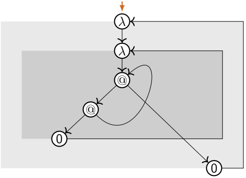
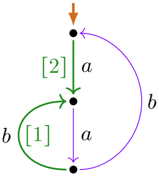
From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes

Clemens Grabmayer
https://clegra.github.io

Department of Computer Science
GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore
L'Aquila, Italy



DCM'23
Sapienza Università di Roma
July 2, 2023



Overview

1. Compactifying λ-terms with `letrec` (maximal sharing of functional programs)
   - higher-order λ-term graphs

2. Recognizing regular-expression processes
   - LEE-witnesses: graph labelings based on a loop–condition LEE

Overview

1. Compactifying λ-terms with `letrec` (maximal sharing of functional programs)
   - from terms in the λ-calculus with letrec to:
     - higher-order λ-term graphs
     - first-order λ-term graphs
     - λ-NFAs, and λ-DFAs
   - minimization / readback / efficiency / Haskell implementation
2. Recognizing regular-expression processes
     - LEE-witnesses: graph labelings based on a loop–condition LEE

Overview

1. Compactifying λ-terms with `letrec` (maximal sharing of functional programs)
    - from terms in the λ-calculus with letrec to:
        - higher-order λ-term graphs
        - first-order λ-term graphs
        - λ-NFAs, and λ-DFAs
    - minimization / readback / efficiency / Haskell implementation
2. Recognizing regular-expression processes
    - Milner's questions, known results
    - structure-constrained process graphs:
        - LEE-witnesses: graph labelings based on a loop–condition LEE
        - preservation under bisimulation collapse
    - readback: from graph labelings to regular expressions

Overview

- Comparison desiderata

1. Compactifying λ-terms with `letrec` (maximal sharing of functional programs)
    - from terms in the λ-calculus with letrec to:
        - higher-order λ-term graphs
        - first-order λ-term graphs
        - λ-NFAs, and λ-DFAs
    - minimization / readback / efficiency / Haskell implementation
2. Recognizing regular-expression processes
    - Milner's questions, known results
    - structure-constrained process graphs:
        - LEE-witnesses: graph labelings based on a loop–condition LEE
        - preservation under bisimulation collapse
    - readback: from graph labelings to regular expressions

- Comparison results

Comparison original desiderata

λ-calculus with letrec under the unfolding semantics

Well-known: graph representations implemented by compilers

- but were not intended for manipulation under $\underline{\leftrightarrow}$

Not well-known: term graph interpretation that is studied under $\underline{\leftrightarrow}$

Regular expressions under process semantics (bisimilarity $\underline{\leftrightarrow}$)

Comparison original desiderata

λ-calculus with letrec under the unfolding semantics

Well-known: graph representations implemented by compilers

- but were not intended for manipulation under $\underline{\leftrightarrow}$

Not well-known: term graph interpretation that is studied under $\underline{\leftrightarrow}$

Desired: term graph interpretation that:

- has natural correspondence with terms in $\boldsymbol{\lambda}_{\text{letrec}}$
- supports compactification under $\underline{\leftrightarrow}$
- permits efficient translation and readback

Regular expressions under process semantics (bisimilarity $\underline{\leftrightarrow}$)

Comparison original desiderata

λ-calculus with letrec under the unfolding semantics

Well-known: graph representations implemented by compilers

- but were not intended for manipulation under $\underline{\leftrightarrow}$

Not well-known: term graph interpretation that is studied under $\underline{\leftrightarrow}$

Desired: term graph interpretation that:

- has natural correspondence with terms in $\boldsymbol{\lambda}_{\text{letrec}}$
- supports compactification under $\underline{\leftrightarrow}$
- permits efficient translation and readback

Regular expressions under process semantics (bisimilarity $\underline{\leftrightarrow}$)

Given: process graph interpretation $P(\cdot)$, studied under $\underline{\leftrightarrow}$

- not closed under $\underline{\rightarrow}$, and $\underline{\leftrightarrow}$, modulo $\underline{\leftrightarrow}$ incomplete

Comparison original desiderata

λ-calculus with letrec under the unfolding semantics

Well-known: graph representations implemented by compilers

- but were not intended for manipulation under $\underline{\leftrightarrow}$

Not well-known: term graph interpretation that is studied under $\underline{\leftrightarrow}$

Desired: term graph interpretation that:

- has natural correspondence with terms in $\boldsymbol{\lambda}_{\text{letrec}}$
- supports compactification under $\underline{\leftrightarrow}$
- permits efficient translation and readback

Regular expressions under process semantics (bisimilarity $\underline{\leftrightarrow}$)

Given: process graph interpretation $P(\cdot)$, studied under $\underline{\leftrightarrow}$

- not closed under $\underline{\to}$, and $\underline{\leftrightarrow}$, modulo $\underline{\leftrightarrow}$ incomplete

Desired: reason with graphs that are $P(\cdot)$-expressible modulo $\underline{\leftrightarrow}$
(at least with 'sufficiently many')

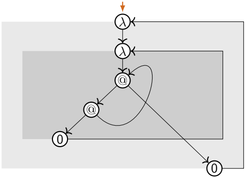
understand incompleteness by a structural graph property

structure constraints (L'Aquila)

Maximal sharing of functional programs

(joint work with Jan Rochel)

$$\begin{tikzcd} L \arrow[r, mapsto, bend left, "\text{interpret}"] & G \arrow[d, Rightarrow, "\text{collapse}"] \\ L_0 & G_0 \arrow[l, mapsto, bend left, "\text{readback}"] \end{tikzcd}$$



Maximal sharing: example (fix)

$$\lambda f.\,\mathsf{let}\ r = f\ (f\ r)\ \mathsf{in}\ r$$

L

Maximal sharing: example (fix)

$$\lambda f.\,\mathsf{let}\ r = f\ (f\ r)\ \mathsf{in}\ r$$

L

L_0

$$\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r$$

Maximal sharing: example (fix)

$$\begin{array}{c} L \\ \text{unfold}\ \big\downarrow \\ M \\ \text{unfold}\ \big\uparrow \\ L_0 \end{array}$$

$$\begin{array}{c} \lambda f.\,\mathsf{let}\ r = f\,(f\ r)\ \mathsf{in}\ r \\ \llbracket\cdot\rrbracket_{\lambda^\infty}\ \big\downarrow \\ \lambda f.\,f\,(f\,(\ldots)) \\ \llbracket\cdot\rrbracket_{\lambda^\infty}\ \big\uparrow \\ \lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r \end{array}$$

Maximal sharing: example (fix)

$$\lambda f.\,\mathsf{let}\ r = f\,(f\ r)\ \mathsf{in}\ r$$

L

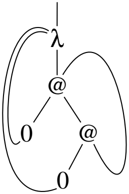
L_0

$$\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r$$

Maximal sharing: example (fix)

$$L \xrightarrow{\text{interpret}} G$$

$$\lambda f.\,\text{let } r = f\,(f\ r) \text{ in } r \xrightarrow{[\![\cdot]\!]_{\mathcal{T}}}$$



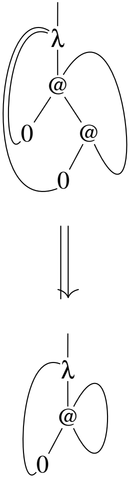
L_0

$$\lambda f.\,\text{let } r = f\ r \text{ in } r$$

Maximal sharing: example (fix)

$$
\begin{tikzcd}
L \arrow[rr, mapsto, bend left, "\text{interpret}"] & & G \arrow[d, Rightarrow, "\text{collapse}"] \\
L_0 & & G_0
\end{tikzcd}
$$

$$\lambda f.\,\mathsf{let}\ r = f\ (f\ r)\ \mathsf{in}\ r \quad \xmapsto{[\![\cdot]\!]_{\mathcal{T}}}$$

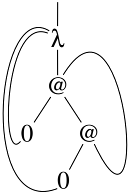


$$\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r$$

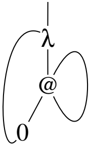
Maximal sharing: example (fix)

$$
\begin{tikzcd}
L \arrow[r, mapsto, bend left, "\text{interpret}"] & G \arrow[d, Rightarrow, "\text{collapse}"] \\
L_0 & G_0 \arrow[l, mapsto, bend left, "\text{readback}"]
\end{tikzcd}
$$

$\lambda f.\,\text{let } r = f\ (f\ r) \text{ in } r \xmapsto{\ [\![\cdot]\!]_{\mathcal{T}}\ }$

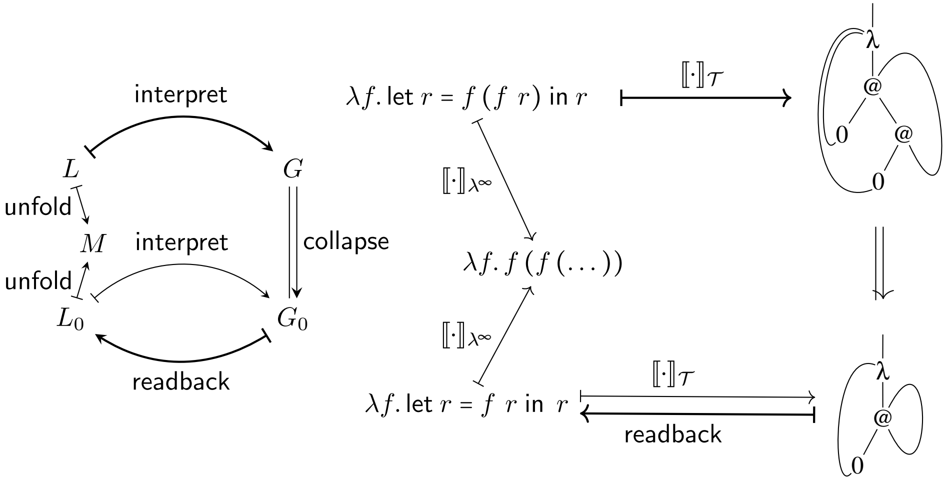


$\Downarrow$



$\lambda f.\,\text{let } r = f\ r \text{ in } r \xleftarrow[\text{readback}]{}$

Maximal sharing: example (fix)



Maximal sharing: the method

$$L \xmapsto{[\![\cdot]\!]_{\mathcal{H}}} \mathcal{G}$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$

Maximal sharing: the method

$$\begin{tikzcd} L \arrow[r, mapsto, "\llbracket\cdot\rrbracket_{\mathcal{H}}"] & \mathcal{G} \arrow[r, mapsto] & G \end{tikzcd}$$

1. term graph interpretation $\llbracket\cdot\rrbracket$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = \llbracket L\rrbracket_{\mathcal{H}}$
   b. first-order term graph $G = \llbracket L\rrbracket_{\mathcal{T}}$

Maximal sharing: the method

$$\begin{tikzcd} L \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & \mathcal{G} \arrow[r, mapsto] & G \end{tikzcd}$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$

Maximal sharing: the method

$$
\begin{tikzcd}
L \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & \mathcal{G} \arrow[r, mapsto] & G \arrow[d, Rightarrow] \\
 & & G_0
\end{tikzcd}
$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G into G_0

Maximal sharing: the method

$$
\begin{tikzcd}
L \arrow[r, mapsto, "\llbracket\cdot\rrbracket_{\mathcal{H}}"] \arrow[rr, mapsto, bend left, "\llbracket\cdot\rrbracket_{\mathcal{T}}"] & \mathcal{G} \arrow[r, mapsto] \arrow[d, Rrightarrow] & G \arrow[d, Rrightarrow] \\
& \mathcal{G}_0 \arrow[r, mapsto] & G_0
\end{tikzcd}
$$

1. term graph interpretation $\llbracket\cdot\rrbracket$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph $\mathcal{G} = \llbracket L\rrbracket_{\mathcal{H}}$
   b. first-order term graph $G = \llbracket L\rrbracket_{\mathcal{T}}$
2. bisimulation collapse $|\!\!\downarrow$
   of f-o term graph G into G_0

Maximal sharing: the method

$$\begin{tikzcd}
L \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & \mathcal{G} \arrow[r, mapsto] \arrow[d, Rightarrow] & G \arrow[d, Rightarrow] \\
L_0 & \mathcal{G}_0 \arrow[r, mapsto] & G_0 \arrow[ll, mapsto, bend left, "\mathsf{rb}"]
\end{tikzcd}$$

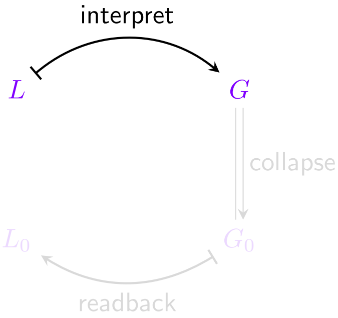
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G into G_0
3. readback rb
   of f-o term graph G_0
   yielding program $L_0 = \mathsf{rb}(G_0)$.

Maximal sharing: the method

$$
\begin{tikzcd}
L \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"'] & \mathcal{G} \arrow[r, mapsto] \arrow[dd, Rightarrow] & G \arrow[dd, Rightarrow] \\
M & {[\![\cdot]\!]_{\mathcal{T}}} & \\
L_0 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"] \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left] & \mathcal{G}_0 \arrow[r, mapsto] & G_0 \arrow[ll, mapsto, bend left, "\mathsf{rb}"]
\end{tikzcd}
$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $\downdownarrows$
   of f-o term graph G into G_0
3. readback rb
   of f-o term graph G_0
   yielding program $L_0 = \mathsf{rb}(G_0)$.

Interpretation



Running example

instead of:

$\lambda f.\, \mathsf{let}\ r = f\,(f\,r)\ \mathsf{in}\ r \quad \longmapsto_{\text{max-sharing}} \quad \lambda f.\, \mathsf{let}\ r = f\,r\ \mathsf{in}\ r$

we use:

$\lambda x.\, \lambda f.\, \mathsf{let}\ r = f\,(f\,r\,x)\,x\ \mathsf{in}\ r \quad \longmapsto_{\text{max-sharing}} \quad \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$

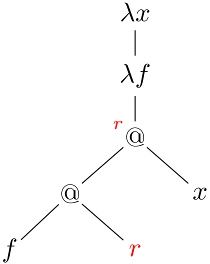
$L \quad \longmapsto_{\text{max-sharing}} \quad L_0$

Graph interpretation (example 1)

$L_0 = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\, r\, x\ \mathsf{in}\ r$

Graph interpretation (example 1)

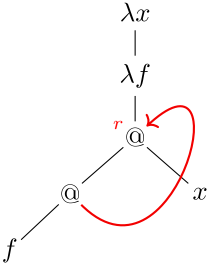
$L_0 = \lambda x.\, \lambda f.\,$ let $r = f\; r\; x$ in r



syntax tree

Graph interpretation (example 1)

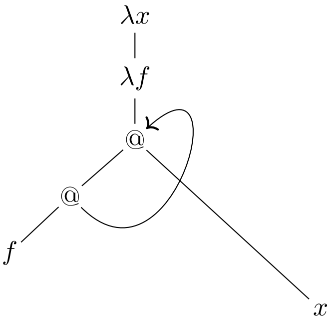
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 1)

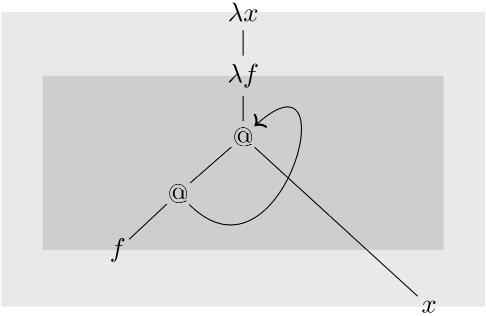
$L_0 = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$



syntax tree (+ recursive backlink)

Graph interpretation (example 1)

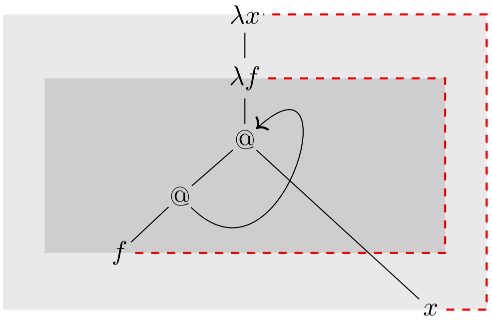
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

Graph interpretation (example 1)

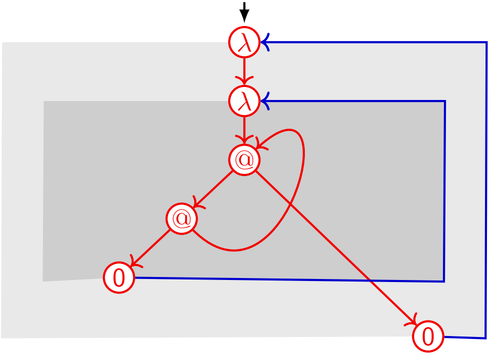
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink, + scopes, + binding links)

Graph interpretation (example 1)

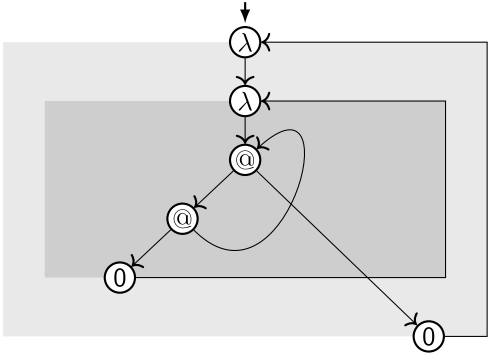
$L_0 = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 1)

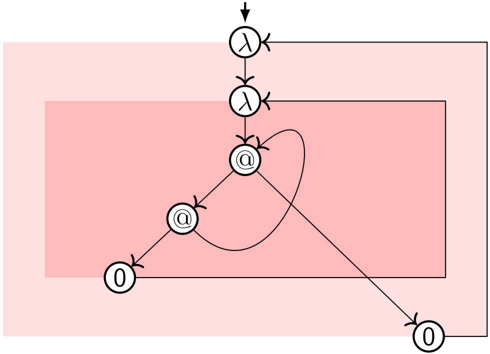
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 1)

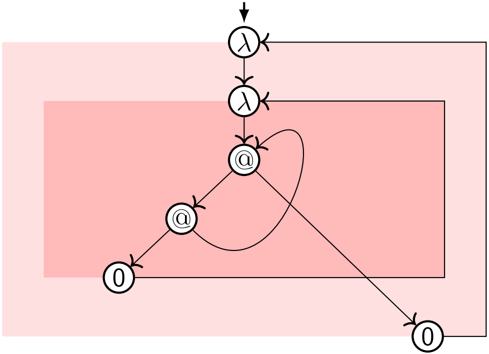
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



first-order term graph (+ scope sets)

Graph interpretation (example 1)

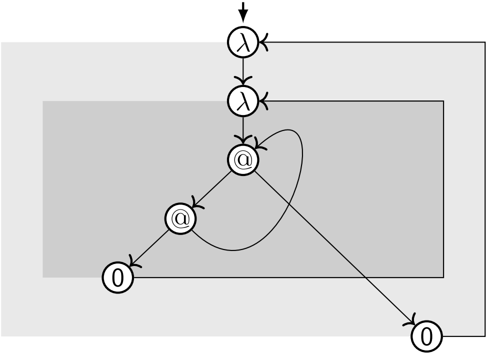
$L_0 = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\, r\, x\ \mathsf{in}\ r$



higher-order term graph (with scope sets, Blom [2003])

Graph interpretation (example 1)

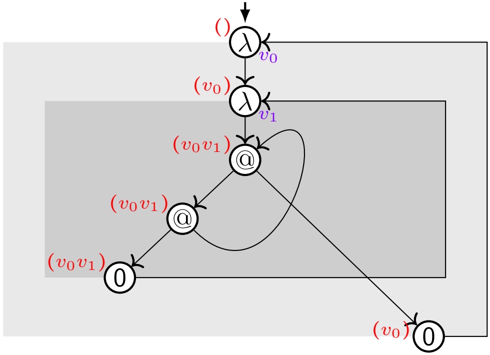
$L_0 = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$



higher-order term graph (with scope sets, Blom [2003])

Graph interpretation (example 1)

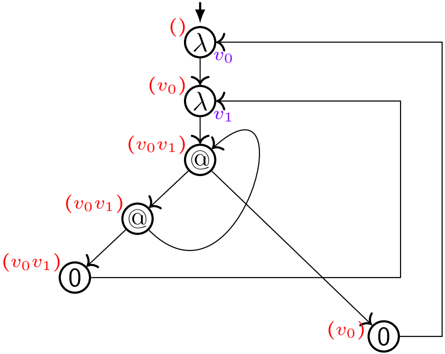
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



higher-order term graph (with scope sets, + abstraction-prefix function)

Graph interpretation (example 1)

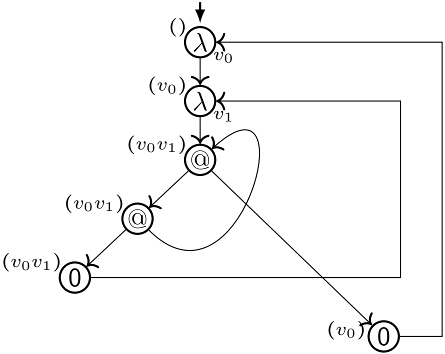
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



higher-order term graph (with abstraction-prefix function)

Graph interpretation (example 1)

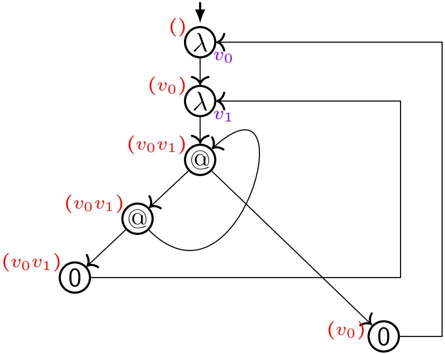
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



λ-higher-order-term-graph $[\![L_0]\!]_{\mathcal{H}}$

Graph interpretation (example 1)

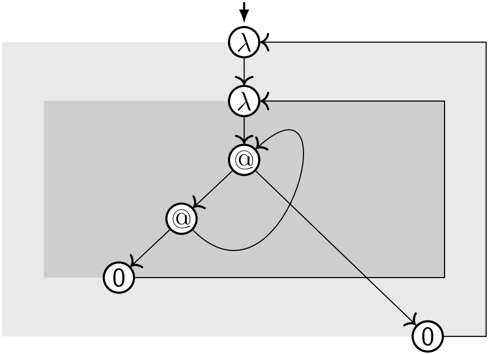
$L_0 = \lambda x. \lambda f. \, \mathsf{let}\ r = f\, r\, x\ \mathsf{in}\ r$



first-order term graph (+ abstraction-prefix function)

Graph interpretation (example 1)

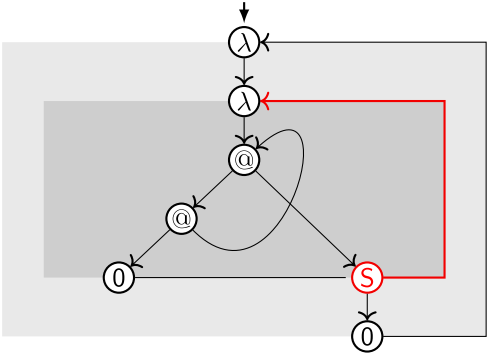
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 1)

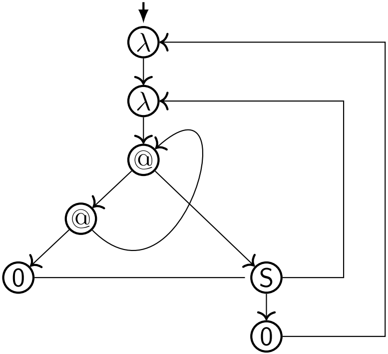
$L_0 = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,r\,x\ \mathsf{in}\ r$



first-order term graph with scope vertices with backlinks (+ scope sets)

Graph interpretation (example 1)

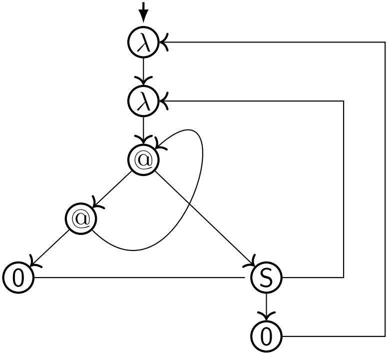
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



first-order term graph with scope vertices with backlinks

Graph interpretation (example 1)

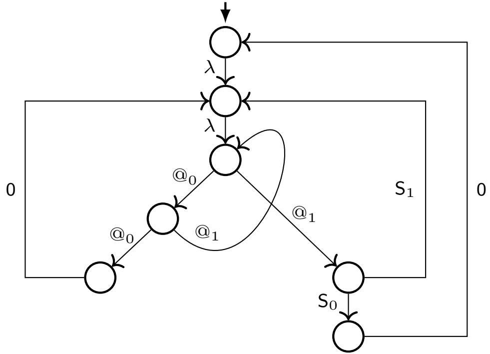
$L_0 = \lambda x.\, \lambda f.\,$ let $r = f\, r\, x$ in r



λ-term-graph $[\![L_0]\!]_{\mathcal{T}}$

Graph interpretation (example 1)

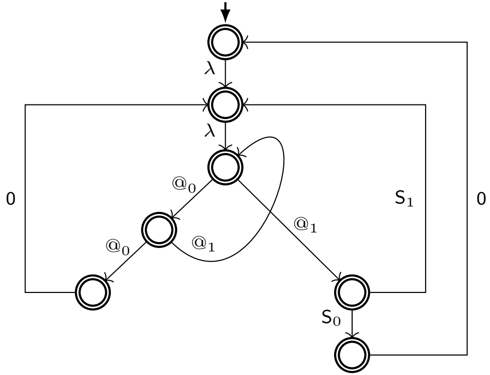
$L_0 = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\, r\, x\ \mathsf{in}\ r$



incomplete DFA

Graph interpretation (example 1)

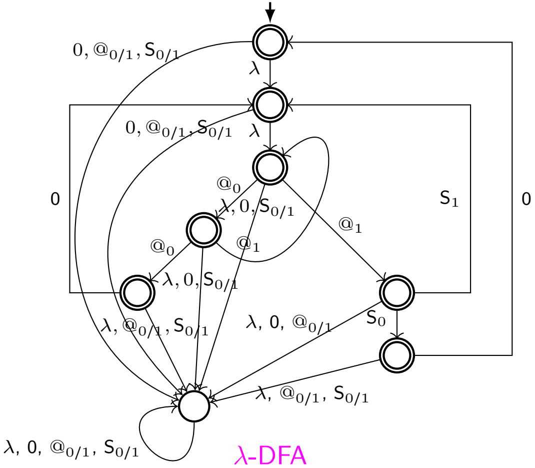
$L_0 = \lambda x. \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



incomplete λ-DFA

Graph interpretation (example 1)

$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$

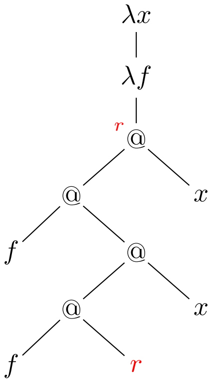


Graph interpretation (example 2)

$L = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,(f\,r\,x)\,x\ \mathsf{in}\ r$

Graph interpretation (example 2)

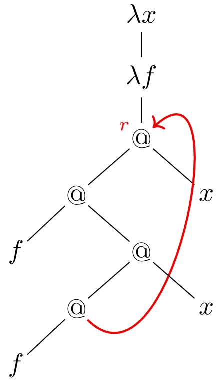
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree

Graph interpretation (example 2)

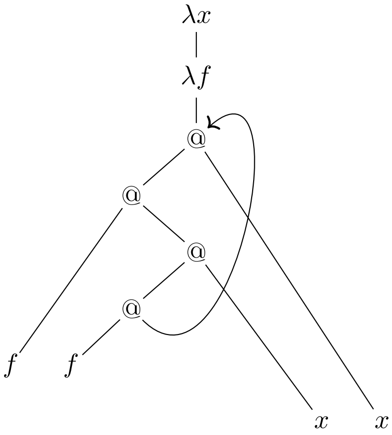
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 2)

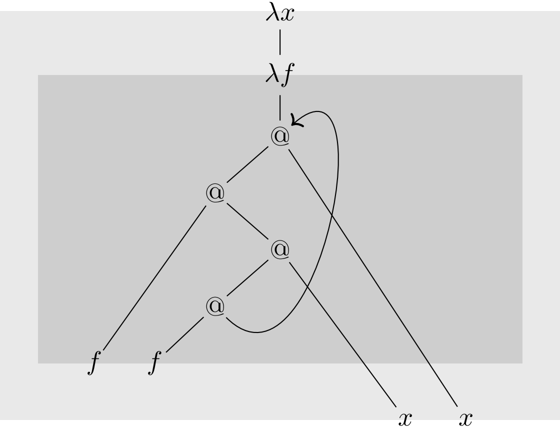
$L = \lambda x.\, \lambda f.\,$ let $r = f\,(f\,r\,x)\,x$ in r



syntax tree (+ recursive backlink)

Graph interpretation (example 2)

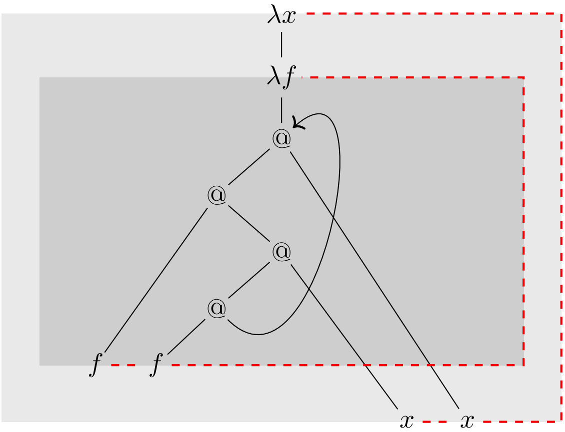
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

Graph interpretation (example 2)

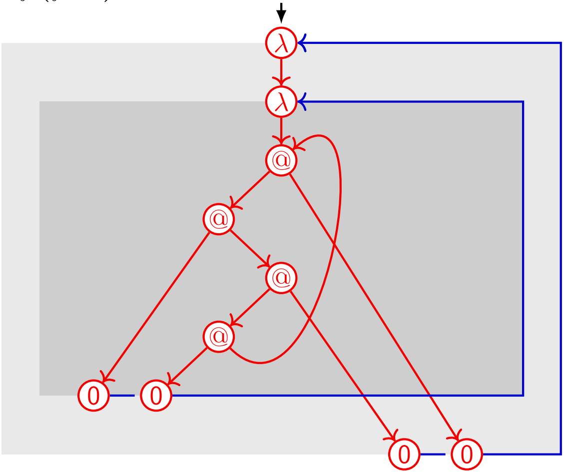
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink, + scopes, + binding links)

Graph interpretation (example 2)

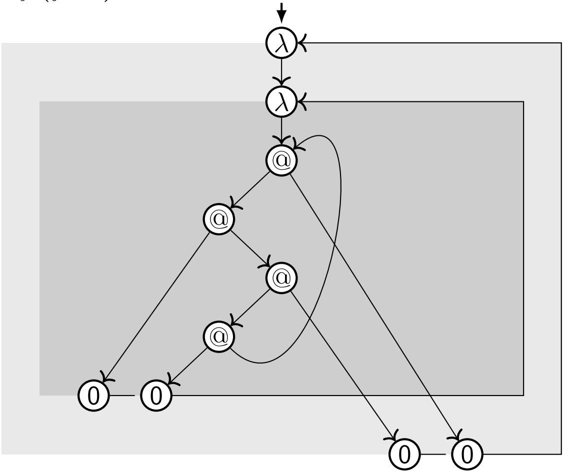
$L = \lambda x.\, \lambda f.\,$ let $r = f\,(f\,r\,x)\,x$ in r



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 2)

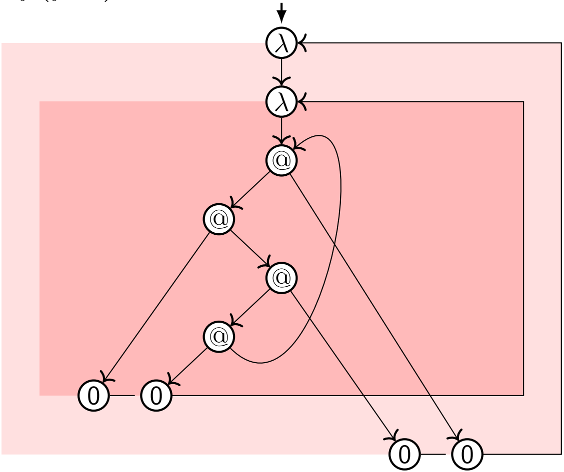
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 2)

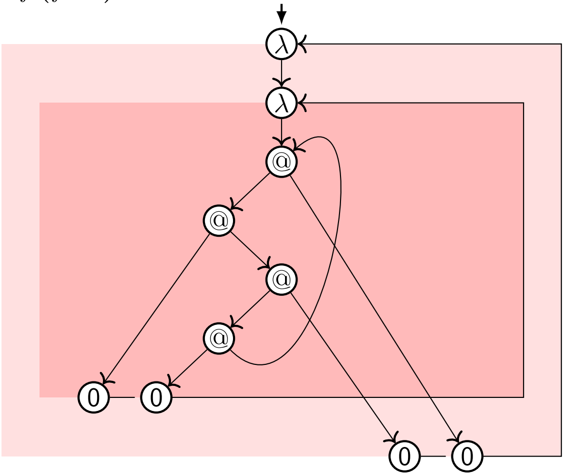
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph (+ scope sets)

Graph interpretation (example 2)

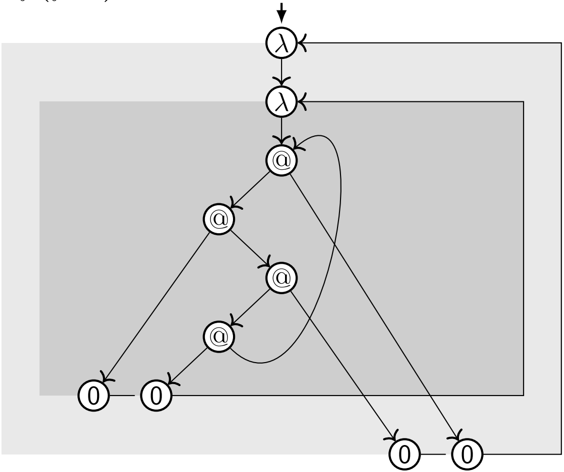
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

Graph interpretation (example 2)

