no-labels.Red-F } (\text{fst } ' \mathcal{N})$

**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*  $\Rightarrow$  *nat* **where**

*nat-of-DL-label* *Passive* = 2

| *nat-of-DL-label* *YY* = 1

| *nat-of-DL-label* *Active* = 0

**definition**  $DL\text{-}Prec\text{-}L :: DL\text{-}label \Rightarrow DL\text{-}label \Rightarrow bool$  (**infix**  $\langle \sqsubset L \rangle$  50) **where**  
 $DL\text{-}Prec\text{-}L\ l\ l' \longleftrightarrow nat\text{-}of\text{-}DL\text{-}label\ l < nat\text{-}of\text{-}DL\text{-}label\ l'$

**locale**  $discount\text{-}loop = labeled\text{-}lifting\text{-}intersection\ Bot\text{-}F\ Inf\text{-}F\ Bot\text{-}G\ Q\ entails\text{-}q\ Inf\text{-}G\text{-}q\ Red\text{-}I\text{-}q$   
 $Red\text{-}F\text{-}q\ \mathcal{G}\text{-}F\text{-}q\ \mathcal{G}\text{-}I\text{-}q$

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

$Bot\text{-}F :: 'f\ set$   
**and**  $Inf\text{-}F :: 'f\ inference\ set$   
**and**  $Bot\text{-}G :: 'g\ set$   
**and**  $Q :: 'q\ set$   
**and**  $entails\text{-}q :: 'q \Rightarrow 'g\ set \Rightarrow 'g\ set \Rightarrow bool$   
**and**  $Inf\text{-}G\text{-}q :: 'q \Rightarrow 'g\ inference\ set$   
**and**  $Red\text{-}I\text{-}q :: 'q \Rightarrow 'g\ set \Rightarrow 'g\ inference\ set$   
**and**  $Red\text{-}F\text{-}q :: 'q \Rightarrow 'g\ set \Rightarrow 'g\ set$   
**and**  $\mathcal{G}\text{-}F\text{-}q :: 'q \Rightarrow 'f \Rightarrow 'g\ set$   
**and**  $\mathcal{G}\text{-}I\text{-}q :: 'q \Rightarrow 'f\ inference \Rightarrow 'g\ inference\ set\ option$

**+ fixes**

$Equiv\text{-}F :: 'f \Rightarrow 'f \Rightarrow bool$  (**infix**  $\langle \doteq \rangle$  50) **and**  
 $Prec\text{-}F :: 'f \Rightarrow 'f \Rightarrow bool$  (**infix**  $\langle \prec \cdot \rangle$  50)

**assumes**

$equiv\text{-}equiv\text{-}F: equivp\ (\doteq)$  **and**  
 $wf\text{-}prec\text{-}F: wfp\ (\prec \cdot)\ transp\ (\prec \cdot)$  **and**  
 $compat\text{-}equiv\text{-}prec: C1 \doteq D1 \Longrightarrow C2 \doteq D2 \Longrightarrow C1 \prec \cdot C2 \Longrightarrow D1 \prec \cdot D2$  **and**  
 $equiv\text{-}F\text{-}grounding: q \in Q \Longrightarrow C1 \doteq C2 \Longrightarrow \mathcal{G}\text{-}F\text{-}q\ q\ C1 \subseteq \mathcal{G}\text{-}F\text{-}q\ q\ C2$  **and**  
 $prec\text{-}F\text{-}grounding: q \in Q \Longrightarrow C2 \prec \cdot C1 \Longrightarrow \mathcal{G}\text{-}F\text{-}q\ q\ C1 \subseteq \mathcal{G}\text{-}F\text{-}q\ q\ C2$  **and**  
 $static\text{-}ref\text{-}comp: statically\text{-}complete\text{-}calculus\ Bot\text{-}F\ Inf\text{-}F\ (\models \cap \mathcal{G})$   
 $no\text{-}labels.Red\text{-}I\text{-}\mathcal{G}\ no\text{-}labels.Red\text{-}F\text{-}\mathcal{G}\text{-}empty$  **and**  
 $inf\text{-}have\text{-}prems: \iota F \in Inf\text{-}F \Longrightarrow prems\text{-}of\ \iota F \neq []$

**begin**

**lemma**  $po\text{-}DL\text{-}Prec\text{-}L: transp\ (\sqsubset L)\ asymp\ (\sqsubset L)$   
 $\langle proof \rangle$

**lemma**  $wfp\text{-}DL\text{-}Prec\text{-}L: wfp\ (\sqsubset L)$   
 $\langle proof \rangle$

**lemma**  $Active\text{-}minimal: l2 \neq Active \Longrightarrow Active\ \sqsubset L\ l2$   
 $\langle proof \rangle$

**lemma**  $at\text{-}least\text{-}two\text{-}labels: \exists l2. Active\ \sqsubset L\ l2$   
 $\langle proof \rangle$

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

**notation**  $lgc.step$  (**infix**  $\langle \rightsquigarrow LGC \rangle$  50)

### 3.2 Basic Definitions and Lemmas

**abbreviation**  $c\text{-}dot\text{-}succ :: 'f \Rightarrow 'f \Rightarrow bool$  (**infix**  $\langle \cdot \succ \rangle$  50) **where**  
 $C \cdot \succ C' \equiv C' \prec \cdot C$

**abbreviation**  $sqsupset :: DL\text{-}label \Rightarrow DL\text{-}label \Rightarrow bool$  (**infix**  $\langle \sqsubset L \rangle$  50) **where**  
 $l \sqsubset L\ l' \equiv l' \sqsubset L\ l$



**fun** *labeled-formulas-of* :: 'f set × 'f set × 'f set ⇒ ('f × DL-label) set **where**  
*labeled-formulas-of* (P, Y, A) = {(C, Passive) | C. C ∈ P} ∪ {(C, YY) | C. C ∈ Y} ∪  
{(C, Active) | C. C ∈ A}

**lemma** *labeled-formulas-of-alt-def*:

*labeled-formulas-of* (P, Y, A) =  
(λC. (C, Passive)) ' P ∪ (λC. (C, YY)) ' Y ∪ (λC. (C, Active)) ' A  
⟨proof⟩

**fun**

*state* :: 'f inference set × 'f set × 'f set × 'f set ⇒ 'f inference set × ('f × DL-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, (λC. (C, Passive)) ' P ∪ (λC. (C, YY)) ' Y ∪ (λC. (C, Active)) ' A)  
⟨proof⟩

**inductive**

*DL* :: 'f inference set × ('f × DL-label) set ⇒ 'f inference set × ('f × DL-label) set ⇒ bool  
(**infix** <~DL> 50)

**where**

*compute-infer*:  $\iota \in \text{no-labels.Red-I } (A \cup \{C\}) \implies$   
*state* (T ∪ {ι}, P, {}, A) ~DL *state* (T, P, {C}, A)  
| *choose-p*: *state* (T, P ∪ {C}, {}, A) ~DL *state* (T, P, {C}, A)  
| *delete-fwd*:  $C \in \text{no-labels.Red-F } A \vee (\exists C' \in A. C' \preceq C) \implies$   
*state* (T, P, {C}, A) ~DL *state* (T, P, {}, A)  
| *simplify-fwd*:  $C \in \text{no-labels.Red-F } (A \cup \{C'\}) \implies$   
*state* (T, P, {C}, A) ~DL *state* (T, P, {C'}, A)  
| *delete-bwd*:  $C' \in \text{no-labels.Red-F } \{C\} \vee C' \succ C \implies$   
*state* (T, P, {C}, A ∪ {C'}) ~DL *state* (T, P, {C}, A)  
| *simplify-bwd*:  $C' \in \text{no-labels.Red-F } \{C, C''\} \implies$   
*state* (T, P, {C}, A ∪ {C'}) ~DL *state* (T, P ∪ {C''}, {C}, A)  
| *schedule-infer*:  $T' = \text{no-labels.Inf-between } A \{C\} \implies$   
*state* (T, P, {C}, A) ~DL *state* (T ∪ T', P, {}, A ∪ {C})  
| *delete-orphan-infers*:  $T' \cap \text{no-labels.Inf-from } A = \{\} \implies$   
*state* (T ∪ T', P, Y, A) ~DL *state* (T, P, Y, A)

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

**lemma** *PYA-add-passive-formula[simp]*:

*labeled-formulas-of* (P, Y, A) ∪ {(C, Passive)} = *labeled-formulas-of* (P ∪ {C}, Y, A)  
⟨proof⟩

**lemma** *P0A-add-y-formula[simp]*:

*labeled-formulas-of* (P, {}, A) ∪ {(C, YY)} = *labeled-formulas-of* (P, {C}, A)  
⟨proof⟩

**lemma** *PYA-add-active-formula[simp]*:

*labeled-formulas-of* (P, Y, A) ∪ {(C', Active)} = *labeled-formulas-of* (P, Y, A ∪ {C'})  
⟨proof⟩

