Maximal Sharing in the Lambda Calculus with letrec

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

Clemens Grabmayer

VU University Amsterdam (Dept. of CS)

Jan Rochel

Be Sport, Paris
(Utrecht University (Dept. of CS))

TCS Seminar, VU University

6 October 2016

maximal sharing: example (fix)

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

L

maximal sharing: example (fix)

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

L

L_0

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

maximal sharing: example (fix)

$$\begin{array}{ccc} L & & \\ \text{unfold} \downarrow & & \\ M & & \\ \text{unfold} \uparrow & & \\ L_0 & & \end{array}$$

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

maximal sharing: example (fix)

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

L

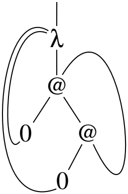
L_0

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

maximal sharing: example (fix)

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

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



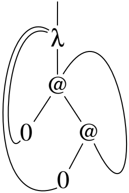
L_0

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

maximal sharing: example (fix)

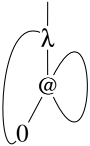
$$
\begin{tikzcd}
L \arrow[rr, bend left, mapsto, "\text{interpret}"] & & G \arrow[d, Rightarrow, "\text{collapse}"] \\
L_0 & & G_0
\end{tikzcd}
$$

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



$\Downarrow$

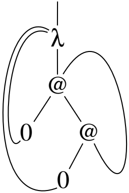
$\lambda f.\,\text{let } r = f\ r \text{ in } r$

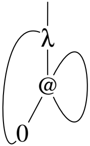


maximal sharing: example (fix)

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

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



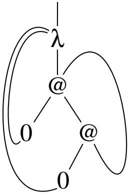


$\lambda f.\,\text{let } r = f\;r \text{ in } r \xleftarrow[\text{readback}]{}$

maximal sharing: example (fix)

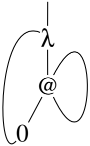
$$
\begin{tikzcd}
L \arrow[rr, bend left, mapsto, "\text{interpret}"] \arrow[d, mapsto, "\text{unfold}"'] & & G \arrow[dd, Rightarrow, "\text{collapse}"] \\
M & & \\
L_0 \arrow[u, mapsto, "\text{unfold}"] \arrow[rr, bend left, mapsto, "\text{interpret}"] & & G_0 \arrow[ll, bend left, mapsto, "\text{readback}"]
\end{tikzcd}
$$

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



$$\lambda f.\,\text{let } r = f\,(f\;r) \text{ in } r \xmapsto{[\![\cdot]\!]_{\lambda^\infty}} \lambda f.\,f\,(f\,(\dots)) \xleftarrow{[\![\cdot]\!]_{\lambda^\infty}} \lambda f.\,\text{let } r = f\;r \text{ in } r$$

$$\lambda f.\,\text{let } r = f\;r \text{ in } r \;\underset{\text{readback}}{\overset{[\![\cdot]\!]_{\mathcal{T}}}{\rightleftarrows}}$$



motivation, questions, and results

motivation

- desirable: increase sharing in programs
  - code that is as compact as possible
  - avoid duplication of reduction work at run-time
- useful: check equality of unfolding semantics of programs

questions

(1): how to maximize sharing in programs?

(2): how to check for unfolding equivalence?

we restrict to λ_{letrec}, the λ-calculus with letrec

- as abstraction & syntactical core of functional languages

our results:

- efficient methods solving questions (1) and (2) for λ_{letrec}

motivation, questions, and results

motivation

- desirable: increase sharing in programs
  - code that is as compact as possible
  - avoid duplication of reduction work at run-time
- useful: check equality of unfolding semantics of programs

questions

(1): how to maximize sharing in programs?

(2): how to check for unfolding equivalence?

we restrict to λ_{letrec}, the λ-calculus with letrec

- as abstraction & syntactical core of functional languages

our results:

- efficient methods solving questions (1) and (2) for λ_{letrec}

motivation, questions, and results

motivation

- desirable: increase sharing in programs
  - code that is as compact as possible
  - avoid duplication of reduction work at run-time
- useful: check equality of unfolding semantics of programs

questions

(1): how to maximize sharing in programs?

(2): how to check for unfolding equivalence?

we restrict to λ_{letrec}, the λ-calculus with letrec

- as abstraction & syntactical core of functional languages

our results:

- efficient methods solving questions (1) and (2) for λ_{letrec}

motivation, questions, and results

motivation

- desirable: increase sharing in programs
  - code that is as compact as possible
  - avoid duplication of reduction work at run-time
- useful: check equality of unfolding semantics of programs

questions

(1): how to maximize sharing in programs?

