Towards Small-step Compilation Schemas for SOS

Ferdinand Vesely

Department of Computer Science
Swansea University

27th Nordic Workshop on Programming Theory

October 21-23, 2015
Reykjavík, Iceland

What?

Motivation

What?

Motivation

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \textbf{print}(t) \xrightarrow{\ell} \textbf{print}(t')}$$

$$\frac{\text{Value}\, v}{\rho \vdash \textbf{print}(v) \xrightarrow{\text{out } v} v}$$

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \textbf{if}(e, s, t) \xrightarrow{\ell} \textbf{if}(e', s, t)}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{true}, s, t) \xrightarrow{\tau} s}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{false}, s, t) \xrightarrow{\tau} t}$$

What?

Motivation

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \textbf{print}(t) \xrightarrow{\ell} \textbf{print}(t')}$$

$$\frac{\text{Value}\, v}{\rho \vdash \textbf{print}(v) \xrightarrow{\text{out } v} v}$$

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \textbf{if}(e, s, t) \xrightarrow{\ell} \textbf{if}(e', s, t)}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{true}, s, t) \xrightarrow{\tau} s}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{false}, s, t) \xrightarrow{\tau} t}$$

$$[\![\textbf{print}(\ldots)]\!] = \ldots$$

$$[\![\textbf{if}(\ldots)]\!] = \ldots$$

What?

Motivation

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \mathbf{print}(t) \xrightarrow{\ell} \mathbf{print}(t')}$$

$$\frac{\mathsf{Value}\, v}{\rho \vdash \mathbf{print}(v) \xrightarrow{\text{out } v} v}$$

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \mathbf{if}(e, s, t) \xrightarrow{\ell} \mathbf{if}(e', s, t)}$$

$$\frac{}{\rho \vdash \mathbf{if}(\mathbf{true}, s, t) \xrightarrow{\tau} s}$$

$$\frac{}{\rho \vdash \mathbf{if}(\mathbf{false}, s, t) \xrightarrow{\tau} t}$$

$$\mathbf{if}(\mathbf{true}, \mathbf{print}(\text{"a"}), \mathbf{print}(\text{"b"}))$$

$$[\![\mathbf{print}(\ldots)]\!] = \ldots$$

$$[\![\mathbf{if}(\ldots)]\!] = \ldots$$

What?

Motivation

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \textbf{print}(t) \xrightarrow{\ell} \textbf{print}(t')}$$

$$\frac{\textsf{Value}\ v}{\rho \vdash \textbf{print}(v) \xrightarrow{\text{out } v} v}$$

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \textbf{if}(e, s, t) \xrightarrow{\ell} \textbf{if}(e', s, t)}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{true}, s, t) \xrightarrow{\tau} s}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{false}, s, t) \xrightarrow{\tau} t}$$

$$\textbf{if}(\textbf{true}, \textbf{print}(\text{"a"}), \textbf{print}(\text{"b"}))$$

$$[\![\textbf{print}(\ldots)]\!] = \ldots$$
$$[\![\textbf{if}(\ldots)]\!] = \ldots$$

```
if0:
  %tmp0 ← 1
  case %tmp0 2
  jump if1_0
  jump if1_1
  halt
if1_0:
  %tmp1 ← "b"
  out %tmp1
  jump tmp3
if1_1:
  %tmp2 ← "a"
  out %tmp2
  jump tmp3
tmp3:
  %tmp3 ←
      phi %tmp1 %tmp2
  halt
```

How?

