A Reversible Semantics for Janus

Ivan Lanese[1] Germán Vidal[2]

[1]Olas Team, University of Bologna & Inria
[2]VRAIN, Universitat Politècnica de València

18th Conference on Reversible Computation
July 9–10, 2026, Torino, Italia

Reversible Computation & Janus

Reversible computation is a programming paradigm that allows execution to proceed both forwards and backwards

Janus: a paradigmatic reversible programming language

- imperative language with assignments, conditionals, loops
- execution can be performed deterministically forwards and backwards

$$x \mathrel{+}= 5 \quad \leftrightarrow \quad x \mathrel{-}= 5$$

$$\text{if } (i > 0) \text{ then } i \mathrel{-}= 1 \text{ else } i \mathrel{-}= 1 \text{ fi } (i \geq 0)$$

$$\updownarrow$$

$$\text{if } (i \geq 0) \text{ then } i \mathrel{+}= 1 \text{ else } i \mathrel{+}= 1 \text{ fi } (i > 0)$$

Reversible Computation & Janus

Reversible computation is a programming paradigm that allows execution to proceed both forwards and backwards

Janus: a paradigmatic reversible programming language

- imperative language with assignments, conditionals, loops
- execution can be performed deterministically forwards and backwards

$$x \mathrel{+}= 5 \quad \leftrightarrow \quad x \mathrel{-}= 5$$

$$\text{if } (i > 0) \text{ then } i \mathrel{-}= 1 \text{ else } i \mathrel{-}= 1 \text{ fi } (i \geq 0)$$

$$\updownarrow$$

$$\text{if } (i \geq 0) \text{ then } i \mathrel{+}= 1 \text{ else } i \mathrel{+}= 1 \text{ fi } (i > 0)$$

A statement inverter has been defined [Yokoyama, Glück, PEPM07]:

$$\begin{array}{rcl}
\mathcal{I}_{op}[\![+]\!] ::= - \quad \mathcal{I}_{op}[\![-]\!] & ::= & + \quad \mathcal{I}_{op}[\![\hat{}]\!] ::= \hat{} \\
\mathcal{I}[\![x \oplus = e]\!] & ::= & x\ \mathcal{I}_{op}[\![\oplus]\!] = e \\
\mathcal{I}[\![x[e_1] \oplus = e_2]\!] & ::= & x[e_1]\ \mathcal{I}_{op}[\![\oplus]\!] = e_2 \\
\mathcal{I}[\![\text{if } e_1 \text{ then } s_1 \text{ else } s_2 \text{ fi } e_2]\!] & ::= & \text{if } e_2 \text{ then } \mathcal{I}[\![s_1]\!] \text{ else } \mathcal{I}[\![s_2]\!] \text{ fi } e_1 \\
\mathcal{I}[\![\text{from } e_1 \text{ do } s_1 \text{ loop } s_2 \text{ until } e_2]\!] & ::= & \text{from } e_2 \text{ do } \mathcal{I}[\![s_1]\!] \text{ loop } \mathcal{I}[\![s_2]\!] \text{ until } e_1 \\
\mathcal{I}[\![\text{call } id]\!] & ::= & \text{uncall } id \\
\mathcal{I}[\![\text{uncall } id]\!] & ::= & \text{call } id \\
\mathcal{I}[\![\text{skip}]\!] & ::= & \text{skip} \\
\mathcal{I}[\![s_1\ s_2]\!] & ::= & \mathcal{I}[\![s_2]\!]\ \mathcal{I}[\![s_1]\!] \\
\mathcal{I}[\![\text{procedure } id\ s]\!] & ::= & \text{procedure } id^{-1}\ \mathcal{I}[\![s]\!]
\end{array}$$

such that:

$$\sigma \vdash s \Downarrow \sigma' \ \text{ iff } \ \sigma' \vdash \mathcal{I}[\![s]\!] \Downarrow \sigma$$

Motivation

Janus has a well-established big-step semantics [Yokoyama, Glück, PEPM07]

- simple, intuitive, good for proving formal properties, ...
- but worse for analyzing nontermination, exposing intermediate steps, debugging, concurrency, etc

A small-step semantics was recently introduced [Lami,Lanese,Stefani, RC24]

- configurations include the statement to be executed
- uses contexts to identify the next statement

Drawback: The small-step semantics is irreversible!