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

**lemma** *active-subset-of-setOfFormulasWithLabelDiffActive:*

$l \neq \text{Active} \implies \text{active-subset } \{(C', l)\} = \{\}$

*<proof>*

### 3.3 Refinement

**lemma** *dl-compute-infer-in-lgc:*

**assumes**  $\iota \in \text{no-labels.Red-I}\mathcal{G} (A \cup \{C\})$

**shows**  $\text{state } (T \cup \{\iota\}, P, \{\}, A) \rightsquigarrow \text{LGC state } (T, P, \{C\}, A)$

*<proof>*

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

*<proof>*

**lemma** *dl-delete-fwd-in-lgc:*

**assumes**  $(C \in \text{no-labels.Red-F } A) \vee (\exists C' \in A. C' \preceq C)$

**shows**  $\text{state } (T, P, \{C\}, A) \rightsquigarrow \text{LGC state } (T, P, \{\}, A)$

*<proof>*

**lemma** *dl-simplify-fwd-in-lgc:*

**assumes**  $C \in \text{no-labels.Red-F}\mathcal{G} (A \cup \{C'\})$

**shows**  $\text{state } (T, P, \{C\}, A) \rightsquigarrow \text{LGC state } (T, P, \{C'\}, A)$

*<proof>*

**lemma** *dl-delete-bwd-in-lgc:*

**assumes**  $C' \in \text{no-labels.Red-F}\mathcal{G} \{C\} \vee C' \succ C$

**shows**  $\text{state } (T, P, \{C\}, A \cup \{C'\}) \rightsquigarrow \text{LGC state } (T, P, \{C\}, A)$

*<proof>*

**lemma** *dl-simplify-bwd-in-lgc:*

**assumes**  $C' \in \text{no-labels.Red-F}\mathcal{G} \{C, C''\}$

**shows**  $\text{state } (T, P, \{C\}, A \cup \{C'\}) \rightsquigarrow \text{LGC state } (T, P \cup \{C''\}, \{C\}, A)$

*<proof>*

**lemma** *dl-schedule-infer-in-lgc:*

**assumes**  $T' = \text{no-labels.Inf-between } A \{C\}$

**shows**  $\text{state } (T, P, \{C\}, A) \rightsquigarrow \text{LGC state } (T \cup T', P, \{\}, A \cup \{C\})$

*<proof>*

**lemma** *dl-delete-orphan-infers-in-lgc:*

**assumes**  $T' \cap \text{no-labels.Inf-from } A = \{\}$

**shows**  $\text{state } (T \cup T', P, Y, A) \rightsquigarrow \text{LGC state } (T, P, Y, A)$

*<proof>*

**theorem** *DL-step-imp-LGC-step:*  $TM \rightsquigarrow \text{DL } TM' \implies TM \rightsquigarrow \text{LGC } TM'$

*<proof>*

### 3.4 Completeness

**theorem**

**assumes**

*dl-chain:*  $\text{chain } (\rightsquigarrow \text{DL}) \text{ Sts}$  **and**

*act:*  $\text{active-subset } (\text{snd } (\text{lhd } \text{Sts})) = \{\}$  **and**

*pas:*  $\text{passive-subset } (\text{Liminf-llist } (\text{lmap } \text{snd } \text{Sts})) = \{\}$  **and**

*no-prems-init:*  $\forall \iota \in \text{Inf-F. } \text{prems-of } \iota = [] \longrightarrow \iota \in \text{fst } (\text{lhd } \text{Sts})$  **and**

```

    final-sched: Liminf-llist (lmap fst Sts) = {}
shows
    DL-Liminf-saturated: saturated (Liminf-llist (lmap snd Sts)) and
    DL-complete-Liminf: B ∈ Bot-F ⇒ fst ‘ snd (lhd Sts) ⊨∩G {B} ⇒
      ∃ BL ∈ Bot-FL. BL ∈ Liminf-llist (lmap snd Sts) and
    DL-complete: B ∈ Bot-F ⇒ fst ‘ snd (lhd Sts) ⊨∩G {B} ⇒
      ∃ i. enat i < llength Sts ∧ (∃ BL ∈ Bot-FL. BL ∈ 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 (λy xs. if y ∈ set xs then xs else xs @ [y]) ys xs)) ⊆ set (drop k (xs @ ys))
⟨proof⟩

```

```

lemma fold-maybe-append-removeAll:
assumes y ∈ set xs
shows fold (λy xs. if y ∈ set xs then xs else xs @ [y]) (removeAll y ys) xs =
  fold (λy xs. if y ∈ set xs then xs else xs @ [y]) ys xs
⟨proof⟩

```

### 4.2 More on Relational Chains over Lazy Lists

```

definition finitely-often :: ('a ⇒ 'a ⇒ bool) ⇒ 'a llist ⇒ bool where
  finitely-often R xs ←→
  (∃ i. ∀ j. i ≤ j → enat (Suc j) < llength xs → ¬ R (lnth xs j) (lnth xs (Suc j)))

```

```

abbreviation infinitely-often :: ('a ⇒ 'a ⇒ bool) ⇒ 'a llist ⇒ bool where
  infinitely-often R xs ≡ ¬ finitely-often R xs

```

```

lemma infinitely-often-alt-def:
  infinitely-often R xs ←→
  (∀ i. ∃ j. i ≤ j ∧ enat (Suc j) < llength xs ∧ R (lnth xs j) (lnth xs (Suc j)))
⟨proof⟩