Overview of the idea

- translate steps of open terms
- 1 abstract state $\approx$ 1 block
- *atomic* blocks
- block might contain many instructions
- terminated by jumps or a halt
- non-determinism in semantics – non-deterministic schema

Basic Examples

Printing stuff

Consider **print**:

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \mathbf{print}(t) \xrightarrow{\ell} \mathbf{print}(t')} \qquad \frac{\mathsf{Value}\, v}{\rho \vdash \mathbf{print}(v) \xrightarrow{\mathbf{out}\ v} v}$$

If there is a sequence of n transitions for a term t (actually $\langle \rho, t \rangle$):

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

then the computation of **print**(t) proceeds as follows:

$$\mathbf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathbf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathbf{print}(v) \xrightarrow{\mathbf{out}\ v} v$$

Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$

$$\boxed{?} \xrightarrow{[\![\ell_1]\!]} \cdots \xrightarrow{[\![\ell_{n-1}]\!]} \boxed{?} \xrightarrow{[\![\ell_n]\!]} \boxed{?} \xrightarrow{[\![\textbf{out}\ v]\!]} \boxed{?}$$

Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$

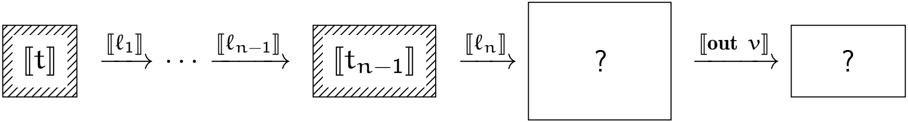
$$\boxed{[\![t]\!]} \xrightarrow{[\![\ell_1]\!]} \cdots \xrightarrow{[\![\ell_{n-1}]\!]} \boxed{[\![t_{n-1}]\!]} \xrightarrow{[\![\ell_n]\!]} \boxed{?} \xrightarrow{[\![\textbf{out}\ v]\!]} \boxed{?}$$

Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$

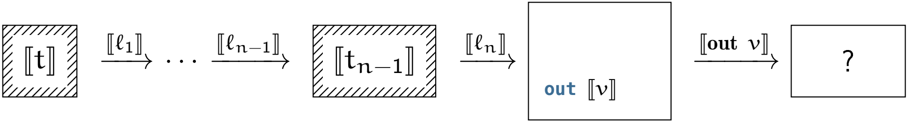


Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$

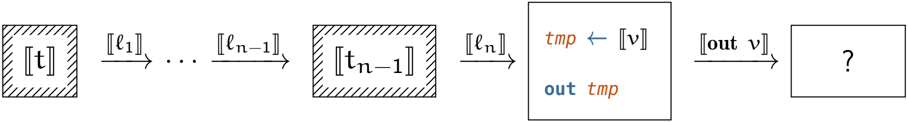


Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\mathbf{out}\ v} v$$

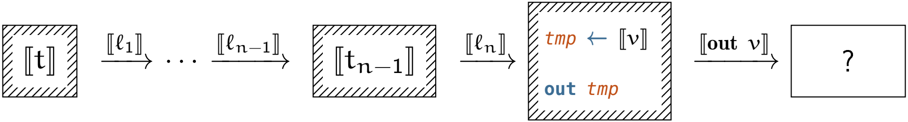


Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$

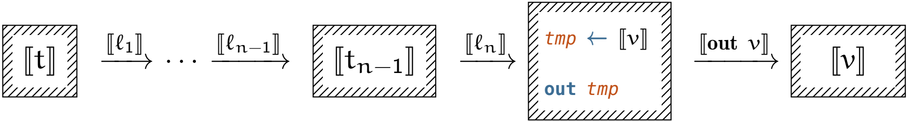


Basic Examples

Printing stuff

$$t \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} t_{n-1} \xrightarrow{\ell_n} v$$

$$\mathsf{print}(t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_{n-1}} \mathsf{print}(t_{n-1}) \xrightarrow{\ell_n} \mathsf{print}(v) \xrightarrow{\textbf{out}\ v} v$$



Basic Examples

Schema for print

Schema defined in terms of *translators*:

- operations: code, next, label, jumps
- translators have states
  - correspond to one or more states of SOS computation
- e.g., translator for **print**: $\text{tr}_{\textbf{print}}$
- translator state: $[\![\textbf{print}(t)]\!] = \text{tr}_{\textbf{print}}([\![t]\!])$