E.g., if e_1 then s_1 else s_2 fi e_2 reduces to s_1 when e_1 is true

Motivation

Janus has a well-established big-step semantics [Yokoyama, Glück, PEPM07]

- simple, intuitive, good for proving formal properties, ...
- but worse for analyzing nontermination, exposing intermediate steps, debugging, concurrency, etc

A small-step semantics was recently introduced [Lami,Lanese,Stefani, RC24]

- configurations include the statement to be executed
- uses contexts to identify the next statement

Drawback: The small-step semantics is irreversible!

E.g., if e_1 then s_1 else s_2 fi e_2 reduces to s_1 when e_1 is true

Motivation

Janus has a well-established big-step semantics [Yokoyama, Glück, PEPM07]

- simple, intuitive, good for proving formal properties, ...
- but worse for analyzing nontermination, exposing intermediate steps, debugging, concurrency, etc

A small-step semantics was recently introduced [Lami,Lanese,Stefani, RC24]

- configurations include the statement to be executed
- uses contexts to identify the next statement

Drawback: The small-step semantics is irreversible!

E.g., if e_1 then s_1 else s_2 fi e_2 reduces to s_1 when e_1 is true

Our goal

Define a reversible small-step semantics for Janus

Why?

- program understanding
  - reversing an statement vs executing this statement backwards
- satisfy the "loop lemma" (any reduction has an inverse)
- reversible debugging
- ...

Our goal

Define a reversible small-step semantics for Janus

Why?

- program understanding
  - reversing an statement vs executing this statement backwards
- satisfy the "loop lemma" (any reduction has an inverse)
- reversible debugging
- ...

How to make small-step semantics reversible?

Option 1: Add a "history" to configurations

- expensive memory usage
- not strictly necessary since Janus itself is reversible!

Option 2 (Our Approach):

- statement $\rightarrow$ "program counter" (PC)

(very few examples of high-level lang. semantics based on a PC)

Move from $\langle \sigma, s, \pi \rangle$ to $\langle \sigma, \ell_{last}, \ell_{next}, \pi \rangle$ where

- ℓ_{last}: last executed statement
- ℓ_{next}: next statement (program counter)

How to make small-step semantics reversible?

Option 1: Add a "history" to configurations

- expensive memory usage
- not strictly necessary since Janus itself is reversible!

Option 2 (Our Approach):

- statement $\rightarrow$ "program counter" (PC)

(very few examples of high-level lang. semantics based on a PC)

Move from $\langle \sigma, s, \pi \rangle$ to $\langle \sigma, \ell_{last}, \ell_{next}, \pi \rangle$ where

- ℓ_{last}: last executed statement
- ℓ_{next}: next statement (program counter)

Some auxiliary notions

To manage the program counter, we label elementary blocks (conditions, assignments, calls, etc):

$$\text{if } [e_1]^{\ell_1} \text{ then } s_1 \text{ else } s_2 \text{ fi } [e_2]^{\ell_2}$$

We use functions: $entry(s)$, $exit(s)$, and $flow(s)$ to extract the CFG

Some auxiliary notions

To manage the program counter, we label elementary blocks (conditions, assignments, calls, etc):

$$\text{if } [e_1]^{\ell_1} \text{ then } s_1 \text{ else } s_2 \text{ fi } [e_2]^{\ell_2}$$

We use functions: $\mathit{entry}(s)$, $\mathit{exit}(s)$, and $\mathit{flow}(s)$ to extract the CFG