(2): how to check for unfolding equivalence?

we restrict to $\boldsymbol{\lambda}_{\text{letrec}}$, the λ-calculus with letrec

- as abstraction & syntactical core of functional languages

our results:

- efficient methods solving questions (1) and (2) for $\boldsymbol{\lambda}_{\text{letrec}}$

outline

- methods consist of the steps:

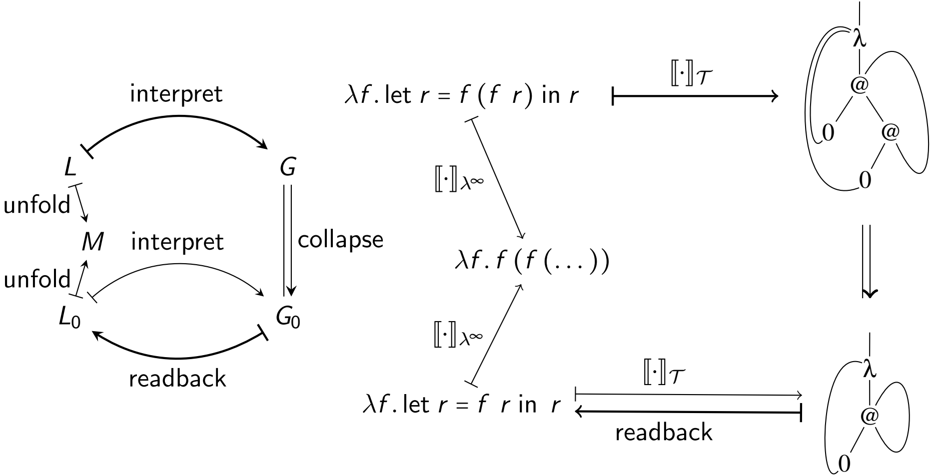
  interpretation of $\boldsymbol{\lambda}_{\text{letrec}}$-terms as term graphs

  - higher-order: λ-ho-term-graphs
  - first-order: λ-term-graphs

  bisimilarity & bisimulation collapse of λ-term-graphs

  readback of λ-term-graphs as $\boldsymbol{\lambda}_{\text{letrec}}$-terms
- implementation
- complexity
- extensions and applications

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

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

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

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

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

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

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

maximal sharing: the method

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

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

maximal sharing: the method

$$\begin{tikzcd}
L \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & \mathcal{G} \arrow[r, mapsto] \arrow[d, "|\!\downarrow"'] & G \arrow[d, "|\!\downarrow"'] \\
L_0 & \mathcal{G}_0 \arrow[r, mapsto] & G_0 \arrow[ll, mapsto, bend left, "\mathrm{rb}"]
\end{tikzcd}$$

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 $L_0 = \mathrm{rb}(G_0)$.

maximal sharing: the method

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

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 $L_0 = \mathrm{rb}(G_0)$.

contribution

conceptually

- reason about syntactically expressed sharing
  via an adequate term graph semantics
- reduction to problems accessible by standard methods

contribution

conceptually

- reason about syntactically expressed sharing
  via an adequate term graph semantics
- reduction to problems accessible by standard methods

maximal sharing method

- extends 'maximal sharing'
  from first-order terms to higher-order terms (with binding)
- significantly extends common subexpression elimination
- is targeted at maximizing sharing statically
  - with respect to the unfolding semantics
  - not: organize/maximize sharing dynamically during evaluation

unfolding equivalence: example

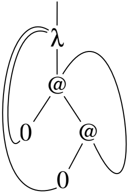
$$
\begin{tikzcd}
L_1 \arrow[d, mapsto, "\text{unfold}"', "?"] \\
M \\
L_2 \arrow[u, mapsto, "\text{unfold}", "?"']
\end{tikzcd}
$$

$$
\begin{tikzcd}
\lambda f.\,\text{let } r = f\,(f\ \ r) \text{ in } r \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"'] \\
\lambda f.\,f\,(f\,(\ldots)) \\
\lambda f.\,\text{let } r = f\ \ r \text{ in } \ r \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"]
\end{tikzcd}
$$