```

```

lemma infinitely-often-lifting:
assumes
  r-imp-s: ∀ x x'. R (f x) (f x') → 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  $\Rightarrow$  'e **and**  
*add* :: 'e  $\Rightarrow$  'q  $\Rightarrow$  'q **and**  
*remove* :: 'e  $\Rightarrow$  'q  $\Rightarrow$  'q **and**  
*felems* :: 'q  $\Rightarrow$  'e fset

**assumes**

*felems-empty[simp]*: *felems empty* =  $\{\}$  **and**  
*felems-not-empty*:  $Q \neq \text{empty} \Longrightarrow \text{felems } Q \neq \{\}$  **and**  
*select-in-felems[simp]*:  $Q \neq \text{empty} \Longrightarrow \text{select } Q \in \text{felems } Q$  **and**  
*felems-add[simp]*: *felems (add e Q)* =  $\{e\} \cup \text{felems } Q$  **and**  
*felems-remove[simp]*: *felems (remove e Q)* = *felems Q*  $- \{e\}$  **and**  
*add-again*:  $e \in \text{felems } Q \Longrightarrow \text{add } e \text{ } Q = Q$

**begin**

**abbreviation** *elems* :: 'q  $\Rightarrow$  'e set **where**

*elems Q*  $\equiv$  fset (*felems Q*)

**lemma** *elems-empty*: *elems empty* =  $\{\}$

*<proof>*

**lemma** *formula-not-empty[simp]*:  $Q \neq \text{empty} \Longrightarrow \text{elems } Q \neq \{\}$

*<proof>*

**lemma**

*elems-add*: *elems (add e Q)* =  $\{e\} \cup \text{elems } Q$  **and**  
*elems-remove*: *elems (remove e Q)* = *elems Q*  $- \{e\}$   
*<proof>*

**lemma** *elems-fold-add[simp]*: *elems (fold add es Q)* = *set es*  $\cup$  *elems Q*

*<proof>*

**lemma** *elems-fold-remove[simp]*: *elems (fold remove es Q)* = *elems Q*  $-$  *set es*

*<proof>*

**inductive** *queue-step* :: 'q  $\Rightarrow$  'q  $\Rightarrow$  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  $\Rightarrow$  'q  $\Rightarrow$  bool **where**

*select-queue-stepI*:  $Q \neq \text{empty} \Longrightarrow \text{select-queue-step } Q (\text{remove } (\text{select } Q) \text{ } Q)$

end

**locale** *fair-prover-queue* = *prover-queue empty select add remove felems*

**for**

*empty* :: 'q **and**

*select* :: 'q  $\Rightarrow$  'e **and**

*add* :: 'e  $\Rightarrow$  'q  $\Rightarrow$  'q **and**

*remove* :: 'e  $\Rightarrow$  'q  $\Rightarrow$  'q **and**

*felems* :: 'q  $\Rightarrow$  'e fset +

**assumes** *fair*: *chain queue-step Qs  $\Rightarrow$  infinitely-often select-queue-step Qs  $\Rightarrow$*

*lhd Qs = empty  $\Rightarrow$  Liminf-list (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  $\lambda y xs$ . if  $y \in \text{set } xs$  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  $\lambda y xs$ . if  $y \in \text{set } xs$  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 G-F-q 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**  
*G-F-q* :: 'q ⇒ 'f ⇒ 'g set **and**  
*G-I-q* :: 'q ⇒ 'f *inference* ⇒ 'g *inference* set option **and**  
*Equiv-F* :: 'f ⇒ 'f ⇒ bool (**infix** <=> 50) **and**  
*Prec-F* :: 'f ⇒ 'f ⇒ bool (**infix** << 50) **and**  
*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**

*wfp-Prec-S*: *wfp* (<S) **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*:  $\text{irrefl } \{(x, y). x \prec_S y\}$   
 ⟨proof⟩

## 5.2 Basic Definitions and Lemmas

**abbreviation** *passive-of* :: ('p, 'f) DLf-state  $\Rightarrow$  'p **where**  
*passive-of* St  $\equiv$  fst St

**abbreviation** *yy-of* :: ('p, 'f) DLf-state  $\Rightarrow$  'f option **where**  
*yy-of* St  $\equiv$  fst (snd St)

**abbreviation** *active-of* :: ('p, 'f) DLf-state  $\Rightarrow$  'f fset **where**  
*active-of* St  $\equiv$  snd (snd St)

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

**definition** *passive-formulas-of* :: 'p  $\Rightarrow$  '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  $\Rightarrow$  'f set **where**  
*all-formulas-of* St  $\equiv$  passive-formulas-of (passive-of St)  $\cup$  set-option (yy-of St)  $\cup$   
 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  $\iota$ ) P) = { $\iota$ }  $\cup$  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  $\circ$  Passive-Inference)  $\iota$ s P) = passive-inferences-of P  $\cup$  set  $\iota$ s  
 ⟨proof⟩

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

**lemma** *passive-inferences-of-remove-Passive-Inference[simp]*:  
 passive-inferences-of (remove (Passive-Inference  $\iota$ ) 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  $\circ$  Passive-Inference)  $\iota$ s P) = passive-inferences-of P - set  $\iota$ s

⟨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) ιs 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) ιs 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*:  $l\text{length } Sts = \infty$  **and**  
*full*: *full-chain*  $(\rightsquigarrow DLf) Sts$  **and**  
*inv*: *DLf-invariant*  $(lhd\ Sts)$   
**shows**  $Liminf\text{-l}list\ (lmap\ (set\text{-option} \circ yy\text{-of})\ Sts) = \{\}$   
 ⟨*proof*⟩

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

**lemma** *fair-DL-Liminf-passive-empty*:  
**assumes**  
*len*:  $l\text{length } Sts = \infty$  **and**  
*full*: *full-chain*  $(\rightsquigarrow DLf) Sts$  **and**  
*init*: *is-initial-DLf-state*  $(lhd\ Sts)$   
**shows**  $Liminf\text{-l}list\ (lmap\ (elems \circ passive\text{-of})\ Sts) = \{\}$   
 ⟨*proof*⟩

**lemma** *fair-DL-Liminf-passive-formulas-empty*:  
**assumes**  
*len*:  $l\text{length } Sts = \infty$  **and**  
*full*: *full-chain*  $(\rightsquigarrow DLf) Sts$  **and**  
*init*: *is-initial-DLf-state*  $(lhd\ Sts)$   
**shows**  $Liminf\text{-l}list\ (lmap\ (passive\text{-formulas-of} \circ passive\text{-of})\ Sts) = \{\}$   
 ⟨*proof*⟩

**lemma** *fair-DL-Liminf-passive-inferences-empty*:  
**assumes**  
*len*:  $l\text{length } Sts = \infty$  **and**  
*full*: *full-chain*  $(\rightsquigarrow DLf) Sts$  **and**  
*init*: *is-initial-DLf-state*  $(lhd\ Sts)$   
**shows**  $Liminf\text{-l}list\ (lmap\ (passive\text{-inferences-of} \circ passive\text{-of})\ Sts) = \{\}$   
 ⟨*proof*⟩

**theorem**  
**assumes**  
*full*: *full-chain*  $(\rightsquigarrow DLf) Sts$  **and**  
*init*: *is-initial-DLf-state*  $(lhd\ Sts)$   
**shows**  
*fair-DL-Liminf-saturated*: *saturated*  $(labeled\text{-formulas-of } (Liminf\text{-fstate } Sts))$  **and**  
*fair-DL-complete-Liminf*:  $B \in Bot\text{-}F \implies passive\text{-formulas-of } (passive\text{-of } (lhd\ Sts)) \models_{\cap\mathcal{G}} \{B\} \implies$   
 $\exists B' \in Bot\text{-}F. B' \in formulas\text{-union } (Liminf\text{-fstate } Sts)$  **and**  
*fair-DL-complete*:  $B \in Bot\text{-}F \implies passive\text{-formulas-of } (passive\text{-of } (lhd\ Sts)) \models_{\cap\mathcal{G}} \{B\} \implies$   
 $\exists i. enat\ i < l\text{length } Sts \wedge (\exists B' \in Bot\text{-}F. B' \in all\text{-formulas-of } (lth\ Sts\ i))$   
 ⟨*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 G-F-q 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
  G-F-q :: 'q  $\Rightarrow$  'f  $\Rightarrow$  'g set and
  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  $\langle \prec S \rangle$  50)
assumes
  wfp-Prec-S: wfp ( $\prec S$ ) and
  transp-Prec-S: transp ( $\prec S$ ) and
  finite-Inf-between: finite A  $\implies$  finite (no-labels.Inf-between A {C})
begin

sublocale fifo-prover-queue
   $\langle$ proof $\rangle$ 

sublocale fair-discount-loop 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 [] hd  $\lambda y$  xs. if  $y \in$  set xs then xs else xs @ [y] removeAll fset-of-list Prec-S
   $\langle$ proof $\rangle$ 

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*  $\Rightarrow$  *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*  $\Rightarrow$  *OL-label*  $\Rightarrow$  *bool* (**infix**  $\langle \sqsubset L \rangle$  50) **where**  
*OL-Prec-L* *l l'*  $\longleftrightarrow$  *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*  
{ $\iota_{FL}$  :: (*f*  $\times$  *OL-label*) *inference*. *Infer* (*map fst* (*prems-of*  $\iota_{FL}$ )) (*fst* (*concl-of*  $\iota_{FL}$ ))  $\in$  *Inf-F*}  
**for**  
*Bot-F* :: '*f* set  
**and** *Inf-F* :: '*f* *inference* set  
**and** *Bot-G* :: '*g* set  
**and** *Q* :: '*q* set  
**and** *entails-q* :: '*q*  $\Rightarrow$  '*g* set  $\Rightarrow$  '*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** *G-F-q* :: '*q*  $\Rightarrow$  '*f*  $\Rightarrow$  '*g* set  
**and** *G-I-q* :: '*q*  $\Rightarrow$  '*f* *inference*  $\Rightarrow$  '*g* *inference* set option  
+ **fixes**  
*Equiv-F* :: '*f*  $\Rightarrow$  '*f*  $\Rightarrow$  *bool* (**infix**  $\langle \doteq \rangle$  50) **and**  
*Prec-F* :: '*f*  $\Rightarrow$  '*f*  $\Rightarrow$  *bool* (**infix**  $\langle \prec \cdot \rangle$  50)  
**assumes**  
*equiv-equiv-F*: *equivp* ( $\doteq$ ) **and**  
*wfp-prec-F*: *wfp* ( $\prec \cdot$ ) *transp* ( $\prec \cdot$ ) **and**  
*compat-equiv-prec*: *C1*  $\doteq$  *D1*  $\Longrightarrow$  *C2*  $\doteq$  *D2*  $\Longrightarrow$  *C1*  $\prec \cdot$  *C2*  $\Longrightarrow$  *D1*  $\prec \cdot$  *D2* **and**  
*equiv-F-grounding*: *q*  $\in$  *Q*  $\Longrightarrow$  *C1*  $\doteq$  *C2*  $\Longrightarrow$  *G-F-q* *q* *C1*  $\subseteq$  *G-F-q* *q* *C2* **and**  
*prec-F-grounding*: *q*  $\in$  *Q*  $\Longrightarrow$  *C2*  $\prec \cdot$  *C1*  $\Longrightarrow$  *G-F-q* *q* *C1*  $\subseteq$  *G-F-q* *q* *C2* **and**  
*static-ref-comp*: *statically-complete-calculus* *Bot-F* *Inf-F* ( $\models \cap \mathcal{G}$ )  
*no-labels.Red-I-G* *no-labels.Red-F-G-empty* **and**  
*inf-have-prems*:  $\iota F \in$  *Inf-F*  $\Longrightarrow$  *prems-of*  $\iota F \neq \square$

**begin**

**lemma** *transp-OL-Prec-L*: *transp* ( $\sqsubset L$ )  
 $\langle$ *proof* $\rangle$

**lemma** *wfp-OL-Prec-L*: *wfp* ( $\sqsubset L$ )  
 $\langle$ *proof* $\rangle$

**lemma** *Active-minimal*: *l2*  $\neq$  *Active*  $\Longrightarrow$  *Active*  $\sqsubset L$  *l2*  
 $\langle$ *proof* $\rangle$

**lemma** *at-least-two-labels*:  $\exists$  *l2*. *Active*  $\sqsubset L$  *l2*  
 $\langle$ *proof* $\rangle$

**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*  
 $\langle$ *proof* $\rangle$

**notation** *gc.step* (**infix**  $\langle \rightsquigarrow GC \rangle$  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) =  
 (λC. (C, New)) ' N ∪ (λC. (C, XX)) ' X ∪ (λC. (C, Passive)) ' P ∪ (λC. (C, YY)) ' Y ∪  
 (λC. (C, Active)) ' A  
 ⟨proof⟩

**inductive** *OL* :: ('f × OL-label) set ⇒ ('f × OL-label) set ⇒ bool (**infix** <~OL> 50) **where**

*choose-n*:  $C \notin N \implies \text{state } (N \cup \{C\}, \{\}, P, \{\}, A) \sim_{OL} \text{state } (N, \{C\}, P, \{\}, A)$   
| *delete-fwd*:  $C \in \text{no-labels.Red-F } (P \cup A) \vee (\exists C' \in P \cup A. C' \preceq C) \implies$   
 $\text{state } (N, \{C\}, P, \{\}, A) \sim_{OL} \text{state } (N, \{\}, P, \{\}, A)$   
| *simplify-fwd*:  $C \in \text{no-labels.Red-F } (P \cup A \cup \{C'\}) \implies$   
 $\text{state } (N, \{C\}, P, \{\}, A) \sim_{OL} \text{state } (N, \{C'\}, P, \{\}, A)$   
| *delete-bwd-p*:  $C' \in \text{no-labels.Red-F } \{C\} \vee C \prec C' \implies$   
 $\text{state } (N, \{C\}, P \cup \{C'\}, \{\}, A) \sim_{OL} \text{state } (N, \{C\}, P, \{\}, A)$   
| *simplify-bwd-p*:  $C' \in \text{no-labels.Red-F } \{C, C''\} \implies$   
 $\text{state } (N, \{C\}, P \cup \{C'\}, \{\}, A) \sim_{OL} \text{state } (N \cup \{C''\}, \{C\}, P, \{\}, A)$   
| *delete-bwd-a*:  $C' \in \text{no-labels.Red-F } \{C\} \vee C \prec C' \implies$   
 $\text{state } (N, \{C\}, P, \{\}, A \cup \{C'\}) \sim_{OL} \text{state } (N, \{C\}, P, \{\}, A)$   
| *simplify-bwd-a*:  $C' \in \text{no-labels.Red-F } (\{C, C''\}) \implies$   
 $\text{state } (N, \{C\}, P, \{\}, A \cup \{C'\}) \sim_{OL} \text{state } (N \cup \{C''\}, \{C\}, P, \{\}, A)$   
| *transfer*:  $\text{state } (N, \{C\}, P, \{\}, A) \sim_{OL} \text{state } (N, \{\}, P \cup \{C\}, \{\}, A)$   
| *choose-p*:  $C \notin P \implies \text{state } (\{\}, \{\}, P \cup \{C\}, \{\}, A) \sim_{OL} \text{state } (\{\}, \{\}, P, \{C\}, A)$   
| *infer*:  $\text{no-labels.Inf-between } A \{C\} \subseteq \text{no-labels.Red-I } (A \cup \{C\} \cup M) \implies$   
 $\text{state } (\{\}, \{\}, P, \{C\}, A) \sim_{OL} \text{state } (M, \{\}, P, \{\}, A \cup \{C\})$

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

**lemma** *active-subset-of-setOfFormulasWithLabelDiffActive*:

$l \neq \text{Active} \implies \text{active-subset } \{(C', l)\} = \{\}$   
 ⟨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*:  $state (N \cup \{C\}, \{\}, P, \{\}, A) \sim_{GC} state (N, \{C\}, P, \{\}, A)$   
 ⟨proof⟩

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

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

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

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

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

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

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

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

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

**theorem** *OL-step-imp-GC-step*:  $M \sim_{OL} M' \implies M \sim_{GC} M'$   
 ⟨proof⟩

## 6.3 Completeness

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