Example

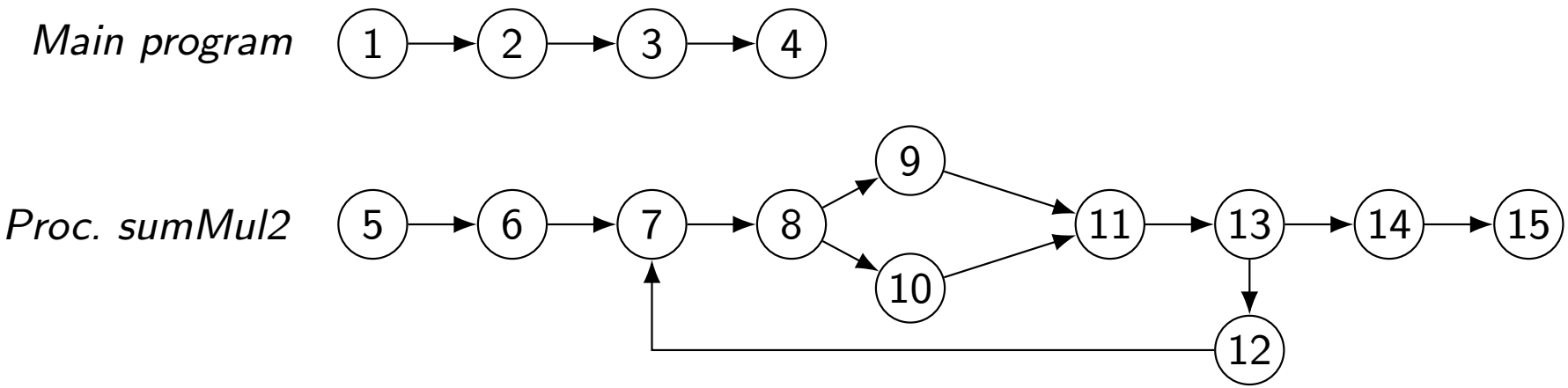
Sum2: a program computing the sum of the multiples of 2 up to n

```
[start]1
[n += 2]2
[call sumMul2]3
[stop]4
```

```
procedure sumMul2
    [start]5
    [i += 1]6
    from [i == 1]7 do
      if [(i % 2) == 0]8 then [total += i]9
                         else [skip]10
      fi [(i % 2) == 0]11
    loop
      [i += 1]12
    until [i >= n]13
    [n += total]14
    [stop]15
```

Example

Sum2: a program computing the sum of the multiples of 2 up to n



Note: no explicit call/return edges in the CFG

- these are handled dynamically using the stack

An irreversible small-step semantics

We first define a small-step semantics

- use stacks for storing continuations
- equivalent to previous semantics (big-step and small-step)
- irreversible

E.g., for procedure calls:

$$\textsc{Call}\ \frac{}{\langle \sigma, \text{call } id, \pi\rangle \to \langle \sigma, \Gamma(id), \pi\rangle}$$

E.g., for conditionals:

$$\textsc{IfTrue1}\ \frac{\sigma \vdash e_1 \Downarrow v_1 \qquad \text{is\_true?}(v_1)}{\langle \sigma, \text{if } e_1 \text{ then } s_1 \text{ else } s_2 \text{ fi } e_2, \pi\rangle \to \langle \sigma, s_1, \text{if\_true}(e_2) : \pi\rangle}$$

$$\textsc{IfTrue2}\ \frac{\sigma \vdash e_2 \Downarrow v_2 \qquad \text{is\_true?}(v_2)}{\langle \sigma, \text{skip}, \text{if\_true}(e_2) : \pi\rangle \to \langle \sigma, \text{skip}, \pi\rangle}$$

An irreversible small-step semantics

We first define a small-step semantics

- use stacks for storing continuations
- equivalent to previous semantics (big-step and small-step)
- irreversible

E.g., for procedure calls:

$$\textsc{Call}\ \frac{}{\langle \sigma, \mathsf{call}\ id, \pi \rangle \to \langle \sigma, \Gamma(id), \pi \rangle}$$