Basic Examples

Schema for print

$$\frac{\rho \vdash t \xrightarrow{\ell} t'}{\rho \vdash \mathbf{print}(t) \xrightarrow{\ell} \mathbf{print}(t')}$$

$$\frac{\mathsf{Value}\, v}{\rho \vdash \mathbf{print}(v) \xrightarrow{\mathbf{out}\ v} v}$$

$$\mathsf{code}[\![\mathbf{print}(t)]\!] = \begin{cases} \mathsf{code}[\![t]\!] & \text{if } \mathsf{next}[\![t]\!] \neq \mathsf{none} \\ \mathsf{code}[\![t]\!] \cdot \mathbf{out}\ \mathsf{label}[\![t]\!] & \text{if } \mathsf{next}[\![t]\!] = \mathsf{none} \end{cases}$$

$$\mathsf{next}[\![\mathbf{print}(t)]\!] = \begin{cases} \mathsf{tr}_{\mathbf{print}}(\mathsf{next}[\![t]\!]) & \text{if } \mathsf{next}[\![t]\!] \neq \mathsf{none} \\ [\![t]\!] & \text{if } \mathsf{next}[\![t]\!] = \mathsf{none} \end{cases}$$

Basic Examples

Let-bindings

$$\frac{\rho \vdash t_1 \xrightarrow{\ell} t_1'}{\rho \vdash \mathbf{let}(i, t_1, t_2) \xrightarrow{\ell} \mathbf{let}(i, t_1', t_2)}$$

$$\frac{\text{Value } v_1 \qquad \rho[i \mapsto v_1] \vdash t_2 \xrightarrow{\ell} t_2'}{\rho \vdash \mathbf{let}(i, v_1, t_2) \xrightarrow{\ell} \mathbf{let}(i, v_1, t_2')}$$

$$\frac{\text{Value } v_1 \qquad \text{Value } v_2}{\rho \vdash \mathbf{let}(i, v_1, v_2) \xrightarrow{\tau} v_2}$$

Basic Examples

Let-bindings

$$\text{code}[\![\textbf{let}(i, t_1, t_2)]\!] =$$

Basic Examples

Let-bindings

$\text{code}[\![\textbf{let}(i, t_1, t_2)]\!] =$

1. $\text{code}[\![t_1]\!]$
   - if $\text{next}[\![t_1]\!] \neq \text{none}$

$$\frac{\rho \vdash t_1 \xrightarrow{\ell} t_1'}{\rho \vdash \textbf{let}(i, t_1, t_2) \xrightarrow{\ell} \textbf{let}(i, t_1', t_2)}$$

Basic Examples

Let-bindings

$\text{code}[\![\textbf{let}(i, t_1, t_2)]\!] =$

$$\frac{\text{Value}\ \ v_1 \qquad \rho[i \mapsto v_1] \vdash t_2 \xrightarrow{\ell} t_2'}{\rho \vdash \textbf{let}(i, v_1, t_2) \xrightarrow{\ell} \textbf{let}(i, v_1, t_2')}$$

1. $\text{code}[\![t_1]\!]$
   - if $\text{next}[\![t_1]\!] \neq \text{none}$
2. $\text{code}\ tr_{iv} \cdot$ **push_env** $tmp \cdot \text{code}[\![t_2]\!] \cdot$ **pop_env**
   - if $\text{next}[\![t_1]\!] = \text{none}$ and $\text{next}[\![t_2]\!] \neq \text{none}$,
   - $tr_{iv} = [\![\{i \mapsto t_1\}]\!]$,
   - $tmp = \text{label}(tr_{iv})$

Basic Examples

Let-bindings

$\text{code}[\![\textbf{let}(i, t_1, t_2)]\!] =$

$$\frac{\text{Value } v_1 \qquad \rho[i \mapsto v_1] \vdash t_2 \xrightarrow{\ell} t_2'}{\rho \vdash \textbf{let}(i, v_1, t_2) \xrightarrow{\ell} \textbf{let}(i, v_1, t_2')}$$