shows

*OL-Liminf-saturated*: *saturated (Liminf-llist Sts)* **and**  
*OL-complete-Liminf*:  $B \in \text{Bot-F} \implies \text{fst } \text{‘} \text{ lhd Sts} \models_{\cap \mathcal{G}} \{B\} \implies$   
 $\exists BL \in \text{Bot-FL}. BL \in \text{Liminf-llist Sts}$  **and**  
*OL-complete*:  $B \in \text{Bot-F} \implies \text{fst } \text{‘} \text{ lhd Sts} \models_{\cap \mathcal{G}} \{B\} \implies$   
 $\exists i. \text{enat } i < \text{llength Sts} \wedge (\exists BL \in \text{Bot-FL}. BL \in \text{lnth Sts } i)$

*<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 G-F-q 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**

*G-F-q* :: *'q*  $\Rightarrow$  *'f*  $\Rightarrow$  *'g set* **and**

*G-I-q* :: *'q*  $\Rightarrow$  *'f inference*  $\Rightarrow$  *'g inference set option* **and**

*Equiv-F* :: *'f*  $\Rightarrow$  *'f*  $\Rightarrow$  *bool* (**infix**  $\langle \Rightarrow \rangle$  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**  $\langle \prec S \rangle$  50)

**assumes**

*wfp-Prec-S*: *wfp* ( $\prec S$ ) **and**

*transp-Prec-S*: *transp* ( $\prec S$ ) **and**

*finite-Inf-between*: *finite A*  $\implies$  *finite (no-labels.Inf-between A {C})*

**begin**



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

**lemma** *irreflp-Prec-S*:  $\text{irreflp } (\prec_S)$   
 ⟨proof⟩

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

## 7.2 Basic Definitions and Lemmas

**abbreviation** *new-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'f fset}$  **where**  
 $\text{new-of St} \equiv \text{fst St}$

**abbreviation** *xx-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'f option}$  **where**  
 $\text{xx-of St} \equiv \text{fst (snd St)}$

**abbreviation** *passive-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'p}$  **where**  
 $\text{passive-of St} \equiv \text{fst (snd (snd St))}$

**abbreviation** *yy-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'f option}$  **where**  
 $\text{yy-of St} \equiv \text{fst (snd (snd (snd St)))}$

**abbreviation** *active-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'f fset}$  **where**  
 $\text{active-of St} \equiv \text{snd (snd (snd (snd St)))}$

**abbreviation** *all-formulas-of* ::  $(\text{'p}, \text{'f}) \text{ OLf-state} \Rightarrow \text{'f set}$  **where**  
 $\text{all-formulas-of St} \equiv \text{fset (new-of St)} \cup \text{set-option (xx-of St)} \cup \text{elems (passive-of St)} \cup$   
 $\text{set-option (yy-of St)} \cup \text{fset (active-of St)}$

**fun** *fstate* ::  $\text{'f fset} \times \text{'f option} \times \text{'p} \times \text{'f option} \times \text{'f fset} \Rightarrow (\text{'f} \times \text{OL-label}) \text{ set}$  **where**  
 $\text{fstate } (N, X, P, Y, A) = \text{state (fset } N, \text{set-option } X, \text{elems } P, \text{set-option } Y, \text{fset } A)$

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

### definition

*Liminf-fstate* ::  $(\text{'p}, \text{'f}) \text{ OLf-state llist} \Rightarrow \text{'f set} \times \text{'f set} \times \text{'f set} \times \text{'f set} \times \text{'f set}$   
**where**

$\text{Liminf-fstate Sts} =$   
 $(\text{Liminf-llist (lmap (fset} \circ \text{new-of) Sts)},$   
 $\text{Liminf-llist (lmap (set-option} \circ \text{xx-of) Sts)},$   
 $\text{Liminf-llist (lmap (elems} \circ \text{passive-of) Sts)},$   
 $\text{Liminf-llist (lmap (set-option} \circ \text{yy-of) Sts)},$   
 $\text{Liminf-llist (lmap (fset} \circ \text{active-of) Sts)})$

**lemma** *Liminf-fstate-commute*:  $\text{Liminf-llist (lmap fstate Sts)} = \text{state (Liminf-fstate Sts)}$   
 ⟨proof⟩

**fun** *state-union* ::  $\text{'f set} \times \text{'f set} \times \text{'f set} \times \text{'f set} \times \text{'f set} \Rightarrow \text{'f set}$  **where**  
 $\text{state-union } (N, X, P, Y, A) = N \cup X \cup P \cup Y \cup A$

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

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

### 7.3 Initial State and Invariant

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

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

**lemma** *initial-OLf-invariant*: *is-initial-OLf-state* St  $\implies$  *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*  $\Rightarrow$  *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  $\longleftrightarrow$  ( $\forall$  St'.  $\neg$  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*  $(N, X, P, Y, A) \rightsquigarrow_{OL} \text{fstate} (N', X', P', Y', A')$

*<proof>*

**lemma** *fair-OL-step-imp-GC-step*:

$(N, X, P, Y, A) \rightsquigarrow_{OLf} (N', X', P', Y', A') \implies$

$\text{fstate} (N, X, P, Y, A) \rightsquigarrow_{GC} \text{fstate} (N', X', P', Y', A')$

*<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* ::  $(\text{'f} \times \text{OL-label}) \text{ set} \Rightarrow (\text{'f} \times \text{OL-label}) \text{ set} \Rightarrow \text{bool}$  (**infix**  $\rightsquigarrow_{IL}$  50)

**where**

*ol*:  $St \rightsquigarrow_{OL} St' \implies St \rightsquigarrow_{IL} St'$

| *red-by-children*:  $C \in \text{no-labels.Red-F} (A \cup M) \vee (M = \{C\} \wedge C' \prec \cdot C) \implies$

$\text{state} (\{\}, \{\}, P, \{C\}, A) \rightsquigarrow_{IL} \text{state} (M, \{\}, P, \{\}, A)$

### 8.2 Refinement

**lemma** *red-by-children-in-GC*:

**assumes**  $C \in \text{no-labels.Red-F} (A \cup M) \vee (M = \{C\} \wedge C' \prec \cdot C)$

**shows**  $\text{state} (\{\}, \{\}, P, \{C\}, A) \rightsquigarrow_{GC} \text{state} (M, \{\}, P, \{\}, A)$

*<proof>*

**theorem** *IL-step-imp-GC-step*:  $M \rightsquigarrow_{IL} M' \implies M \rightsquigarrow_{GC} M'$

*<proof>*

### 8.3 Completeness

**theorem**

**assumes**

*il-chain*:  $\text{chain} (\rightsquigarrow_{IL}) \text{ Sts}$  **and**

*act*:  $\text{active-subset} (\text{lhs Sts}) = \{\}$  **and**

*pas*:  $\text{passive-subset} (\text{Liminf-list Sts}) = \{\}$

**shows**

*IL-Liminf-saturated: saturated (Liminf-llist Sts) and*  
*IL-complete-Liminf:  $B \in \text{Bot-F} \implies \text{fst } \text{‘ lhd Sts} \models_{\cap \mathcal{G}} \{B\} \implies$*   
 *$\exists BL \in \text{Bot-FL}. BL \in \text{Liminf-llist Sts}$  and*  
*IL-complete:  $B \in \text{Bot-F} \implies \text{fst } \text{‘ lhd Sts} \models_{\cap \mathcal{G}} \{B\} \implies$*   
 *$\exists i. \text{enat } i < \text{llength Sts} \wedge (\exists BL \in \text{Bot-FL}. BL \in \text{lth Sts } i)$*   
 <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**  $\langle \rightsquigarrow \text{ILf} \rangle$  50) **where**

*ol: St*  $\rightsquigarrow \text{OLf St}' \implies \text{St} \rightsquigarrow \text{ILf St}'$

| *red-by-children: C*  $\in$  *no-labels.Red-F* (*fset A*  $\cup$  *fset M*)  $\vee$  *fset M* = {*C'*}  $\wedge$  *C'*  $\prec$  *C*  $\implies$   
 ({||}, *None*, *P*, *Some C*, *A*)  $\rightsquigarrow \text{ILf}$  (*M*, *None*, *P*, *None*, *A*)

### 9.3 Initial State and Invariant

**lemma** *step-ILf-invariant:*

**assumes** *St*  $\rightsquigarrow \text{ILf St}'$

**shows** *OLf-invariant St'*

<proof>

**lemma** *chain-ILf-invariant-lnth:*

**assumes**

*chain: chain* ( $\rightsquigarrow \text{ILf}$ ) *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* ( $\rightsquigarrow \text{ILf}$ ) *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  $\longleftrightarrow$   $(\forall St'. \neg St \rightsquigarrow_{ILf} St')$   
 ⟨proof⟩

## 9.5 Refinement

**lemma** *fair-IL-step-imp-IL-step*:  
**assumes** *ilf*:  $(N, X, P, Y, A) \rightsquigarrow_{ILf} (N', X', P', Y', A')$   
**shows** *fstate*  $(N, X, P, Y, A) \rightsquigarrow_{IL} \text{fstate } (N', X', P', Y', A')$   
 ⟨proof⟩

**lemma** *fair-IL-step-imp-GC-step*:  
 $(N, X, P, Y, A) \rightsquigarrow_{ILf} (N', X', P', Y', A') \implies$   
*fstate*  $(N, X, P, Y, A) \rightsquigarrow_{GC} \text{fstate } (N', X', P', Y', A')$   
 ⟨proof⟩

## 9.6 Completeness

**fun** *mset-of-fstate* ::  $('p, 'f)$  OLf-state  $\Rightarrow$  '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  $\Rightarrow$  'f multiset  $\Rightarrow$  bool (**infix**  $\prec\prec S$  50) **where**  
 $\prec\prec S \equiv \text{multp } (\prec S)$

**lemma** *wfP-Precprec-S*: wfP  $(\prec\prec S)$   
 ⟨proof⟩

**definition** *Less1-state* ::  $('p, 'f)$  OLf-state  $\Rightarrow$   $('p, 'f)$  OLf-state  $\Rightarrow$  bool (**infix**  $\sqsubset 1$  50) **where**  
 $St' \sqsubset 1 St \longleftrightarrow$   
*mset-of-fstate*  $St' \prec\prec S \text{mset-of-fstate } St$   
 $\vee$  (*mset-of-fstate*  $St' = \text{mset-of-fstate } St$   
 $\wedge$  (*mset-set* (fset (new-of  $St'$ ))  $\prec\prec S$  *mset-set* (fset (new-of St))  
 $\vee$  (*mset-set* (fset (new-of  $St'$ )) = *mset-set* (fset (new-of St))  
 $\wedge$  *mset-set* (set-option (xx-of  $St'$ ))  $\prec\prec S$  *mset-set* (set-option (xx-of St))))))

**lemma** *wfP-Less1-state*: wfP  $(\sqsubset 1)$   
 ⟨proof⟩

**definition** *Less2-state* ::  $('p, 'f)$  OLf-state  $\Rightarrow$   $('p, 'f)$  OLf-state  $\Rightarrow$  bool (**infix**  $\sqsubset 2$  50) **where**  
 $St' \sqsubset 2 St \equiv$   
*mset-set* (set-option (yy-of  $St'$ ))  $\prec\prec S$  *mset-set* (set-option (yy-of St))  
 $\vee$  (*mset-set* (set-option (yy-of  $St'$ )) = *mset-set* (set-option (yy-of St))  
 $\wedge$   $St' \sqsubset 1 St$ )

**lemma** *wfP-Less2-state*: wfP  $(\sqsubset 2)$   
 ⟨proof⟩

**lemma** *fair-IL-Liminf-yy-empty*:  
**assumes**  
*full*: *full-chain* ( $\rightsquigarrow$ ILf) *Sts* **and**  
*inv*: *OLf-invariant* (lhd *Sts*)  
**shows** *Liminf-llist* (*lmap* (*set-option*  $\circ$  *yy-of*) *Sts*) = {}  
<proof>

**lemma** *xx-nonempty-OLf-step-imp-Precprec-S*:  
**assumes**  
*step*: *St*  $\rightsquigarrow$ OLf *St'* **and**  
*xx*: *xx-of St*  $\neq$  None **and**  
*xx'*: *xx-of St'*  $\neq$  None  
**shows** *mset-of-fstate St'*  $\prec\prec_S$  *mset-of-fstate St*  
<proof>

**lemma** *xx-nonempty-ILf-step-imp-Precprec-S*:  
**assumes**  
*step*: *St*  $\rightsquigarrow$ ILf *St'* **and**  
*xx*: *xx-of St*  $\neq$  None **and**  
*xx'*: *xx-of St'*  $\neq$  None  
**shows** *mset-of-fstate St'*  $\prec\prec_S$  *mset-of-fstate St*  
<proof>

**lemma** *fair-IL-Liminf-xx-empty*:  
**assumes**  
*len*: *llength Sts* =  $\infty$  **and**  
*full*: *full-chain* ( $\rightsquigarrow$ ILf) *Sts* **and**  
*inv*: *OLf-invariant* (lhd *Sts*)  
**shows** *Liminf-llist* (*lmap* (*set-option*  $\circ$  *xx-of*) *Sts*) = {}  
<proof>

**lemma** *xx-nonempty-OLf-step-imp-Less1-state*:  
**assumes** *step*: (*N*, *Some C*, *P*, *Y*, *A*)  $\rightsquigarrow$ OLf (*N'*, *Some C'*, *P'*, *Y'*, *A'*) (**is** *?bef*  $\rightsquigarrow$ OLf *?aft*)  
**shows** *?aft*  $\sqsubset_1$  *?bef*  
<proof>

**lemma** *yy-empty-OLf-step-imp-Less1-state*:  
**assumes**  
*step*: *St*  $\rightsquigarrow$ OLf *St'* **and**  
*yy*: *yy-of St* = None **and**  
*yy'*: *yy-of St'* = None  
**shows** *St'*  $\sqsubset_1$  *St*  
<proof>

**lemma** *yy-empty-ILf-step-imp-Less1-state*:  
**assumes**  
*step*: *St*  $\rightsquigarrow$ ILf *St'* **and**  
*yy*: *yy-of St* = None **and**  
*yy'*: *yy-of St'* = None  
**shows** *St'*  $\sqsubset_1$  *St*  
<proof>

**lemma** *fair-IL-Liminf-new-empty*:  
**assumes**  
*len*: *llength Sts* =  $\infty$  **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 =  $\infty$  **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:  $B \in \text{Bot-F} \implies \text{fset} (\text{new-of} (\text{lhd Sts})) \models_{\cap \mathcal{G}} \{B\} \implies$   
 $\exists B' \in \text{Bot-F}. B' \in \text{state-union} (\text{Liminf-fstate Sts})$  **and**  
 fair-IL-complete:  $B \in \text{Bot-F} \implies \text{fset} (\text{new-of} (\text{lhd Sts})) \models_{\cap \mathcal{G}} \{B\} \implies$   
 $\exists i. \text{enat } i < \text{llength Sts} \wedge (\exists B' \in \text{Bot-F}. B' \in \text{all-formulas-of} (\text{lth Sts } i))$   
 <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*:  $B \in \text{Bot-}F \implies \text{fset}(\text{new-of}(\text{lhd } Sts)) \models_{\cap \mathcal{G}} \{B\} \implies$

$\exists B' \in \text{Bot-}F. B' \in \text{state-union}(\text{Liminf-fstate } Sts)$  **and**

*fair-OL-complete*:  $B \in \text{Bot-}F \implies \text{fset}(\text{new-of}(\text{lhd } Sts)) \models_{\cap \mathcal{G}} \{B\} \implies$

$\exists i. \text{enat } i < \text{llength } Sts \wedge (\exists B' \in \text{Bot-}F. B' \in \text{all-formulas-of}(\text{lth } Sts \ i))$

*<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 G-F-q 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
```

```
  G-F-q :: 'q  $\Rightarrow$  'f  $\Rightarrow$  'g set and
```

```
  G-I-q :: 'q  $\Rightarrow$  'f inference  $\Rightarrow$  'g inference set option and
```

```
  Equiv-F :: 'f  $\Rightarrow$  'f  $\Rightarrow$  bool (infix  $\langle \Rightarrow \rangle$  50) and
```

```
  Prec-F :: 'f  $\Rightarrow$  'f  $\Rightarrow$  bool (infix  $\langle \prec \rangle$  50) +
```

```
fixes
```

```
  Prec-S :: 'f  $\Rightarrow$  'f  $\Rightarrow$  bool (infix  $\langle \prec S \rangle$  50)
```

```
assumes
```