E.g., for conditionals:

$$\textsc{IfTrue1}\ \frac{\sigma \vdash e_1 \Downarrow v_1 \qquad \mathsf{is\_true?}(v_1)}{\langle \sigma, \mathsf{if}\ e_1\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}\ e_2, \pi \rangle \to \langle \sigma, s_1, \mathsf{if\_true}(e_2) : \pi \rangle}$$

$$\textsc{IfTrue2}\ \frac{\sigma \vdash e_2 \Downarrow v_2 \qquad \mathsf{is\_true?}(v_2)}{\langle \sigma, \mathsf{skip}, \mathsf{if\_true}(e_2) : \pi \rangle \to \langle \sigma, \mathsf{skip}, \pi \rangle}$$

An irreversible small-step semantics

We first define a small-step semantics

- use stacks for storing continuations
- equivalent to previous semantics (big-step and small-step)
- irreversible

E.g., for procedure calls:

$$\textsc{Call}\ \frac{}{\langle \sigma, \mathsf{call}\ id, \pi\rangle \to \langle \sigma, \Gamma(id), \pi\rangle}$$

E.g., for conditionals:

$$\textsc{IfTrue1}\ \frac{\sigma \vdash e_1 \Downarrow v_1 \qquad \mathsf{is\_true?}(v_1)}{\langle \sigma, \mathsf{if}\ e_1\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}\ e_2, \pi\rangle \to \langle \sigma, s_1, \mathsf{if\_true}(e_2) : \pi\rangle}$$

$$\textsc{IfTrue2}\ \frac{\sigma \vdash e_2 \Downarrow v_2 \qquad \mathsf{is\_true?}(v_2)}{\langle \sigma, \mathsf{skip}, \mathsf{if\_true}(e_2) : \pi\rangle \to \langle \sigma, \mathsf{skip}, \pi\rangle}$$

From an irreversible small-step semantics to a reversible forward small-step semantics...

Main idea:

- statement $\rightarrow$ "program counter" (PC)

Move from $\langle \sigma, s, \pi \rangle$ to $\langle \sigma, \ell_{last}, \ell_{next}, \pi \rangle$ where

- ℓ_{last}: last executed statement
- ℓ_{next}: next statement (program counter)

Reversible Forward Semantics ($\rightharpoonup$)

E.g., for procedure calls:

$$\overrightarrow{\textit{Call}}\ \frac{\textit{block}(\ell') = (\text{call } \textit{id}) \qquad \textit{entry}(\Gamma(\textit{id})) \longrightarrow \ell'' \in \textit{CFG}}{\langle \sigma, \ell, \ell', \pi \rangle \rightharpoonup \langle \sigma, \textit{entry}(\Gamma(\textit{id})), \ell'', \text{call}(\ell') : \pi \rangle}$$

$$\overrightarrow{\textit{Return}}\ \frac{\textit{block}(\ell') = \text{stop} \qquad \ell'' \longrightarrow \ell''' \in \textit{CFG}}{\langle \sigma, \ell, \ell', \text{call}(\ell'') : \pi \rangle \rightharpoonup \langle \sigma, \ell'', \ell''', \pi \rangle}$$

E.g., for conditionals:

$$\overrightarrow{\textit{IfTrue1}}\ \frac{\begin{array}{c}\text{ctx}(\ell_1) = (\text{if } [e_1]^{\ell_1} \text{ then } s_1 \text{ else } s_2 \text{ fi } [e_2]^{\ell_2}) \\ \sigma \vdash e_1 \Downarrow v_1 \qquad \text{is\_true?}(v_1)\end{array}}{\langle \sigma, \ell, \ell_1, \pi \rangle \rightharpoonup \langle \sigma, \ell_1, \textit{entry}(s_1), \text{if\_true}(\ell_1, \ell_2) : \pi \rangle}$$

$$\overrightarrow{\textit{IfTrue2}}\ \frac{\textit{block}(\ell_2) = e_2 \qquad \sigma \vdash e_2 \Downarrow v_2 \qquad \text{is\_true?}(v_2) \qquad \ell_2 \longrightarrow \ell_2' \in \textit{CFG}}{\langle \sigma, \ell, \ell_2, \text{if\_true}(\ell_1, \ell_2) : \pi \rangle \rightharpoonup \langle \sigma, \ell_2, \ell_2', \pi \rangle}$$

Reversible Forward Semantics ($\rightharpoonup$)