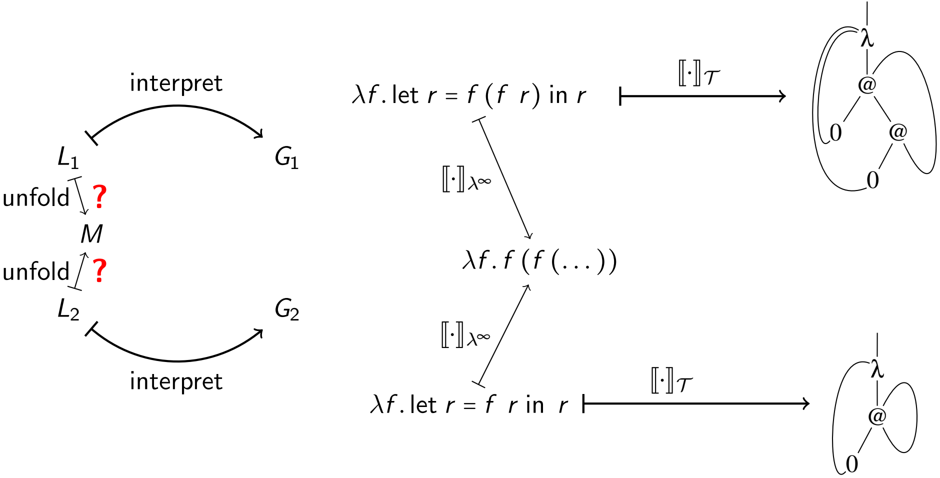
unfolding equivalence: example

$$
\begin{tikzcd}
L_1 \arrow[rr, mapsto, bend left, "\text{interpret}"] \arrow[d, mapsto, "\text{unfold}"', "\color{red}{?}"] & & G_1 \\
M & & \\
L_2 \arrow[u, mapsto, "\text{unfold}", "\color{red}{?}"'] & &
\end{tikzcd}
$$

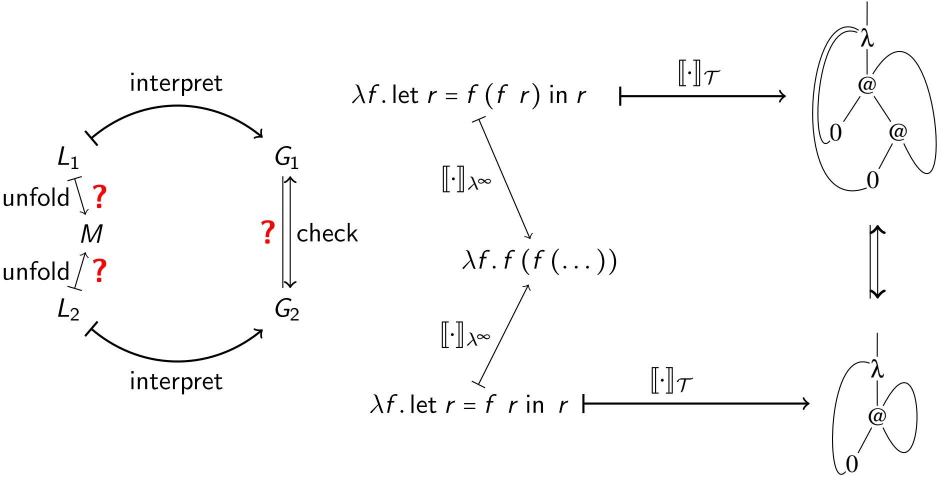
$$
\begin{tikzcd}
\lambda f.\,\text{let } r = f\,(f\ \ r) \text{ in } r \arrow[rr, mapsto, "[\![\cdot]\!]_{\mathcal{T}}"] \arrow[d, mapsto, "[\![\cdot]\!]_{\lambda^\infty}"'] & & {} \\
\lambda f.\,f\,(f\,(\ldots)) & & \\
\lambda f.\,\text{let } r = f\ \ r \text{ in } \ r \arrow[u, mapsto, "[\![\cdot]\!]_{\lambda^\infty}"] & &
\end{tikzcd}
$$



unfolding equivalence: the method



unfolding equivalence: the method



unfolding equivalence: the method

$$
\begin{array}{c}
L_1 \\
[\![\cdot]\!]_{\lambda^\infty} \downarrow \color{red}{\textbf{?}} \\
M \\
[\![\cdot]\!]_{\lambda^\infty} \uparrow \color{red}{\textbf{?}} \\
L_2
\end{array}
$$

unfolding equivalence: the method

$$\begin{tikzcd}
L_1 \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\color{red}{?}"] & \mathcal{G}_1 \arrow[r, mapsto] & G_1 \\
M & & \\
L_2 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}", "\color{red}{?}"'] & &
\end{tikzcd}$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:
   a. higher-order term graphs
      $\mathcal{G}_1 = [\![L_1]\!]_{\mathcal{H}}$
   b. first-order term graphs
      $G_1 = [\![L_1]\!]_{\mathcal{T}}$

unfolding equivalence: the method

$$\begin{tikzcd}
L_1 \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "?"] & \mathcal{G}_1 \arrow[r, mapsto] & G_1 \\
M & & \\
L_2 \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend right, "{[\![\cdot]\!]_{\mathcal{T}}}"'] \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}", "?"'] & \mathcal{G}_2 \arrow[r, mapsto] & G_2
\end{tikzcd}$$