*wfp-Prec-S*: *wfp* ( $\prec S$ ) **and**  
*transp-Prec-S*: *transp* ( $\prec S$ ) **and**  
*finite-Inf-between*: *finite*  $A \implies$  *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 G-F-q G-I-q*  
*Equiv-F Prec-F [] hd λy xs. if y ∈ 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*  $\Rightarrow$  '*f inference set* **where**  
*flat-inferences-of*  $T = \bigcup \{lset \iota \mid \iota. \iota \in \# T\}$

**fun**  
*zl-state* :: '*f inference llist multiset*  $\times$  '*f inference set*  $\times$  '*f set*  $\times$  '*f set*  $\times$  '*f set*  $\Rightarrow$   
 '*f inference set*  $\times$  ('*f*  $\times$  *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*  $T - D$ , ( $\lambda C. (C, Passive)$ ) '  $P \cup (\lambda C. (C, YY))$  '  $Y \cup (\lambda C. (C, Active))$  '  $A$ )  
 ⟨*proof*⟩

**inductive**

*ZL* :: '*f inference set*  $\times$  ('*f*  $\times$  *DL-label*) *set*  $\Rightarrow$  '*f inference set*  $\times$  ('*f*  $\times$  *DL-label*) *set*  $\Rightarrow$  *bool*  
 (**infix**  $\langle \rightsquigarrow ZL \rangle$  50)

**where**

*compute-infer*:  $\iota 0 \in$  *no-labels.Red-I* ( $A \cup \{C\}$ )  $\implies$

*zl-state* ( $T + \{\#LCons \iota 0 \iota s\#\}$ ,  $D, P, \{C\}, A$ )  $\rightsquigarrow ZL$  *zl-state* ( $T + \{\#\iota s\#\}$ ,  $D \cup \{\iota 0\}$ ,  $P \cup \{C\}, \{C\}$ ,  $A$ )

| *choose-p*: *zl-state* ( $T, D, P \cup \{C\}, \{C\}, A$ )  $\rightsquigarrow ZL$  *zl-state* ( $T, D, P, \{C\}, A$ )

| *delete-fwd*:  $C \in$  *no-labels.Red-F*  $A \vee (\exists C' \in A. C' \preceq C) \implies$

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

## 11.2 Refinement

**lemma** *zl-compute-infer-in-lgc:*

**assumes**  $\iota 0 \in \text{no-labels.Red-I } (A \cup \{C\})$   
**shows**  $zl\text{-state } (T + \{\#LCons \iota 0 \iota s\# \}, D, P, \{\}, A) \sim_{LGC}$   
 $zl\text{-state } (T + \{\#\iota s\# \}, D \cup \{\iota 0\}, P \cup \{C\}, \{\}, A)$

*<proof>*

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

*<proof>*

**lemma** *zl-delete-fwd-in-lgc:*

**assumes**  $C \in \text{no-labels.Red-F } A \vee (\exists C' \in A. C' \preceq C)$   
**shows**  $zl\text{-state } (T, D, P, \{C\}, A) \sim_{LGC} zl\text{-state } (T, D, P, \{\}, A)$

*<proof>*

**lemma** *zl-simplify-fwd-in-lgc:*

**assumes**  $C \in \text{no-labels.Red-F-G } (A \cup \{C'\})$   
**shows**  $zl\text{-state } (T, D, P, \{C\}, A) \sim_{LGC} zl\text{-state } (T, D, P, \{C'\}, A)$

*<proof>*

**lemma** *zl-delete-bwd-in-lgc:*

**assumes**  $C' \in \text{no-labels.Red-F-G } \{C\} \vee C' \cdot \succ C$   
**shows**  $zl\text{-state } (T, D, P, \{C\}, A \cup \{C'\}) \sim_{LGC} zl\text{-state } (T, D, P, \{C\}, A)$

*<proof>*

**lemma** *zl-simplify-bwd-in-lgc:*

**assumes**  $C' \in \text{no-labels.Red-F-G } \{C, C''\}$   
**shows**  $zl\text{-state } (T, D, P, \{C\}, A \cup \{C'\}) \sim_{LGC} zl\text{-state } (T, D, P \cup \{C''\}, \{C\}, A)$