E.g., for procedure calls:

$$\overrightarrow{\textit{Call}}\ \frac{\textit{block}(\ell') = (\text{call } \textit{id}) \qquad \textit{entry}(\Gamma(\textit{id})) \longrightarrow \ell'' \in \textit{CFG}}{\langle \sigma, \ell, \ell', \pi \rangle \rightharpoonup \langle \sigma, \textit{entry}(\Gamma(\textit{id})), \ell'', \text{call}(\ell') : \pi \rangle}$$

$$\overrightarrow{\textit{Return}}\ \frac{\textit{block}(\ell') = \text{stop} \qquad \ell'' \longrightarrow \ell''' \in \textit{CFG}}{\langle \sigma, \ell, \ell', \text{call}(\ell'') : \pi \rangle \rightharpoonup \langle \sigma, \ell'', \ell''', \pi \rangle}$$

E.g., for conditionals:

$$\overrightarrow{\textit{IfTrue1}}\ \frac{\begin{array}{c}\text{ctx}(\ell_1) = (\text{if } [e_1]^{\ell_1} \text{ then } s_1 \text{ else } s_2 \text{ fi } [e_2]^{\ell_2}) \\ \sigma \vdash e_1 \Downarrow v_1 \qquad \text{is\_true?}(v_1)\end{array}}{\langle \sigma, \ell, \ell_1, \pi \rangle \rightharpoonup \langle \sigma, \ell_1, \textit{entry}(s_1), \text{if\_true}(\ell_1, \ell_2) : \pi \rangle}$$

$$\overrightarrow{\textit{IfTrue2}}\ \frac{\textit{block}(\ell_2) = e_2 \qquad \sigma \vdash e_2 \Downarrow v_2 \qquad \text{is\_true?}(v_2) \qquad \ell_2 \longrightarrow \ell_2' \in \textit{CFG}}{\langle \sigma, \ell, \ell_2, \text{if\_true}(\ell_1, \ell_2) : \pi \rangle \rightharpoonup \langle \sigma, \ell_2, \ell_2', \pi \rangle}$$

From a reversible forward semantics to a reversible backward semantics...

- now consider pointer to last executed statement

$$\langle \sigma, \ell, \ell', \pi \rangle \Rightarrow \langle \sigma, \ell, \ell', \pi \rangle$$

- Inverse CFG: $\boxed{CFG \Rightarrow CFG^{-1}}$

- Effects of rules are swapped, e.g.,
  - $\overrightarrow{Call} \approx \overleftarrow{Return}$,
  - $\overrightarrow{Return} \approx \overleftarrow{Call}$,
  - $\overrightarrow{IfTrue1} \approx \overleftarrow{IfTrue2}$,
  - ...

Reversible Backward Semantics ($\leftharpoondown$)

E.g., for procedure calls:

$$\overleftarrow{\textit{Call}}\ \frac{\textit{block}(\ell_s) = \text{start} \qquad \ell' \longrightarrow \ell \in \textit{CFG}^{-1}}{\langle \sigma, \ell_s, \ell'', \text{call}(\ell') : \pi \rangle \leftharpoondown \langle \sigma, \ell, \ell', \pi \rangle}$$

$$\overleftarrow{\textit{Return}}\ \frac{\textit{block}(\ell'') = (\text{call } \textit{id}) \qquad \textit{exit}(\Gamma(\textit{id})) \longrightarrow \ell \in \textit{CFG}^{-1}}{\langle \sigma, \ell'', \ell''', \pi \rangle \leftharpoondown \langle \sigma, \ell, \textit{exit}(\Gamma(\textit{id})), \text{call}(\ell'') : \pi \rangle}$$