1. $\text{code}[\![t_1]\!]$
   - if $\text{next}[\![t_1]\!] \neq \text{none}$
2. $\text{code}\ tr_{iv} \cdot \textbf{push\_env}\ \ tmp \cdot \text{code}[\![t_2]\!] \cdot \textbf{pop\_env}$
   - if $\text{next}[\![t_1]\!] = \text{none}$ and $\text{next}[\![t_2]\!] \neq \text{none}$,
   - $tr_{iv} = [\![\{i \mapsto t_1\}]\!]$,
   - $tmp = \text{label}(tr_{iv})$

name of temporary holding the binding

Basic Examples

Let-bindings

$\text{code}[\![\textbf{let}(i, t_1, t_2)]\!] =$

1. $\text{code}[\![t_1]\!]$
   - if $\text{next}[\![t_1]\!] \neq \text{none}$

$$\frac{\text{Value}\ v_1 \qquad \text{Value}\ v_2}{\rho \vdash \textbf{let}(i, v_1, v_2) \xrightarrow{\tau} v_2}$$

2. $\text{code}\ tr_{iv} \cdot \textbf{push\_env}\ \ tmp \cdot \text{code}[\![t_2]\!] \cdot \textbf{pop\_env}$
   - if $\text{next}[\![t_1]\!] = \text{none}$ and $\text{next}[\![t_2]\!] \neq \text{none}$,
   - $tr_{iv} = [\![\{i \mapsto t_1\}]\!]$,
   - $tmp = \text{label}(tr_{iv})$

name of temporary holding the binding

3. ε
   - if $\text{next}[\![t_1]\!] \neq \text{none}$ and $\text{next}[\![t_2]\!] \neq \text{none}$
   - also: $\text{next}[\![\textbf{let}(i, t_1, t_2)]\!] = [\![t_2]\!]$ and $\text{jumps}[\![\textbf{let}(i, t_1, t_2)]\!] = \textbf{jump}\ \ \text{label}[\![t_2]\!]$

Top-level Translator

- translating the top-level phrase
- invoke translator for the outermost construct
- push jumps to the end
  - exit point should be at the end block
- if no jumps – final state – issue **`halt`**

Conditional Branching

- for conditionals – need to translate both branches and join them

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \mathbf{if}(e, s, t) \xrightarrow{\ell} \mathbf{if}(e', s, t)}$$

$$\frac{}{\rho \vdash \mathbf{if}(\mathbf{true}, s, t) \xrightarrow{\tau} s} \qquad \frac{}{\rho \vdash \mathbf{if}(\mathbf{false}, s, t) \xrightarrow{\tau} t}$$

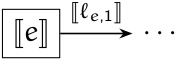
Conditional Branching

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \mathbf{if}(e, s, t) \xrightarrow{\ell} \mathbf{if}(e', s, t)}$$

$[\![e]\!]$

Conditional Branching

$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \mathbf{if}(e, s, t) \xrightarrow{\ell} \mathbf{if}(e', s, t)}$$



Conditional Branching

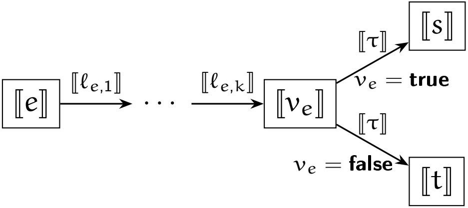
$$\frac{\rho \vdash e \xrightarrow{\ell} e'}{\rho \vdash \mathbf{if}(e, s, t) \xrightarrow{\ell} \mathbf{if}(e', s, t)}$$

$$\boxed{[\![e]\!]} \xrightarrow{[\![\ell_{e,1}]\!]} \cdots \xrightarrow{[\![\ell_{e,k}]\!]} \boxed{[\![v_e]\!]}$$