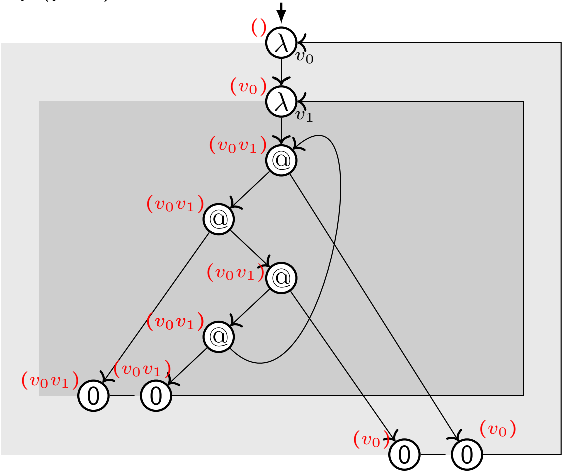
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

Graph interpretation (example 2)

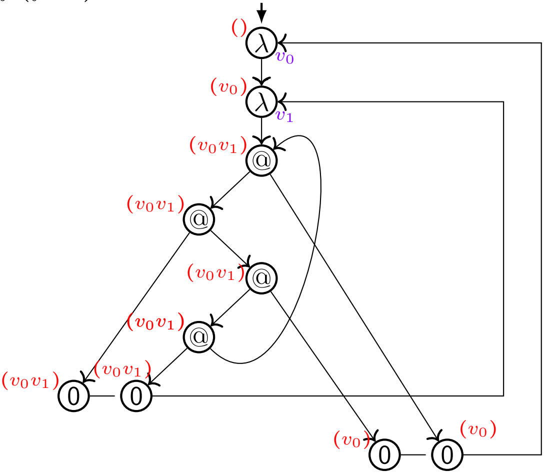
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



higher-order term graph (with scope sets, + abstraction-prefix function)

Graph interpretation (example 2)

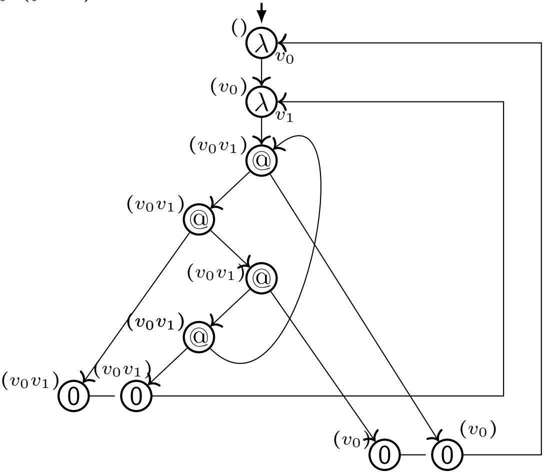
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



higher-order term graph (with abstraction-prefix function)

Graph interpretation (example 2)

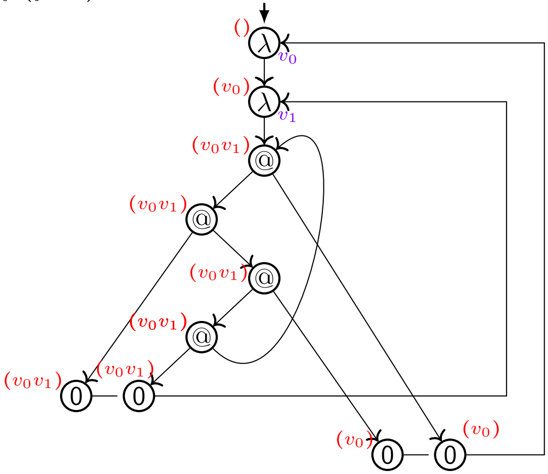
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



λ-higher-order-term-graph $[\![L]\!]_{\mathcal{H}}$

Graph interpretation (example 2)

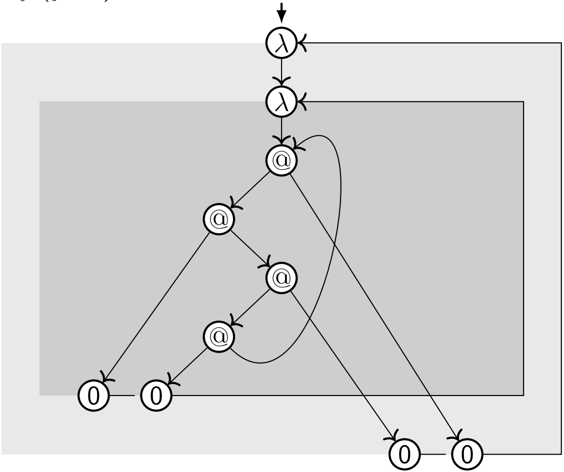
$L = \lambda x.\,\lambda f.\,$let $r = f\,(f\,r\,x)\,x$ in r



first-order term graph (+ abstraction-prefix function)

Graph interpretation (example 2)

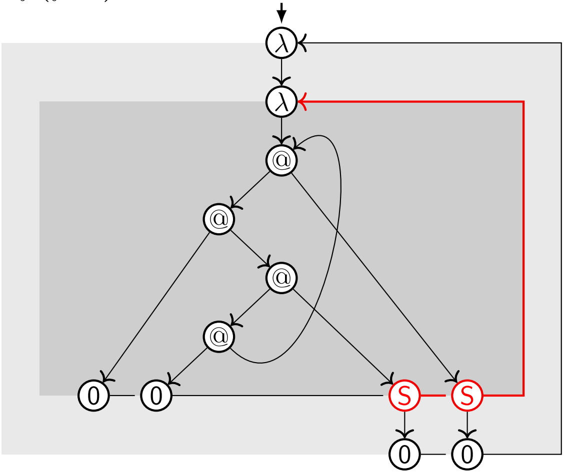
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

Graph interpretation (example 2)

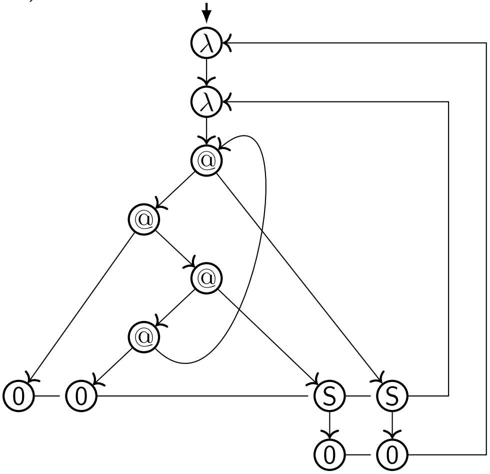
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph with scope vertices with backlinks (+ scope sets)

Graph interpretation (example 2)

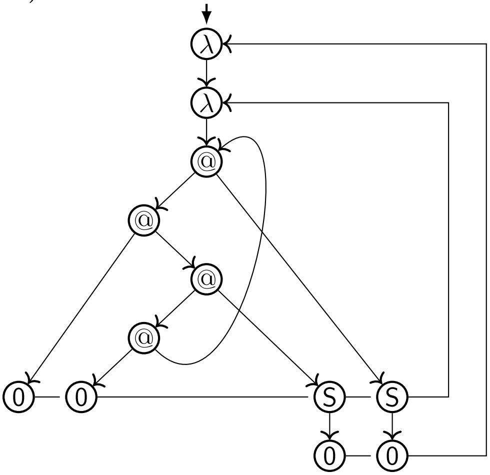
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph with scope vertices with backlinks

Graph interpretation (example 2)

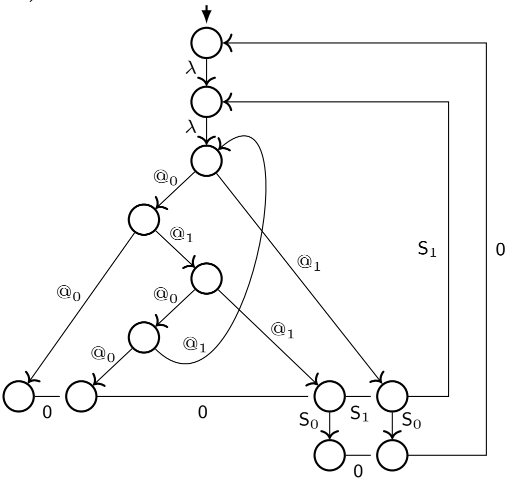
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



λ-term-graph $[\![L]\!]_{\mathcal{T}}$

Graph interpretation (example 2)

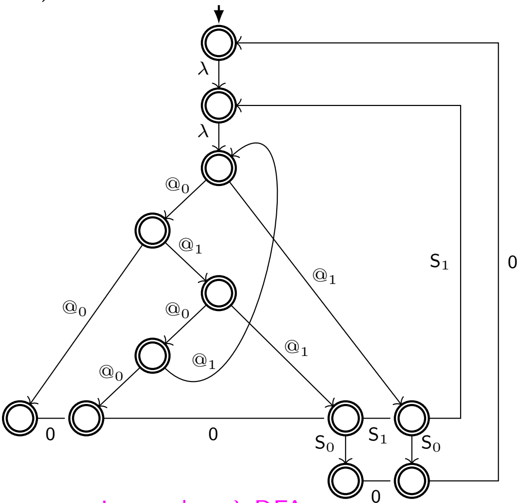
$L = \lambda x. \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



incomplete DFA

Graph interpretation (example 2)

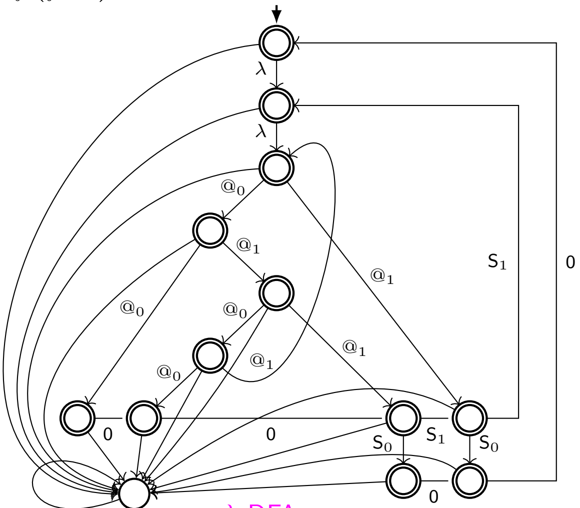
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



incomplete λ-DFA

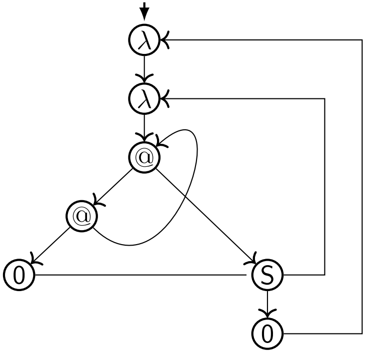
Graph interpretation (example 2)

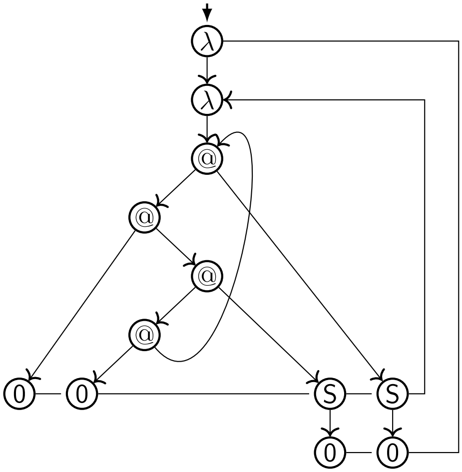
L = $\lambda x.\, \lambda f.$ let r = $f\,(f\,r\,x)\,x$ in r



λ-DFA

Graph interpretation (examples 1 and 2)





$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Interpretation $[\![\cdot]\!]_{\mathcal{T}}$: properties (cont.)

interpretation $\boldsymbol{\lambda}_{\text{letrec}}$-term $L \longmapsto \lambda$-term-graph $[\![L]\!]_{\mathcal{T}}$

- defined by induction on structure of L
- similar analysis as fully-lazy lambda-lifting
- yields eager-scope λ-term-graphs: $\sim$ minimal scopes

Theorem

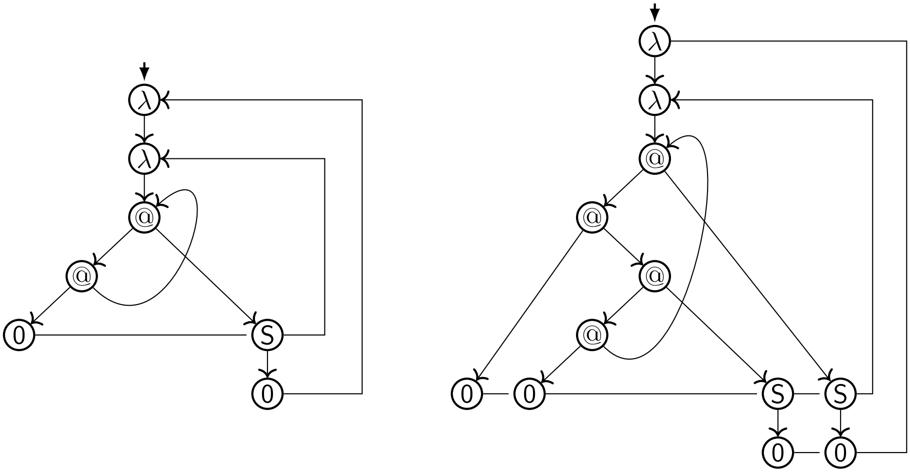
For $\boldsymbol{\lambda}_{\text{letrec}}$-terms L_1 and L_2 it holds: Equality of infinite unfolding coincides with bisimilarity of λ-term-graph interpretations:

$$[\![L_1]\!]_{\lambda^\infty} = [\![L_2]\!]_{\lambda^\infty} \iff [\![L_1]\!]_{\mathcal{T}} \;\underline{\leftrightarrow}\; [\![L_2]\!]_{\mathcal{T}}$$

Collapse

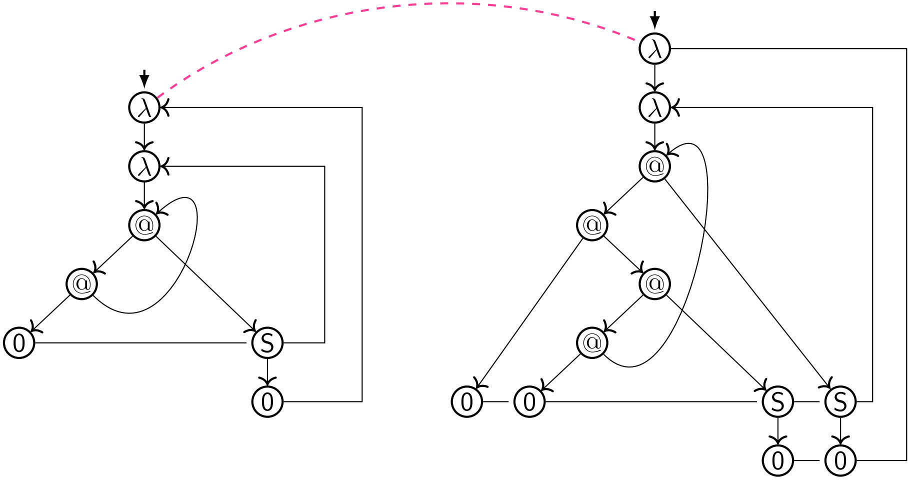
$$\begin{array}{ccc} L & \xrightarrow{\text{interpret}} & G \\ & & \Downarrow \text{collapse} \\ L_0 & \xleftarrow{\text{readback}} & G_0 \end{array}$$

Bisimulation check between λ-term-graphs



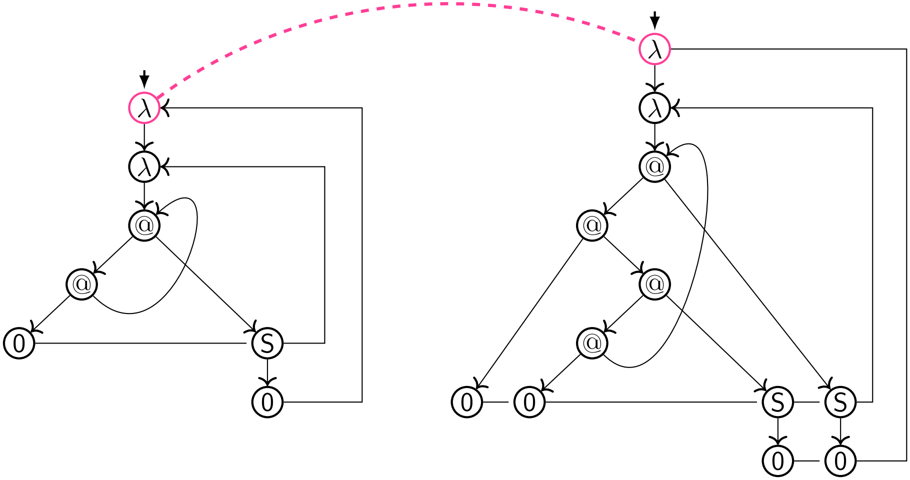
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



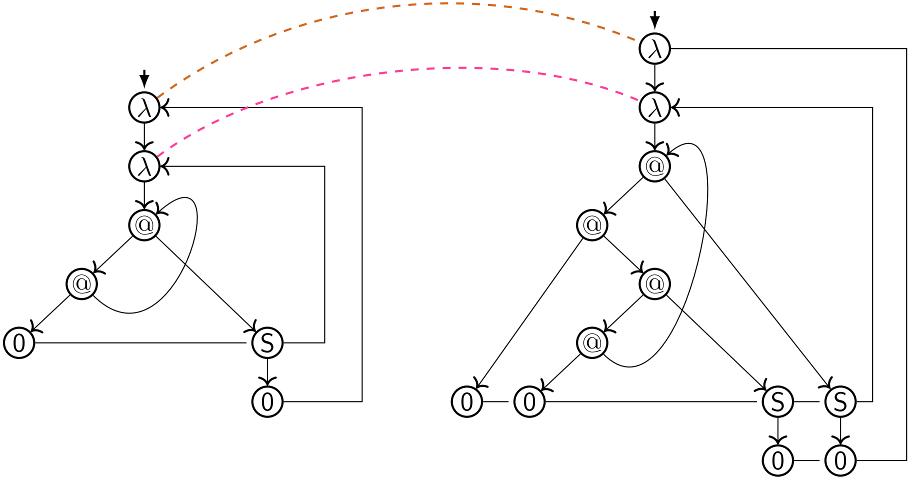
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



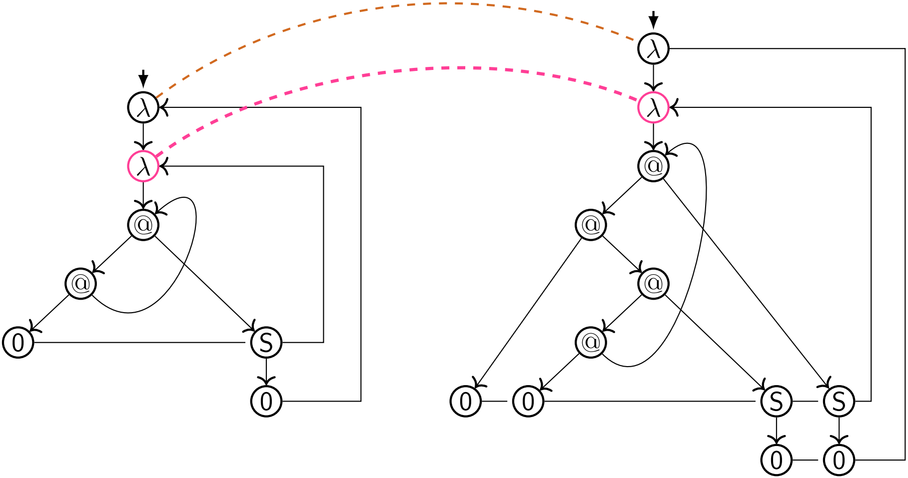
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



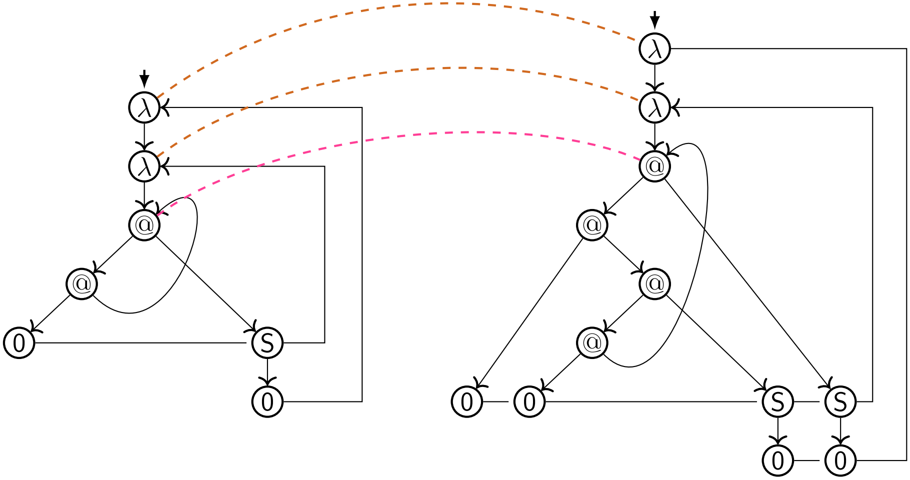
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



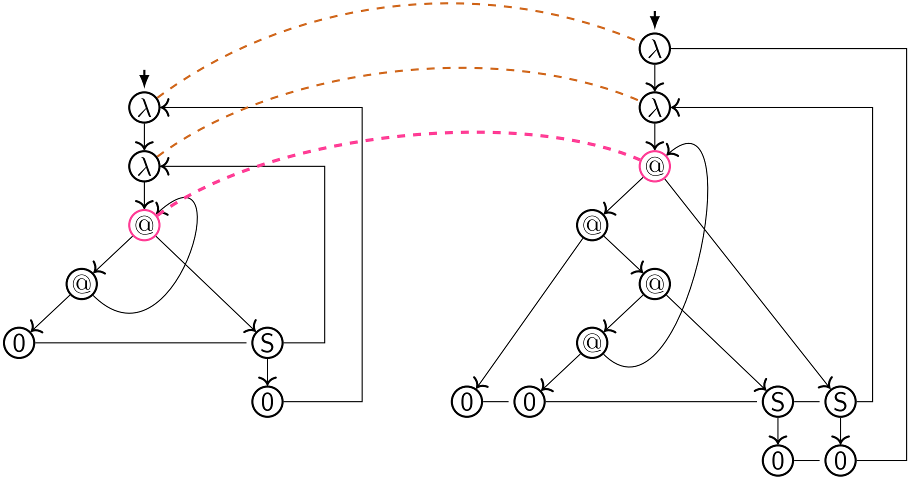
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



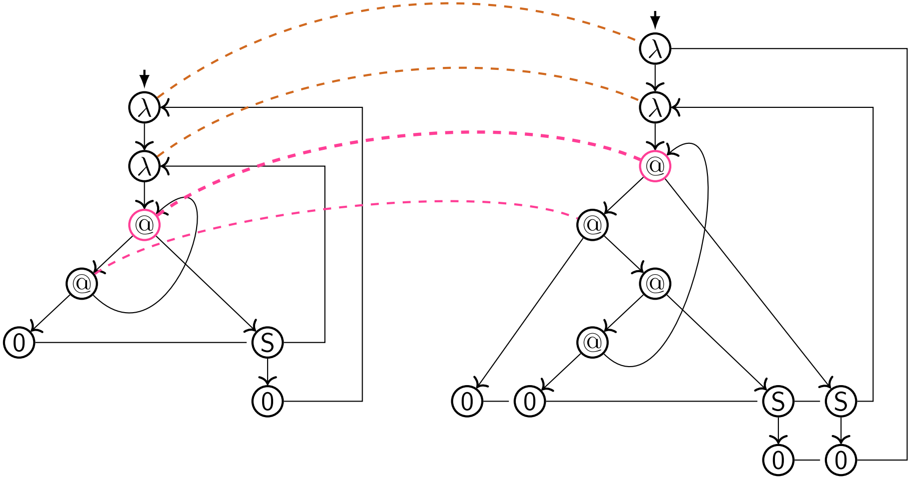
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



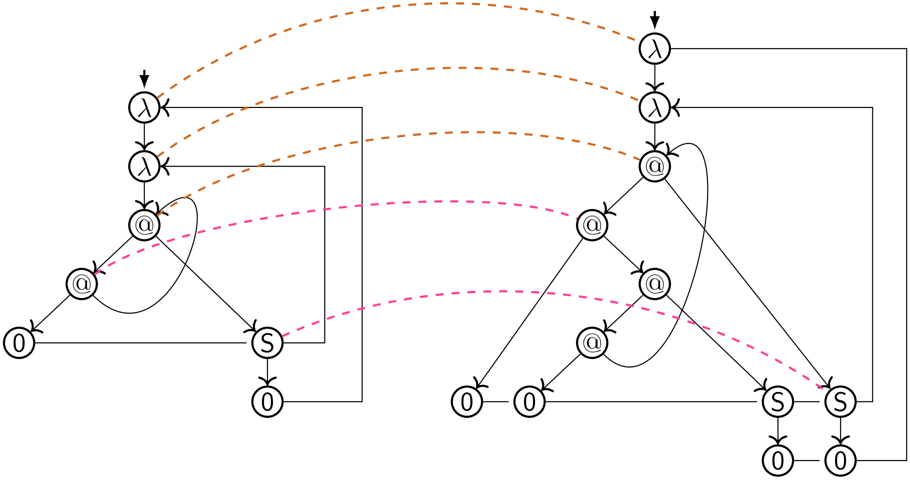
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



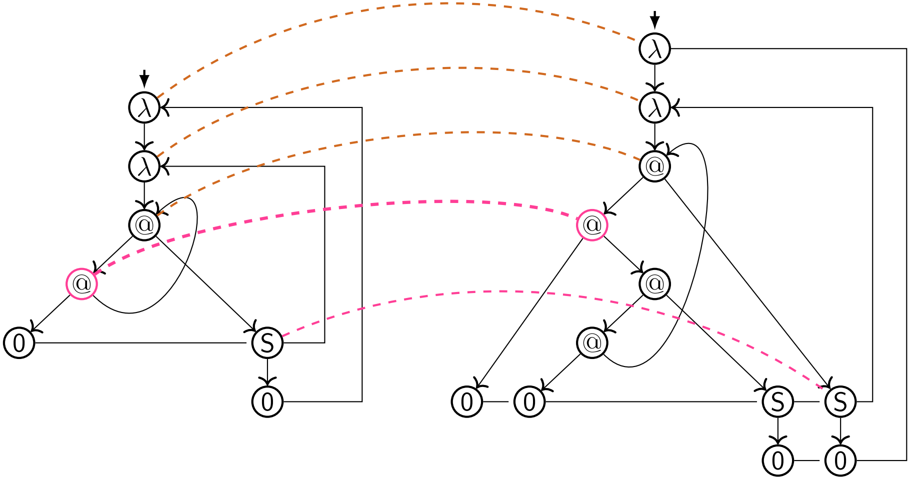
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



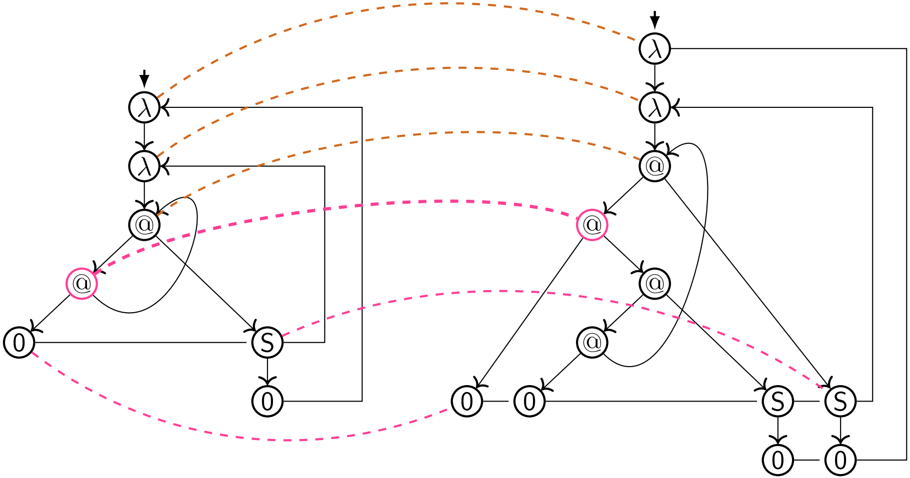
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



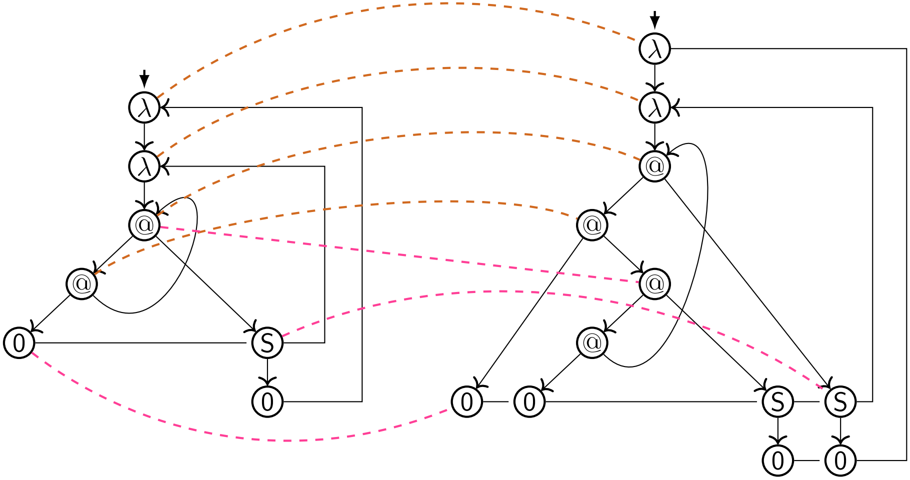
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



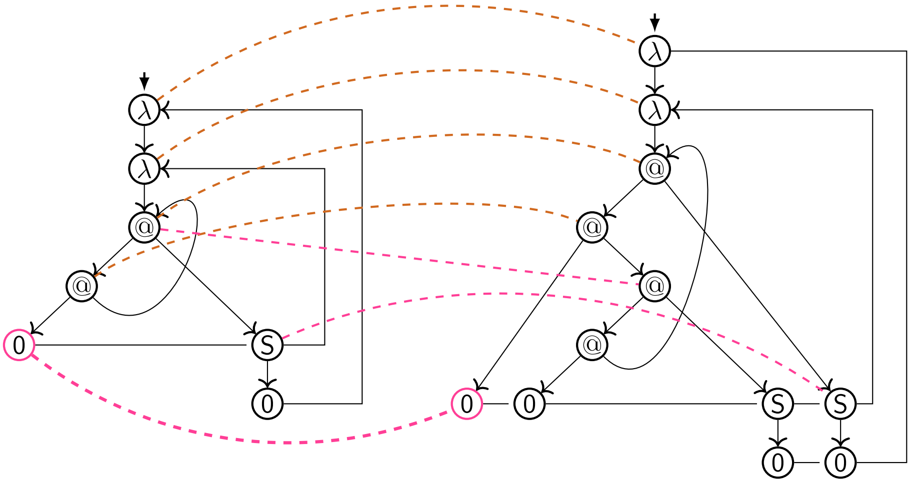
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



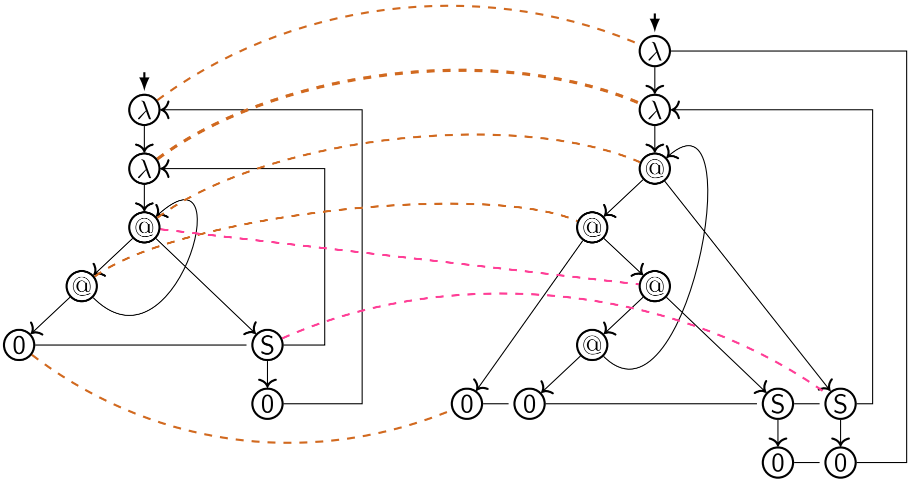
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



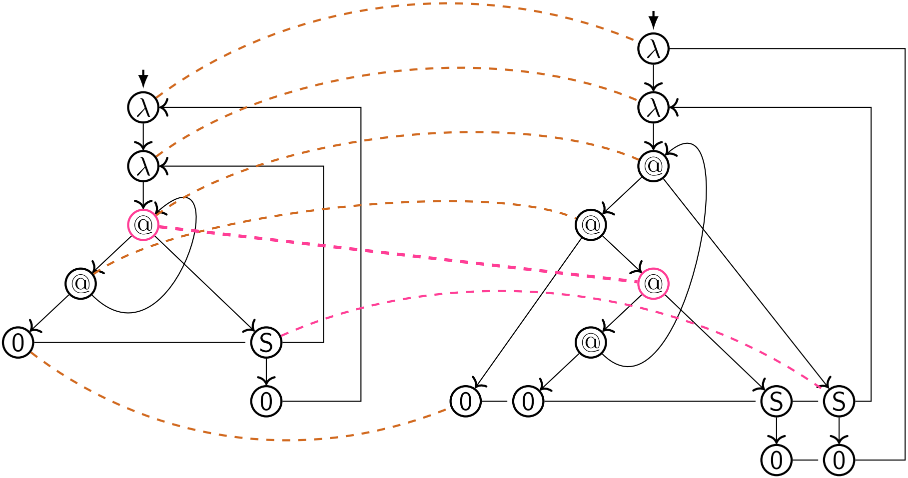
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



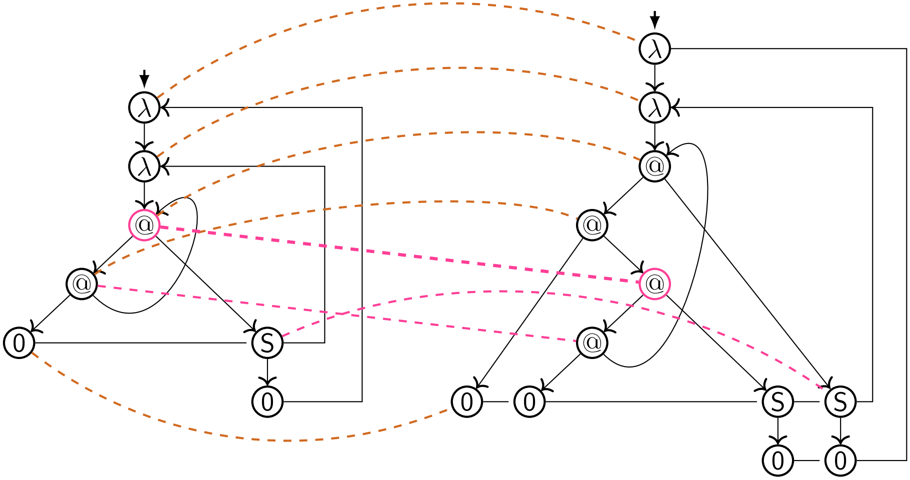
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