E.g., for conditionals:

$$\overleftarrow{\textit{IfTrue1}}\ \frac{\textit{block}(\ell_1) = e_1 \quad \sigma \vdash e_1 \Downarrow v_1 \qquad \text{is\_true?}(v_1) \qquad \ell_1 \longrightarrow \ell \in \textit{CFG}^{-1}}{\langle \sigma, \ell_1, \ell_1', \text{if\_true}(\ell_1, \ell_2) : \pi \rangle \leftharpoondown \langle \sigma, \ell, \ell_1, \pi \rangle}$$

$$\overleftarrow{\textit{IfTrue2}}\ \frac{\text{ctx}(\ell_2) = (\text{if } [e_1]^{\ell_1} \text{ then } s_1 \text{ else } s_2 \text{ fi } [e_2]^{\ell_2}) \quad \sigma \vdash e_2 \Downarrow v_2 \qquad \text{is\_true?}(v_2)}{\langle \sigma, \ell_2, \ell_2', \pi \rangle \leftharpoondown \langle \sigma, \textit{exit}(s_1), \ell_2, \text{if\_true}(\ell_1, \ell_2) : \pi \rangle}$$

Reversible Backward Semantics ($\leftharpoondown$)

E.g., for procedure calls:

$$\overleftarrow{\textit{Call}}\ \frac{\textit{block}(\ell_s) = \mathsf{start} \qquad \ell' \longrightarrow \ell \in \textit{CFG}^{-1}}{\langle \sigma, \ell_s, \ell'', \mathsf{call}(\ell') : \pi \rangle \leftharpoondown \langle \sigma, \ell, \ell', \pi \rangle}$$

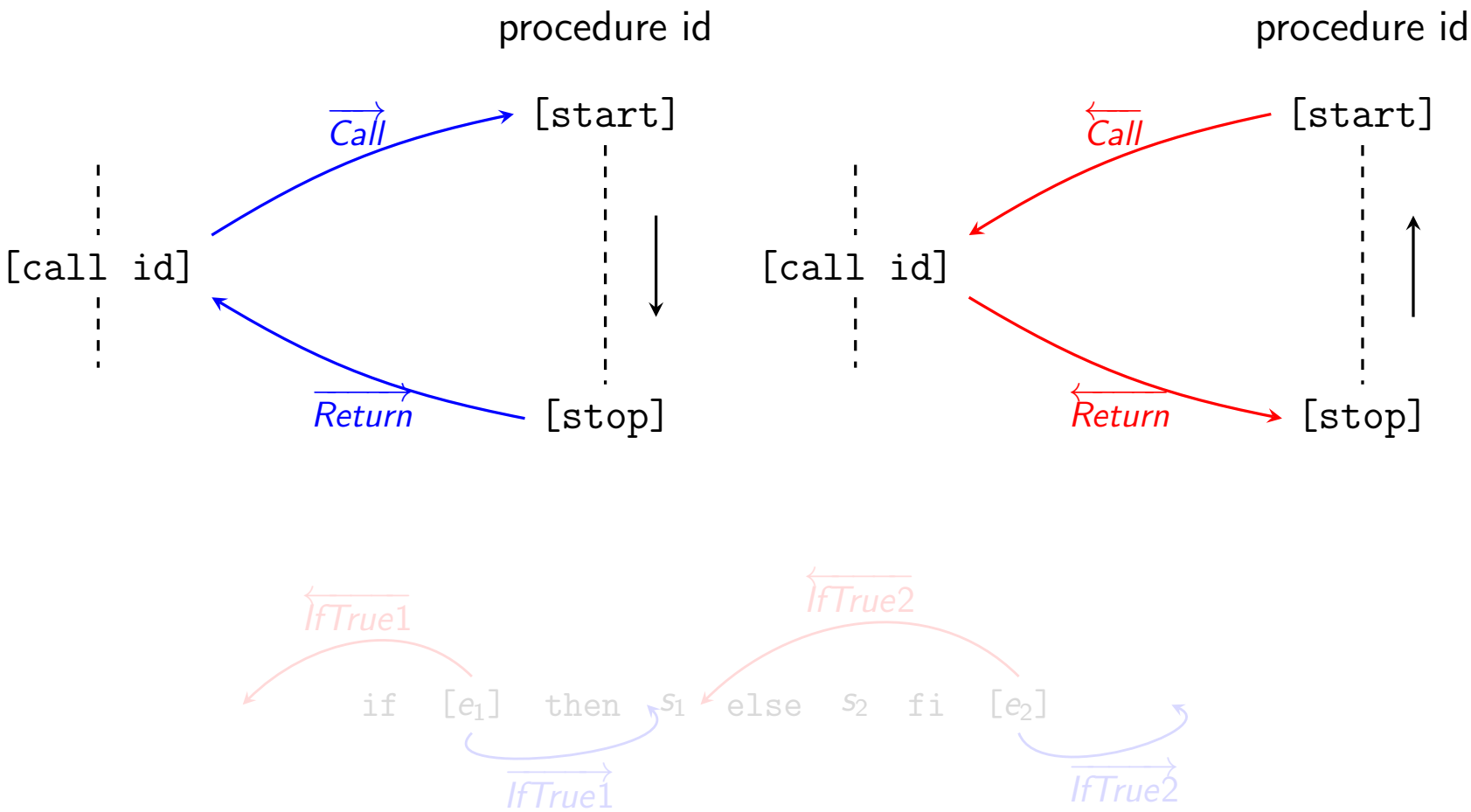
$$\overleftarrow{\textit{Return}}\ \frac{\textit{block}(\ell'') = (\mathsf{call}\ \textit{id}) \qquad \textit{exit}(\Gamma(\textit{id})) \longrightarrow \ell \in \textit{CFG}^{-1}}{\langle \sigma, \ell'', \ell''', \pi \rangle \leftharpoondown \langle \sigma, \ell, \textit{exit}(\Gamma(\textit{id})), \mathsf{call}(\ell'') : \pi \rangle}$$