Conditional Branching

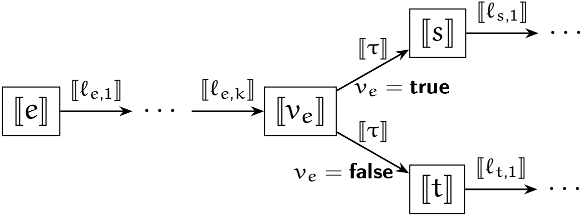
$$\frac{}{\rho \vdash \textbf{if}(\textbf{true}, s, t) \xrightarrow{\tau} s}$$

$$\frac{}{\rho \vdash \textbf{if}(\textbf{false}, s, t) \xrightarrow{\tau} t}$$



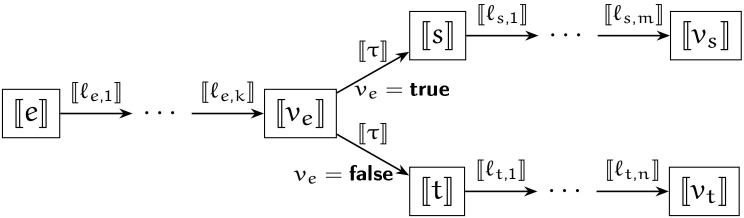
Conditional Branching

$(s \xrightarrow{\ell} s')$ $(t \xrightarrow{\ell} t')$

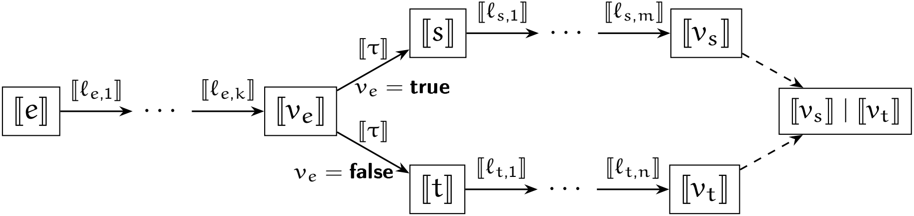


Conditional Branching

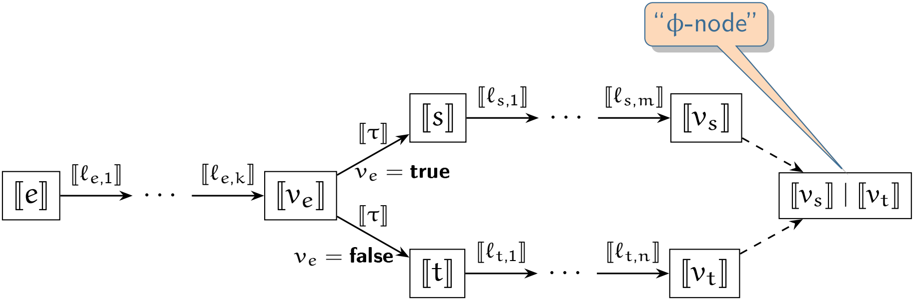
$(s \xrightarrow{\ell} s')$ $(t \xrightarrow{\ell} t')$



Conditional Branching



Conditional Branching



An Example Translation

$$\mathsf{if}(\mathsf{true}, \mathsf{print}(\text{"a"}), \mathsf{print}(\text{"b"}))$$

An Example Translation

$$\textsf{if}(\textsf{true}, \textsf{print}(\text{"a"}), \textsf{print}(\text{"b"}))$$

```
if0:
  %tmp0 ← 1        ; [[true]]
  case %tmp0 2     ; pc ← pc + min(%tmp0, 2) + 1
  jump if1_0       ; [[false]] branch
  jump if1_1       ; [[true]] branch
  halt             ; otherwise: stuck
```