*<proof>*

**lemma** *zl-schedule-infer-in-lgc:*

**assumes**  $\text{flat-inferences-of } T' = \text{no-labels.Inf-between } A \{C\}$   
**shows**  $zl\text{-state } (T, D, P, \{C\}, A) \sim_{LGC}$   
 $zl\text{-state } (T + T', D - \text{flat-inferences-of } T', P, \{\}, A \cup \{C\})$

*<proof>*

**lemma** *zl-delete-orphan-infers-in-lgc:*

**assumes**  $\text{inter: lset } \iota s \cap \text{no-labels.Inf-from } A = \{\}$   
**shows**  $zl\text{-state } (T + \{\#\iota s\# \}, D, P, Y, A) \sim_{LGC} zl\text{-state } (T, D \cup \text{lset } \iota s, P, Y, A)$

*<proof>*

**theorem** *ZL-step-imp-LGC-step:*  $St \sim_{ZL} St' \implies St \sim_{LGC} St'$

*<proof>*

## 11.3 Completeness

**theorem**

**assumes**

*zl-chain*:  $\text{chain } (\sim\text{ZL}) \text{ } S\text{ts}$  **and**

*act*:  $\text{active-subset } (\text{snd } (\text{lhd } S\text{ts})) = \{\}$  **and**

*pas*:  $\text{passive-subset } (\text{Liminf-llist } (\text{lmap } \text{snd } S\text{ts})) = \{\}$  **and**

*no-prems-init*:  $\forall \iota \in \text{Inf-}F. \text{prems-of } \iota = [] \longrightarrow \iota \in \text{fst } (\text{lhd } S\text{ts})$  **and**

*final-sched*:  $\text{Liminf-llist } (\text{lmap } \text{fst } S\text{ts}) = \{\}$

**shows**

*ZL-Liminf-saturated*:  $\text{saturated } (\text{Liminf-llist } (\text{lmap } \text{snd } S\text{ts}))$  **and**

*ZL-complete-Liminf*:  $B \in \text{Bot-}F \Longrightarrow \text{fst } ' \text{snd } (\text{lhd } S\text{ts}) \models_{\cap\mathcal{G}} \{B\} \Longrightarrow$

$\exists BL \in \text{Bot-FL}. BL \in \text{Liminf-llist } (\text{lmap } \text{snd } S\text{ts})$  **and**

*ZL-complete*:  $B \in \text{Bot-}F \Longrightarrow \text{fst } ' \text{snd } (\text{lhd } S\text{ts}) \models_{\cap\mathcal{G}} \{B\} \Longrightarrow$

$\exists i. \text{enat } i < \text{llength } S\text{ts} \wedge (\exists BL \in \text{Bot-FL}. BL \in \text{snd } (\text{lnth } S\text{ts } i))$

*<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**

$z \in \text{set } (\text{take } m \text{ } xs)$

**shows**  $z \in \text{set } (\text{take } m \text{ } (\text{remove1 } y \text{ } xs))$

*<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]*:  $\text{llists } \text{empty} = \{\#\}$  **and**

*llists-not-empty*:  $Q \neq \text{empty} \Longrightarrow \text{llists } Q \neq \{\#\}$  **and**

*llists-add[simp]*:  $\text{llists } (\text{add-llist } es \text{ } Q) = \text{llists } Q + \{\#es\# \}$  **and**

*llist-remove[simp]*:  $\text{llists } (\text{remove-llist } es \text{ } Q) = \text{llists } Q - \{\#es\# \}$  **and**

*llists-pick-elem*:  $(\exists es \in \# \text{ llists } Q. es \neq \text{LNil}) \Longrightarrow$

$\exists e \text{ } es. \text{LCons } e \text{ } es \in \# \text{ llists } Q \wedge \text{fst } (\text{pick-elem } Q) = e$

$\wedge \text{llists } (\text{snd } (\text{pick-elem } Q)) = \text{llists } Q - \{\#\text{LCons } e \text{ } es\# \} + \{\#es\# \}$

**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 (lth 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* × *'f inference set* × *'p* × *'f option* × *'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 G-F-q 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* ⇒ *'g set* ⇒ *'g set* ⇒ *bool* **and**

*Inf-G-q* :: *'q* ⇒ *'g inference set* **and**

*Red-I-q* :: *'q* ⇒ *'g set* ⇒ *'g inference set* **and**

*Red-F-q* :: *'q* ⇒ *'g set* ⇒ *'g set* **and**

*G-F-q* :: *'q* ⇒ *'f* ⇒ *'g set* **and**

*G-I-q* :: *'q* ⇒ *'f inference* ⇒ *'g inference set option* **and**

*Equiv-F* :: *'f* ⇒ *'f* ⇒ *bool* (**infix** <≐> 50) **and**

*Prec-F* :: *'f* ⇒ *'f* ⇒ *bool* (**infix** <<> 50) **and**

*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* ⇒ *'p* ⇒ *'p* **and**

*p-felems* :: *'p* ⇒ *'f fset* +

**fixes**

*Prec-S* :: *'f* ⇒ *'f* ⇒ *bool* (**infix** <<S> 50)

**assumes**

*wfp-Prec-S*: *wfp* (<S>) **and**

*transp-Prec-S*: *transp* (<S>) **and**

*countable-Inf-between*: *finite A* ⇒ *countable (no-labels.Inf-between A {C})*

**begin**

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

**lemma** *irreflp-Prec-S*:  $\text{irreflp } (\prec_S)$   
 ⟨proof⟩

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

## 13.2 Basic Definitions and Lemmas

**abbreviation** *todo-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'t$  **where**  
*todo-of*  $St \equiv \text{fst } St$

**abbreviation** *done-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'f$  inference set **where**  
*done-of*  $St \equiv \text{fst } (\text{snd } St)$

**abbreviation** *passive-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'p$  **where**  
*passive-of*  $St \equiv \text{fst } (\text{snd } (\text{snd } St))$

**abbreviation** *yy-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'f$  option **where**  
*yy-of*  $St \equiv \text{fst } (\text{snd } (\text{snd } (\text{snd } St)))$

**abbreviation** *active-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'f$  fset **where**  
*active-of*  $St \equiv \text{snd } (\text{snd } (\text{snd } (\text{snd } St)))$

**abbreviation** *all-formulas-of* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $'f$  set **where**  
*all-formulas-of*  $St \equiv \text{passive.elems } (\text{passive-of } St) \cup \text{set-option } (\text{yy-of } St) \cup \text{fset } (\text{active-of } St)$

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

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

**definition**  
*Liminf-zl-fstate* ::  $(t, 'p, 'f)$  ZLf-state llist  $\Rightarrow$   $'f$  set  $\times$   $'f$  set  $\times$   $'f$  set  
**where**  
*Liminf-zl-fstate*  $Sts =$   
 $(\text{Liminf-llist } (\text{lmap } (\text{passive.elems } \circ \text{passive-of}) Sts),$   
 $\text{Liminf-llist } (\text{lmap } (\text{set-option } \circ \text{yy-of}) Sts),$   
 $\text{Liminf-llist } (\text{lmap } (\text{fset } \circ \text{active-of}) Sts))$

**lemma** *Liminf-zl-fstate-commute*:  
 $\text{Liminf-llist } (\text{lmap } (\text{snd } \circ \text{zl-fstate}) Sts) = \text{labeled-formulas-of } (\text{Liminf-zl-fstate } Sts)$   
 ⟨proof⟩

**fun** *formulas-union* ::  $'f$  set  $\times$   $'f$  set  $\times$   $'f$  set  $\Rightarrow$   $'f$  set **where**  
*formulas-union*  $(P, Y, A) = P \cup Y \cup A$

**inductive**  
*fair-ZL* ::  $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$   $(t, 'p, 'f)$  ZLf-state  $\Rightarrow$  bool (**infix**  $\langle \rightsquigarrow \text{ZL} \rangle$  50)  
**where**  
*compute-infer*:  $(\exists \iota s \in \# t\text{-llists } T. \iota s \neq \text{LNil}) \implies t\text{-pick-elem } T = (\iota 0, T') \implies$   
 $\iota 0 \in \text{no-labels.Red-I } (\text{fset } A \cup \{C\}) \implies$   
 $(T, D, P, \text{None}, A) \rightsquigarrow \text{ZL} (T', D \cup \{\iota 0\}, p\text{-add } C P, \text{None}, A)$   
 | *choose-p*:  $P \neq p\text{-empty} \implies$   
 $(T, D, P, \text{None}, A) \rightsquigarrow \text{ZL} (T, D, p\text{-remove } (p\text{-select } P) P, \text{Some } (p\text{-select } P), A)$

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

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

The step below is slightly more general than the corresponding step in  $(\rightsquigarrow \text{ZLf})$ , 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**  
 $P \neq p\text{-empty} \implies$   
*choose-p-step*  $(T, D, P, \text{None}, A) (T, D', p\text{-remove } (p\text{-select } P) P, \text{Some } (p\text{-select } P), A)$

### 13.3 Initial State and Invariant

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

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

**lemma** *initial-ZLf-invariant*:  
**assumes** *is-initial-ZLf-state*  $St$   
**shows** *ZLf-invariant*  $St$   
 $\langle \text{proof} \rangle$

**lemma** *step-ZLf-invariant*:  
**assumes**  
 $inv$ : *ZLf-invariant*  $St$  **and**  
 $step$ :  $St \rightsquigarrow \text{ZLf } St'$   
**shows** *ZLf-invariant*  $St'$   
 $\langle \text{proof} \rangle$

**lemma** *chain-ZLf-invariant-lnth*:  
**assumes**  
 $chain$ :  $chain (\rightsquigarrow \text{ZLf}) Sts$  **and**  
 $fair\text{-hd}$ : *ZLf-invariant*  $(\text{lhd } Sts)$  **and**  
 $i\text{-lt}$ :  $\text{enat } i < \text{llength } Sts$   
**shows** *ZLf-invariant*  $(\text{lnth } Sts i)$   
 $\langle \text{proof} \rangle$



**lemma** *chain-ZLf-invariant-llast*:

**assumes**

*chain*:  $\text{chain } (\rightsquigarrow \text{ZLf}) \text{ } Sts$  **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*  $\Rightarrow$  *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*  $\longleftrightarrow$   $(\forall St'. \neg St \rightsquigarrow \text{ZLf } St')$