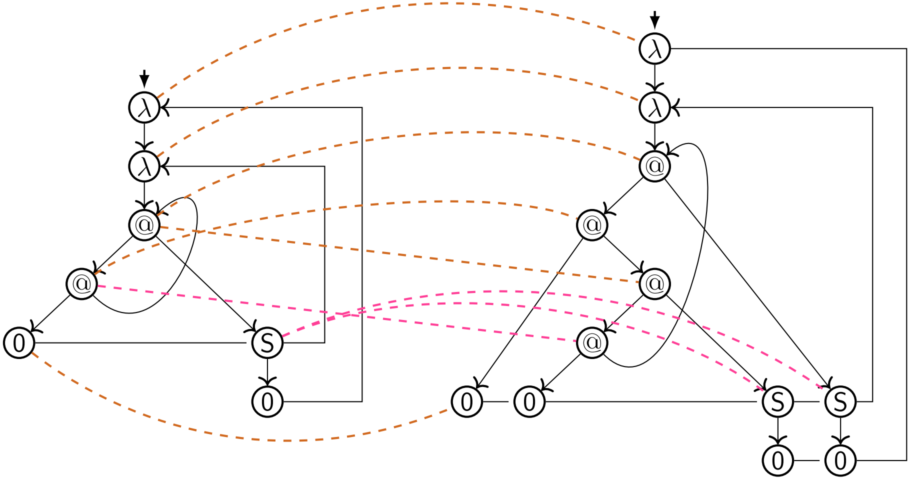
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



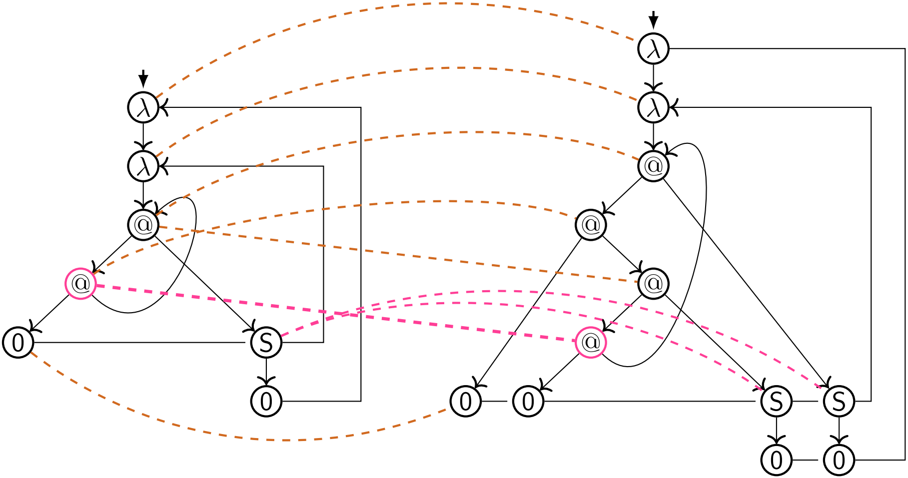
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



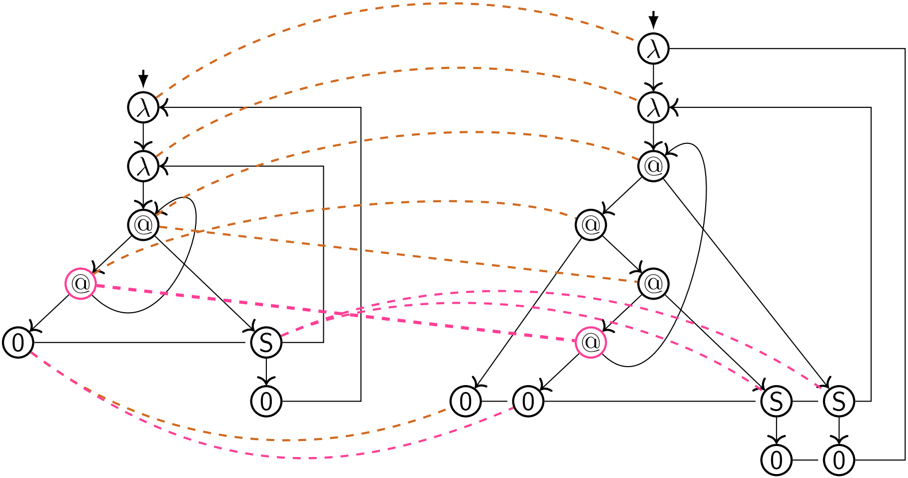
$[\![L_0]\!]_{\mathcal{T}}$ $\quad$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



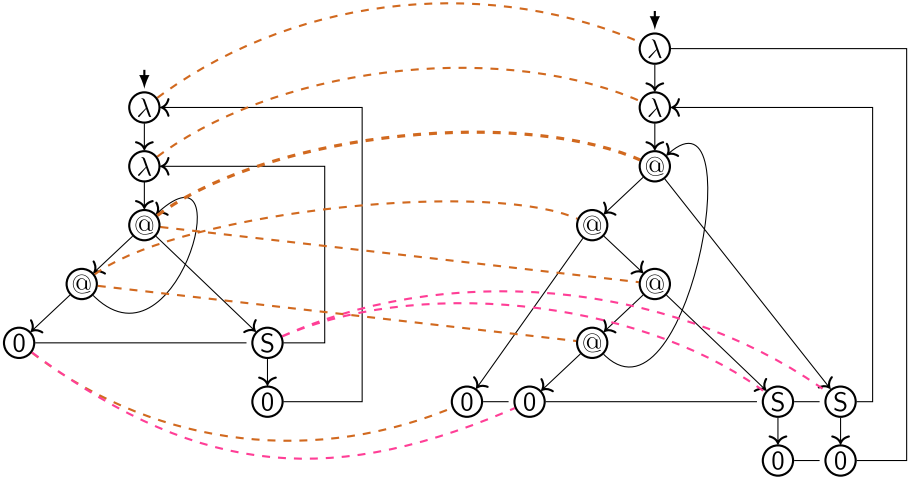
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



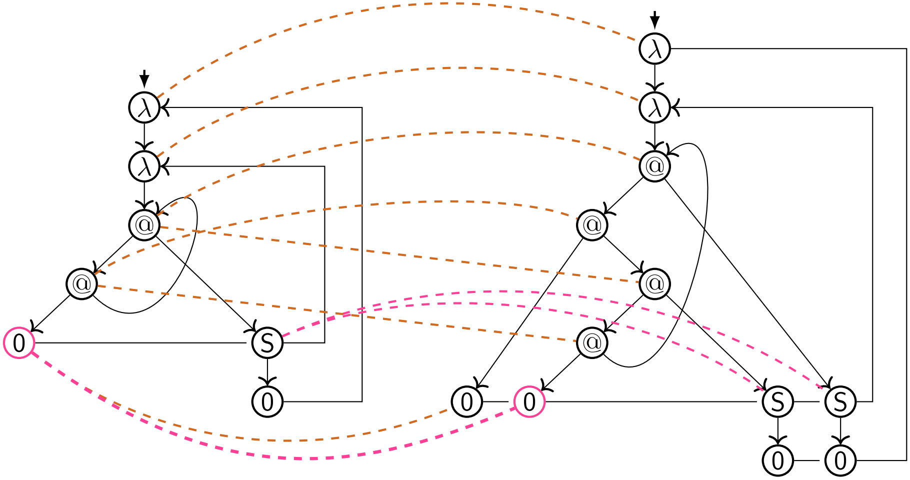
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



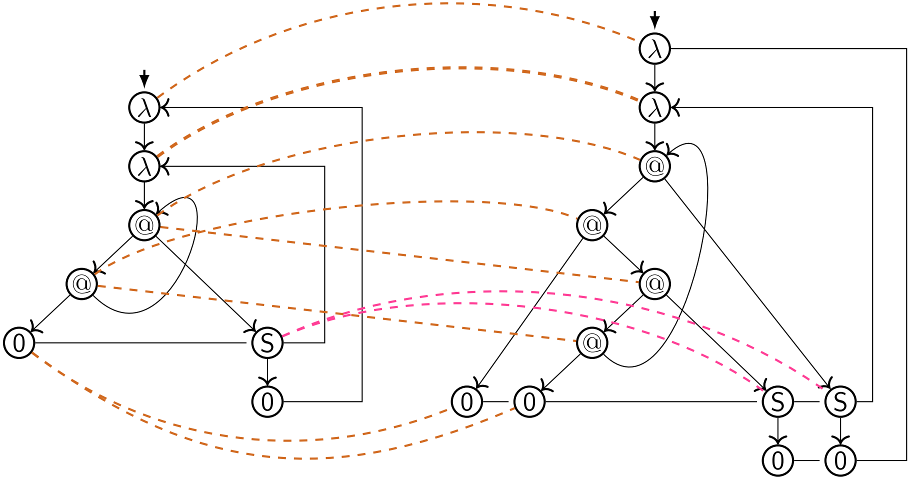
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



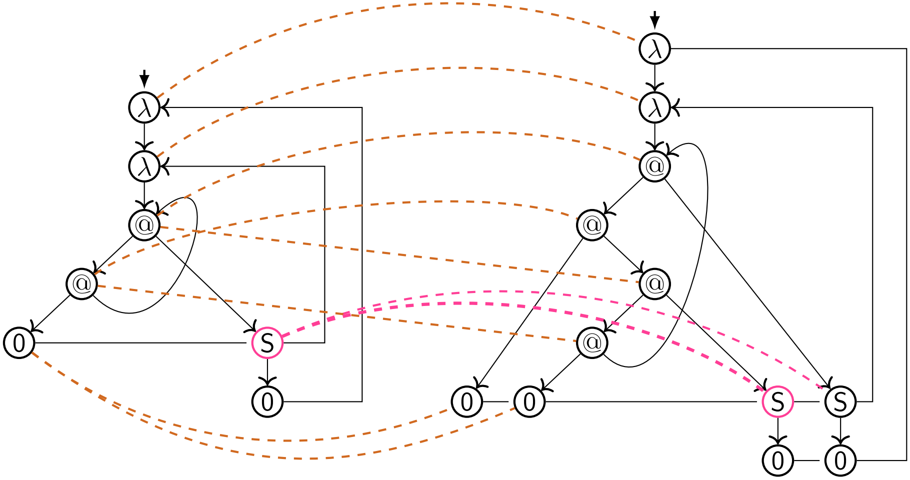
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



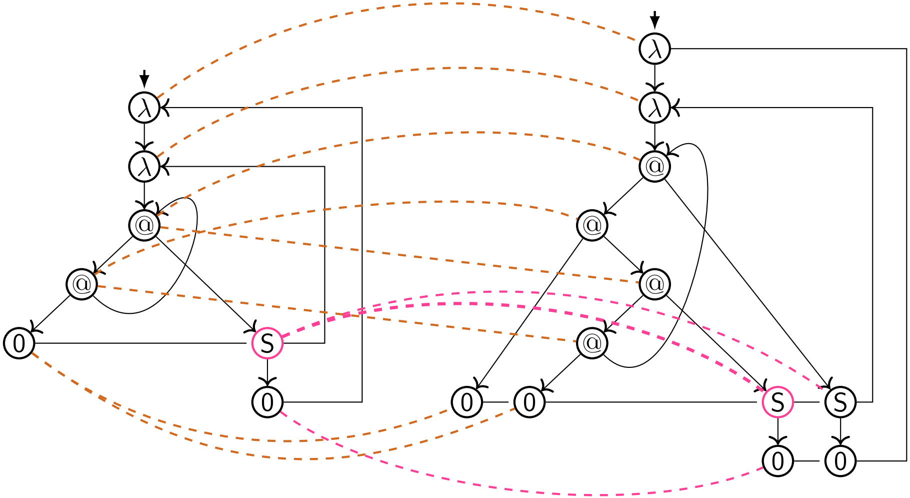
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



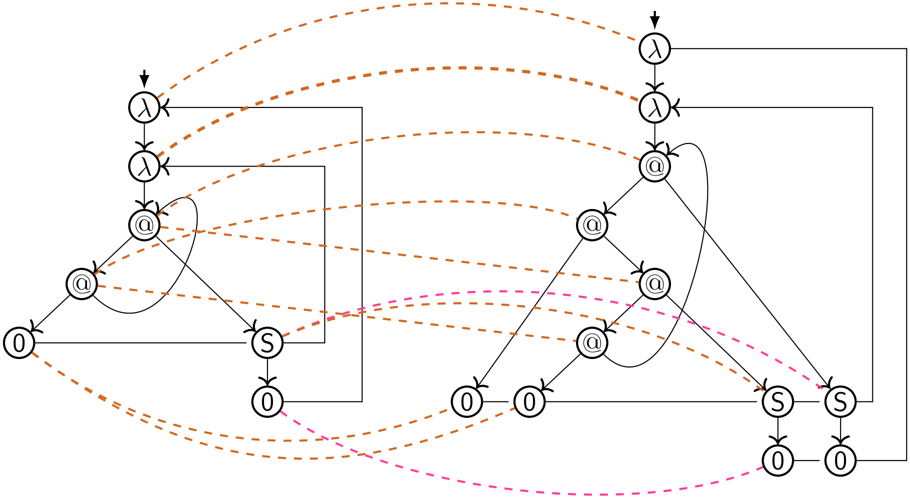
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



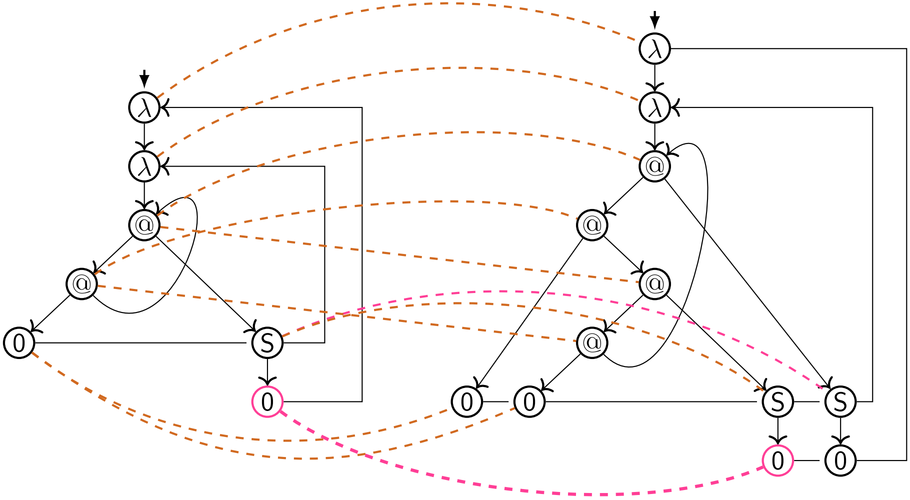
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



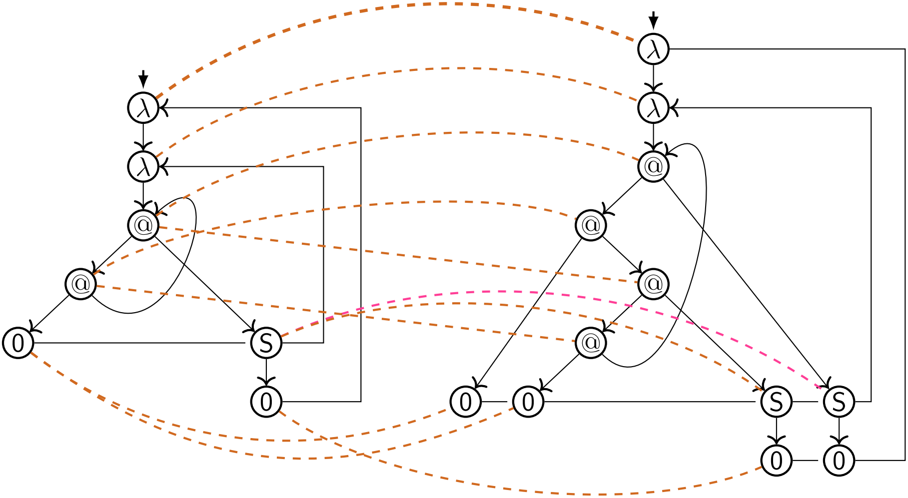
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



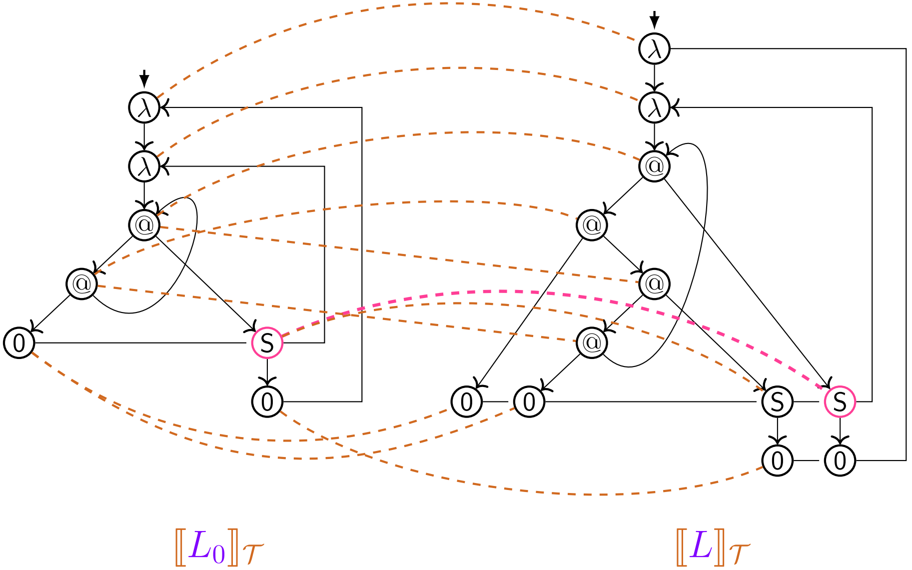
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

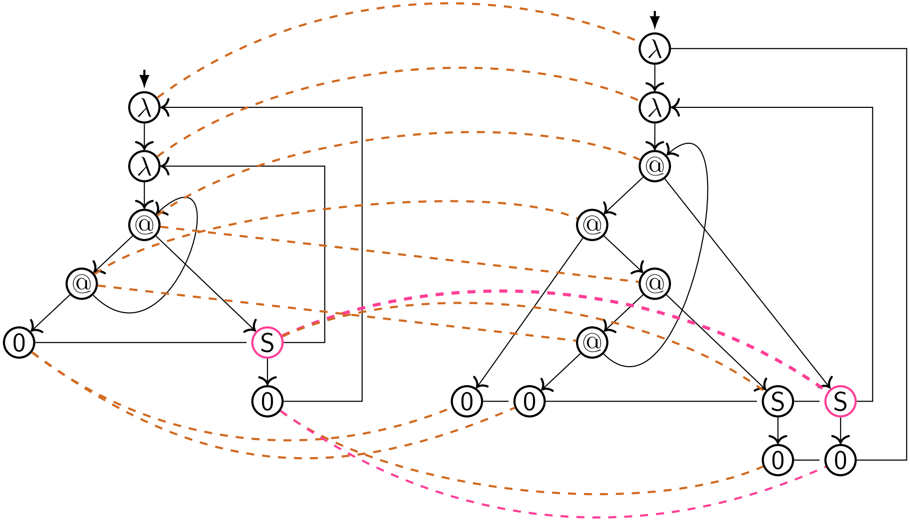


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

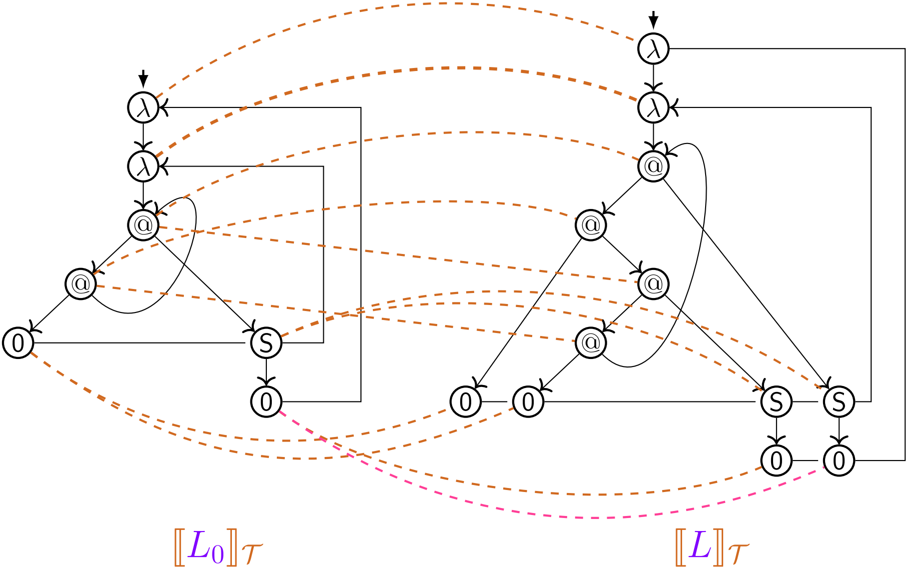


Bisimulation check between λ-term-graphs

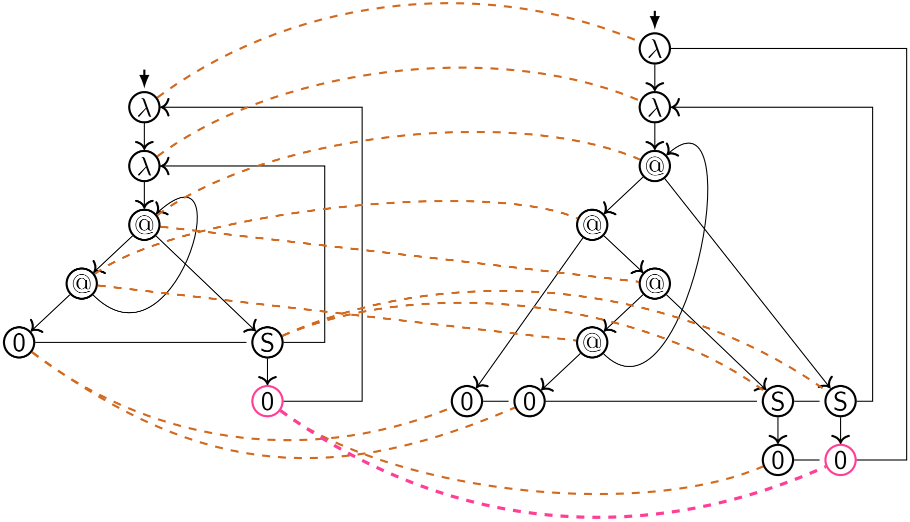


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

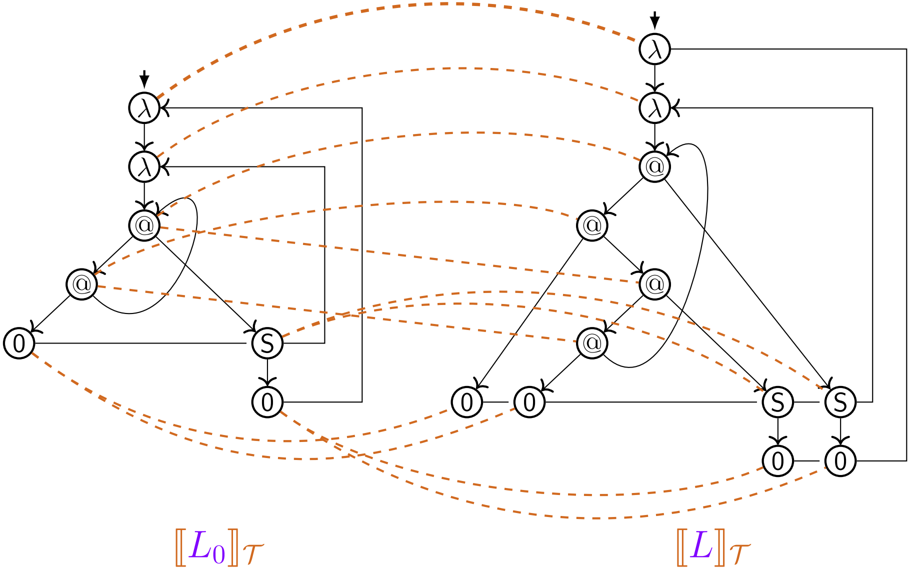


Bisimulation check between λ-term-graphs

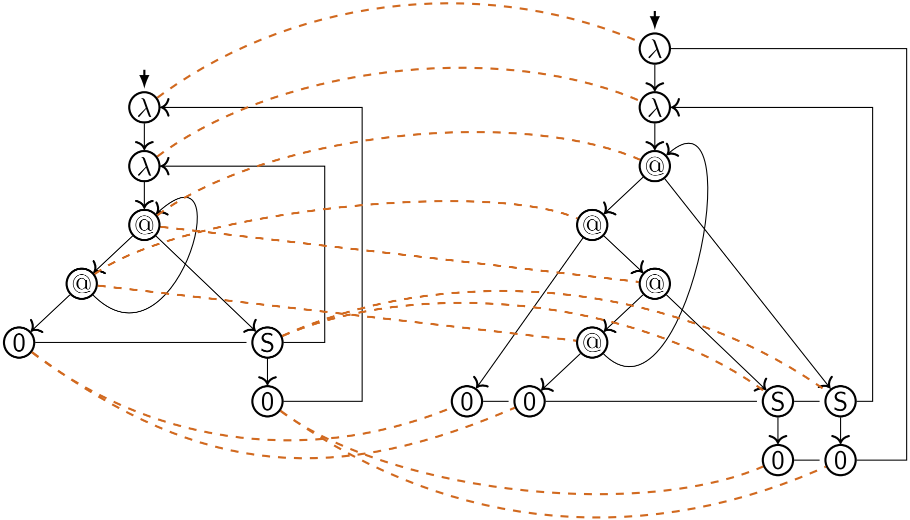


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

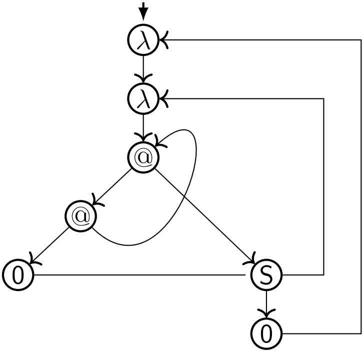


bisimulation between λ-term-graphs

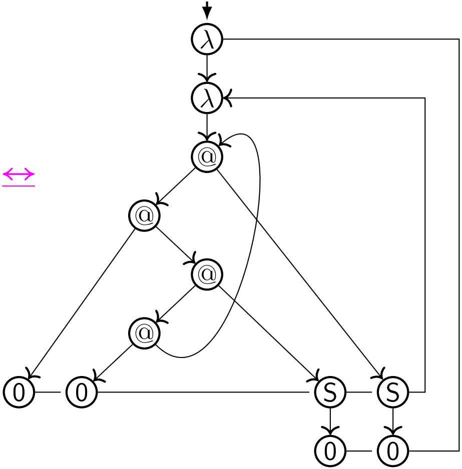


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimilarity between λ-term-graphs

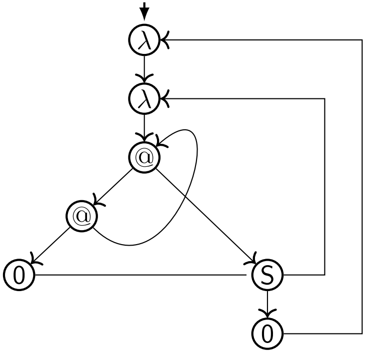


$\underline{\leftrightarrow}$

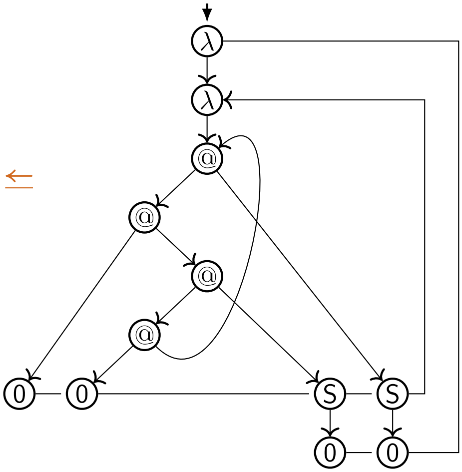


$[\![L_0]\!]_{\mathcal{T}} \quad \underline{\leftrightarrow} \quad [\![L]\!]_{\mathcal{T}}$

functional bisimilarity and bisimulation collapse



$\Leftarrow$



$[\![L_0]\!]_{\mathcal{T}} \quad \Leftarrow \quad [\![L]\!]_{\mathcal{T}}$

Bisimulation collapse: property

Theorem

The class of eager-scope λ-term-graphs
is closed under functional bisimilarity $\rightarrowtail$.

$\Longrightarrow$ For a $\boldsymbol{\lambda}_{\text{letrec}}$-term L
the bisimulation collapse of $[\![L]\!]_{\mathcal{T}}$ is again an eager-scope λ-term-graph.

Readback

$$\begin{tikzcd}
L \arrow[r, mapsto, bend left, "\text{interpret}"] & G \arrow[d, Rightarrow, "\text{collapse}"] \\
L_0 & G_0 \arrow[l, mapsto, bend left, "\text{readback}"]
\end{tikzcd}$$

Readback

defined with property:

$$L \xleftarrow{\text{rb}} G \quad \text{eager-scope}$$

Readback

defined with property:

$$\begin{tikzcd} L \arrow[rr, mapsto, bend left, "[\![\cdot]\!]_{\mathcal{T}}"] & & G \arrow[ll, mapsto, bend left, "\mathrm{rb}"] \end{tikzcd} \quad \text{eager-scope}$$

Readback

defined with property:

$$\begin{tikzcd} L \arrow[rr, mapsto, bend left, "[\![\cdot]\!]_{\mathcal{T}}"] & & G \arrow[ll, mapsto, bend left, "\mathsf{rb}"] \end{tikzcd} \quad \text{eager-scope}$$

Theorem

For all eager-scope λ-term-graphs G:

$$([\![\cdot]\!]_{\mathcal{T}} \circ \mathsf{rb})(G) \simeq G$$

The readback rb is a right-inverse of $[\![\cdot]\!]_{\mathcal{T}}$ modulo isomorphism $\simeq$.