E.g., for conditionals:

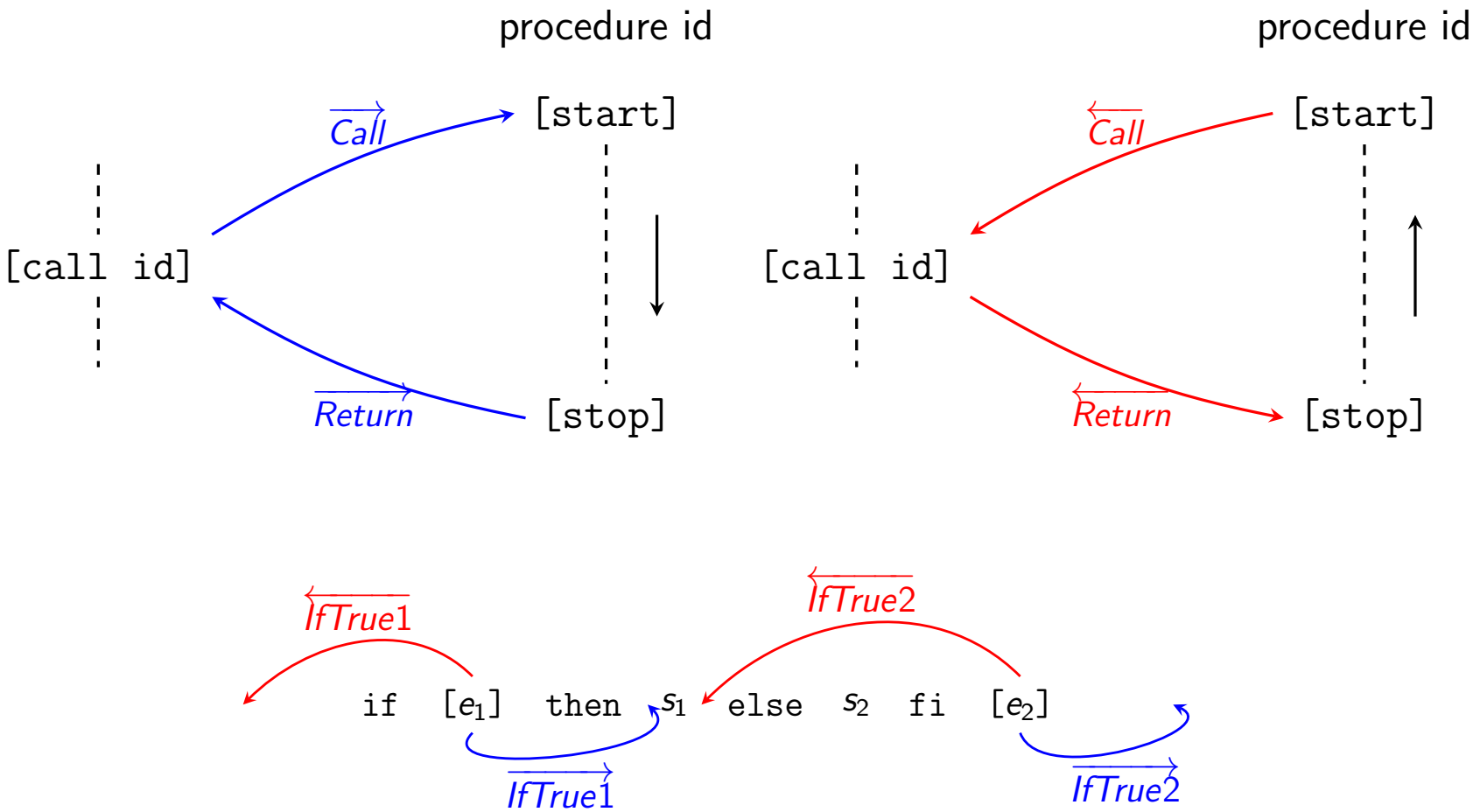
$$\overleftarrow{\textit{IfTrue1}}\ \frac{\begin{array}{c}\textit{block}(\ell_1) = e_1 \\ \sigma \vdash e_1 \Downarrow v_1 \qquad \mathsf{is\_true?}(v_1) \qquad \ell_1 \longrightarrow \ell \in \textit{CFG}^{-1}\end{array}}{\langle \sigma, \ell_1, \ell_1', \mathsf{if\_true}(\ell_1, \ell_2) : \pi \rangle \leftharpoondown \langle \sigma, \ell, \ell_1, \pi \rangle}$$