*<proof>*

### 13.5 Refinement

**lemma** *fair-ZL-step-imp-ZL-step*:

**assumes** *zlf*:  $(T, D, P, Y, A) \rightsquigarrow \text{ZLf } (T', D', P', Y', A')$

**shows** *zl-fstate*  $(T, D, P, Y, A) \rightsquigarrow \text{ZL } \text{zl-fstate } (T', D', P', Y', A')$

*<proof>*

**lemma** *fair-ZL-step-imp-GC-step*:

$(T, D, P, Y, A) \rightsquigarrow \text{ZLf } (T', D', P', Y', A') \Longrightarrow$

*zl-fstate*  $(T, D, P, Y, A) \rightsquigarrow \text{LGC } \text{zl-fstate } (T', D', P', Y', A')$

*<proof>*

### 13.6 Completeness

**fun** *mset-of-zl-fstate* :: (*t*, *p*, *f*) *ZLf-state*  $\Rightarrow$  *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*  $\Rightarrow$  *f* *multiset*  $\Rightarrow$  *bool* (**infix**  $\prec \prec S$  50) **where**

$\prec \prec S \equiv \text{multp } (\prec S)$

**lemma** *wfP-Precprec-S*: *wfP*  $(\prec \prec S)$

*<proof>*

**definition** *Less-state* :: (*t*, *p*, *f*) *ZLf-state*  $\Rightarrow$  (*t*, *p*, *f*) *ZLf-state*  $\Rightarrow$  *bool* (**infix**  $\sqsubset$  50)

**where**

$St' \sqsubset St \longleftrightarrow$

*mset-of-zl-fstate*  $St' \prec \prec S \text{ } \text{mset-of-zl-fstate } St$

$\vee (\text{mset-of-zl-fstate } St' = \text{mset-of-zl-fstate } St$

$\wedge (\text{mset-set } (\text{passive.elems } (\text{passive-of } St')) \prec \prec S \text{ } \text{mset-set } (\text{passive.elems } (\text{passive-of } St)))$

$\vee (\text{passive.elems } (\text{passive-of } St') = \text{passive.elems } (\text{passive-of } St)$

$\wedge (\text{mset-set } (\text{set-option } (\text{yy-of } St')) \prec \prec S \text{ } \text{mset-set } (\text{set-option } (\text{yy-of } St)))$

$\vee (\text{mset-set } (\text{set-option } (\text{yy-of } St')) = \text{mset-set } (\text{set-option } (\text{yy-of } St)))$

$\wedge \text{size } (t\text{-llists } (\text{todo-of } St')) < \text{size } (t\text{-llists } (\text{todo-of } St))))))$

**lemma** *wfP-Less-state*: *wfP*  $(\sqsubset)$

*<proof>*

**lemma** *non-compute-infer-ZLf-step-imp-Less-state:*

**assumes**

*step*:  $St \rightsquigarrow ZLf\ St'$  **and**

*non-ci*:  $\neg \text{compute-infer-step } St\ St'$

**shows**  $St' \sqsubset St$

*<proof>*

**lemma** *yy-nonempty-ZLf-step-imp-Less-state:*

**assumes**

*step*:  $St \rightsquigarrow ZLf\ St'$  **and**

*yy*: *yy-of*  $St \neq \text{None}$

**shows**  $St' \sqsubset St$

*<proof>*

**lemma** *fair-ZL-Liminf-yy-empty:*

**assumes**

*len*:  $\text{length } Sts = \infty$  **and**

*full*: *full-chain*  $(\rightsquigarrow ZLf)\ Sts$  **and**

*inv*: *ZLf-invariant*  $(\text{lhs } Sts)$

**shows**  $\text{Liminf-llist } (\text{lmap } (\text{set-option} \circ \text{yy-of})\ Sts) = \{\}$

*<proof>*

**lemma** *ZLf-step-imp-passive-queue-step:*

**assumes**  $St \rightsquigarrow ZLf\ St'$

**shows** *passive.queue-step*  $(\text{passive-of } St)\ (\text{passive-of } St')$

*<proof>*

**lemma** *choose-p-step-imp-select-passive-queue-step:*

**assumes** *choose-p-step*  $St\ St'$

**shows** *passive.select-queue-step*  $(\text{passive-of } St)\ (\text{passive-of } St')$

*<proof>*

**lemma** *fair-ZL-Liminf-passive-empty:*

**assumes**

*len*:  $\text{length } Sts = \infty$  **and**

*full*: *full-chain*  $(\rightsquigarrow ZLf)\ Sts$  **and**

*init*: *is-initial-ZLf-state*  $(\text{lhs } Sts)$  **and**

*fair*: *infinitely-often compute-infer-step*  $Sts \longrightarrow$  *infinitely-often choose-p-step*  $Sts$

**shows**  $\text{Liminf-llist } (\text{lmap } (\text{passive elems} \circ \text{passive-of})\ Sts) = \{\}$

*<proof>*

**lemma** *ZLf-step-imp-todo-queue-step:*

**assumes**  $St \rightsquigarrow ZLf\ St'$

**shows** *todo.lqueue-step*  $(\text{todo-of } St,\ \text{done-of } St)\ (\text{todo-of } St',\ \text{done-of } St')$

*<proof>*

**lemma** *fair-ZL-Liminf-todo-empty:*

**assumes**

*len*:  $\text{length } Sts = \infty$  **and**

*full*: *full-chain*  $(\rightsquigarrow ZLf)\ Sts$  **and**

*init*: *is-initial-ZLf-state*  $(\text{lhs } Sts)$

**shows**  $\text{Liminf-llist } (\text{lmap } (\lambda St.\ \text{flat-inferences-of } (t\text{-llists } (\text{todo-of } St)) - \text{done-of } St)\ Sts) =$

$\{\}$

*<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*:  $B \in \text{Bot-F} \implies \text{passive elems (passive-of (lhd Sts))} \models_{\cap \mathcal{G}} \{B\} \implies$

$\exists B' \in \text{Bot-F}. B' \in \text{formulas-union (Liminf-zl-fstate Sts)}$  **and**

*fair-ZL-complete*:  $B \in \text{Bot-F} \implies \text{passive elems (passive-of (lhd Sts))} \models_{\cap \mathcal{G}} \{B\} \implies$

$\exists i. \text{enat } i < \text{llength Sts} \wedge (\exists B' \in \text{Bot-F}. B' \in \text{all-formulas-of (lnth Sts } i))$

*<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* =  $'t \times 'p \times 'f \text{ option} \times 'f \text{ fset}$

**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 G-F-q

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 ::  $'f \text{ set}$  **and**

Inf-F ::  $'f \text{ inference set}$  **and**

Bot-G ::  $'g \text{ set}$  **and**

Q ::  $'q \text{ set}$  **and**

entails-q ::  $'q \Rightarrow 'g \text{ set} \Rightarrow 'g \text{ set} \Rightarrow \text{bool}$  **and**

Inf-G-q ::  $'q \Rightarrow 'g \text{ inference set}$  **and**

Red-I-q ::  $'q \Rightarrow 'g \text{ set} \Rightarrow 'g \text{ inference set}$  **and**

Red-F-q ::  $'q \Rightarrow 'g \text{ set} \Rightarrow 'g \text{ set}$  **and**

G-F-q ::  $'q \Rightarrow 'f \Rightarrow 'g \text{ set}$  **and**

G-I-q ::  $'q \Rightarrow 'f \text{ inference} \Rightarrow 'g \text{ inference set option}$  **and**

Equiv-F ::  $'f \Rightarrow 'f \Rightarrow \text{bool}$  (**infix**  $\langle \doteq \rangle$  50) **and**

Prec-F ::  $'f \Rightarrow 'f \Rightarrow \text{bool}$  (**infix**  $\langle \prec \cdot \rangle$  50) **and**

t-empty ::  $'t$  **and**

t-add-llist ::  $'f \text{ inference llist} \Rightarrow 't \Rightarrow 't$  **and**

t-remove-llist ::  $'f \text{ inference llist} \Rightarrow 't \Rightarrow 't$  **and**

t-pick-elem ::  $'t \Rightarrow 'f \text{ inference} \times 't$  **and**

t-llists ::  $'t \Rightarrow 'f \text{ inference llist multiset}$  **and**

p-empty ::  $'p$  **and**

p-select ::  $'p \Rightarrow 'f$  **and**

p-add ::  $'f \Rightarrow 'p \Rightarrow 'p$  **and**

$p\text{-remove} :: 'f \Rightarrow 'p \Rightarrow 'p \text{ and}$   
 $p\text{-felems} :: 'p \Rightarrow 'f \text{ fset and}$   
 $\text{Prec-}S :: 'f \Rightarrow 'f \Rightarrow \text{bool (infix } \langle \prec S \rangle 50)$

**begin**

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

**inductive**

$\text{fair-ZL-wo-ghosts} ::$

$('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow ('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow \text{bool}$

(**infix**  $\langle \rightsquigarrow \text{ZLfw} \rangle 50)$

**where**

$\text{compute-infer}: (\exists \iota s \in \# \text{ t-llists } T. \iota s \neq \text{LNil}) \Longrightarrow \text{t-pick-elem } T = (\iota 0, T') \Longrightarrow$   
 $\iota 0 \in \text{no-labels.Red-I (fset } A \cup \{C\}) \Longrightarrow$   
 $(T, P, \text{None}, A) \rightsquigarrow \text{ZLfw } (T', \text{p-add } C \ P, \text{None}, A)$

|  $\text{choose-p}: P \neq \text{p-empty} \Longrightarrow$

$(T, P, \text{None}, A) \rightsquigarrow \text{ZLfw } (T, \text{p-remove } (\text{p-select } P) \ P, \text{Some } (\text{p-select } P), A)$

|  $\text{delete-fwd}: C \in \text{no-labels.Red-F (fset } A) \vee (\exists C' \in \text{fset } A. C' \preceq C) \Longrightarrow$

$(T, P, \text{Some } C, A) \rightsquigarrow \text{ZLfw } (T, P, \text{None}, A)$

|  $\text{simplify-fwd}: C' \prec S C \Longrightarrow C \in \text{no-labels.Red-F (fset } A \cup \{C'\}) \Longrightarrow$

$(T, P, \text{Some } C, A) \rightsquigarrow \text{ZLfw } (T, P, \text{Some } C', A)$

|  $\text{delete-bwd}: C' \notin A \Longrightarrow C' \in \text{no-labels.Red-F } \{C\} \vee C' \succ C \Longrightarrow$