Readback

defined with property:

$$\begin{tikzcd} L \arrow[r, mapsto, bend left, "\llbracket\cdot\rrbracket_{\mathcal{T}}"] & G \arrow[l, mapsto, bend left, "\mathsf{rb}"] \end{tikzcd} \quad \text{eager-scope}$$

Theorem

For all eager-scope λ-term-graphs G:

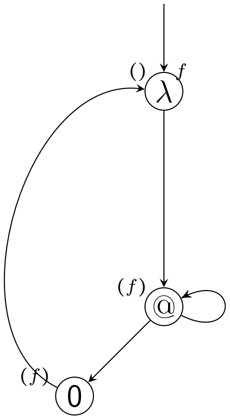
$$(\llbracket\cdot\rrbracket_{\mathcal{T}} \circ \mathsf{rb})(G) \simeq G$$

The readback rb *is a right-inverse of* $\llbracket\cdot\rrbracket_{\mathcal{T}}$ *modulo isomorphism* $\simeq$*.*

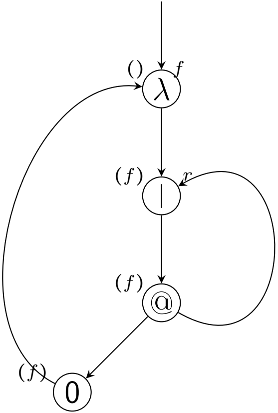
idea:

1. construct a spanning tree T of G
2. using local rules, in a bottom-up traversal of T synthesize $L = \mathsf{rb}(G)$

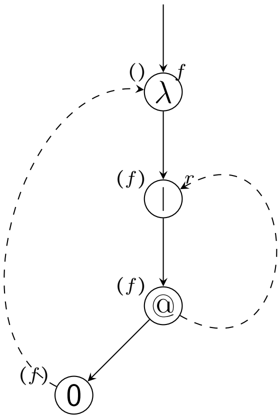
Readback: example (fix)



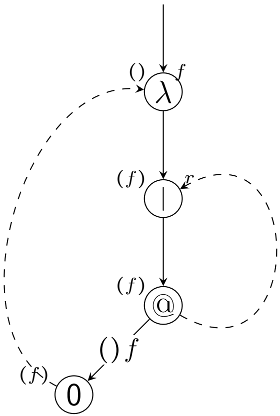
Readback: example (fix)



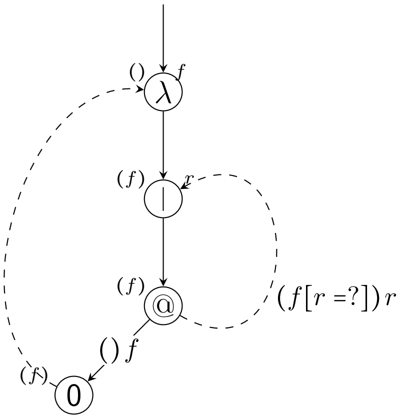
Readback: example (fix)



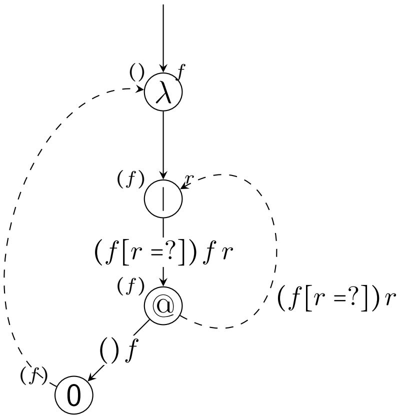
Readback: example (fix)



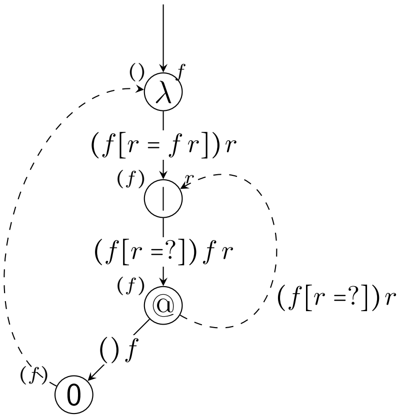
Readback: example (fix)



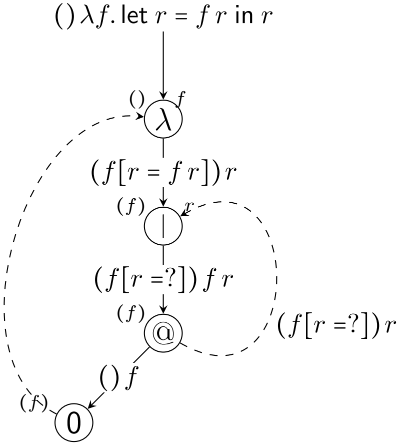
Readback: example (fix)



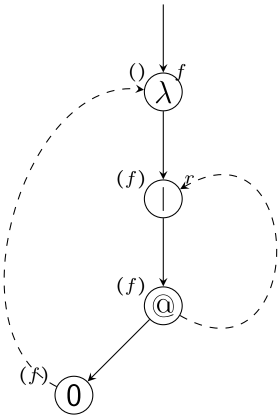
Readback: example (fix)



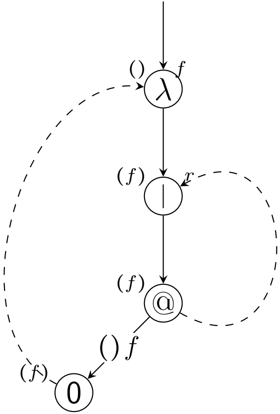
Readback: example (fix)

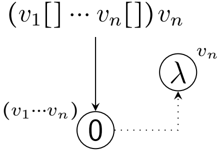


Readback: example (fix)

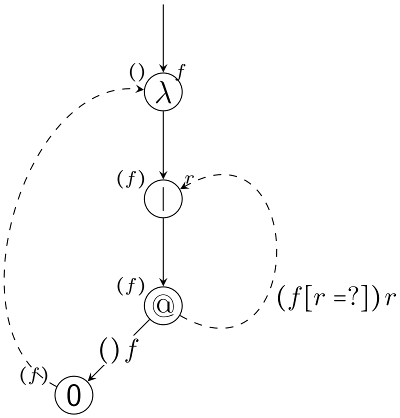


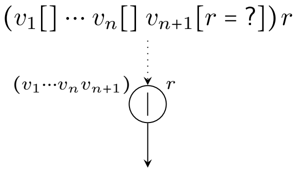
Readback: example (fix)



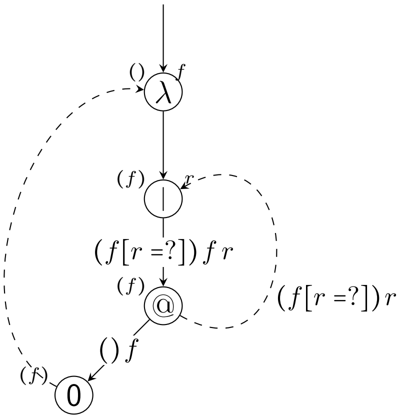


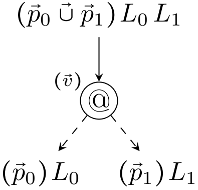
Readback: example (fix)



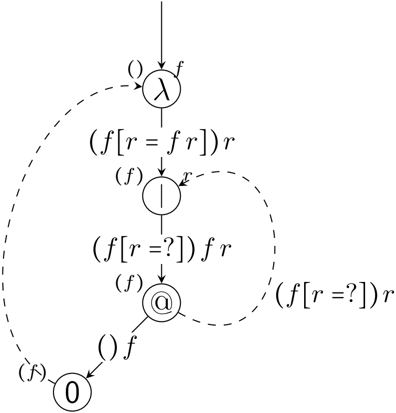


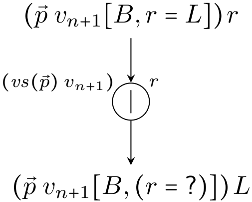
Readback: example (fix)



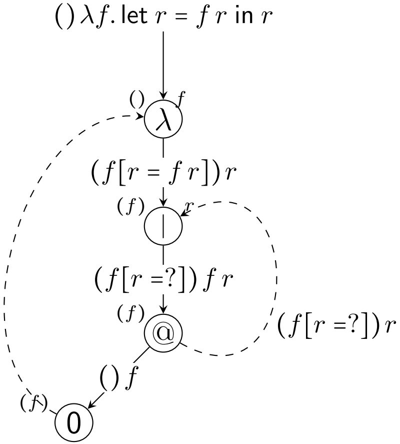


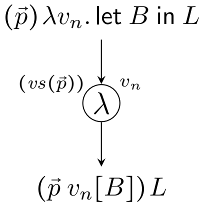
Readback: example (fix)





Readback: example (fix)





Maximal sharing: complexity

$$\begin{tikzcd}
L \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & & G \arrow[d, Downarrow] \\
L_0 & & G_0 \arrow[ll, mapsto, bend left, "\mathsf{rb}"]
\end{tikzcd}$$

1. interpretation

   of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L with $|L| = n$

   as λ-term-graph $G = [\![L]\!]_{\mathcal{T}}$

   ▶ in time $O(n^2)$, size $|G| \in O(n^2)$.

2. bisimulation collapse $|\!\!\downarrow$

   of f-o term graph G into G_0

   ▶ in time $O(|G| \log |G|) = O(n^2 \log n)$

3. readback rb

   of f-o term graph G_0

   yielding $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term $L_0 = \mathsf{rb}(G_0)$.

   ▶ in time $O(|G| \log |G|) = O(n^2 \log n)$

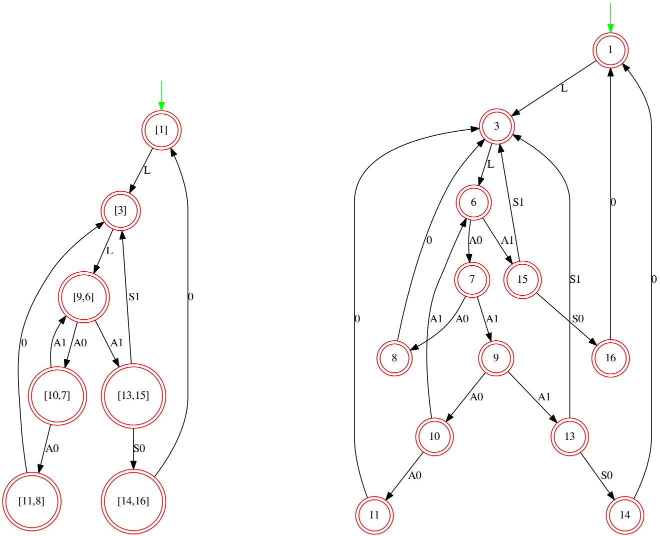
Theorem

Computing a maximally compact form $L_0 = (\mathsf{rb} \circ |\!\!\downarrow \circ [\![\cdot]\!]_{\mathcal{T}})(L)$ of L for a $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L requires time $O(n^2 \log n)$, where $|L| = n$.

Demo: console output

```
jan:~/papers/maxsharing-ICFP/talks/ICFP-2014> maxsharing running.l
λ-letrec-term:
λx. λf. let r = f (f r x) x in r

derivation:
                ---------- 0                ------- 0
                (x f[r]) f    (x f[r]) r    (x) x
                ------------------------ @  ---------- S
                (x f[r]) f r                (x f[r]) x
------------ 0  ---------------------------------------- @    ------- 0
(x f[r]) f      (x f[r]) f r x                                (x) x
---------------------------------------------------------- @  ---------- S
(x f[r]) f (f r x)                                            (x f[r]) x
-------------------------------------------------------------------------- @
(x f[r]) f (f r x) x                                                          (x f[r]) r
---------------------------------------------------------------------------------------- let
(x f) let r = f (f r x) x in r
-------------------------------------------------------------------------------------------- λ
(x) λf. let r = f (f r x) x in r
---------------------------------------------------------------------------------------------- λ
() λx. λf. let r = f (f r x) x in r

writing DFA to file: running-dfa.pdf

readback of DFA:
λx. λy. let F = y (y F x) x in F

writing minimised DFA to file: running-mindfa.pdf

readback of minimised DFA:
λx. λy. let F = y F x in F
jan:~/papers/maxsharing-ICFP/talks/ICFP-2014>
```

Demo: generated λ-NFAs

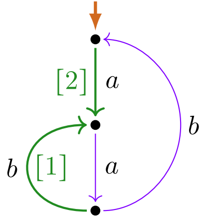


Resources (maximal sharing)

- tool maxsharing on `hackage.haskell.org`
- articles and reports
  - Maximal Sharing in the Lambda Calculus with Letrec
    - ICFP 2014 paper
    - accompanying report arXiv:1401.1460
  - Term Graph Representations for Cyclic Lambda Terms
    - TERMGRAPH 2013 proceedings
    - extended report arXiv:1308.1034
  - Vincent van Oostrom, CG: Nested Term Graphs
    - TERMGRAPH 2014 post-proceedings in EPTCS 183
- thesis Jan Rochel
  - Unfolding Semantics of the Untyped λ-Calculus with letrec
    - Ph.D. Thesis, Utrecht University, 2016

Process interpretation of regular expressions

(based on joint work with Wan Fokkink)



Regular expressions *(S.C. Kleene, 1951)*

Definition

The set $\mathrm{Reg}(A)$ of regular expressions over alphabet A is defined by the grammar:

$$e, f \;::=\; 0 \mid 1 \mid a \mid (e + f) \mid (e \cdot f) \mid (e^{\star}) \qquad \text{(for } a \in A\text{).}$$

Regular expressions *(S.C. Kleene, 1951)*

Definition

The set $\mathrm{Reg}(A)$ of regular expressions over alphabet A is defined by the grammar:

$$e, f \;::=\; 0 \mid 1 \mid a \mid (e + f) \mid (e \cdot f) \mid (e^{\star}) \qquad \text{(for } a \in A\text{)}.$$

Note, here:

- symbol 0 instead of $\varnothing$
- symbol 1 used (often dropped, definable as $0^{\star}$)
- no complementation operation $\overline{e}$
  - which is not expressible under language interpretation

Language semantics $[\![\cdot]\!]_L$ of reg. expr's *(Copi–Elgot–Wright, 1958)*

$\mathbf{0} \xmapsto{L}$ empty language $\varnothing$

$\mathbf{1} \xmapsto{L} \{\epsilon\}$ (ϵ the empty word)

$a \xmapsto{L} \{a\}$

Language semantics $[\![\cdot]\!]_L$ of reg. expr's *(Copi–Elgot–Wright, 1958)*

$\mathbf{0} \xmapsto{L}$ empty language $\varnothing$

$\mathbf{1} \xmapsto{L} \{\epsilon\}$ (ϵ the empty word)

$a \xmapsto{L} \{a\}$

$e + f \xmapsto{L}$ union of $L(e)$ and $L(f)$

$e \cdot f \xmapsto{L}$ element-wise concatenation of $L(e)$ and $L(f)$

$e^* \xmapsto{L}$ set of words formed by concatenating words in $L(e)$, and adding the empty word ϵ

Language semantics $[\![\cdot]\!]_L$ of reg. expr's *(Copi–Elgot–Wright, 1958)*

$\mathbf{0} \xmapsto{L}$ empty language $\varnothing$

$\mathbf{1} \xmapsto{L} \{\epsilon\}$ (ϵ the empty word)

$a \xmapsto{L} \{a\}$

$e + f \xmapsto{L}$ union of $L(e)$ and $L(f)$

$e \cdot f \xmapsto{L}$ element-wise concatenation of $L(e)$ and $L(f)$

$e^* \xmapsto{L}$ set of words formed by concatenating words in $L(e)$, and adding the empty word ϵ

$[\![e]\!]_L := L(e)$ (language defined by e)

Process semantics of regular expressions $[\![\cdot]\!]_P$ *(Milner, 1984)*

$\mathbf{0} \quad \xmapsto{P} \quad$ deadlock $\boldsymbol{\delta}$, no termination

$\mathbf{1} \quad \xmapsto{P} \quad$ empty-step process $\boldsymbol{\epsilon}$, then terminate

$a \quad \xmapsto{P} \quad$ atomic action a, then terminate

Process semantics of regular expressions $[\![\cdot]\!]_P$ (Milner, 1984)

$\mathbf{0} \overset{P}{\longmapsto}$ deadlock $\boldsymbol{\delta}$, no termination

$\mathbf{1} \overset{P}{\longmapsto}$ empty-step process $\boldsymbol{\epsilon}$, then terminate

$a \overset{P}{\longmapsto}$ atomic action a, then terminate

$e + f \overset{P}{\longmapsto}$ (*choice*) execute $P(e)$ or $P(f)$

$e \cdot f \overset{P}{\longmapsto}$ (*sequentialization*) execute $P(e)$, then $P(f)$

$e^* \overset{P}{\longmapsto}$ (*iteration*) repeat (terminate or execute $P(e)$)

Process semantics of regular expressions $[\![\cdot]\!]_P$ (Milner, 1984)

$\mathbf{0} \stackrel{P}{\longmapsto}$ deadlock $\boldsymbol{\delta}$, no termination

$\mathbf{1} \stackrel{P}{\longmapsto}$ empty-step process $\boldsymbol{\epsilon}$, then terminate

$a \stackrel{P}{\longmapsto}$ atomic action a, then terminate

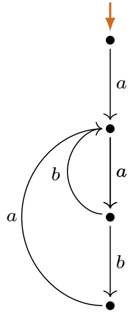
$e + f \stackrel{P}{\longmapsto}$ (*choice*) execute $P(e)$ or $P(f)$

$e \cdot f \stackrel{P}{\longmapsto}$ (*sequentialization*) execute $P(e)$, then $P(f)$

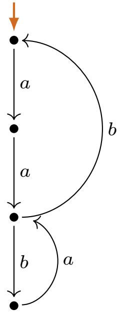
$e^* \stackrel{P}{\longmapsto}$ (*iteration*) repeat (terminate or execute $P(e)$)

$[\![e]\!]_P \;:=\; [P(e)]_{\underline{\leftrightarrow}}$ (bisimilarity equivalence class of process $P(e)$)

Process interpretation of regular expressions (examples)

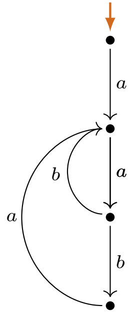


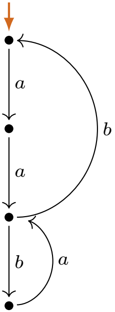
$P(a(a(b + ba))^*0)$



$P((aa(ba)^*b)^*0)$

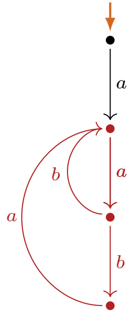
Process interpretation of regular expressions (examples)



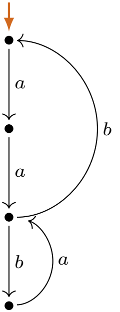


$P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$ $\qquad$ $P((a \cdot a \cdot (b \cdot a)^* \cdot b)^* \cdot 0)$

Process interpretation of regular expressions (examples)

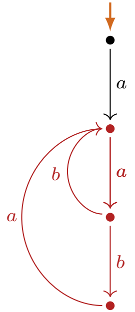


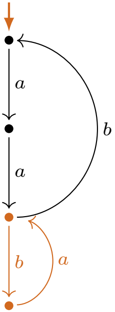
$P(a \cdot (a \cdot (b + b \cdot a))^{*} \cdot 0)$



$P((a \cdot a \cdot (b \cdot a)^{*} \cdot b)^{*} \cdot 0)$

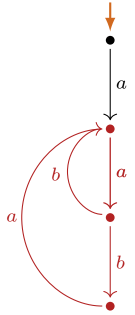
Process interpretation of regular expressions (examples)

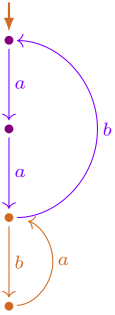




$P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$ $\qquad$ $P((a \cdot a \cdot (b \cdot a)^* \cdot b)^* \cdot 0)$

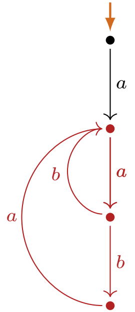
Process interpretation of regular expressions (examples)



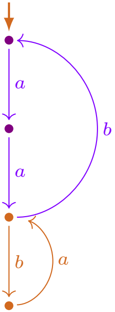


$P(a \cdot (a \cdot (b + b \cdot a))^{*} \cdot 0)$ $\qquad$ $P((a \cdot a \cdot (b \cdot a)^{*} \cdot b)^{*} \cdot 0)$

Process interpretation of regular expressions (examples)

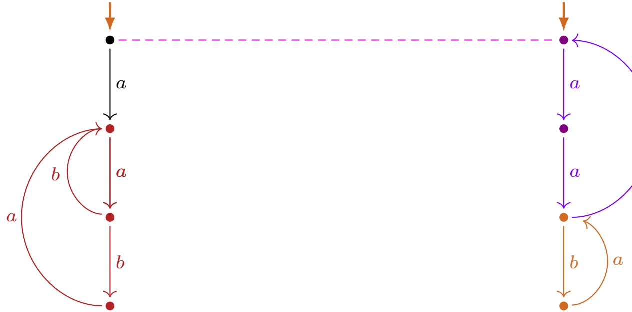


$P(a(a(b+ba))^{*}0)$



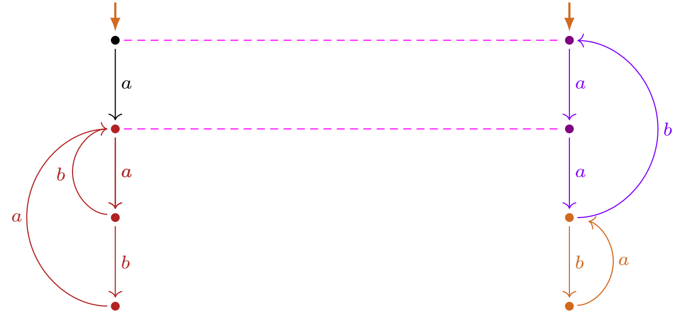
$P((aa(ba)^{*}b)^{*}0)$

Process interpretation of regular expressions (examples)



$P(a(a(b+ba))^{*}0)$ $\qquad$ $P((aa(ba)^{*}b)^{*}0)$

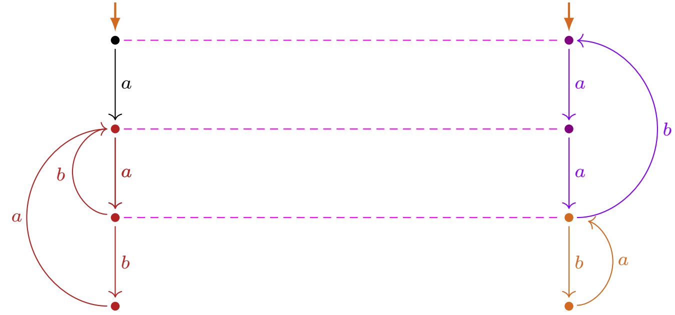
Process interpretation of regular expressions (examples)



$P(a(a(b+ba))^{*}0)$

$P((aa(ba)^{*}b)^{*}0)$

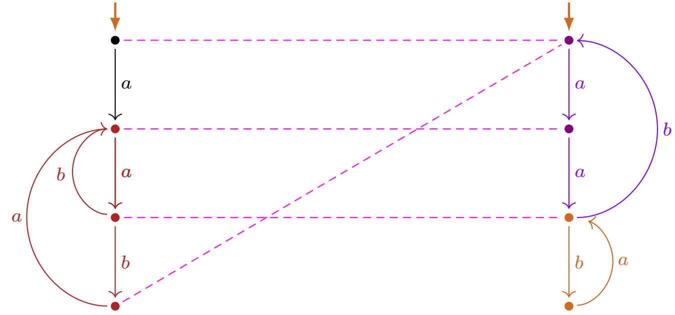
Process interpretation of regular expressions (examples)



$P(a(a(b+ba))^{*}0)$

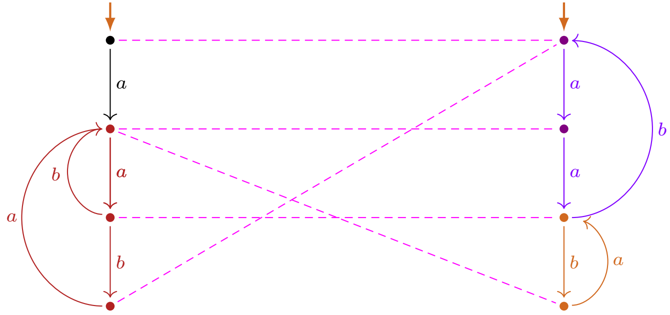
$P((aa(ba)^{*}b)^{*}0)$

Process interpretation of regular expressions (examples)



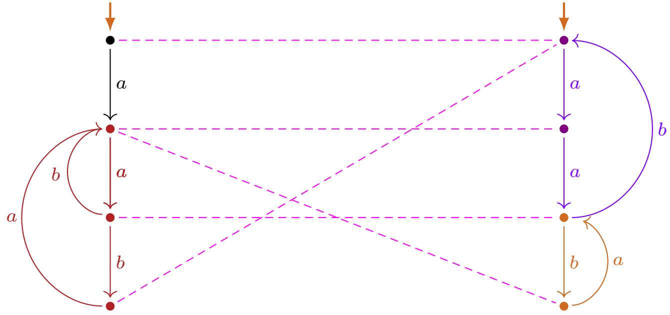
$P(a(a(b+ba))^*0)$ $\qquad$ $P((aa(ba)^*b)^*0)$

Process interpretation of regular expressions (examples)



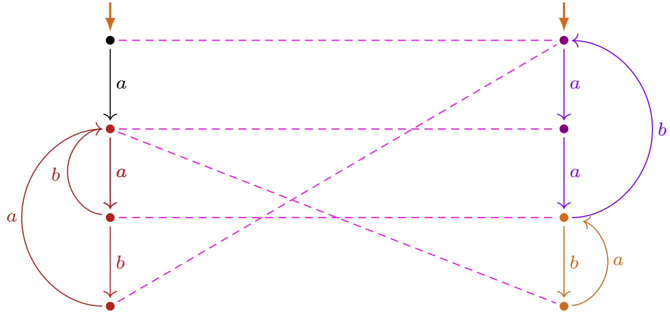
$P(a(a(b+ba))^*0)$ $\quad$ $P((aa(ba)^*b)^*0)$

Process interpretation of regular expressions (examples)



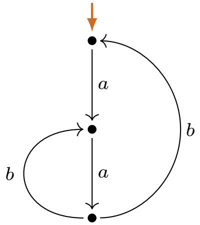
$$P(a(a(b+ba))^{*}0) \quad \underline{\leftrightarrow} \quad P((aa(ba)^{*}b)^{*}0)$$

Process interpretation of regular expressions (examples)

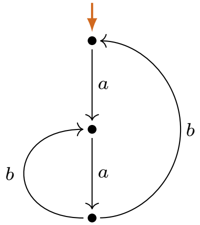


$a(a(b+ba))^{*}0 \qquad \underline{\leftrightarrow}_{P} \qquad (aa(ba)^{*}b)^{*}0$

Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)

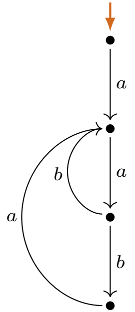


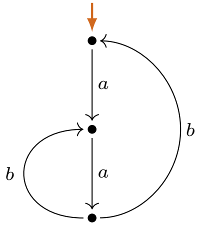
Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)



$? \in im(P(\cdot))\ ?$

Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)



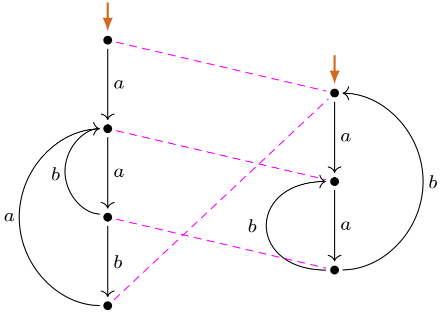


$\in im(P(\cdot))$

$P(\cdot)$-expressible

? $\in im(P(\cdot))$?

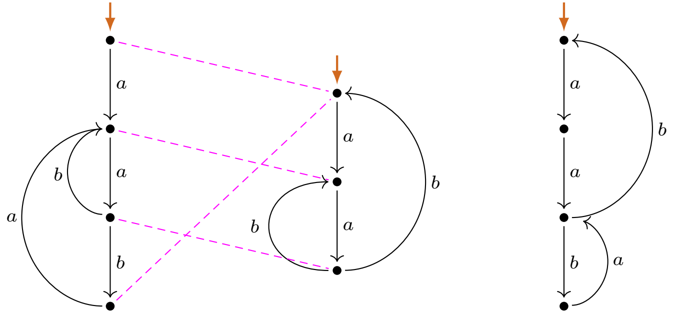
Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)



$\in im(P(\cdot))$ $\quad$? $\in im(P(\cdot))$?

$P(\cdot)$-expressible

Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)

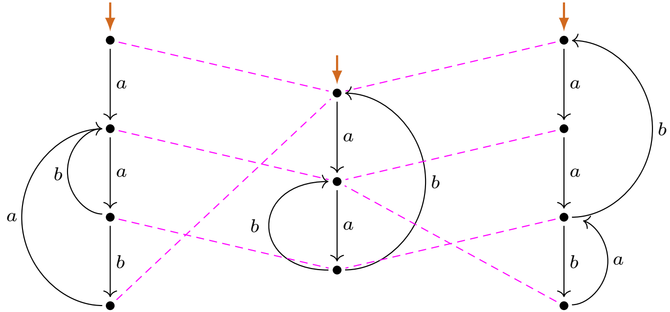


$\in im(P(\cdot))$ $P(\cdot)$-expressible

$? \in im(P(\cdot))\ ?$

$\in im(P(\cdot))$ $P(\cdot)$-expressible

Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)



$\in im(P(\cdot))$

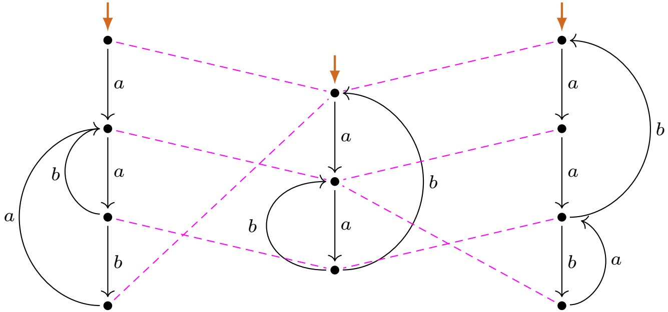
$P(\cdot)$-expressible

? $\in im(P(\cdot))$?

$\in im(P(\cdot))$

$P(\cdot)$-expressible

Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)

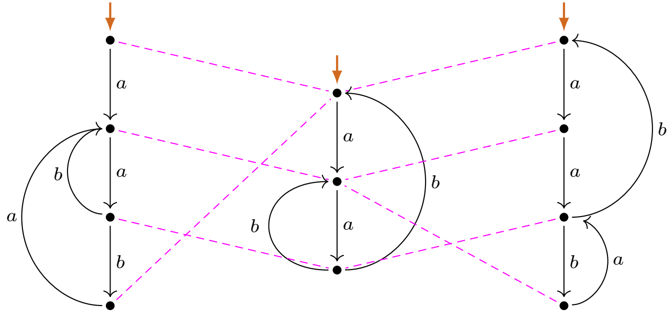


$\in im(P(\cdot))$ — $P(\cdot)$-expressible

? $\in im(P(\cdot))$? — $P(\cdot)$-expressible modulo $\underline{\leftrightarrow}$

$\in im(P(\cdot))$ — $P(\cdot)$-expressible

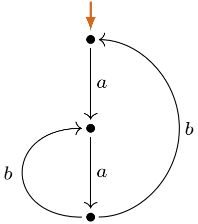
Expressible process graphs (under bisimulation $\underline{\leftrightarrow}$)



$\in im(P(\cdot))$	? $\in im(P(\cdot))$?	$\in im(P(\cdot))$
$P(\cdot)$-expressible	$P(\cdot)$-expressible modulo $\underline{\leftrightarrow}$	$P(\cdot)$-expressible
$[\![\cdot]\!]_P$-expressible	$[\![\cdot]\!]_P$-expressible	$[\![\cdot]\!]_P$-expressible

Properties of P and $[\![\cdot]\!]_P$

- Not every finite-state process is $P(\cdot)$-expressible.

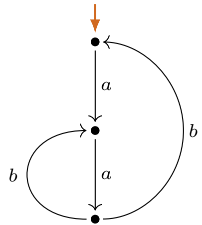


? $P(\cdot)$-expressible ?

$[\![\cdot]\!]_P$-expressible

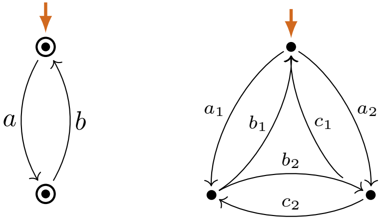
Properties of P and $[\![\cdot]\!]_P$

- Not every finite-state process is $P(\cdot)$-expressible.



? $P(\cdot)$-expressible ?

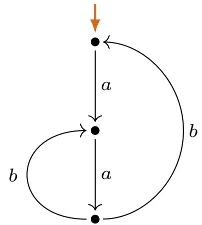
$[\![\cdot]\!]_P$-expressible

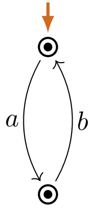


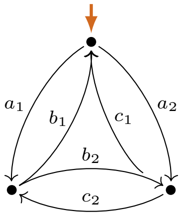
not $P(\cdot)$-expressible

Properties of P and $[\![\cdot]\!]_P$

- **Not** every finite-state process is $P(\cdot)$-expressible.
- **Not** every finite-state process is $[\![\cdot]\!]_P$-expressible.







? $P(\cdot)$-expressible ?

$[\![\cdot]\!]_P$-expressible

not $P(\cdot)$-expressible

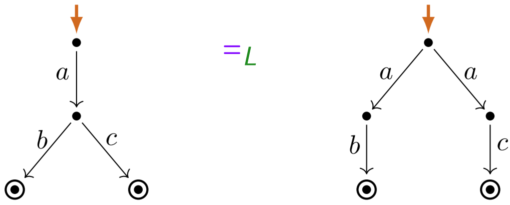
not $[\![\cdot]\!]_P$-expressible

Properties of P and $[\![\cdot]\!]_P$

- Not every finite-state process is $P(\cdot)$-expressible.
- Not every finite-state process is $[\![\cdot]\!]_P$-expressible.
- Fewer identities hold for $\underline{\leftrightarrow}_P$ than for $=_L$: $\underline{\leftrightarrow}_P \subsetneqq =_L$.

Properties of P and $[\![\cdot]\!]_P$

- Not every finite-state process is $P(\cdot)$-expressible.
- Not every finite-state process is $[\![\cdot]\!]_P$-expressible.
- Fewer identities hold for $\underline{\leftrightarrow}_P$ than for $=_L$: $\quad \underline{\leftrightarrow}_P \subsetneqq =_L$.



$$a \cdot (b + c) \quad =_L \quad a \cdot b + a \cdot c$$

Properties of P and $[\![\cdot]\!]_P$

- Not every finite-state process is $P(\cdot)$-expressible.
- Not every finite-state process is $[\![\cdot]\!]_P$-expressible.
- Fewer identities hold for $\underline{\leftrightarrow}_P$ than for $=_L$: $\quad \underline{\leftrightarrow}_P \subsetneqq =_L$.

$$a \cdot (b + c) \quad \not\underline{\leftrightarrow}_P \quad a \cdot b + a \cdot c$$

Complete axiomatization of $=_L$ *(Aanderaa/Salomaa, 1965/66)*

Axioms :

(B1)	$e + (f + g) = (e + f) + g$	(B7)	$e \cdot 1 = e$
(B2)	$(e \cdot f) \cdot g = e \cdot (f \cdot g)$	(B8)	$e \cdot 0 = 0$
(B3)	$e + f = f + e$	(B9)	$e + 0 = e$
(B4)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(B10)	$e^* = 1 + e \cdot e^*$
(B5)	$e \cdot (f + g) = e \cdot f + e \cdot g$	(B11)	$e^* = (1 + e)^*$
(B6)	$e + e = e$		

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g}\ \text{FIX} \qquad (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Sound and unsound axioms with respect to $\underline{\leftrightarrow}_P$

Axioms :

(B1)	$e + (f + g) = (e + f) + g$	(B7)	$e \cdot 1 = e$
(B2)	$(e \cdot f) \cdot g = e \cdot (f \cdot g)$	(B8)	$e \cdot 0 = 0$
(B3)	$e + f = f + e$	(B9)	$e + 0 = e$
(B4)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(B10)	$e^* = 1 + e \cdot e^*$
(B5)	$e \cdot (f + g) = e \cdot f + e \cdot g$	(B11)	$e^* = (1 + e)^*$
(B6)	$e + e = e$		

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g}\ \text{FIX} \quad (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Sound and unsound axioms with respect to $\underline{\leftrightarrow}_P$

Axioms :

(B1)	$e + (f + g) = (e + f) + g$	(B7)	$e \cdot 1 = e$
(B2)	$(e \cdot f) \cdot g = e \cdot (f \cdot g)$	(B8)	$e \cdot 0 = 0$
(B3)	$e + f = f + e$	(B9)	$e + 0 = e$
(B4)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(B10)	$e^* = 1 + e \cdot e^*$
(B5)	$e \cdot (f + g) = e \cdot f + e \cdot g$	(B11)	$e^* = (1 + e)^*$
(B6)	$e + e = e$	(B8)′	$0 \cdot e = 0$

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g}\ \text{FIX} \quad (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Adaptation for $\underline{\leftrightarrow}_P$ *(Milner, 1984)* (Mil = Mil⁻ + RSP*)

Axioms :

(B1)	$e + (f + g) = (e + f) + g$	(B7)	$e \cdot 1 = e$
(B2)	$(e \cdot f) \cdot g = e \cdot (f \cdot g)$		
(B3)	$e + f = f + e$	(B9)	$e + 0 = e$
(B4)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(B10)	$e^* = 1 + e \cdot e^*$
		(B11)	$e^* = (1 + e)^*$
(B6)	$e + e = e$	(B8)′	$0 \cdot e = 0$

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g} \; \mathsf{RSP}^* \; (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Adaptation for $\underline{\leftrightarrow}_P$ *(Milner, 1984)* (Mil = Mil$^-$ + RSP*)

Axioms :

(B1)	$e + (f + g) = (e + f) + g$	(B7)	$e \cdot 1 = e$
(B2)	$(e \cdot f) \cdot g = e \cdot (f \cdot g)$	(B8)′	$0 \cdot e = 0$
(B3)	$e + f = f + e$	(B9)	$e + 0 = e$
(B4)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(B10)	$e^* = 1 + e \cdot e^*$
		(B11)	$e^* = (1 + e)^*$
(B6)	$e + e = e$		

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g}\ \text{RSP}^* \quad (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Adaptation for $\underline{\leftrightarrow}_P$ *(Milner, 1984)* (Mil = Mil⁻ + RSP*)

Axioms :

$$
\begin{array}{llll}
\text{(B1)} & e + (f + g) = (e + f) + g & \text{(B7)} & e \cdot 1 = e \\
\text{(B2)} & (e \cdot f) \cdot g = e \cdot (f \cdot g) & \text{(B8)}' & 0 \cdot e = 0 \\
\text{(B3)} & e + f = f + e & \text{(B9)} & e + 0 = e \\
\text{(B4)} & (e + f) \cdot g = e \cdot g + f \cdot g & \text{(B10)} & e^* = 1 + e \cdot e^* \\
 & & \text{(B11)} & e^* = (1 + e)^* \\
\text{(B6)} & e + e = e & &
\end{array}
$$

Inference rules : equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g}\ \mathsf{RSP}^* \ (\text{if } \underbrace{\{\epsilon\} \notin L(f)}_{\textit{non-empty-word property}})$$

Milner's questions

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

Milner's questions

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

- Mil is complete for perpetual–loop expressions *(Fokkink, 1996)*

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

- Mil is complete for perpetual–loop expressions *(Fokkink, 1996)*
- Mil is complete when restricted to 0-free and 1-return-less expressions *(Corradini, De Nicola, Labella, 2002)*

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

- Mil is complete for perpetual–loop expressions *(Fokkink, 1996)*
- Mil is complete when restricted to 0-free and 1-return-less expressions *(Corradini, De Nicola, Labella, 2002)*
- Mil^{-} + one of two stronger rules (than RSP^{*}) is complete *(G, 2006)*

Milner's questions, and results

Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

- Mil is complete for perpetual–loop expressions *(Fokkink, 1996)*
- Mil is complete when restricted to 0-free and 1-return-less expressions *(Corradini, De Nicola, Labella, 2002)*
- Mil^{-} + one of two stronger rules (than RSP^{*}) is complete *(G, 2006)*
- Mil is complete when restricted to 1-free expressions *(G, Fokkink, 2020)*

Milner's questions, and results

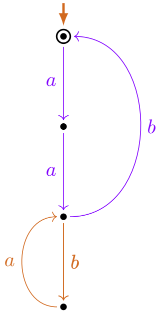
Q1. Recognition: Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P$-expressibility?