$$\overleftarrow{\textit{IfTrue2}}\ \frac{\begin{array}{c}\mathsf{ctx}(\ell_2) = (\mathsf{if}\ [e_1]^{\ell_1}\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}\ [e_2]^{\ell_2}) \\ \sigma \vdash e_2 \Downarrow v_2 \qquad \mathsf{is\_true?}(v_2)\end{array}}{\langle \sigma, \ell_2, \ell_2', \pi \rangle \leftharpoondown \langle \sigma, \textit{exit}(s_1), \ell_2, \mathsf{if\_true}(\ell_1, \ell_2) : \pi \rangle}$$

Reversible Backward Semantics ($\leftarrow$)



Reversible Backward Semantics ($\leftarrow$)



Some properties

soundness and completeness

Let s be a program with $\mathit{entry}(s) \longrightarrow \ell \in \mathit{CFG}$. Then,

$$\langle \epsilon, s, [\,] \rangle \rightarrow^* \langle \sigma, \mathsf{skip}, [\,] \rangle$$

if and only if

$$\langle \epsilon, \mathit{entry}(s), \ell, [\,] \rangle \rightharpoonup^* \langle \sigma, \ell', \ell'', [\,] \rangle \quad \text{with } \mathit{block}(\ell'') = \mathsf{stop}$$

loop lemma

$$\langle \sigma, \ell_1, \ell_2, \pi \rangle \rightharpoonup \langle \sigma', \ell_2, \ell_3, \pi' \rangle \quad \text{iff} \quad \langle \sigma', \ell_2, \ell_3, \pi' \rangle \leftharpoondown \langle \sigma, \ell_1, \ell_2, \pi \rangle$$

Concluding Remarks

Contributions:

- introduced a reversible semantics for Janus (based on the use of a "program counter")
- proved equivalence and loop lemma

Future work:

- dealing with errors
- defining a reversible debugger for Janus

Concluding Remarks

Contributions:

- introduced a reversible semantics for Janus
  (based on the use of a "program counter")
- proved equivalence and loop lemma

Future work:

- dealing with errors
- defining a reversible debugger for Janus

Thanks for your attention!

Questions?