$(T, P, \text{Some } C, A \mid \cup \{|C'\}) \rightsquigarrow \text{ZLfw } (T, P, \text{Some } C, A)$

|  $\text{simplify-bwd}: C' \notin A \Longrightarrow C'' \prec S C' \Longrightarrow C' \in \text{no-labels.Red-F } \{C, C''\} \Longrightarrow$

$(T, P, \text{Some } C, A \mid \cup \{|C'\}) \rightsquigarrow \text{ZLfw } (T, \text{p-add } C'' \ P, \text{Some } C, A)$

|  $\text{schedule-infer}: \text{flat-inferences-of (mset } \iota s s) = \text{no-labels.Inf-between (fset } A) \{C\} \Longrightarrow$

$(T, P, \text{Some } C, A) \rightsquigarrow \text{ZLfw } (\text{fold t-add-llist } \iota s \ T, P, \text{None}, A \mid \cup \{|C|\})$

|  $\text{delete-orphan-infers}: \iota s \in \# \text{ t-llists } T \Longrightarrow \text{lset } \iota s \cap \text{no-labels.Inf-from (fset } A) = \{\} \Longrightarrow$

$(T, P, Y, A) \rightsquigarrow \text{ZLfw } (\text{t-remove-llist } \iota s \ T, P, Y, A)$

**inductive**

$\text{compute-infer-step} ::$

$('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow ('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow \text{bool}$

**where**

$(\exists \iota s \in \# \text{ t-llists } T. \iota s \neq \text{LNil}) \Longrightarrow \text{t-pick-elem } T = (\iota 0, T') \Longrightarrow$

$\iota 0 \in \text{no-labels.Red-I (fset } A \cup \{C\}) \Longrightarrow$

$\text{compute-infer-step } (T, P, \text{None}, A) (T', \text{p-add } C \ P, \text{None}, A)$

**inductive**

$\text{choose-p-step} :: ('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow ('t, 'p, 'f) \text{ ZLf-wo-ghosts-state} \Rightarrow \text{bool}$

**where**

$P \neq \text{p-empty} \Longrightarrow$

$\text{choose-p-step } (T, P, \text{None}, A) (T, \text{p-remove } (\text{p-select } P) \ P, \text{Some } (\text{p-select } P), A)$

**lemma**  $w\text{-ghosts-compute-infer-step-imp-compute-infer-step}:$

**assumes**  $w\text{-ghosts.compute-infer-step } St \ St'$

**shows**  $\text{compute-infer-step } (w\text{-ghosts-of } St) (w\text{-ghosts-of } St')$

$\langle \text{proof} \rangle$

**lemma**  $\text{choose-p-step-imp-w-ghosts-choose-p-step}:$

**assumes**  $\text{choose-p-step } (w\text{-ghosts-of } St) (w\text{-ghosts-of } St')$

**shows**  $w\text{-ghosts.choose-p-step } St \ St'$

$\langle \text{proof} \rangle$

## 14.2 Basic Definitions and Lemmas

**abbreviation** *todo-of* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  't **where**  
*todo-of St*  $\equiv$  fst St

**abbreviation** *passive-of* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  'p **where**  
*passive-of St*  $\equiv$  fst (snd St)

**abbreviation** *yy-of* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  'f option **where**  
*yy-of St*  $\equiv$  fst (snd (snd St))

**abbreviation** *active-of* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  'f fset **where**  
*active-of St*  $\equiv$  snd (snd (snd St))

**abbreviation** *all-formulas-of* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  'f set **where**  
*all-formulas-of St*  $\equiv$  passive.elems (passive-of St)  $\cup$  set-option (yy-of St)  $\cup$  fset (active-of St)

**definition**

*Liminf-zl-fstate* :: ('t, 'p, 'f) ZLf-wo-ghosts-state llist  $\Rightarrow$  'f set  $\times$  'f set  $\times$  'f set

**where**

*Liminf-zl-fstate Sts* =  
 (Liminf-llist (lmap (passive.elems  $\circ$  passive-of) Sts),  
 Liminf-llist (lmap (set-option  $\circ$  yy-of) Sts),  
 Liminf-llist (lmap (fset  $\circ$  active-of) Sts))

## 14.3 Initial States and Invariants

**inductive** *is-initial-ZLf-wo-ghosts-state* :: ('t, 'p, 'f) ZLf-wo-ghosts-state  $\Rightarrow$  bool **where**  
*flat-inferences-of* (mset *iss*) = no-labels.Inf-from {}  $\implies$   
*is-initial-ZLf-wo-ghosts-state* (fold t-add-llist *iss* 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  $\Rightarrow$  'state  
**and** *R* :: 'state  $\Rightarrow$  'state  $\Rightarrow$  bool (**infix**  $\langle \rightsquigarrow \rangle$  60)  
**and** *R0* :: 'state0  $\Rightarrow$  'state0  $\Rightarrow$  bool (**infix**  $\langle \rightsquigarrow_0 \rangle$  60)  
**assumes** *simul*:  $\bigwedge St0 St'. \text{erase } St0 \rightsquigarrow St' \implies \exists St0'. \text{erase } St0' = St' \wedge St0 \rightsquigarrow_0 St0'$   
**begin**

**definition** *lift* :: 'state0  $\Rightarrow$  'state  $\Rightarrow$  'state0 **where**  
*lift St0 St'* = (SOME St0'. *erase* St0' = St'  $\wedge$  St0  $\rightsquigarrow_0$  St0')

**lemma** *lift*: *erase* St0  $\rightsquigarrow$  St'  $\implies$  *erase* (lift St0 St') = St'  $\wedge$  St0  $\rightsquigarrow_0$  lift St0 St'

$\langle \text{proof} \rangle$

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

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

**primcorec** *theSts0* :: 'state0  $\Rightarrow$  'state llist  $\Rightarrow$  'state0 llist **where**

*theSts0 St0 Sts* =

(*case Sts of*

*LNil*  $\Rightarrow$  *LCons St0 LNil*

  | *LCons St Sts'*  $\Rightarrow$  *LCons St0 (theSts0 (lift St0 St) Sts')*)

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *simul-chain0*:

**assumes** *chain*: *lnull Sts*  $\vee$  (*chain* ( $\rightsquigarrow$ ) *Sts*  $\wedge$  *erase St0*  $\rightsquigarrow$  *lhd Sts*)

**shows**  $\exists$  *Sts0*. *lhd Sts0* = *St0*  $\wedge$  *lmap erase (ltl Sts0)* = *Sts*  $\wedge$  *chain* ( $\rightsquigarrow 0$ ) *Sts0*

$\langle \text{proof} \rangle$

**lemma** *simul-chain*:

**assumes**

*chain*: *chain* ( $\rightsquigarrow$ ) *Sts* **and**

*hd*: *lhd Sts* = *erase St0*

**shows**  $\exists$  *Sts0*. *lhd Sts0* = *St0*  $\wedge$  *lmap erase Sts0* = *Sts*  $\wedge$  *chain* ( $\rightsquigarrow 0$ ) *Sts0*

$\langle \text{proof} \rangle$

**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*

$\langle \text{proof} \rangle$

**lemma** *fair-ZL-step-imp-fair-ZL-wo-ghosts-step*:

**assumes** *St*  $\rightsquigarrow$  *ZLf St'*

**shows** *wo-ghosts-of St*  $\rightsquigarrow$  *ZLfw wo-ghosts-of St'*

$\langle \text{proof} \rangle$

**lemma** *fair-ZL-wo-ghosts-step-imp-fair-ZL-step*:

**assumes** *wo-ghosts-of St0*  $\rightsquigarrow$  *ZLfw St'*

**shows**  $\exists$  *St0'*. *wo-ghosts-of St0'* = *St'  $\wedge$  St0*  $\rightsquigarrow$  *ZLf St0'*

$\langle \text{proof} \rangle$

**interpretation** *bisim*: *bisim wo-ghosts-of* ( $\rightsquigarrow$  *ZLfw*) ( $\rightsquigarrow$  *ZLf*)

$\langle \text{proof} \rangle$

**lemma** *chain-fair-ZL-step-wo-ghosts-imp-chain-fair-ZL-step:*

**assumes** *chain: chain ( $\rightsquigarrow$ ZLfw) Sts*

**shows**  $\exists Sts0. \text{lmap } \text{wo-ghosts-of } Sts0 = Sts \wedge \text{chain } (\rightsquigarrow\text{ZLf}) Sts0 \wedge \text{done-of } (\text{lhd } Sts0) = \{\}$

*<proof>*

**lemma** *full-chain-fair-ZL-step-wo-ghosts-imp-full-chain-fair-ZL-step:*

**assumes** *full-chain ( $\rightsquigarrow$ ZLfw) Sts*

**shows**  $\exists Sts0. Sts = \text{lmap } \text{wo-ghosts-of } Sts0 \wedge \text{full-chain } (\rightsquigarrow\text{ZLf}) Sts0 \wedge \text{done-of } (\text{lhd } Sts0) = \{\}$

*<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:  $B \in \text{Bot-F} \implies$*

*passive.ellems (passive-of (lhd Sts))  $\models \cap \mathcal{G} \{B\} \implies$*

*$\exists B' \in \text{Bot-F}. B' \in \text{formulas-union } (\text{Liminf-zl-fstate } Sts) \text{ and}$*

*fair-ZL-wo-ghosts-complete:  $B \in \text{Bot-F} \implies \text{passive.ellems } (\text{passive-of } (\text{lhd } Sts)) \models \cap \mathcal{G} \{B\} \implies$*

*$\exists i. \text{enat } i < \text{llength } Sts \wedge (\exists B \in \text{Bot-F}. B \in \text{all-formulas-of } (\text{lth } Sts \ i))$*

*<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 G-F-q 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*

*G-F-q :: 'q  $\Rightarrow$  'f  $\Rightarrow$  'g set and*

*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**  $\langle \prec \rangle$  50) +*

**fixes**

*Prec-S :: 'f  $\Rightarrow$  'f  $\Rightarrow$  bool (**infix**  $\langle \prec S \rangle$  50)*

**assumes**

*wfp-Prec-S: wfp ( $\prec S$ ) and*

*transp-Prec-S: transp ( $\prec S$ ) and*

*countable-Inf-between: finite A  $\implies$  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 G-F-q 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**