- definability by well-behaved specifications *(Baeten/Corradini, 2005)*
- that is decidable (super–exponentially) *(Baeten/Corradini/G, 2007)*

Q2. Complete axiomatization: Is Mil complete for $\underline{\leftrightarrow}_P$?

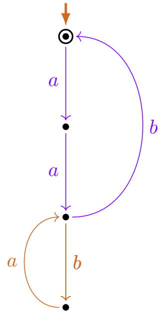
- Mil is complete for perpetual–loop expressions *(Fokkink, 1996)*
- Mil is complete when restricted to 0-free and 1-return-less expressions *(Corradini, De Nicola, Labella, 2002)*
- Mil^- + one of two stronger rules (than RSP^*) is complete *(G, 2006)*
- Mil is complete when restricted to 1-free expressions *(G, Fokkink, 2020)*
- Mil is complete *(G, 2022, proof overview)*

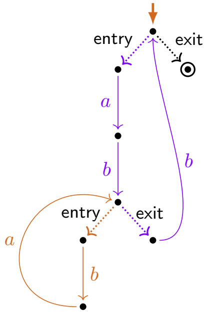
Well-behaved form, looping palm trees



$P((aa(ba)^{*}b)^{*})$

Well-behaved form, looping palm trees

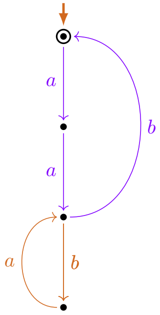


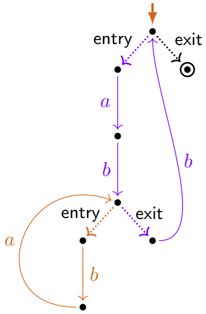


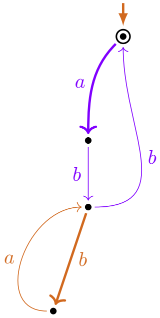
well-behaved form
(Corradini, Baeten)

$P((aa(ba)^*b)^*)$ $P((1 \cdot aa(1 \cdot ba)^* \cdot 1 \cdot b)^*(1 \cdot 1))$

Well-behaved form, looping palm trees







well-behaved form
(Corradini, Baeten)

looping palm tree

$P((aa(ba)^*b)^*)$

$P((1 \cdot aa(1 \cdot ba)^* \cdot 1 \cdot b)^*(1 \cdot 1))$

$P((aa(ba)^*b)^*)$

Loop charts (interpretations of innermost iterations)

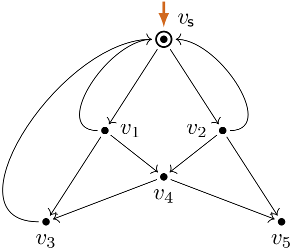
Definition

A process graph is a loop chart if:

L-1.

L-2.

L-3.



Loop charts (interpretations of innermost iterations)

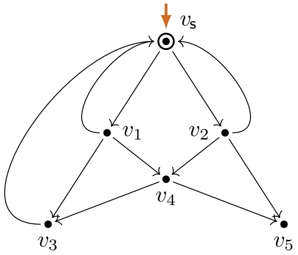
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2.

L-3.



Loop charts (interpretations of innermost iterations)

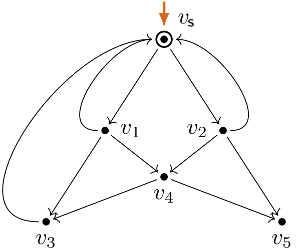
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3.



Loop charts (interpretations of innermost iterations)

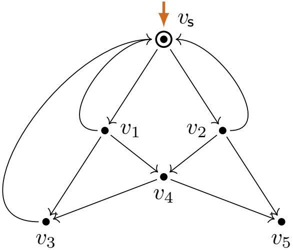
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



Loop charts (interpretations of innermost iterations)

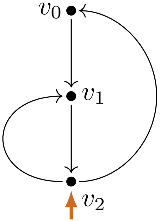
Definition

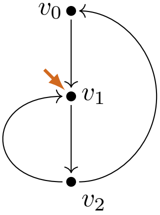
A process graph is a loop chart if:

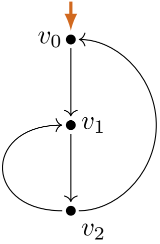
L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.







Loop charts (interpretations of innermost iterations)

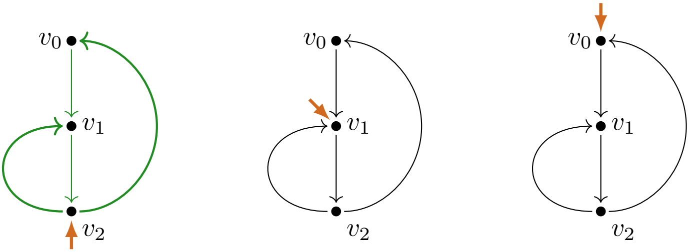
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



Loop charts (interpretations of innermost iterations)

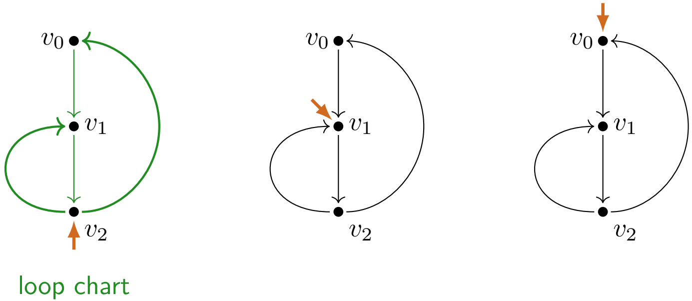
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



Loop charts (interpretations of innermost iterations)

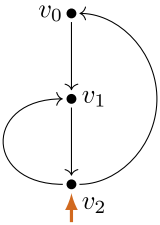
Definition

A process graph is a loop chart if:

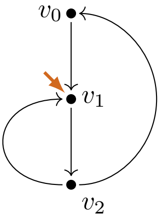
L-1. There is an infinite path from the start vertex.

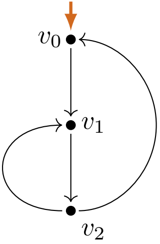
L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



loop chart





Loop charts (interpretations of innermost iterations)

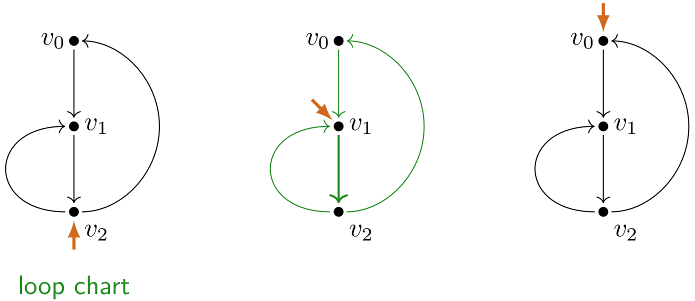
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



Loop charts (interpretations of innermost iterations)

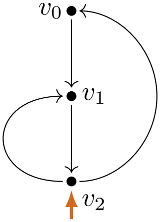
Definition

A process graph is a loop chart if:

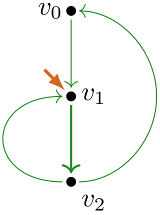
L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

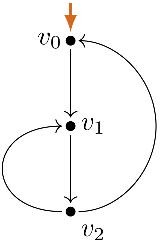
L-3. Termination is only possible at the start vertex.



loop chart



loop chart



Loop charts (interpretations of innermost iterations)

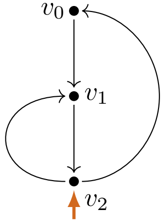
Definition

A process graph is a loop chart if:

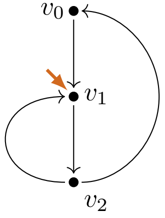
L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

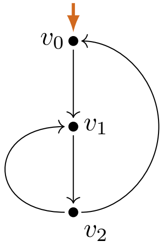
L-3. Termination is only possible at the start vertex.



loop chart



loop chart



Loop charts (interpretations of innermost iterations)

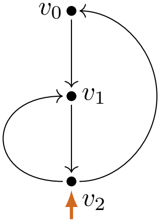
Definition

A process graph is a loop chart if:

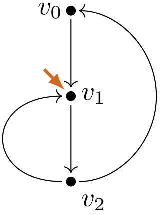
L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

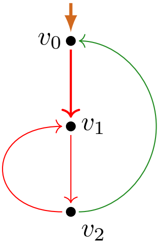
L-3. Termination is only possible at the start vertex.



loop chart



loop chart



Loop charts (interpretations of innermost iterations)

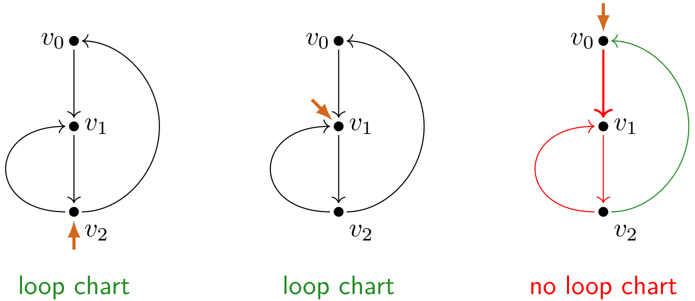
Definition

A process graph is a loop chart if:

L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

L-3. Termination is only possible at the start vertex.



Loop charts (interpretations of innermost iterations)

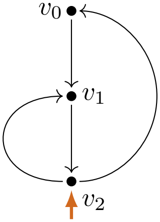
Definition

A process graph is a loop chart if:

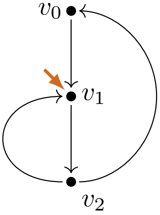
L-1. There is an infinite path from the start vertex.

L-2. Every infinite path from the start vertex returns to it.

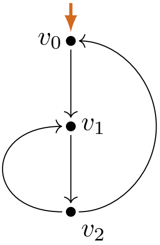
L-3. Termination is only possible at the start vertex.



loop chart

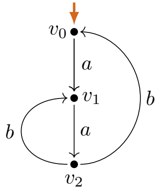


loop chart

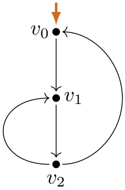


no loop chart

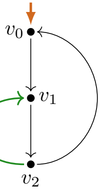
Loop elimination



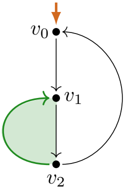
Loop elimination



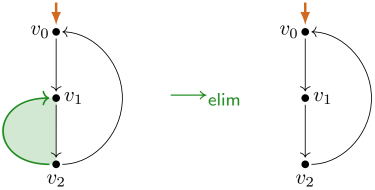
Loop elimination



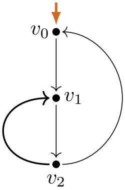
Loop elimination



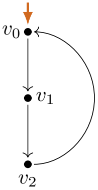
Loop elimination



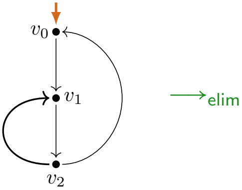
Loop elimination



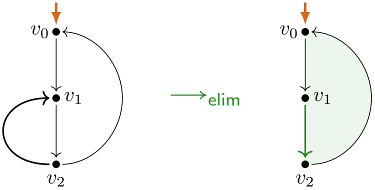
$\longrightarrow_{\text{elim}}$



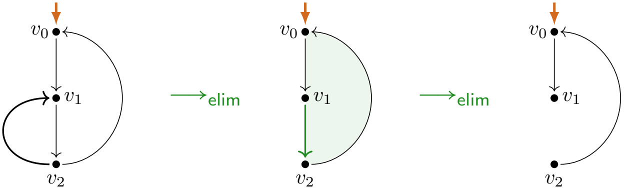
Loop elimination



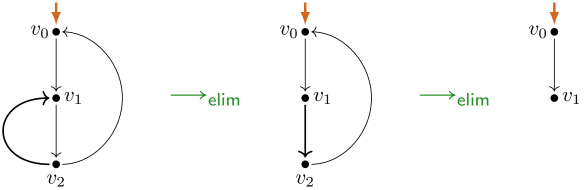
Loop elimination



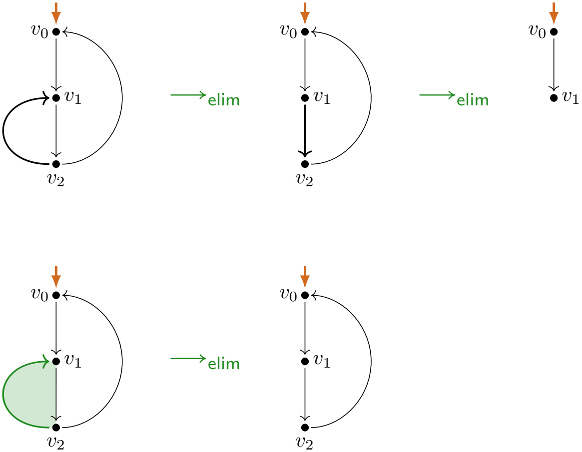
Loop elimination



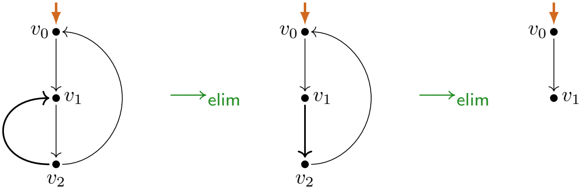
Loop elimination

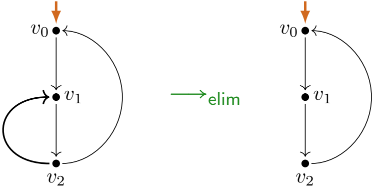


Loop elimination

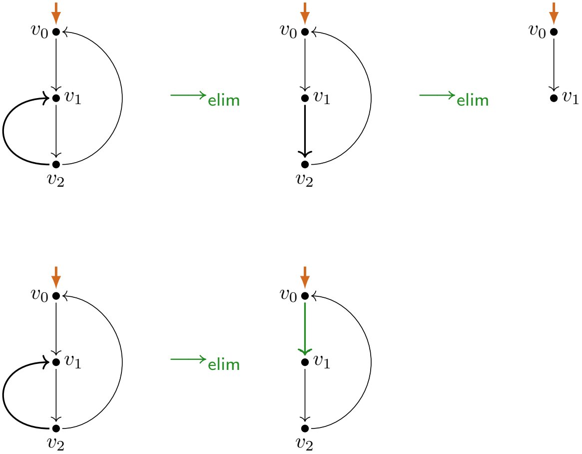


Loop elimination

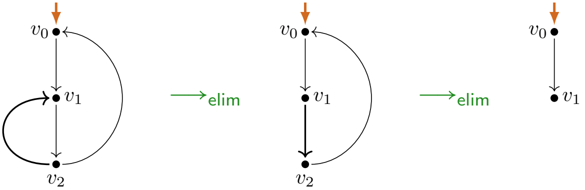


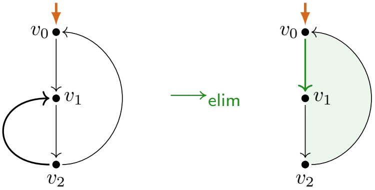


Loop elimination

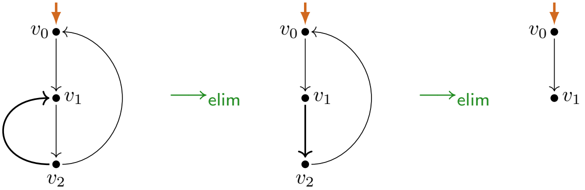


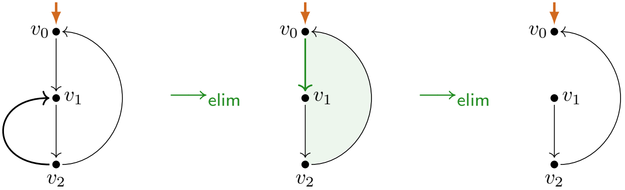
Loop elimination



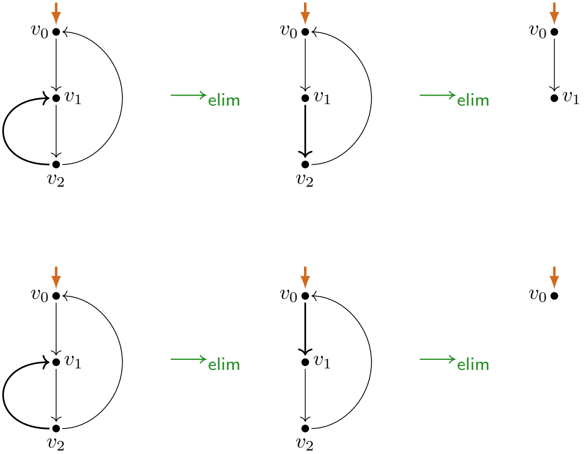


Loop elimination

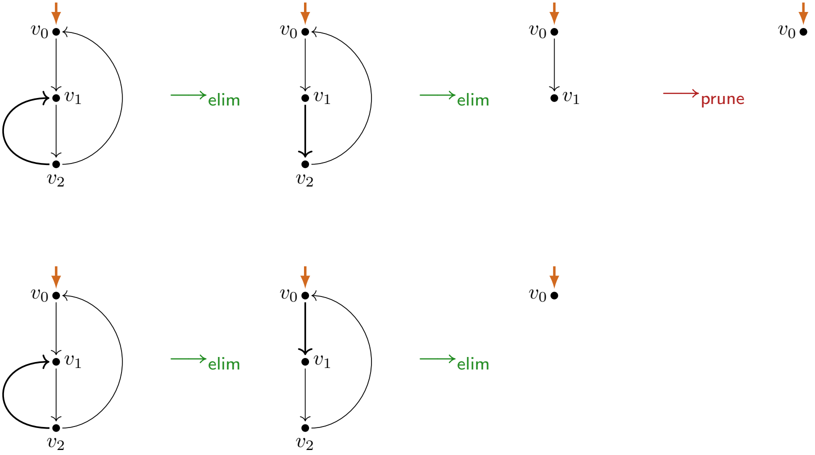




Loop elimination



Loop elimination



Loop elimination, and properties

$\longrightarrow_{\mathrm{elim}}$: eliminate a transition-induced loop by:

- removing the loop-entry transition(s)
- garbage collection

$\longrightarrow_{\mathrm{prune}}$: remove a transition to a deadlocking state

Lemma

(i) $\longrightarrow_{\mathrm{elim}} \cup \longrightarrow_{\mathrm{prune}}$ *is terminating.*

Loop elimination, and properties

$\longrightarrow_{\text{elim}}$: eliminate a transition-induced loop by:

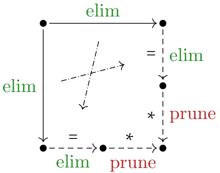
- removing the loop-entry transition(s)
- garbage collection

$\longrightarrow_{\text{prune}}$: remove a transition to a deadlocking state

Lemma

(i) $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$ *is terminating.*

(ii) $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$ *is decreasing, and hence locally confluent.*



Loop elimination, and properties

$\longrightarrow_{\text{elim}}$: eliminate a transition-induced loop by:

- removing the loop-entry transition(s)
- garbage collection

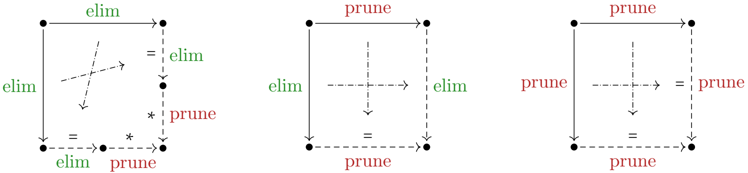
$\longrightarrow_{\text{prune}}$: remove a transition to a deadlocking state

Lemma

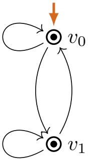
(i) $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$ *is terminating.*

(ii) $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$ *is decreasing, and hence locally confluent.*

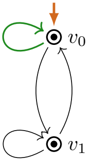
(iii) $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$ *is confluent.*



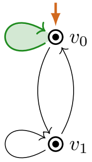
Loop elimination



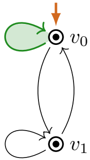
Loop elimination



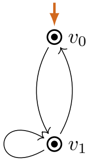
Loop elimination



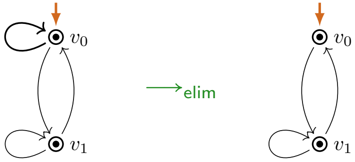
Loop elimination



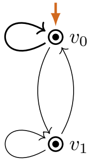
$\longrightarrow_{\text{elim}}$



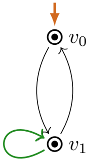
Loop elimination



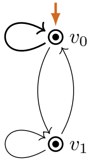
Loop elimination



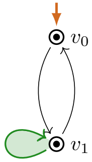
$\longrightarrow_{\text{elim}}$



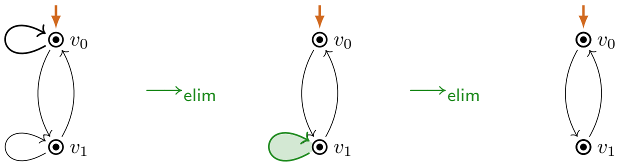
Loop elimination



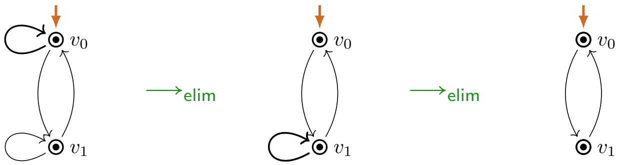
$\longrightarrow_{\text{elim}}$



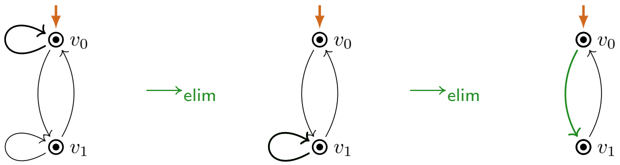
Loop elimination



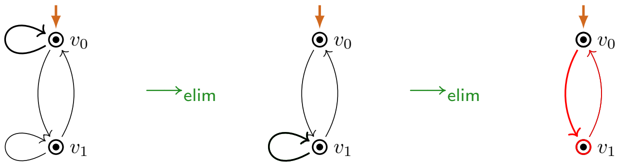
Loop elimination



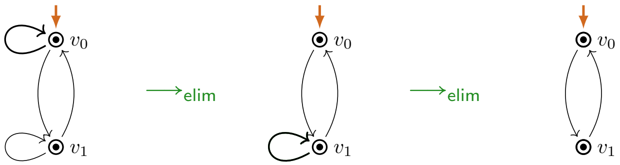
Loop elimination



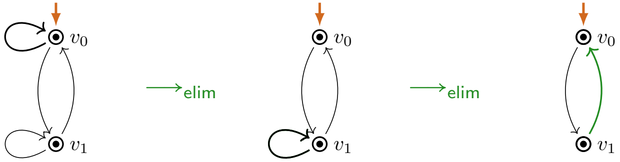
Loop elimination



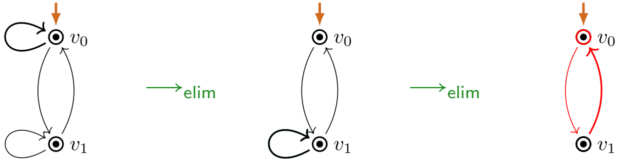
Loop elimination



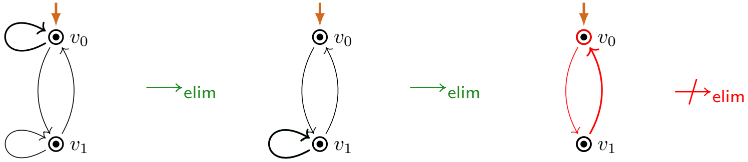
Loop elimination



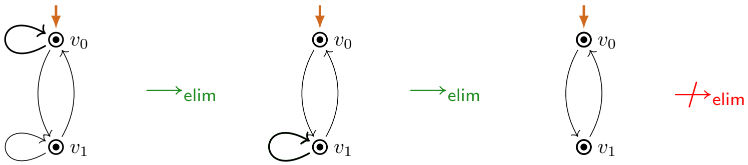
Loop elimination



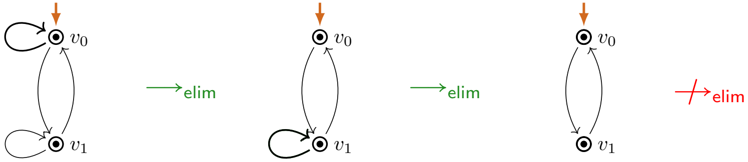
Loop elimination

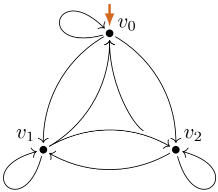


Loop elimination

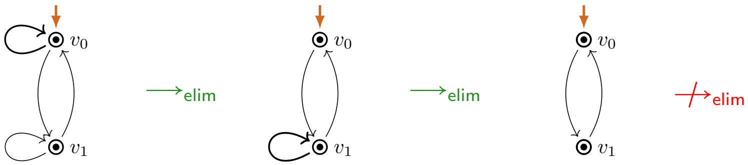


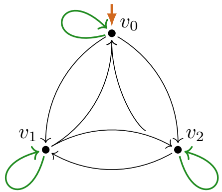
Loop elimination



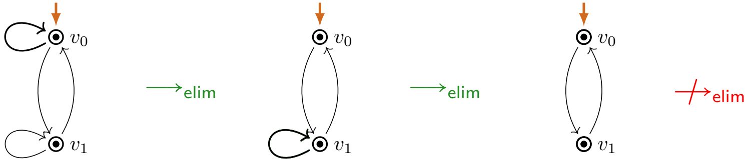


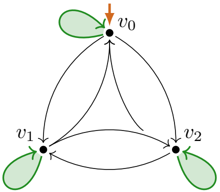
Loop elimination



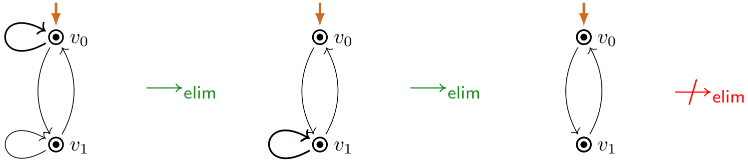


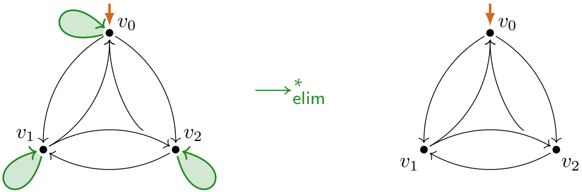
Loop elimination



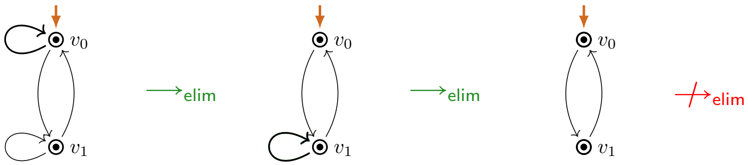


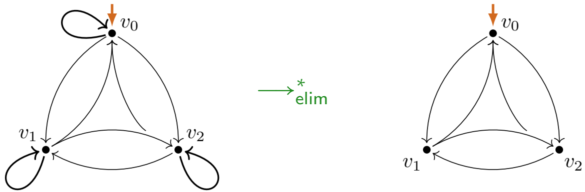
Loop elimination



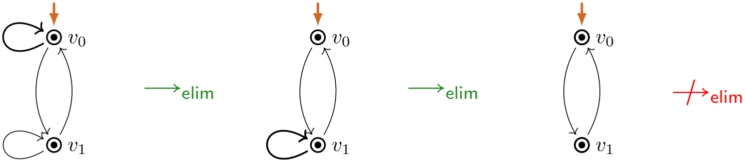


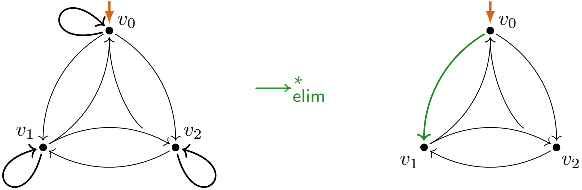
Loop elimination



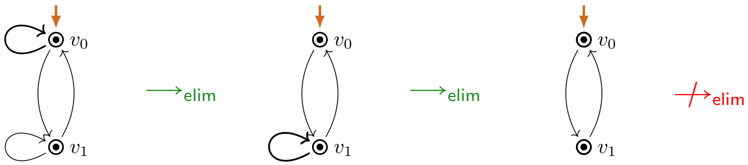


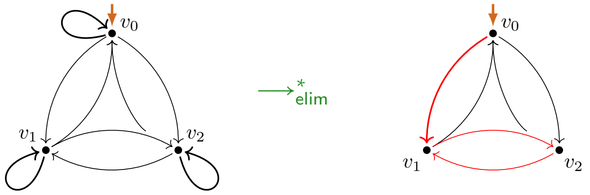
Loop elimination



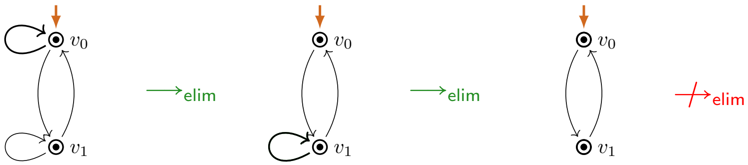


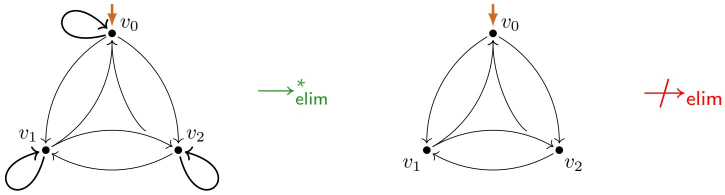
Loop elimination





Loop elimination





Structure property LEE

Definition

A process graph G satisfies LEE (*loop existence and elimination*) if:

$$\exists\, G_0 \,\Big(\, G \longrightarrow^{*}_{\mathsf{elim}} G_0 \not\longrightarrow_{\mathsf{elim}} \\ \wedge\; G_0 \text{ has no infinite trace} \,\Big)\,.$$

Structure property LEE

Definition

A process graph G satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \Big(G \longrightarrow^{*}_{\text{elim}} G_0 \not\longrightarrow_{\text{elim}} \wedge\ G_0 \text{ has no infinite trace} \Big) .$$

Lemma (by using termination and confluence)

For every process graph G the following are equivalent:

(i) LEE(G).

(ii) *There is an $\longrightarrow_{\text{elim}}$ normal form without an infinite trace.*

Structure property LEE

Definition

A process graph G satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \Big(G \longrightarrow^{*}_{\text{elim}} G_0 \not\longrightarrow_{\text{elim}} \\ \wedge\ G_0 \text{ has no infinite trace} \Big) .$$

Lemma (by using termination and confluence)

For every process graph G the following are equivalent:

(i) LEE(G).

(ii) *There is an $\longrightarrow_{\text{elim}}$ normal form without an infinite trace.*

(iii) *There is an $\longrightarrow_{\text{elim,prune}}$ normal form without an infinite trace.*

Structure property LEE

Definition

A process graph G satisfies LEE (*loop existence and elimination*) if:

$$\exists\, G_0 \left(G \longrightarrow^{*}_{\text{elim}} G_0 \not\longrightarrow_{\text{elim}} \;\wedge\; G_0 \text{ has no infinite trace} \right) .$$

Lemma (by using termination and confluence)

For every process graph G the following are equivalent:

(i) LEE(G).

(ii) *There is an $\longrightarrow_{\text{elim}}$ normal form without an infinite trace.*

(iii) *There is an $\longrightarrow_{\text{elim,prune}}$ normal form without an infinite trace.*

(iv) *Every $\longrightarrow_{\text{elim}}$ normal form is without an infinite trace.*

(v) *Every $\longrightarrow_{\text{elim,prune}}$ normal form is without an infinite trace.*

Structure property LEE

Definition

A process graph G satisfies LEE (*loop existence and elimination*) if:

$$\exists\, G_0 \Big(G \longrightarrow^{*}_{\mathsf{elim}} G_0 \not\longrightarrow_{\mathsf{elim}} \\ \wedge\; G_0 \text{ has no infinite trace} \Big) .$$

Lemma (by using termination and confluence)

For every process graph G the following are equivalent:

(i) LEE(G).