An Example Translation

$$\mathsf{if}(\mathsf{true}, \mathsf{print}(\text{"a"}), \mathsf{print}(\text{"b"}))$$

```
if0:
  %tmp0 ← 1         ; [[true]]
  case %tmp0 2      ; pc ← pc + min(%tmp0, 2) + 1
  jump if1_0        ; [[false]] branch
  jump if1_1        ; [[true]] branch
  halt              ; otherwise: stuck
```

```
if1_0:               ; [[print("b")]]
  %tmp1 ← "b"        ; [["b"]]
  out %tmp1
  jump tmp1

tmp1:                ; [["b"]]
  jump tmp3
```

```
if1_1:               ; [[print("a")]]
  %tmp2 ← "a"        ; [["a"]]
  out %tmp2
  jump tmp2

tmp2:                ; [["a"]]
  jump tmp3
```

An Example Translation

$$\textsf{if}(\textsf{true}, \textsf{print}(\text{"a"}), \textsf{print}(\text{"b"}))$$

```
if0:
  %tmp0 ← 1        ; [[true]]
  case %tmp0 2     ; pc ← pc + min(%tmp0, 2) + 1
  jump if1_0       ; [[false]] branch
  jump if1_1       ; [[true]] branch
  halt             ; otherwise: stuck
```

```
if1_0:             ; [[print("b")]]
  %tmp1 ← "b"      ; [["b"]]
  out %tmp1
  jump tmp1

tmp1:              ; [["b"]]
  jump tmp3
```

```
if1_1:             ; [[print("a")]]
  %tmp2 ← "a"      ; [["a"]]
  out %tmp2
  jump tmp2

tmp2:              ; [["a"]]
  jump tmp3
```

```
tmp3:                           ; [["b"]] | [["a"]]
  %tmp3 ← phi %tmp1 %tmp2
  halt                          ; finished
```

Non-determinism

$$\frac{s \xrightarrow{\ell} s'}{\textbf{inter}(s, t) \xrightarrow{\ell} \textbf{inter}(s', t)} \qquad \frac{t \xrightarrow{\ell} t'}{\textbf{inter}(s, t) \xrightarrow{\ell} \textbf{inter}(s, t')}$$

- compile interleavings:

$$[\![\textbf{inter}(s, t)]\!] = [\![\textbf{inter}(s, t)]\!]_l \text{ OR } [\![\textbf{inter}(s, t)]\!]_r$$

where

$$\text{code}[\![\textbf{inter}(s, t)]\!]_l = \text{code}[\![s]\!] \qquad \text{code}[\![\textbf{inter}(s, t)]\!]_r = \text{code}[\![t]\!]$$

$$\text{next}[\![\textbf{inter}(s, t)]\!]_l = \text{tr}_{\textbf{inter}}(\text{next}[\![s]\!], [\![t]\!]) \qquad \text{next}[\![\textbf{inter}(s, t)]\!]_r = \text{tr}_{\textbf{inter}}([\![s]\!], \text{next}[\![t]\!])$$

Iteration

Need to avoid unfolding loops:

- **while**(b, t) may end up after n steps in **while**(b, t) again:

$$\textbf{while}(b, t) \xrightarrow{\ell_1} \cdots \xrightarrow{\ell_n} \textbf{while}(b, t)$$

- corresponds to a jump back to the first block
- but: loops cannot be interleaved

Some Related Work

- calculation (equational derivation) of compilers (Bahr and Hutton, *JFP*, 2015)
- compilation of Esterel and Joy
  - into hardware circuits: Esterel (Berry, *Sadhana*, 1992), Joy (Weber et al., *REX Workshop*, 1993)
  - into sequential code: Esterel (Edwards, *CODES*, 1999)
  - correctness based on SOS semantics

Summary

- idea: compile small-steps into atomic blocks
- each block: execution corresponds to state transition
- non-deterministic compilation schema
- future work:
  - prototype implementation, correctness, optimisation, automation