(ii) *There is an $\longrightarrow_{\mathsf{elim}}$ normal form without an infinite trace.*

(iii) *There is an $\longrightarrow_{\mathsf{elim},\mathsf{prune}}$ normal form without an infinite trace.*

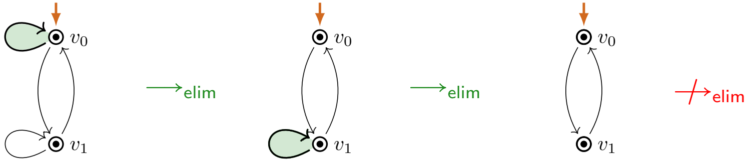
(iv) *Every $\longrightarrow_{\mathsf{elim}}$ normal form is without an infinite trace.*

(v) *Every $\longrightarrow_{\mathsf{elim},\mathsf{prune}}$ normal form is without an infinite trace.*

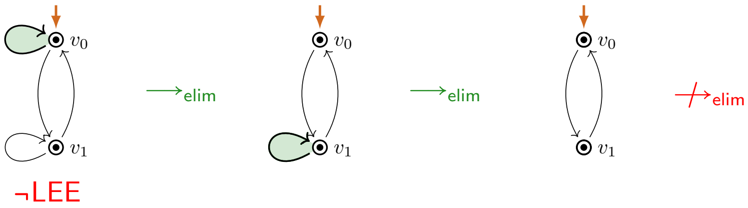
Theorem (efficient decidability)

The problem of deciding LEE(G) for process graphs G is in PTIME.

LEE fails

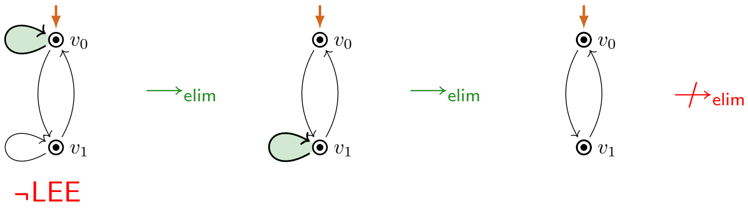

v_0
v_1
$\longrightarrow_{\text{elim}}$
v_0
v_1
$\longrightarrow_{\text{elim}}$
v_0
v_1
$\not\longrightarrow_{\text{elim}}$

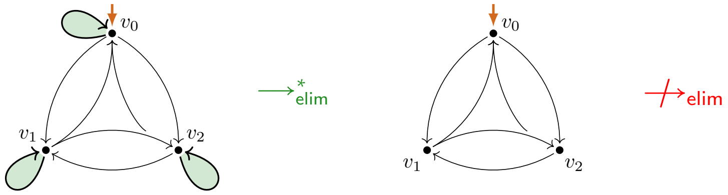
LEE fails



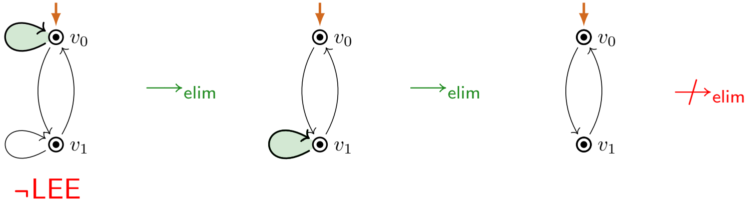
v_0
v_1
¬LEE
$\longrightarrow_{\text{elim}}$
v_0
v_1
$\longrightarrow_{\text{elim}}$
v_0
v_1
$\not\longrightarrow_{\text{elim}}$

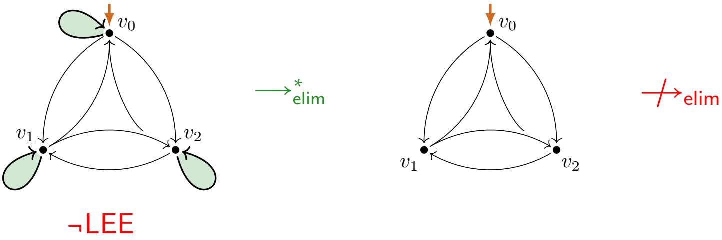
LEE fails



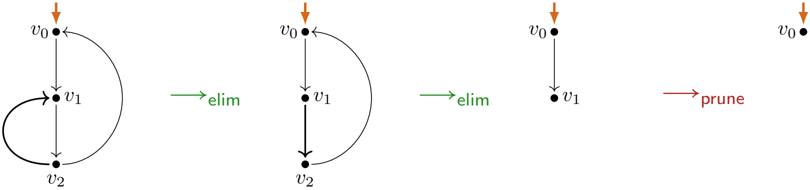


LEE fails

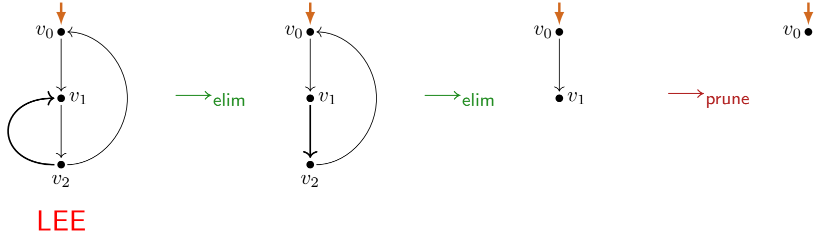

v_0
v_1
elim
v_0
v_1
elim
v_0
v_1
elim
¬LEE


v_0
v_1
v_2
$*$
elim
v_0
v_1
v_2
elim
¬LEE

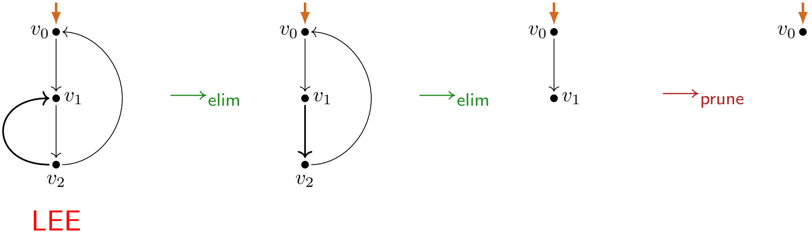
LEE holds



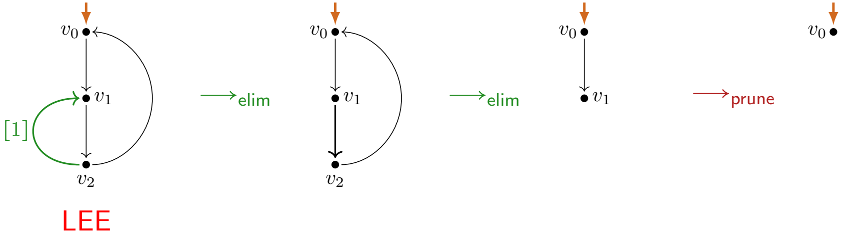
LEE holds



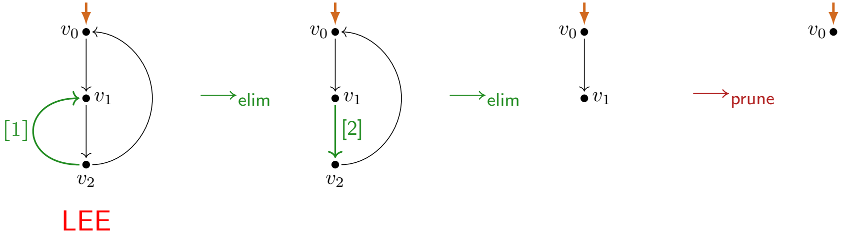
LEE holds / Recording loop elimination



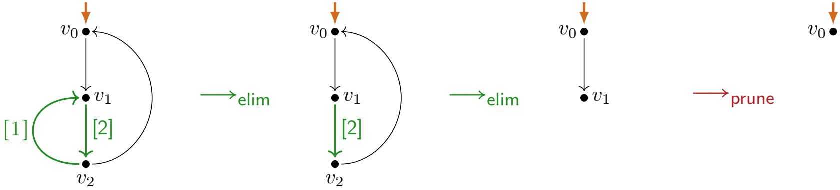
LEE holds / Recording loop elimination



LEE holds / Recording loop elimination

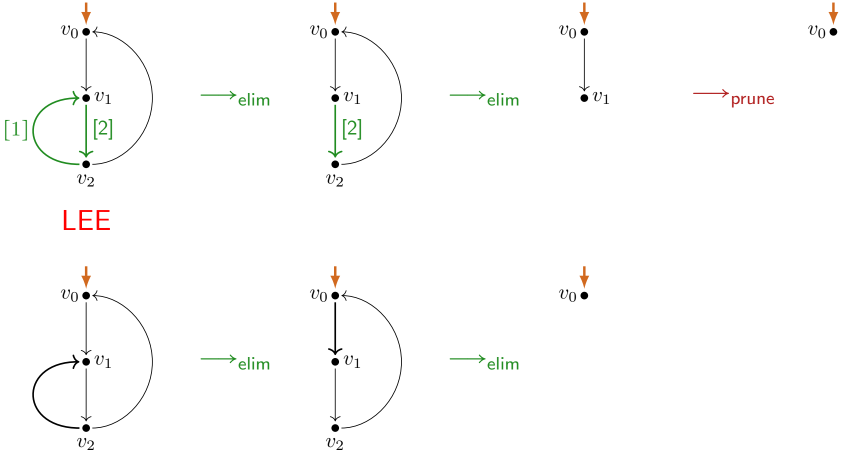


LEE holds / Recording loop elimination

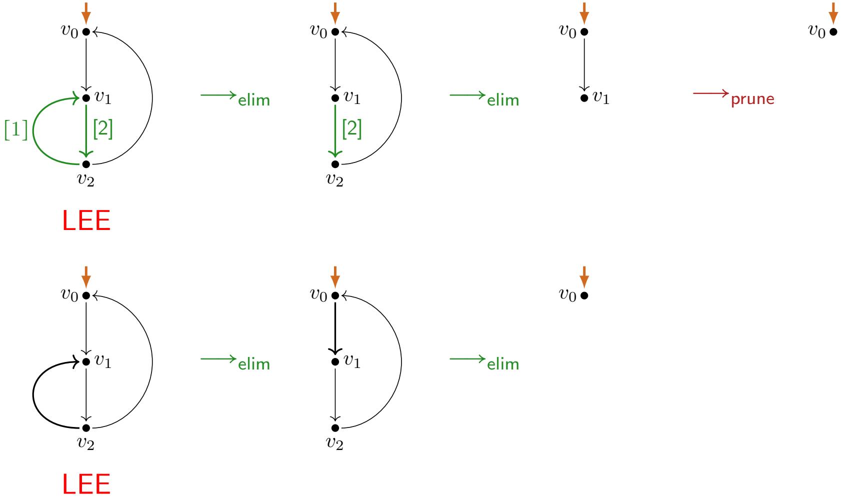


LEE

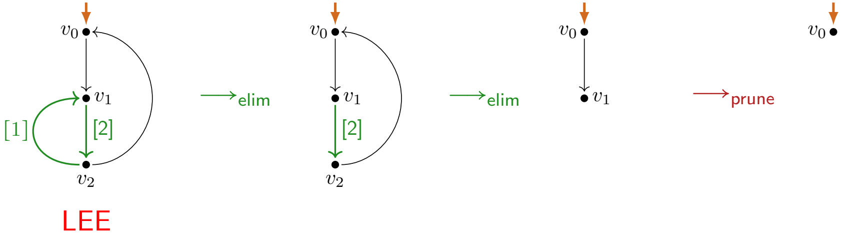
LEE holds / Recording loop elimination

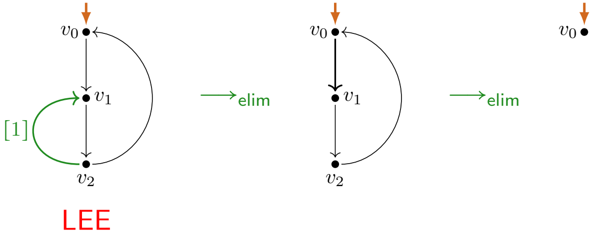


LEE holds / Recording loop elimination

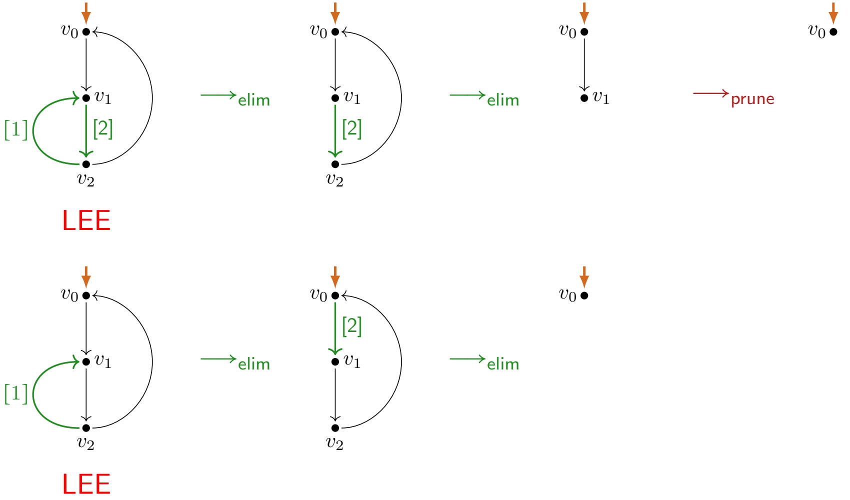


LEE holds / Recording loop elimination

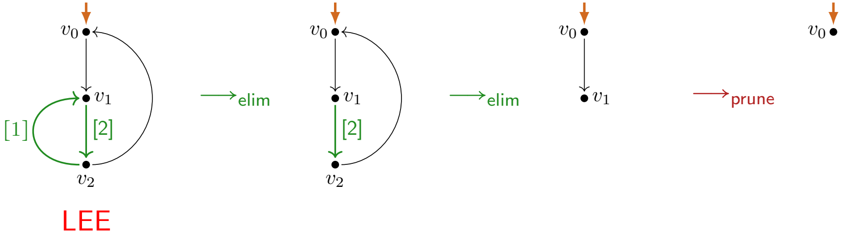


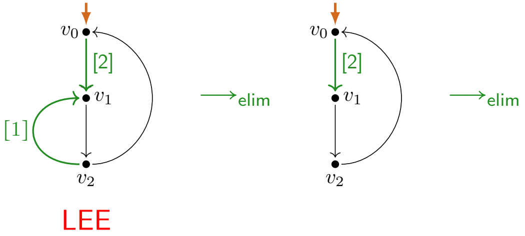


LEE holds / Recording loop elimination

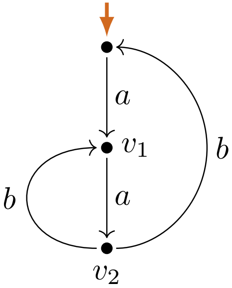


LEE holds / Recording loop elimination



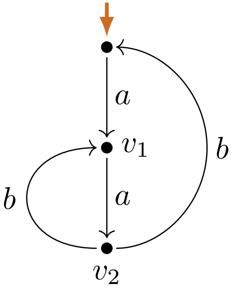


LEE-witness



LEE-witness

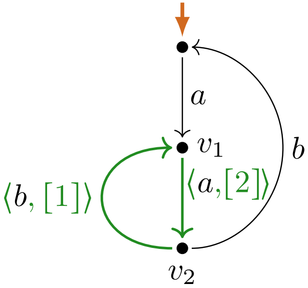
loop–branch labeling: marking transitions $\xrightarrow{a}$ as:



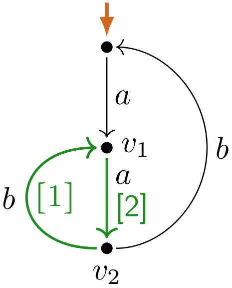
LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$,



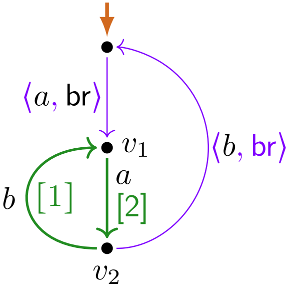
LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,

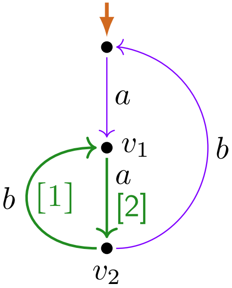
LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$,

LEE-witness



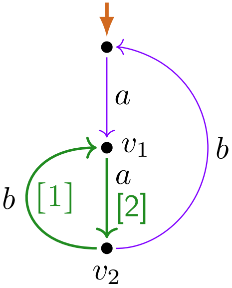
loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.



Definition

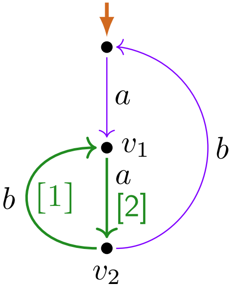
A loop–branch labeling is a LEE-witness, if:

L1.

L2.

L3.

LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

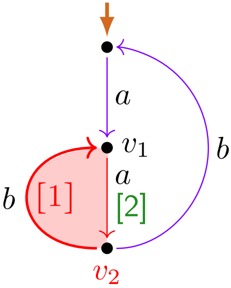
L1.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]}) :=$ subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

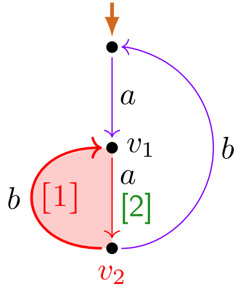
L1.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



$\mathcal{L}(v_2, \to_{[1]}, \to_{\text{br},[>1]})$
is loop subchart

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\text{br}\rangle}$, written $\xrightarrow{a}_{\text{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

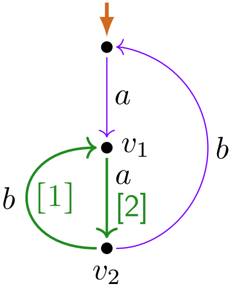
L1.

L2.

L3.

$\mathcal{L}(v, \to_{[n]}, \to_{\text{br},[>n]})$:= subchart induced
by entry steps $\to_{[n]}$ from v
followed by branch steps $\to_{\text{br}}$
or entry steps $\to_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\text{br}\rangle}$, written $\xrightarrow{a}_{\text{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

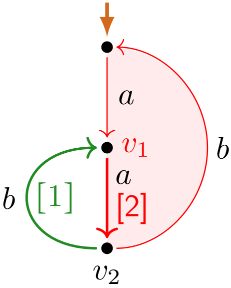
L1.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\text{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\text{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



$\mathcal{L}(v_1, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

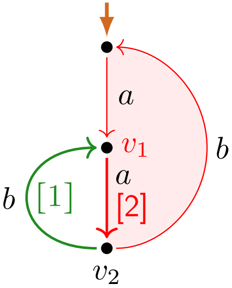
L1.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



$\mathcal{L}(v_1, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$
is loop subchart

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

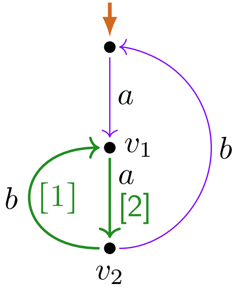
L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \rightarrow_{[n]} \;\Rightarrow\; \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]}) \text{ is a loop subchart} \right)$.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

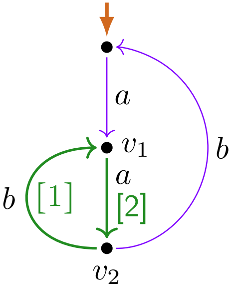
L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \rightarrow_{[n]} \Rightarrow \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]}) \text{ is a loop subchart} \right)$.

L2.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

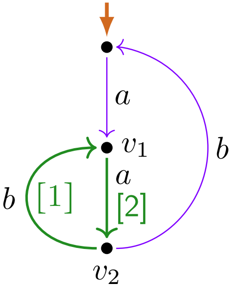
L1. $\forall n \in \mathbb{N} \forall v \in V \left(\begin{array}{c} v \rightarrow_{[n]} \Rightarrow \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]}) \\ \text{is a loop subchart} \end{array} \right)$.

L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

L3.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

L1. $\forall n \in \mathbb{N} \forall v \in V \Big(v \rightarrow_{[n]} \;\Rightarrow\; \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$ is a loop subchart $\Big)$.

L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

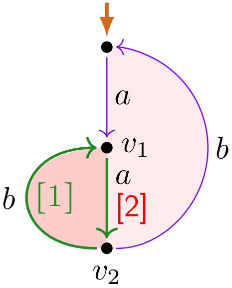
L3. Loop subcharts contained in other loop subcharts have different entry-step levels.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.



$\mathcal{L}(v_2, \to_{[1]}, \to_{\mathsf{br},[>1]})$

$\mathcal{L}(v_1, \to_{[2]}, \to_{\mathsf{br},[>2]})$

Definition

A loop–branch labeling is a LEE-witness, if:

L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \to_{[n]} \Rightarrow \mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br},[>n]}) \text{ is a loop subchart} \right)$.

L2. No infinite $\to_{\mathsf{br}}$ path from start vertex.

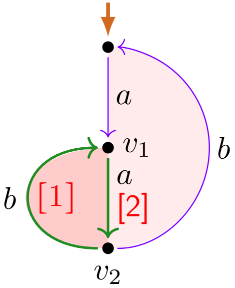
L3. Loop subcharts contained in other loop subcharts have different entry-step levels.

$\mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\to_{[n]}$ from v
followed by branch steps $\to_{\mathsf{br}}$
or entry steps $\to_{[m]}$ with $m > n$,
until v is reached again

LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$

$\mathcal{L}(v_1, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$

Definition

A loop–branch labeling is a LEE-witness, if:

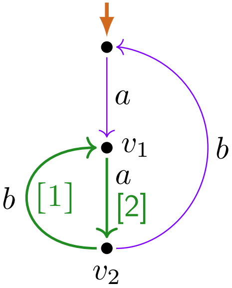
L1. $\forall n \in \mathbb{N} \forall v \in V \Big(v \rightarrow_{[n]} \Rightarrow \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$ is a loop subchart $\Big)$.

L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

L3. $\mathcal{L}(w_i, \rightarrow_{[n_i]}, \rightarrow_{\mathsf{br},[>n_i]})$ for $i \in \{1,2\}$ loop charts $\wedge\, w_1 \neq w_2 \,\wedge\, w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 \neq n_2$.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

LEE-witness



LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

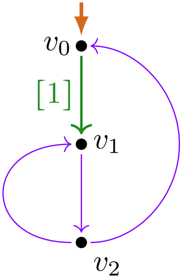
L1. $\forall n \in \mathbb{N} \forall v \in V \Big(v \rightarrow_{[n]} \;\Rightarrow\; \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$ is a loop subchart $\Big)$.

L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

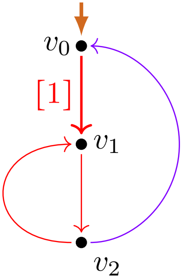
L3. $\mathcal{L}(w_i, \rightarrow_{[n_i]}, \rightarrow_{\mathsf{br},[>n_i]})$ for $i \in \{1,2\}$ loop charts $\wedge\; w_1 \neq w_2 \;\wedge\; w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 \neq n_2$.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

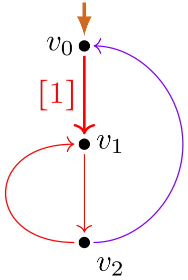
LEE-witness ?



LEE-witness ?



LEE-witness ?



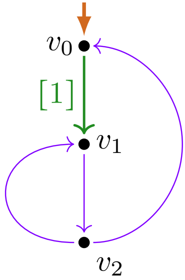
no!

(L1.) violated:

$\mathcal{L}(v_0, \to_{[1]}, \to_{\text{br},[>1]})$

not a loop chart

LEE-witness ?

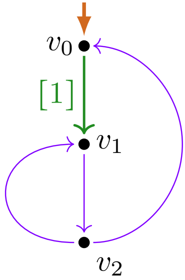


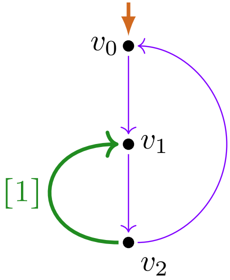
no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$
not a loop chart

LEE-witness ?





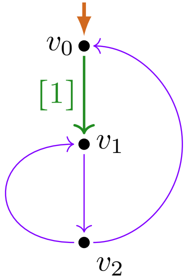
no!

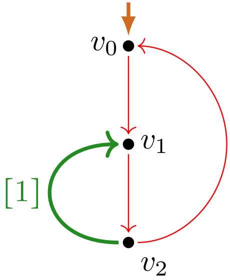
(L1.) violated:

$\mathcal{L}(v_0, \to_{[1]}, \to_{\mathsf{br},[>1]})$

not a loop chart

LEE-witness ?





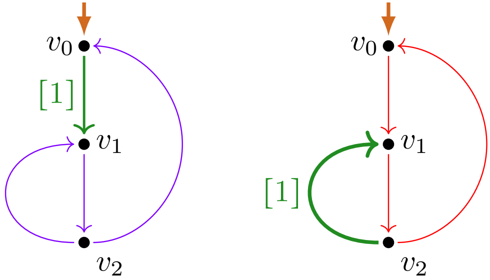
no!

(L1.) violated:

$\mathcal{L}(v_0, \to_{[1]}, \to_{\mathsf{br},[>1]})$

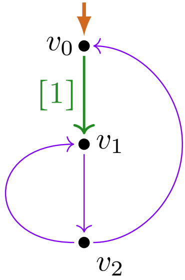
not a loop chart

LEE-witness ?



no!	no!
(L1.) violated:	(L2.) violated:
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$ not a loop chart	infinite $\rightarrow_{\mathsf{br}}$ path from start vertex

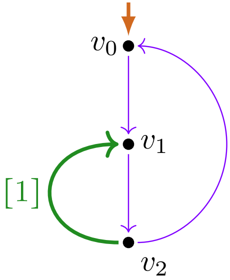
LEE-witness ?



no!

(L1.) violated:

$\mathcal{L}(v_0, \to_{[1]}, \to_{\mathsf{br},[>1]})$
not a loop chart

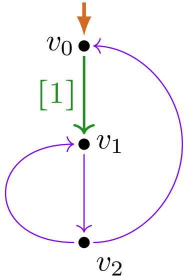


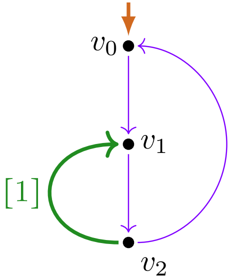
no!

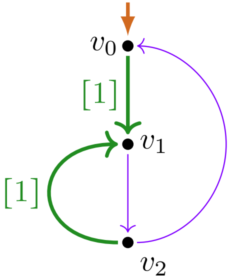
(L2.) violated:

infinite $\to_{\mathsf{br}}$ path
from start vertex

LEE-witness ?







no!

(L1.) violated:

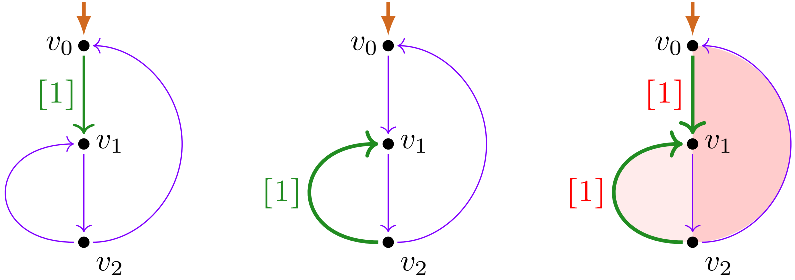
$\mathcal{L}(v_0, \to_{[1]}, \to_{\mathsf{br},[>1]})$
not a loop chart

no!

(L2.) violated:

infinite $\to_{\mathsf{br}}$ path
from start vertex

LEE-witness ?



no!

(L1.) violated:

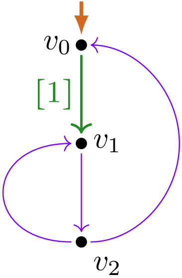
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$
not a loop chart

no!

(L2.) violated:

infinite $\rightarrow_{\mathsf{br}}$ path
from start vertex

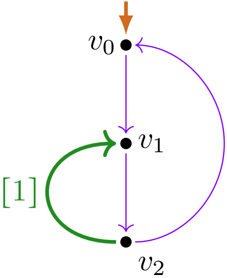
LEE-witness ?



no!

(L1.) violated:

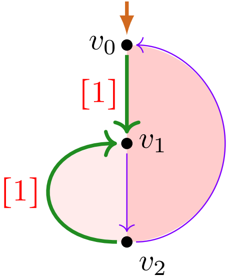
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br},[>1]})$
not a loop chart



no!

(L2.) violated:

infinite $\rightarrow_{\text{br}}$ path
from start vertex

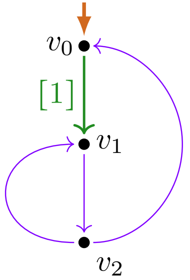


no!

(L3.) violated:

overlapping loop charts
have same level

LEE-witness ?

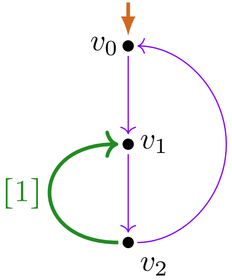


no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$

not a loop chart

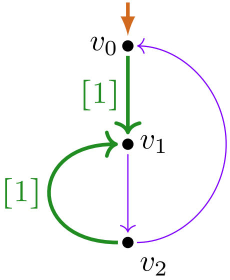


no!

(L2.) violated:

infinite $\rightarrow_{\mathsf{br}}$ path

from start vertex



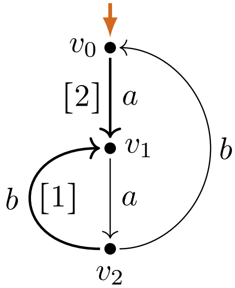
no!

(L3.) violated:

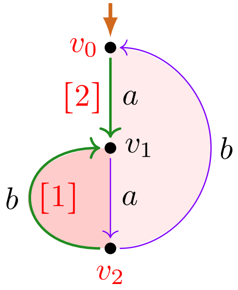
overlapping loop charts

have same level

LEE-witness ?



LEE-witness



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$

$\mathcal{L}(v_0, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$

LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a LEE-witness, if:

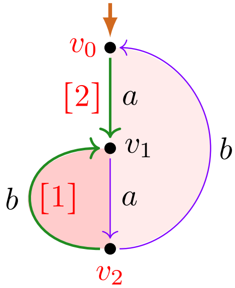
L1. $\forall n \in \mathbb{N} \forall v \in V \big(\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$ is a loop subchart, or trivial $\big)$.

L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

L3. $\mathcal{L}(w_i, \rightarrow_{[n_i]}, \rightarrow_{\mathsf{br},[>n_i]})$ for $i \in \{1,2\}$ loop charts $\wedge\; w_1 \neq w_2 \;\wedge\; w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 \neq n_2$.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

Layered LEE-witness



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$
$\mathcal{L}(v_0, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

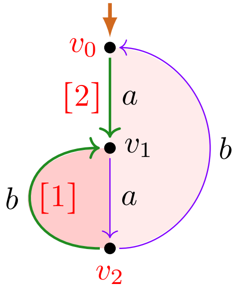
l-L1. $\forall n \in \mathbb{N} \forall v \in V \Big(v \rightarrow_{[n]} \;\Rightarrow\; \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]}) \text{ is a loop subchart} \Big)$.

l-L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

l-L3. $\mathcal{L}(w_i, \rightarrow_{[n_i]}, \rightarrow_{\mathsf{br},[>n_i]})$ for $i \in \{1,2\}$ loop charts $\wedge\; w_1 \neq w_2 \;\wedge\; w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 < n_2$.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br},[>n]})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

Layered LEE-witness



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br},[>1]})$

$\mathcal{L}(v_0, \rightarrow_{[2]}, \rightarrow_{\mathsf{br},[>2]})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

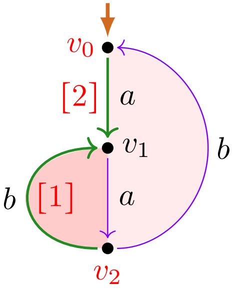
l-L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \rightarrow_{[n]} \Rightarrow \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br}}) \text{ is a loop subchart} \right)$.

l-L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

l-L3. $\mathcal{L}(w_i, \rightarrow_{[n_i]}, \rightarrow_{\mathsf{br}})$ for $i \in \{1,2\}$ loop charts $\wedge\, w_1 \neq w_2 \,\wedge\, w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 < n_2$.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br}})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$
or entry steps $\rightarrow_{[m]}$ with $m > n$,
until v is reached again

Layered LEE-witness



$\mathcal{L}(v_2, \to_{[1]}, \to_{\mathsf{br}})$

$\mathcal{L}(v_0, \to_{[2]}, \to_{\mathsf{br}})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

l-L1. $\forall n \in \mathbb{N} \forall v \in V \Big(v \to_{[n]} \Rightarrow \mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br}})$ is a loop subchart $\Big)$.

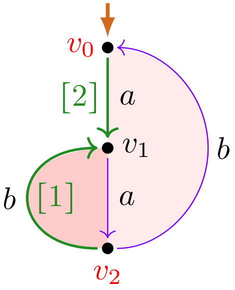
l-L2. No infinite $\to_{\mathsf{br}}$ path from start vertex.

l-L3. $\mathcal{L}(w_i, \to_{[n_i]}, \to_{\mathsf{br}})$ for $i \in \{1,2\}$ loop charts $\wedge\ w_1 \neq w_2\ \wedge\ w_1 \in \mathcal{L}(w_2, \ldots, \ldots) \implies n_1 < n_2$.

$\mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br}}) :=$ subchart induced
by entry steps $\to_{[n]}$ from v
followed by branch steps $\to_{\mathsf{br}}$

until v is reached again

Layered LEE-witness



$\mathcal{L}(v_2, \to_{[1]}, \to_{\mathsf{br}})$
$\mathcal{L}(v_0, \to_{[2]}, \to_{\mathsf{br}})$

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

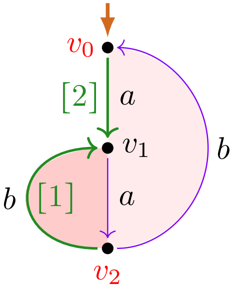
l-L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \to_{[n]} \Rightarrow \mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br}}) \text{ is a loop subchart} \right)$.

l-L2. No infinite $\to_{\mathsf{br}}$ path from start vertex.

l-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \to_{[n]}, \to_{\mathsf{br}})$:= subchart induced
by entry steps $\to_{[n]}$ from v
followed by branch steps $\to_{\mathsf{br}}$

Layered LEE-witness



$\mathcal{L}(v_2, \rightarrow_{[1]}, \rightarrow_{\mathsf{br}})$

$\mathcal{L}(v_0, \rightarrow_{[2]}, \rightarrow_{\mathsf{br}})$

layered LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\mathsf{br}\rangle}$, written $\xrightarrow{a}_{\mathsf{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

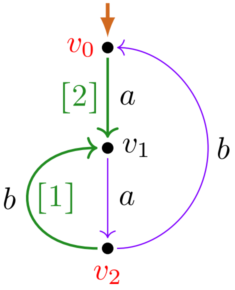
l-L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \rightarrow_{[n]} \Rightarrow \mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br}}) \text{ is a loop subchart} \right)$.

l-L2. No infinite $\rightarrow_{\mathsf{br}}$ path from start vertex.

l-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \rightarrow_{[n]}, \rightarrow_{\mathsf{br}})$:= subchart induced
by entry steps $\rightarrow_{[n]}$ from v
followed by branch steps $\rightarrow_{\mathsf{br}}$

Layered LEE-witness



$\mathcal{L}(v_2, \to_{[1]}, \to_{\text{br}})$
$\mathcal{L}(v_0, \to_{[2]}, \to_{\text{br}})$

layered LEE-witness

loop–branch labeling: marking transitions $\xrightarrow{a}$ as:

- entry steps $\xrightarrow{\langle a,[n]\rangle}$ for $n \in \mathbb{N}$, written $\xrightarrow{a}_{[n]}$,
- branch steps $\xrightarrow{\langle a,\text{br}\rangle}$, written $\xrightarrow{a}_{\text{br}}$ or $\xrightarrow{a}$.

Definition

A loop–branch labeling is a layered LEE-witness, if:

l-L1. $\forall n \in \mathbb{N} \forall v \in V \left(v \to_{[n]} \Rightarrow \mathcal{L}(v, \to_{[n]}, \to_{\text{br}}) \text{ is a loop subchart} \right)$.

l-L2. No infinite $\to_{\text{br}}$ path from start vertex.

l-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \to_{[n]}, \to_{\text{br}}) :=$ subchart induced
by entry steps $\to_{[n]}$ from v
followed by branch steps $\to_{\text{br}}$

LEE versus LEE-witness

Theorem

For every process graph G :

$$\text{LEE}(G) \iff G \text{ has a LEE-witness.}$$

LEE versus LEE-witness

Theorem

For every process graph G :

$$\mathsf{LEE}(G) \iff G \text{ has a } \mathsf{LEE}\text{-witness.}$$

Proof.

$\Rightarrow$: record loop elimination

LEE versus LEE-witness

Theorem

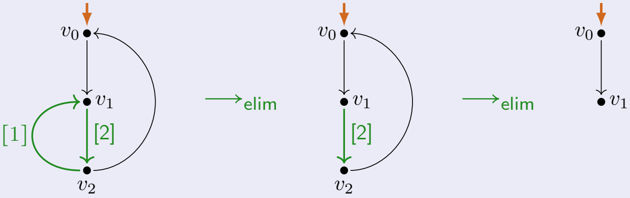
For every process graph G :

$$\mathsf{LEE}(G) \iff G \text{ has a } \mathsf{LEE}\text{-witness.}$$

Proof.

$\Rightarrow$: record loop elimination

$\Leftarrow$: carry out loop-elimination as indicated in the LEE-witness, in *inside–out* direction, e.g.:



LEE and (layered) LEE-witness

Lemma

Every layered LEE-witness is a LEE-witness.

Lemma

Every LEE-witness $\widehat{G}$ of a process graph G
can be transformed by an effective procedure (cut-elimination-like)
into a layered LEE-witness $\widehat{G}'$ of G.

LEE and (layered) LEE-witness

Lemma

Every layered LEE-witness is a LEE-witness.

Lemma

Every LEE-witness $\widehat{G}$ of a process graph G
can be transformed by an effective procedure (cut-elimination-like)
into a layered LEE-witness $\widehat{G}'$ of G.

Lemma

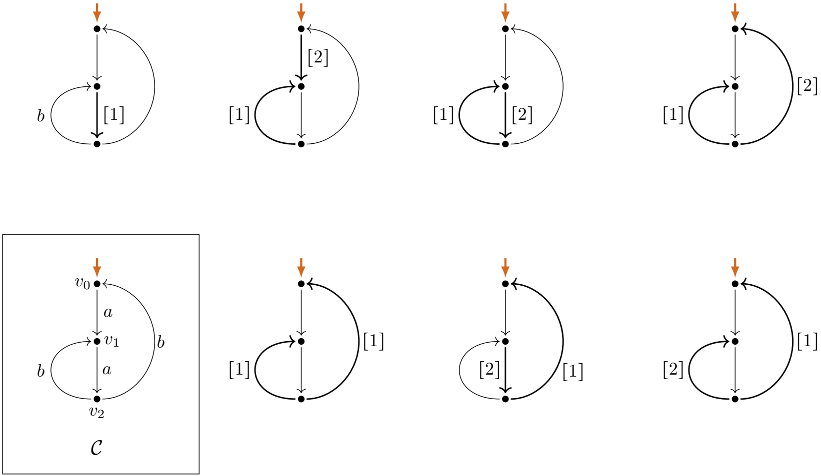
For every process graph G the following are equivalent:

(i) LEE(G).

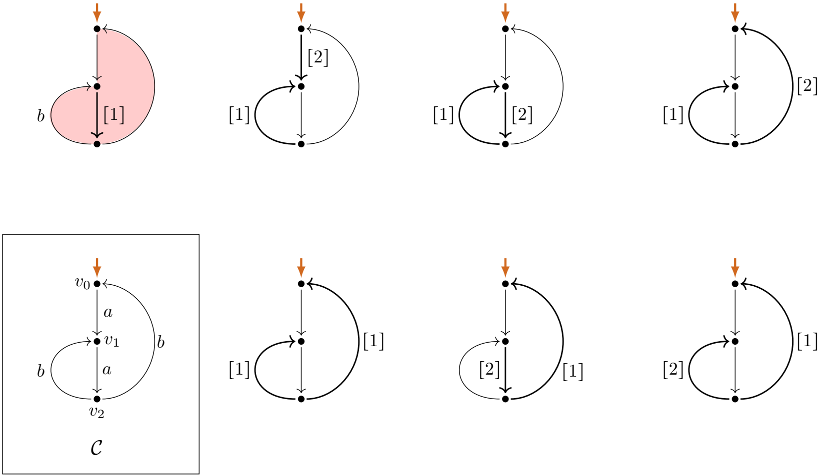
(ii) G *has a LEE-witness.*

(iii) G *has a layered LEE-witness.*

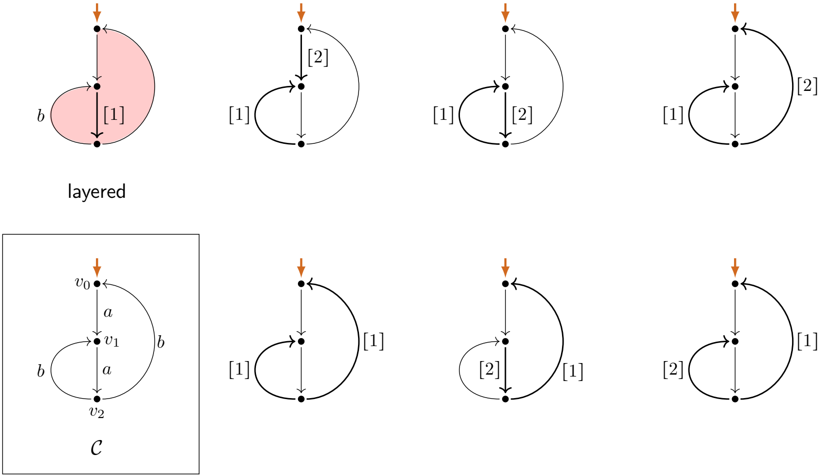
7 LEE-witnesses



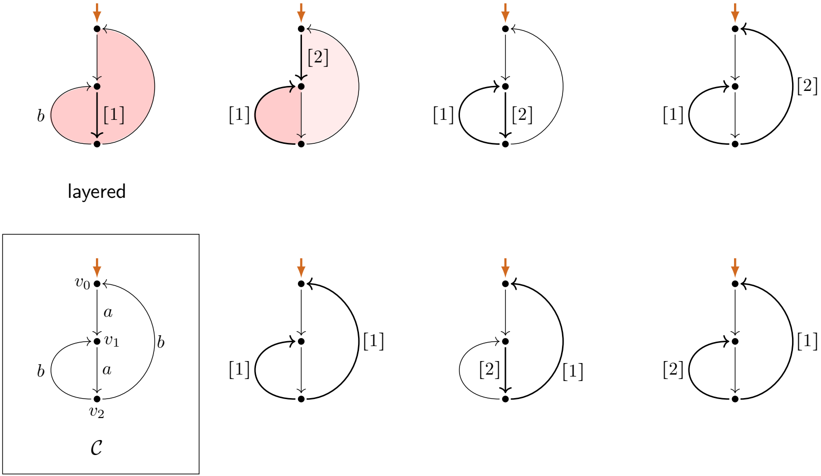
7 LEE-witnesses



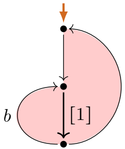
7 LEE-witnesses



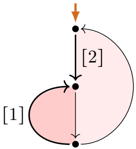
7 LEE-witnesses



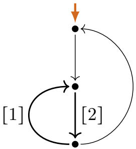
7 LEE-witnesses

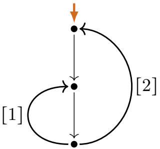


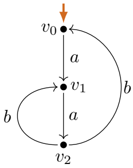
layered



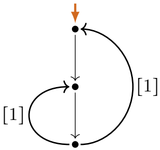
layered

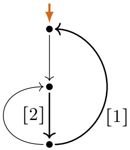


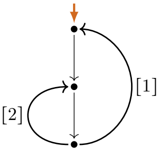




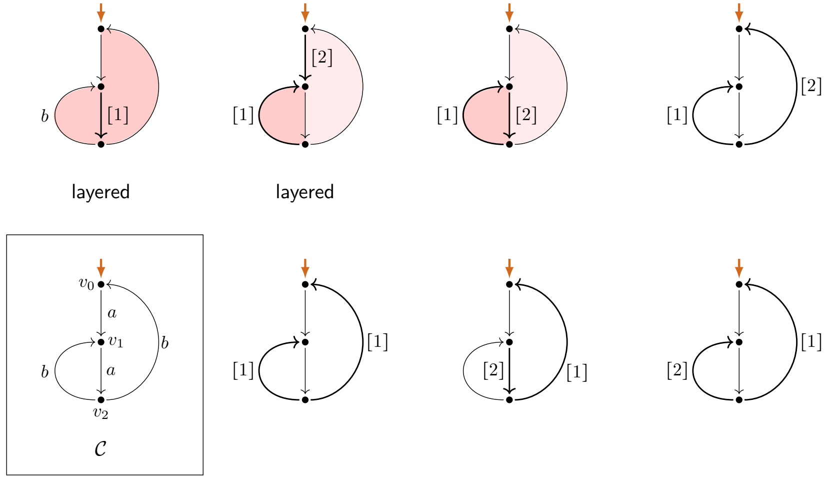
$\mathcal{C}$



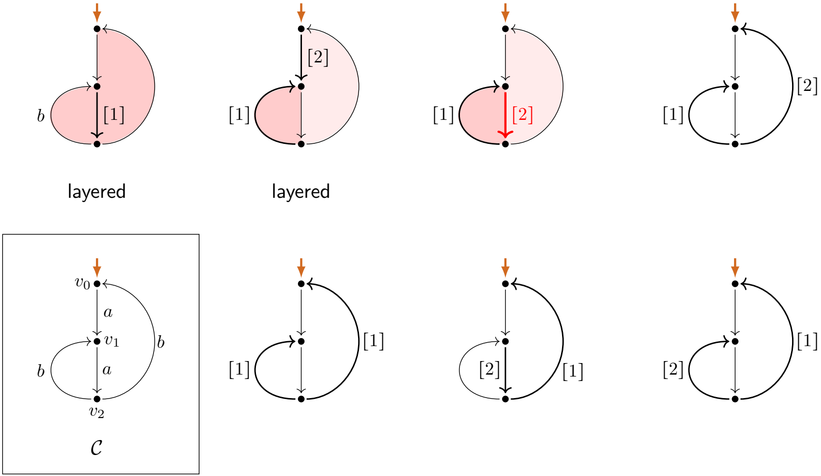




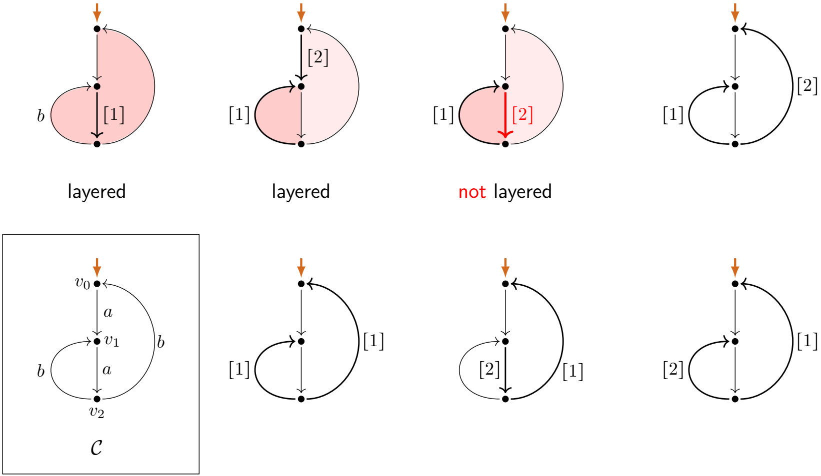
7 LEE-witnesses



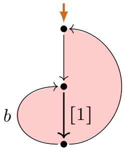
7 LEE-witnesses



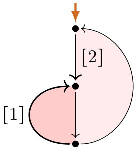
7 LEE-witnesses



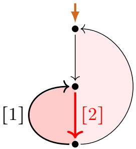
7 LEE-witnesses



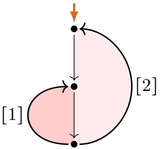
layered

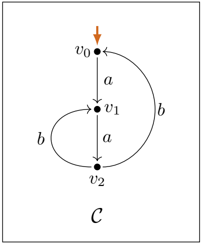


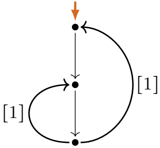
layered

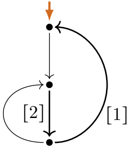


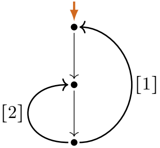
not layered



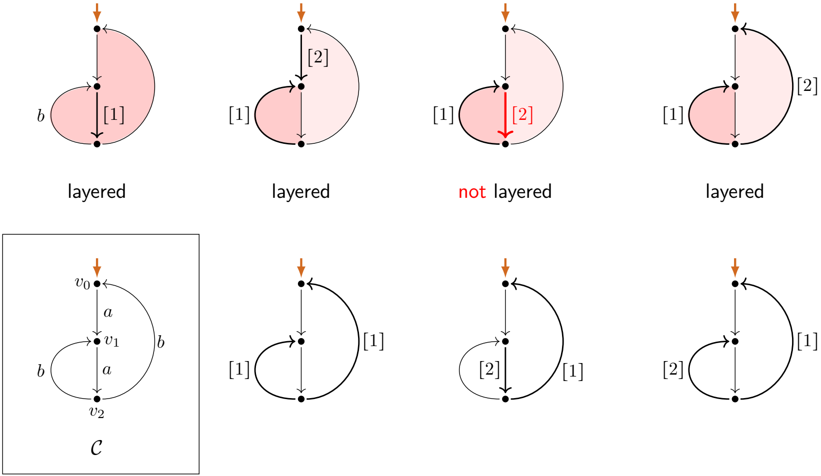




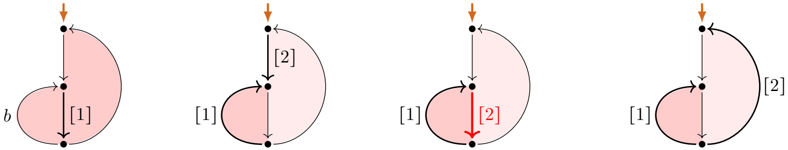




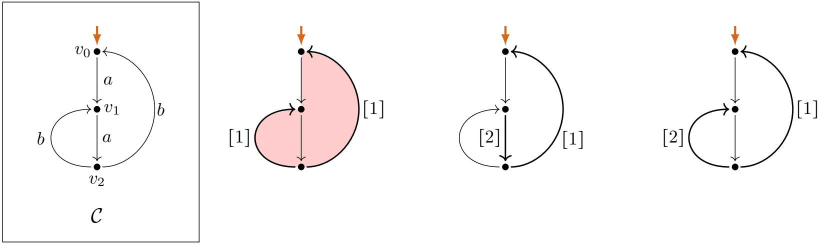
7 LEE-witnesses



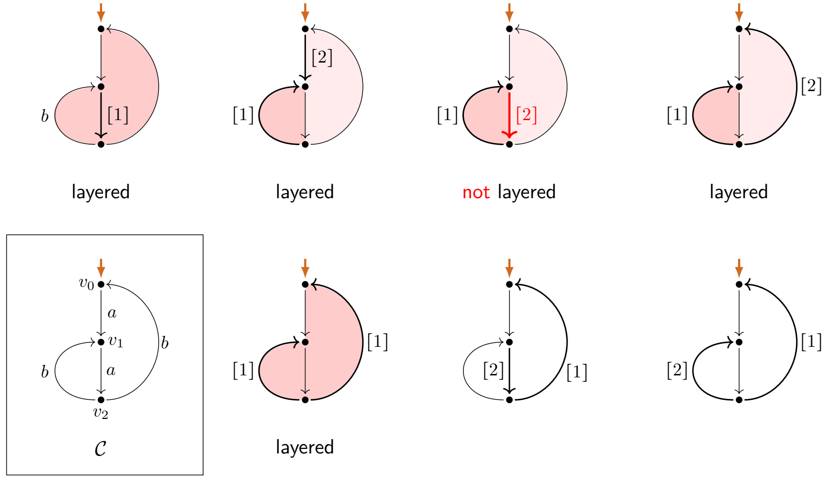
7 LEE-witnesses



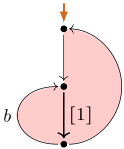
layered layered not layered layered



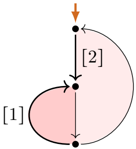
7 LEE-witnesses



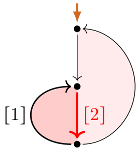
7 LEE-witnesses



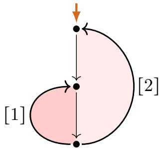
layered



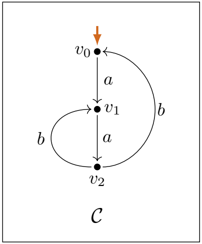
layered

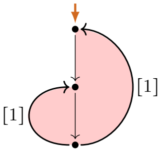


not layered

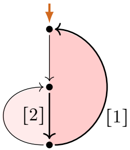


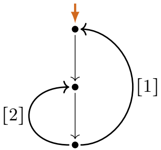
layered



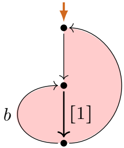


layered

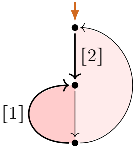




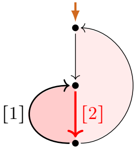
7 LEE-witnesses



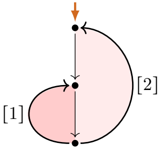
layered



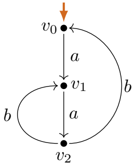
layered



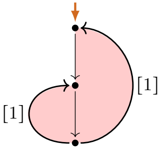
not layered



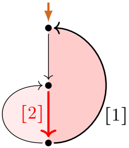
layered

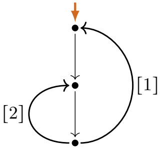


$\mathcal{C}$

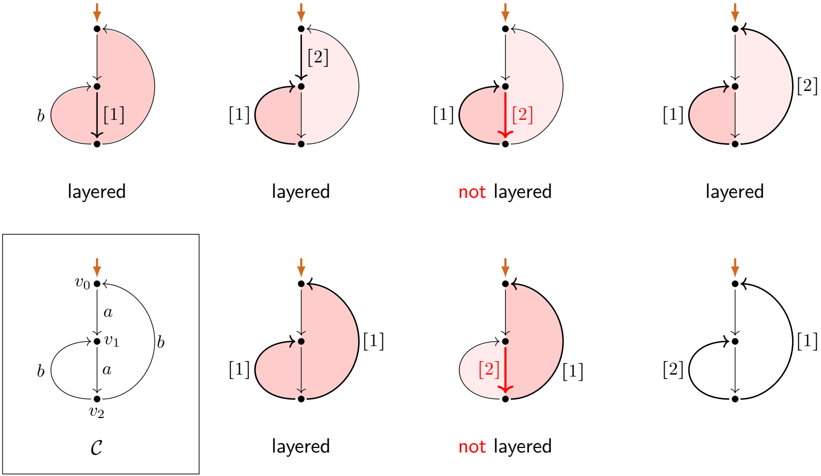


layered

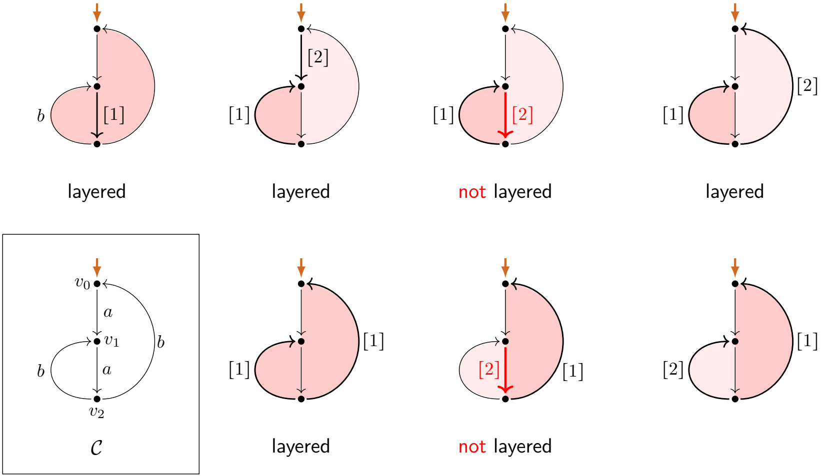




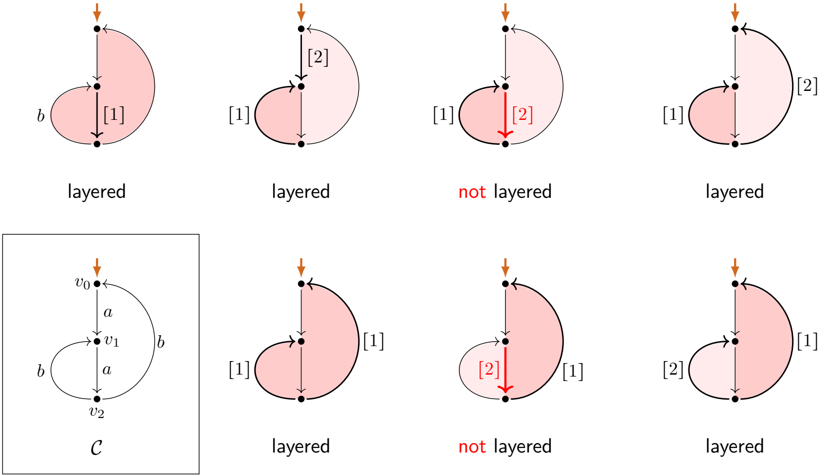
7 LEE-witnesses



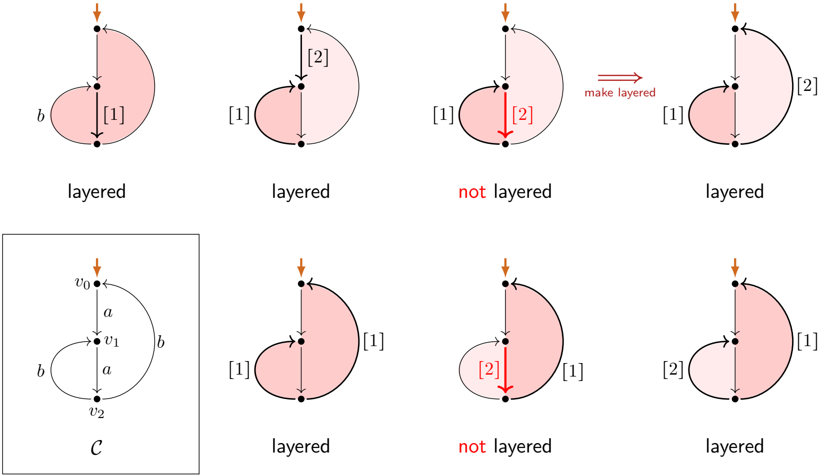
7 LEE-witnesses



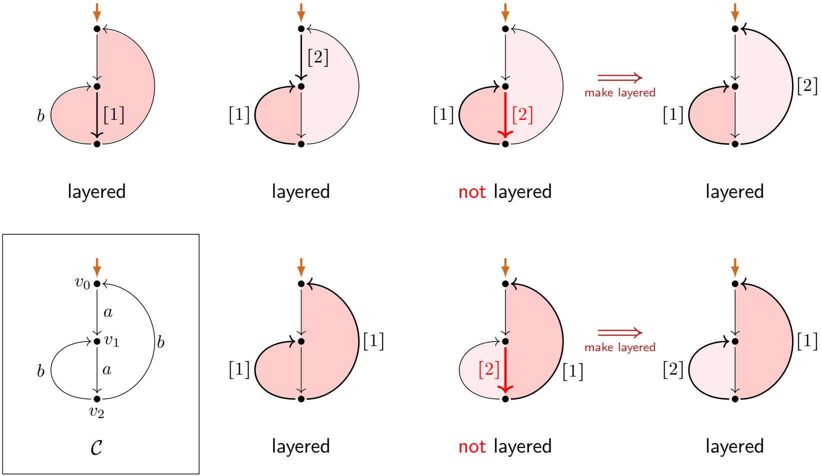
7 LEE-witnesses



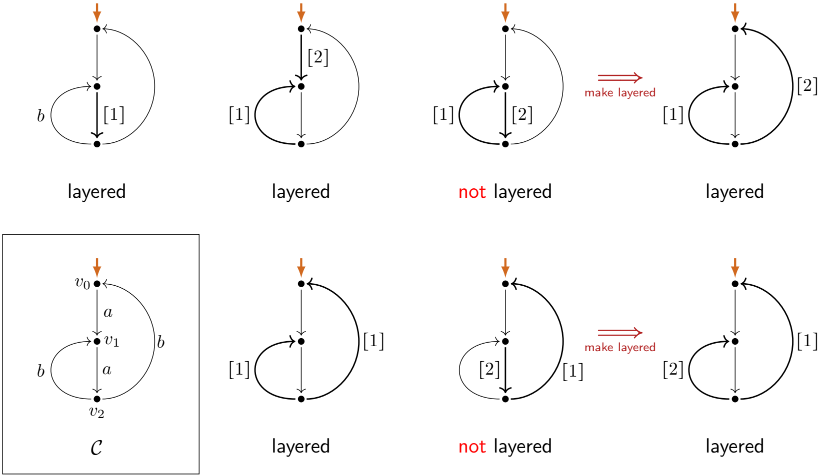
7 LEE-witnesses



7 LEE-witnesses



7 LEE-witnesses



LEE under bisimulation?

LEE under bisimulation

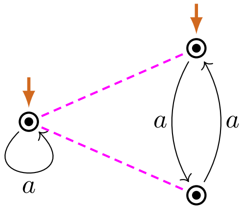
Observation

- LEE is not invariant under bisimulation.

LEE under bisimulation

Observation

- LEE is not invariant under bisimulation.

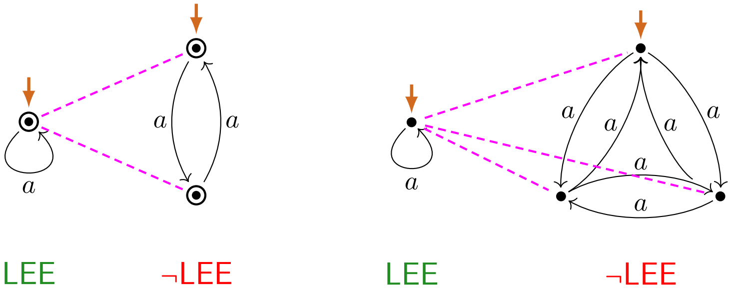


LEE ¬LEE

LEE under bisimulation

Observation

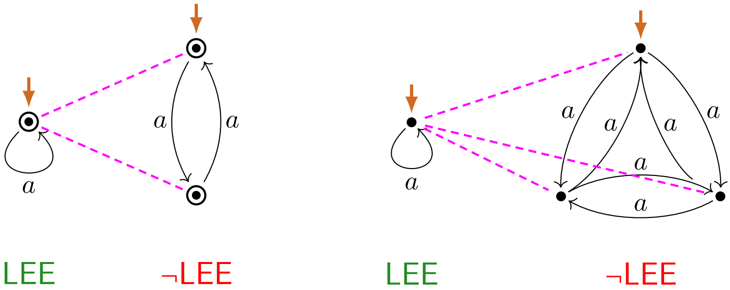
- LEE is not invariant under bisimulation.



LEE under bisimulation

Observation

- LEE is not invariant under bisimulation.
- LEE is not preserved by converse functional bisimulation.



LEE under functional bisimulation

Lemma

(i) LEE *is preserved by functional bisimulations:*

$$\mathsf{LEE}(G_1) \wedge G_1 \mathrel{\underline{\rightarrow}} G_2 \implies \mathsf{LEE}(G_2) \, .$$

LEE under functional bisimulation

Lemma

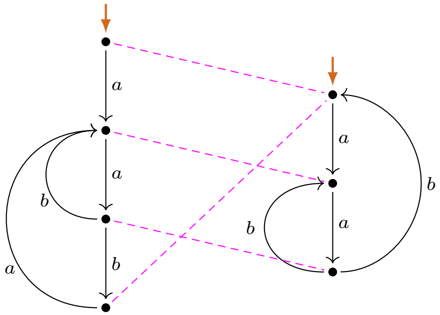
(i) LEE *is preserved by functional bisimulations:*

$$\mathsf{LEE}(G_1) \wedge G_1 \mathrel{\underline{\rightarrow}} G_2 \implies \mathsf{LEE}(G_2) \ .$$

Proof (Idea).

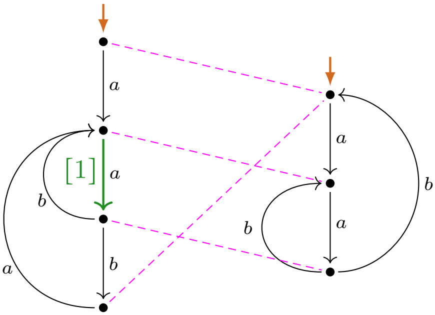
Use loop elimination in G_1 to carry out loop elimination in G_2.

Collapsing LEE-witnesses



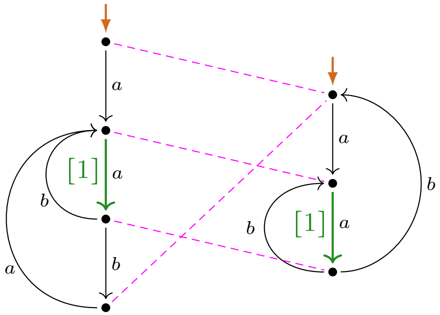
$P(a(a(b+ba))^*0)$

Collapsing LEE-witnesses



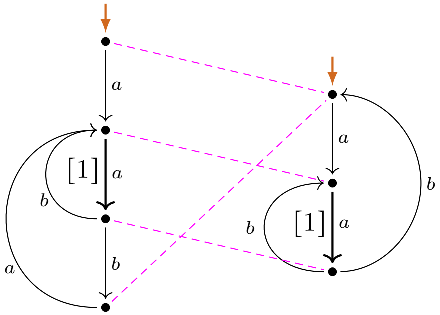
$P(a(a(b+ba))^*0)$

Collapsing LEE-witnesses



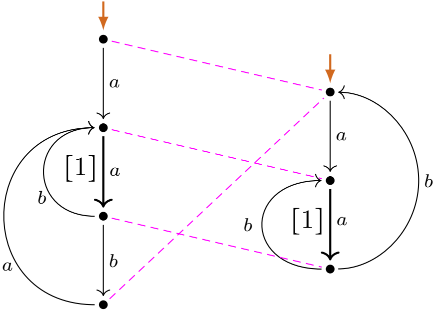
$P(a(a(b+ba))^{*}0)$

Collapsing LEE-witnesses

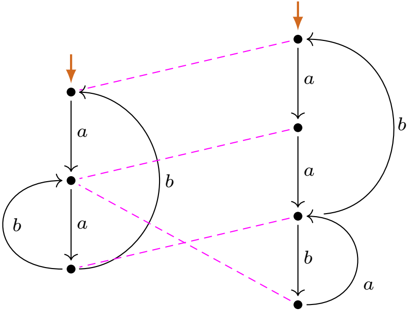


$P(a(a(b+ba))^*0)$

Collapsing LEE-witnesses

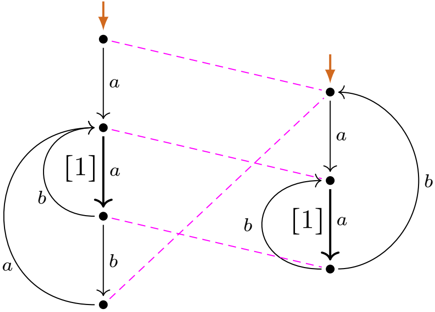


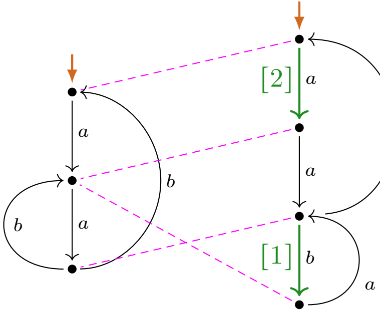
$P(a(a(b+ba))^*0)$



$P((aa(ba)^*b)^*0)$

Collapsing LEE-witnesses

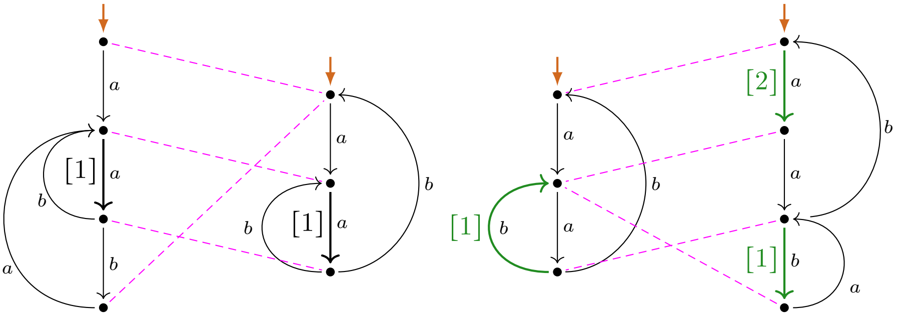




$P(a(a(b+ba))^*0)$

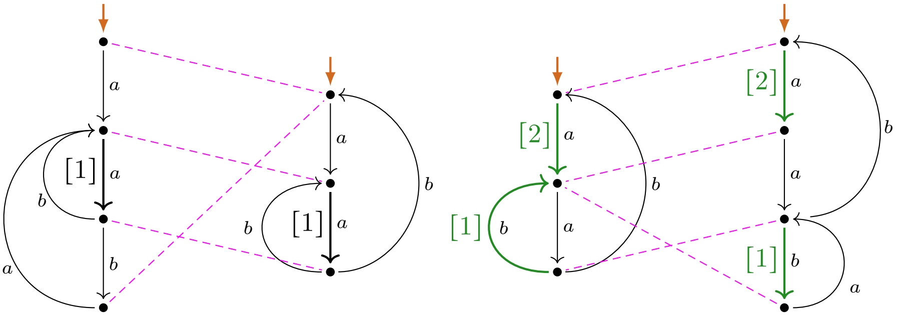
$P((aa(ba)^*b)^*0)$

Collapsing LEE-witnesses



$P(a(a(b+ba))^*0)$ $P((aa(ba)^*b)^*0)$

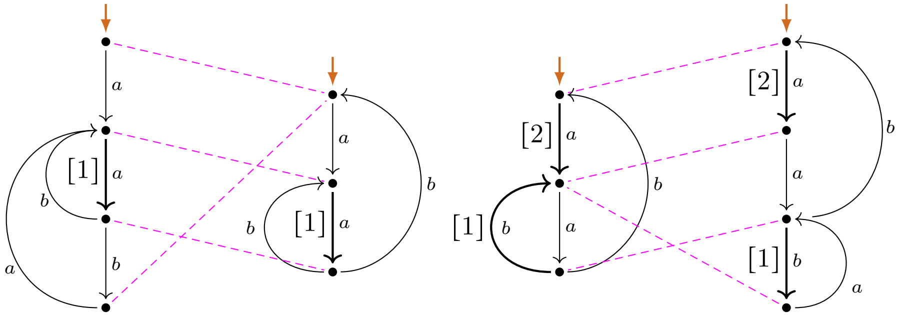
Collapsing LEE-witnesses



$P(a(a(b+ba))^*0)$

$P((aa(ba)^*b)^*0)$

Collapsing LEE-witnesses

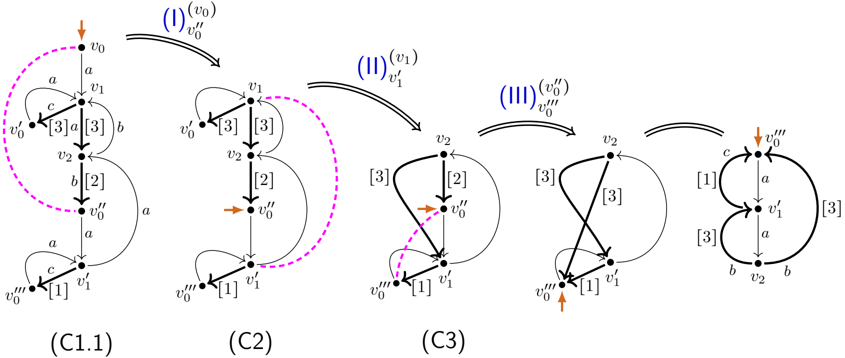


$P(a(a(b+ba))^*0)$

$P((aa(ba)^*b)^*0)$

LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20)

(no 1-transitions!)



Lemma

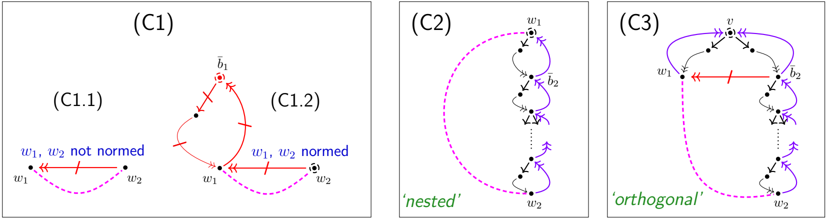
The bisimulation collapse of a LLEE-chart is again a LLEE-chart.

Reduced bisimilarity redundancies in LLEE-charts (no 1-trans.!)

(G/Fokkink, LICS'20)

w_1, w_2 in different scc's

w_1, w_2 in the same scc

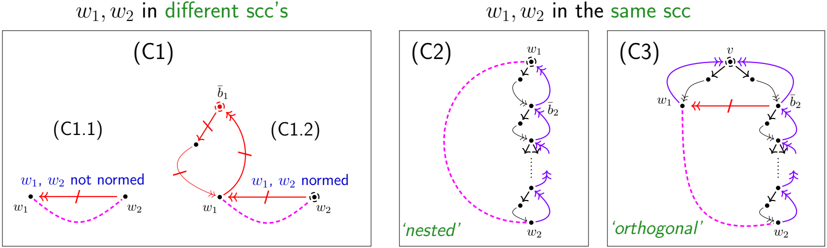


Lemma

Every not collapsed LLEE-chart contains bisimilar vertices $w_1 \neq w_2$ of kind (C1), (C2), *or* (C3) *(a reduced bisimilarity redundancy $\langle w_1, w_2 \rangle$):*

Reduced bisimilarity redundancies in LLEE-charts (no 1-trans.!)

(G/Fokkink, LICS'20)



Lemma

Every not collapsed LLEE-chart contains bisimilar vertices $w_1 \neq w_2$ of kind (C1), (C2), *or* (C3) *(a reduced bisimilarity redundancy $\langle w_1, w_2 \rangle$):*

Lemma

Every reduced bisimilarity redundancy in a LLEE-chart can be eliminated LLEE-preservingly.

LEE under functional bisimulation

Lemma

(i) LEE *is preserved by functional bisimulations:*

$$\mathsf{LEE}(G_1) \wedge G_1 \mathrel{\underline{\to}} G_2 \implies \mathsf{LEE}(G_2) .$$

Idea of Proof for (i)

Use loop elimination in G_1 to carry out loop elimination in G_2.

LEE under functional bisimulation / bisimulation collapse

Lemma

(i) LEE *is preserved by functional bisimulations:*

$$\mathsf{LEE}(G_1) \wedge G_1 \mathrel{\underline{\rightarrow}} G_2 \implies \mathsf{LEE}(G_2)\ .$$

(ii) LEE *is preserved from a process graph to its bisimulation collapse:*

$$\mathsf{LEE}(G) \wedge C \textit{ is bisimulation collapse of } G \implies \mathsf{LEE}(C)\ .$$

Idea of Proof for (i)

Use loop elimination in G_1 to carry out loop elimination in G_2.

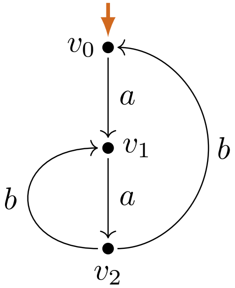
Readback

Lemma

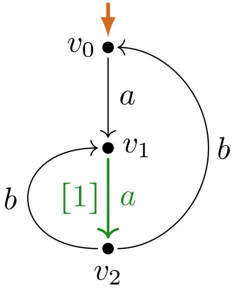
Process graphs with LEE are $P(\cdot)$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathsf{Reg}(A) \left(G \mathrel{\underline{\leftrightarrow}} P(e) \right) .$$

Readback from layered LEE-witness (example)

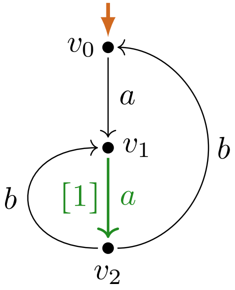


Readback from layered LEE-witness (example)



layered
LEE-witness

Readback from layered LEE-witness (example)

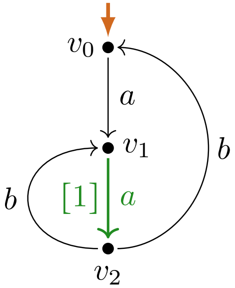


layered LEE-witness

$$
\begin{aligned}
s(v_0) &= 0^* \cdot a \cdot s(v_1) \\
&=_{\mathsf{Mil}^-} a \cdot s(v_1) \\
&=_{\mathsf{Mil}^-} a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_1) &= \big(a \cdot s(v_2, v_1)\big)^* \cdot 0 \\
&=_{\mathsf{Mil}^-} \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_2, v_1) &= 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\
&=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \\
&=_{\mathsf{Mil}^-} b + b \cdot a \\
s(v_1, v_1) &= 1 \\
s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\
&= 0^* \cdot a \cdot 1 \\
&=_{\mathsf{Mil}^-} a
\end{aligned}
$$

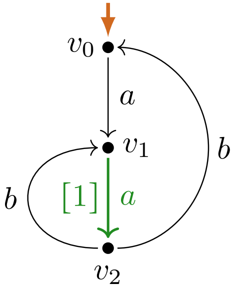
Readback from layered LEE-witness (example)

$$s(v_0) = \quad 0^* \cdot a \cdot s(v_1)$$



layered
LEE-witness

Readback from layered LEE-witness (example)

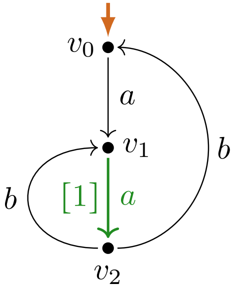


layered
LEE-witness

$$s(v_0) = \quad 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \quad \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

Readback from layered LEE-witness (example)



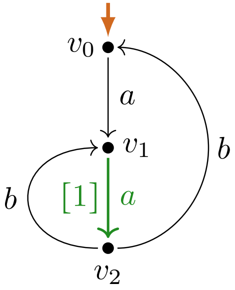
layered
LEE-witness

$$s(v_0) = \quad 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \quad \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$s(v_2, v_1) = \quad 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big)$$

Readback from layered LEE-witness (example)



layered
LEE-witness

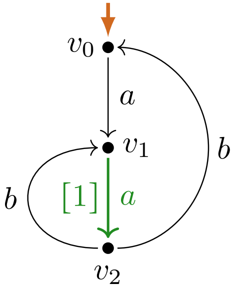
$$s(v_0) = \quad 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \quad \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$s(v_2, v_1) = \quad 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big)$$

$$s(v_1, v_1) = \quad 1$$

Readback from layered LEE-witness (example)



layered
LEE-witness

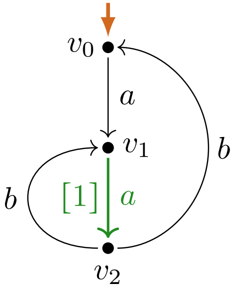
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big)$$

$$s(v_1, v_1) = 1$$
$$s(v_0, v_1) = 0^* \cdot a \cdot s(v_1, v_1)$$

Readback from layered LEE-witness (example)



layered
LEE-witness

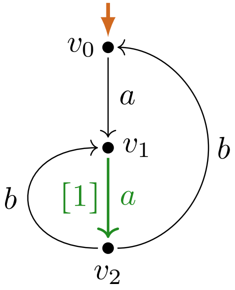
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big)$$

$$\begin{aligned} s(v_1, v_1) &= 1 \\ s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \end{aligned}$$

Readback from layered LEE-witness (example)



layered
LEE-witness

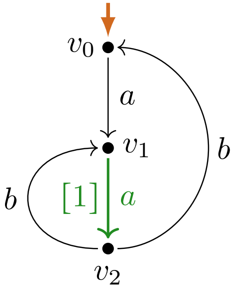
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big)$$

$$\begin{aligned} s(v_1, v_1) &= 1 \\ s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &=_{\mathsf{Mil}^-} a \end{aligned}$$

Readback from layered LEE-witness (example)



layered
LEE-witness

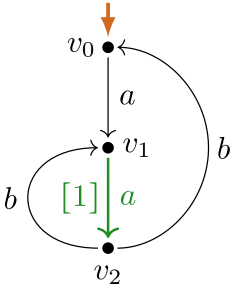
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = \big(a \cdot s(v_2, v_1)\big)^* \cdot 0$$

$$\begin{aligned} s(v_2, v_1) &= 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\ &=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \end{aligned}$$

$$\begin{aligned} s(v_1, v_1) &= 1 \\ s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &=_{\mathsf{Mil}^-} a \end{aligned}$$

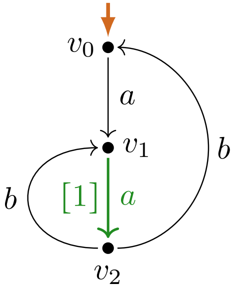
Readback from layered LEE-witness (example)



layered
LEE-witness

$$
\begin{aligned}
s(v_0) &= \quad 0^* \cdot a \cdot s(v_1) \\
s(v_1) &= \quad \big(a \cdot s(v_2, v_1)\big)^* \cdot 0 \\
s(v_2, v_1) &= \quad 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\
&=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \\
&=_{\mathsf{Mil}^-} b + b \cdot a \\
s(v_1, v_1) &= \quad 1 \\
s(v_0, v_1) &= \quad 0^* \cdot a \cdot s(v_1, v_1) \\
&= \quad 0^* \cdot a \cdot 1 \\
&=_{\mathsf{Mil}^-} a
\end{aligned}
$$

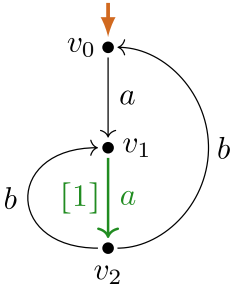
Readback from layered LEE-witness (example)



layered LEE-witness

$$
\begin{aligned}
s(v_0) &= \quad 0^* \cdot a \cdot s(v_1) \\[2em]
s(v_1) &= \quad \big(a \cdot s(v_2, v_1)\big)^* \cdot 0 \\
&=_{\mathsf{Mil}^-} \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_2, v_1) &= \quad 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\
&=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \\
&=_{\mathsf{Mil}^-} b + b \cdot a \\
s(v_1, v_1) &= \quad 1 \\
s(v_0, v_1) &= \quad 0^* \cdot a \cdot s(v_1, v_1) \\
&= \quad 0^* \cdot a \cdot 1 \\
&=_{\mathsf{Mil}^-} a
\end{aligned}
$$

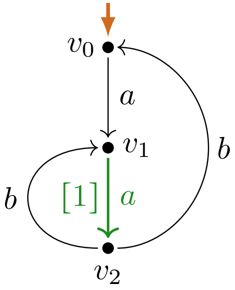
Readback from layered LEE-witness (example)



layered
LEE-witness

$$
\begin{aligned}
s(v_0) &= 0^* \cdot a \cdot s(v_1) \\
&=_{\mathsf{Mil}^-} a \cdot s(v_1) \\[1em]
s(v_1) &= \big(a \cdot s(v_2, v_1)\big)^* \cdot 0 \\
&=_{\mathsf{Mil}^-} \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_2, v_1) &= 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\
&=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \\
&=_{\mathsf{Mil}^-} b + b \cdot a \\
s(v_1, v_1) &= 1 \\
s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\
&= 0^* \cdot a \cdot 1 \\
&=_{\mathsf{Mil}^-} a
\end{aligned}
$$

Readback from layered LEE-witness (example)



layered
LEE-witness

$$
\begin{aligned}
s(v_0) &= 0^* \cdot a \cdot s(v_1) \\
&=_{\mathsf{Mil}^-} a \cdot s(v_1) \\
&=_{\mathsf{Mil}^-} a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_1) &= \big(a \cdot s(v_2, v_1)\big)^* \cdot 0 \\
&=_{\mathsf{Mil}^-} \big(a \cdot (b + b \cdot a)\big)^* \cdot 0 \\
s(v_2, v_1) &= 0^* \cdot \big(b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)\big) \\
&=_{\mathsf{Mil}^-} 0^* \cdot \big(b \cdot 1 + b \cdot a\big) \\
&=_{\mathsf{Mil}^-} b + b \cdot a \\
s(v_1, v_1) &= 1 \\
s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\
&= 0^* \cdot a \cdot 1 \\
&=_{\mathsf{Mil}^-} a
\end{aligned}
$$

1-return-less regular expressions

Lemma

Process graphs with LEE are $P(\cdot)$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathrm{Reg}(A) \left(G \underline{\leftrightarrow} P(e) \right) .$$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_{P}^{\cancel{\mathbf{1r}}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists\, e \in \mathrm{Reg}^{\cancel{\mathbf{1r}}\backslash\star}(A)\,\big(\, G \underline{\leftrightarrow}\, P(e)\,\big)\;.$$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\mathrm{LEE}(G) \implies \exists e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)\left(G \leftrightarrow P(e) \right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\mathbf{1r}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathrm{Reg}^{\mathbf{1r}\backslash\star}(A) \left(G \leftrightarrows P(e) \right) .$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\mathbf{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_{P}^{\cancel{\mathbf{1r}}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathrm{Reg}^{\cancel{\mathbf{1r}}\backslash\star}(A)\left(G \leftrightarrow P(e)\right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{\mathbf{1r}}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)\left(G \leftrightarrows P(e)\right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for <u>no</u> iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, <u>and</u>
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)\left(G \leftrightarrow P(e)\right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-⋆) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)\left(G \leftrightarrows P(e) \right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_{P}^{\cancel{1r}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists\, e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)\,\big(\, G \leftrightarrows P(e)\,\big)\;.$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\text{LEE}(G) \implies \exists\, e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)\, \big(\, G \underline{\leftrightarrow} P(e) \,\big) \,.$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$ ✓

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\mathrm{LEE}(G) \implies \exists\, e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)\left(G \leftrightarrow P(e) \right).$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \left(a \cdot (b + b \cdot a)\right)^* \cdot 0$ ✓
- $(a^* (b^* + c \cdot 0)^*)^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists\, e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)\,\big(\, G \underline{\leftrightarrow}\, P(e)\,\big)\;.$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$ ✓
- $(a^*(b^* + c \cdot 0)^*)^*$ ✗

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{\cancel{1r}\backslash\star}(A) \left(G \leftrightarrow P(e) \right) .$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \left(a \cdot (b + b \cdot a)\right)^* \cdot 0$ ✓
- $(a^*(b^* + c \cdot 0)^*)^*$ ✗
- $(a^*(b^* + c \cdot 0))^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A) \left(G \leftrightarrow P(e) \right) .$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$ ✓
- $(a^*(b^* + c \cdot 0)^*)^*$ ✗
- $(a^*(b^* + c \cdot 0))^*$ ✗

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)\,\big(G \leftrightarrow P(e)\big)\;.$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \text{Reg}^{\cancel{1r}\backslash\star}(A)$) if:

- for no iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, and
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$ ✓
- $(a^*(b^* + c \cdot 0)^*)^*$ ✗
- $(a^*(b^* + c \cdot 0))^*$ ✗
- $(a^*(b + c \cdot 0))^*$

1-return-less regular expressions

Lemma

Process graphs with LEE are $[\![\cdot]\!]_{P}^{\cancel{\mathbf{1r}}\backslash\star}$-expressible:

$$\mathsf{LEE}(G) \implies \exists\, e \in \mathrm{Reg}^{\cancel{\mathbf{1r}}\backslash\star}(A) \left(G \,\underline{\leftrightarrow}\, P(e) \right) .$$

Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression e is 1-return-less(-under-$\star$) ($e \in \mathrm{Reg}^{\cancel{\mathbf{1r}}\backslash\star}(A)$) if:

- for <u>no</u> iteration subexpression f^* of e does $P(f)$ proceed to a process p such that:
  - p has the option to immediately terminate, <u>and</u>
  - p has the option to do a proper step, and terminate later.

Non-/Examples of 1-return-less regular expressions

- $(a \cdot (1 + b))^*$ ✗
- $(a \cdot (0^* + b))^*$ ✗
- $a \cdot \big(a \cdot (b + b \cdot a)\big)^* \cdot 0$ ✓
- $(a^* (b^* + c \cdot 0)^*)^*$ ✗
- $(a^* (b^* + c \cdot 0))^*$ ✗
- $(a^* (b + c \cdot 0))^*$ ✓

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Milners characterization question:

Q1. Which structural property of finite process graphs characterizes $P(\cdot)$-expressibility?

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Milners characterization question restricted:

$\mathrm{Q1}'$. Which structural property of finite process graphs characterizes $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$-expressibility?

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Milners characterization question restricted, and adapted:

Q1$''$. Which structural property of collapsed finite process graphs characterizes $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$-expressibility?

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Answering Milners characterization question restricted, and adapted:

Q1″. Which structural property of collapsed finite process graphs characterizes $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$-expressibility?

- The loop-existence and elimination property LEE.

Characterization of expressibility$^{\cancel{\mathbf{1r}}\backslash\star}$

Theorem

For every process graph G with bisimulation collapse C the following are equivalent:

(i) G *is* $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$*-expressible.*

(ii) LEE(C).

(iii) C *has a* LEE*-witness.*

(iv) C *has a layered* LEE*-witness.*

Answering Milners characterization question restricted, and adapted:

Q1′′. Which structural property of collapsed finite process graphs characterizes $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$-expressibility?

- The loop-existence and elimination property LEE.

Also yields: efficient decision method of $[\![\cdot]\!]_P^{\cancel{\mathbf{1r}}\backslash\star}$-expressibility?

Structure constrained finite process graphs

graphs with LEE / a (layered) LEE-witness

Benefits of the class of process graphs with LEE:

- is closed under $\underline{\to}$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

$=$ graphs that are $[\![\cdot]\!]_{P}^{\cancel{1r}\backslash\star}$-expressible

Benefits of the class of process graphs with LEE:

- is closed under $\underline{\to}$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

by 1-return-less expression $P(\cdot)$-expressible graphs

$\subsetneqq$ graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

$=$ graphs that are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible

Benefits of the class of process graphs with LEE:

- is closed under $\Rightarrow$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

by 1-return-less expression $P(\cdot)$-expressible graphs

$\subsetneqq$ graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

= graphs that are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible

$\subsetneqq$ graphs that are $P(\cdot)$-expressible

Benefits of the class of process graphs with LEE:

- is closed under $\rightarrow$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

by 1-return-less expression $P(\cdot)$-expressible graphs

$\subsetneqq$ graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

$=$ graphs that are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible

$\subsetneqq$ graphs that are $P(\cdot)$-expressible

$\subsetneqq$ finite process graphs

Benefits of the class of process graphs with LEE:

- is closed under $\Rightarrow$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

loop–exit palm trees $\subsetneqq$ by 1-return-less expression $P(\cdot)$-expressible graphs

$\subsetneqq$ graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

$=$ graphs that are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible

$\subsetneqq$ graphs that are $P(\cdot)$-expressible

$\subsetneqq$ finite process graphs

Benefits of the class of process graphs with LEE:

- is closed under $\Rightarrow$
- forth-/back-correspondence with 1-return-less regular expressions

Structure constrained finite process graphs

loop–exit palm trees $\subsetneqq$ by 1-return-less expression $P(\cdot)$-expressible graphs

$\subsetneqq$ graphs with LEE / a (layered) LEE-witness

$\subsetneqq$ graphs whose collapse satisfies LEE

$=$ graphs that are $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible

$\subsetneqq$ graphs that are $P(\cdot)$-expressible

$\subsetneqq$ finite process graphs

Benefits of the class of process graphs with LEE:

- is closed under $\underline{\rightarrow}$
- forth-/back-correspondence with 1-return-less regular expressions

Application to Milner's questions yields partial results:

Q1: characterization/efficient decision of $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressibility

Q2: alternative compl. proof of Mil on 1-return-less expressions (C/DN/L)

Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete

Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions

Clemens Grabmayer (Department of Computer Science, Gran Sasso Science Institute, Viale F. Crispi, 7, 67100 L'Aquila AQ, Italy)

Abstract

We report on a lengthy completeness proof for Robin Milner's proof system Mil (1984) for bisimilarity of regular expressions in the process semantics. Central for our proof are the recognitions:

1. Process graphs with 1-transitions (1-charts) and the loop existence/elimination property LLEE are **not** closed under bisimilation collapse,
2. Such process graphs can be '**crystallized**' to 'near-collapsed' 1-charts with some strongly connected components of 'twin-crystal' form.

The Process Semantics of Regular Expressions

Milner (1984) introduced a process semantics for regular expressions: the interpretation of 0 is *deadlock*, of 1 is an *empty step to termination*, letters a are *atomic actions*, the operators $+$ and $\cdot$ stand for *choice* and *concatenation* of processes, and unary Kleene star $(\cdot)^*$ represents *(unbounded) iteration*. Formally, Milner defined chart (finite process graph) interpretations $\mathcal{C}(e)$ of regular expressions e.

Milner's Proof System

As axiomatization of the relation $e_1 =_{\boldsymbol{P}} e_2$ on regular expressions e_1 and e_2 defined by $\mathcal{C}(e_1) \leftrightarrow \mathcal{C}(e_2)$ (as bisimilarity $\leftrightarrow$ of chart interpretations), Milner asked whether the following system Mil is complete:

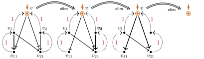
(A1)	$e+(f+g)=(e+f)+g$	(A7)	$e=1\cdot e$
(A2)	$e+0=e$	(A8)	$e=e\cdot 1$
(A3)	$e+f=f+e$	(A9)	$0=0\cdot e$
(A4)	$e+e=e$	(A10)	$e^*=1+e\cdot e^*$
(A5)	$e\cdot(f\cdot g)=(e\cdot f)\cdot g$	(A11)	$e^*=(1+e)^*$
(A6)	$(e+f)\cdot g=e\cdot g+f\cdot g$		

$$\frac{e = f\cdot e + g}{e = f^*\cdot g}\,\text{RSP}^* \quad (\text{if } f \text{ does not terminate immediately})$$

This system is a variation of Salomaa's complete axiom system (1966) for language equality of regular expressions, missing left-distributivity $e\cdot(f+g) = e\cdot f + e\cdot g$ and $e\cdot 0 = 0$, which are unsound here.

Loop Existence and Elimination

The process semantics is incomplete: not every finite process graph is *expressible by* (=bisimilar to the interpretation of) a regular expression. A sufficient condition for expressibility is the *(layered) loop existence and elimination property* LLEE. It is defined via elimination of 'loops' (loop subcharts):



LLEE holds if a graph without infinite behavior can be obtained. Important features of LLEE:

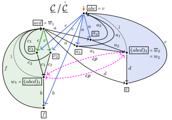
(US) Every guarded LLEE-1-chart (chart, maybe 1-transitions, with LLEE) is uniquely Mil-provably solvable modulo provability in Mil (CALCO 2021).

(IV) The chart interpretation $\mathcal{C}(e)$ of a regular expression e can always be expanded under bisimilarity to a LLEE-1-chart $\underline{\mathcal{C}}(e)$ (TERMGRAPH 2020).

($\text{C}_{\not 1}$) LLEE-charts (without 1-transitions) are preserved by bisimulation collapse (G/Fokkink, LICS'20).

LLEE-preserving Collapse Fails

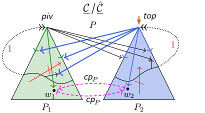
LLEE-1-charts with 1-transitions, however, are **not** preserved under bisimulation collapse. A counterexample is provided by the following LLEE-1-chart $\underline{\mathcal{C}}$:



Identifying the bisimilar vertices w_1 and w_2 yields a chart for which LLEE fails. Also, the subcharts of $\underline{\mathcal{C}}$ that are rooted at w_1 and w_2 are **not** *LLEE-preservingly jointly minimizable under bisimilarity.*

Twin-Crystals

The counterexample to LLEE-preserving collapse is symmetric, and its structure can be abstracted as:



It is a LLEE-1-chart with a single scc (strongly connected component) P that consists of a *pivot part* P_1 below *pivot vertex* ***piv***, and a *top part* P_2 below *top vertex* ***top***. P_1 and P_2 are connected only via transitions from ***piv*** and from ***top***. While both P_1 and P_2 are collapsed, P contains *bisimilarity redundancies* (= distinct bisimilar vertices) such as $\{w_1, w_2\}$ that are linked by a self-inverse counterpart function cp_P. We call such an scc a *twin-crystal*. We have:

(CC) Every Mil-provable solution of a twin-crystal gives rise to a Mil-provable solution of its bisimulation collapse (which often is not a LLEE-1-chart).

Crystallization of LLEE-1-charts

By *crystallization* of a LLEE-1-chart $\underline{\mathcal{C}}$ we mean:

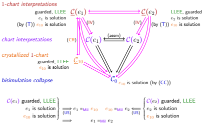
- ▷ a process of minimization of $\underline{\mathcal{C}}$ under bisimilarity by steps that **eliminate most** (all but crystalline) bisimilarity redundancies $\{w_1, w_2\}$, roughly by redirecting transitions that target w_1 over to w_2;
- ▷ hereby only '**reduced**' bisimilarity redundancies can be eliminated LLEE-preservingly, which exist whenever a LLEE-1-chart is **not** collapsed;
- ▷ the result is a *crystallized* LLEE-1-chart that is bisimilar to $\underline{\mathcal{C}}$, and collapsed **apart from** within some its scc's that are twin-crystals.

The crystallization process facilitates to show:

(CR) From every LLEE-1-chart a bisimilar crystallized LLEE-1-chart can be obtained.

Completeness Proof

Let $\mathcal{C}(e_1) \leftrightarrow \mathcal{C}(e_2)$ be bisimilar chart interpretations of regular expressions e_1 and e_2. To secure LLEE, $\mathcal{C}(e_1)$ and $\mathcal{C}(e_2)$ are expanded to their 1-chart interpretations $\underline{\mathcal{C}}(e_1)$ and $\underline{\mathcal{C}}(e_2)$. One of them, say $\underline{\mathcal{C}}(e_1)$, is crystallized to $\mathcal{C}_{10}$. All (1-)charts are linked by (1-)bisimulations to their bisimulation collapse $\mathcal{C}_0$.



From $\mathcal{C}_{10}$ a provable solution e_{10} can be extracted due to LLEE, transferred (T) to the collapse $\mathcal{C}_0$, and then to $\underline{\mathcal{C}}(e_1)$ and $\underline{\mathcal{C}}(e_2)$. On the LLEE-1-charts $\underline{\mathcal{C}}(e_1)$ and $\underline{\mathcal{C}}(e_2)$, e_{10} can be proved equal to the solutions e_1 and e_2 there, respectively. By transitivity, $e_1 =_{\text{Mil}} e_2$ (provability of $e_1 = e_2$ in Mil) follows.

Theorem. *Milner's system* Mil *is complete:* $e_1 =_{\boldsymbol{P}} e_2$ *implies* $e_1 =_{\text{Mil}} e_2$*, for reg. expr's* e_1, e_2.

Next Steps and Projects

- ▷ Monograph project: proof in fine-grained details.
- ▷ Build an animation tool for crystallization.
- ▷ Apply crystallization to find an efficient algorithm for expressibility of finite process graphs by a regular expression modulo bisimilarity.

Contact

clemens.grabmayer@gssi.it

GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

Resources (process interpretation)

- CG: Modeling Terms by Graphs with Structure Constraints
  - TERMGRAPH 2018 Post-Proceedings, EPTCS 288, arXiv:1902.02010.
- CG: Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions
  - TERMGRAPH 2020 Post-Proceedings, EPTCS 334, arXiv:2012.10869.
- CG, Wan Fokkink: A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity
  - LICS 2020, arXiv:2004.12740, video on youtube.
- CG: Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete
  - LICS 2022, arXiv:2209.12188, poster.
- CG: A Coinductive Version/Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity
  - CALCO 2021, arXiv:2108.13104.
  - LMCS 2023, arXiv:2303.14219.

Outlook

correspondences found

- process graphs with LEE
  $\sim$ $P(\cdot)$-interpretations of 1-return-less regular expressions
- process graphs with 1-transitions and with LEE
  $\sim$ $P(\cdot)$-interpretations of regular expressions
- facilitate/may facilitate:
  efficient manipulation/recognition of $P(\cdot)/[\![\cdot]\!]_P$-expressible graphs

slides and resources: `clegra.github.io`

Outlook

correspondences found

- process graphs with LEE
  $\sim$ $P(\cdot)$-interpretations of 1-return-less regular expressions
- process graphs with 1-transitions and with LEE
  $\sim$ $P(\cdot)$-interpretations of regular expressions
- facilitate/may facilitate:
  efficient manipulation/recognition of $P(\cdot)/\llbracket\cdot\rrbracket_P$-expressible graphs

current projects

- PTIME-decidability of LEE (LLEE) and $\llbracket\cdot\rrbracket_P^{\cancel{1r}\backslash\star}$-expressibility
- refinability into LEE-graphs by adding 1-transitions (in PTIME?)
- $\llbracket\cdot\rrbracket_P$-expressibility: $\iff$ expansion and refinability into a crystallized LLEE-1-process-graph (in FPT?)
- full completeness proof of Mil via crystallization
  (two parts: motivation / procedure)

slides and resources: `clegra.github.io`

Comparison results: structure-constrained graphs

λ-calculus with letrec under $=_{\lambda^\infty}$

Not available: graph interpretation that is studied modulo $\underline{\leftrightarrow}$

Regular expressions under $\underline{\leftrightarrow}_P$

Given: graph interpretation $P(\cdot)$, studied modulo bisimulation $\underline{\leftrightarrow}$

- not closed under $\underline{\rightarrow}$, and $\underline{\leftrightarrow}$, incomplete under $\underline{\leftrightarrow}$

Comparison results: structure-constrained graphs

λ-calculus with letrec under $=_{\boldsymbol{\lambda}^\infty}$

Not available: graph interpretation that is studied modulo $\underline{\leftrightarrow}$

Defined: int's $[\![\cdot]\!]_{\mathcal{H}}/[\![\cdot]\!]_{\mathcal{T}}$ as higher-order/first-order λ-term graphs

- closed under $\underline{\to}$ (hence under collapse)
- back-/forth correspondence with λ-calculus with letrec
  - efficient translation and readback
  - translation is inverse of readback

Regular expressions under $\underline{\leftrightarrow}_P$

Given: graph interpretation $P(\cdot)$, studied modulo bisimulation $\underline{\leftrightarrow}$

- not closed under $\underline{\to}$, and $\underline{\leftrightarrow}$, incomplete under $\underline{\leftrightarrow}$

Comparison results: structure-constrained graphs

λ-calculus with letrec under $=_{\lambda^\infty}$

Not available: graph interpretation that is studied modulo $\underline{\leftrightarrow}$

Defined: int's $[\![\cdot]\!]_{\mathcal{H}}/[\![\cdot]\!]_{\mathcal{T}}$ as higher-order/first-order λ-term graphs

- closed under $\underline{\rightarrow}$ (hence under collapse)
- back-/forth correspondence with λ-calculus with letrec
  - efficient translation and readback
  - translation is inverse of readback

Regular expressions under $\underline{\leftrightarrow}_P$

Given: graph interpretation $P(\cdot)$, studied modulo bisimulation $\underline{\leftrightarrow}$

- not closed under $\underline{\rightarrow}$, and $\underline{\leftrightarrow}$, incomplete under $\underline{\leftrightarrow}$

Defined: class of process graphs with LEE / (layered) LEE-witness

- closed under $\underline{\rightarrow}$ (hence under collapse)
- back-/forth correspondence with 1-return-less expr's
- contains the collapse of a process graph G
  $\iff$ G is $[\![\cdot]\!]_P^{\cancel{1r}\backslash\star}$-expressible