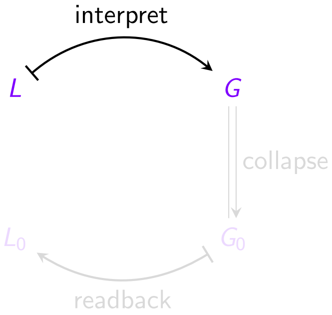
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:
   a. higher-order term graphs
      $\mathcal{G}_1 = [\![L_1]\!]_{\mathcal{H}}$ and $\mathcal{G}_2 = [\![L_2]\!]_{\mathcal{H}}$
   b. first-order term graphs
      $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$

unfolding equivalence: the method

$$
\begin{tikzcd}
L_1 \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\textbf{?}"] & \mathcal{G}_1 \arrow[r, mapsto] & G_1 \arrow[dd, Leftrightarrow, "\textbf{?}"] \\
M & & \\
L_2 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}", "\textbf{?}"'] \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{H}}}"] \arrow[rr, mapsto, bend right, "{[\![\cdot]\!]_{\mathcal{T}}}"'] & \mathcal{G}_2 \arrow[r, mapsto] & G_2
\end{tikzcd}
$$

1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:
   a. higher-order term graphs
      $\mathcal{G}_1 = [\![L_1]\!]_{\mathcal{H}}$ and $\mathcal{G}_2 = [\![L_2]\!]_{\mathcal{H}}$
   b. first-order term graphs
      $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$
2. check bisimilarity
   of f-o term graphs G_1 and G_2

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:

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

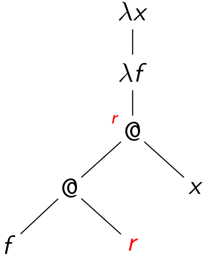
$L \quad \longmapsto_{\text{max-sharing}} \quad L_0$

graph interpretation (example 1)

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

graph interpretation (example 1)

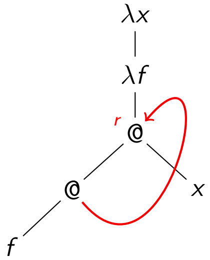
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree

graph interpretation (example 1)

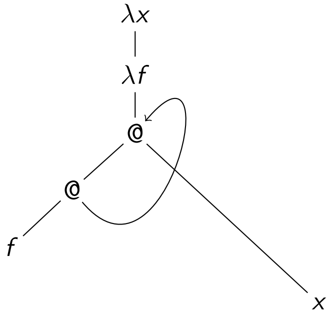
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 1)

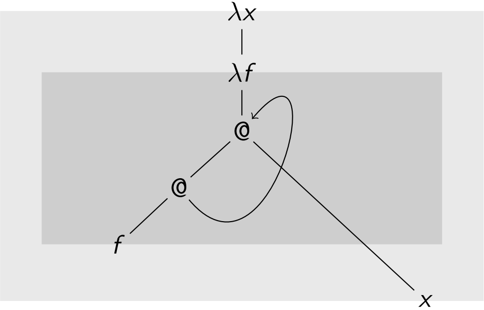
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 1)

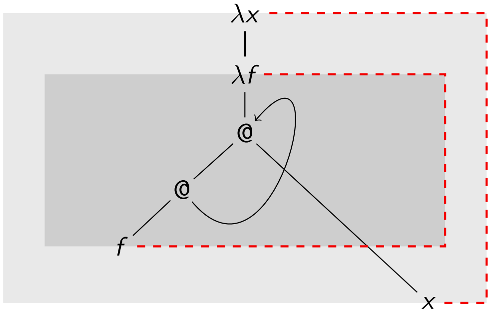
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

graph interpretation (example 1)

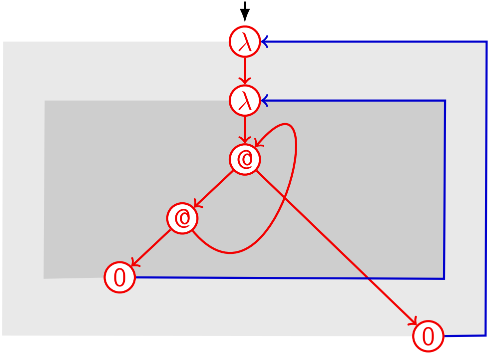
$L_0 = \lambda x.\, \lambda f.\, \mathrm{let}\ r = f\ r\ x\ \mathrm{in}\ r$



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

graph interpretation (example 1)

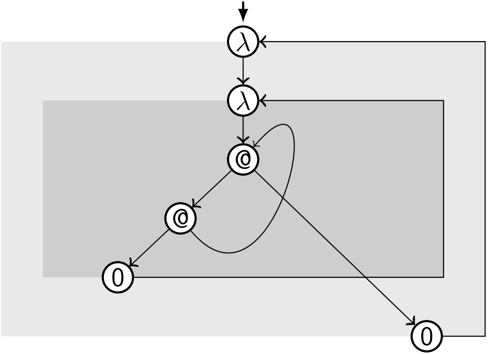
$L_0 = \lambda x. \lambda f. \, \text{let } r = f \, r \, x \text{ in } r$



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

graph interpretation (example 1)

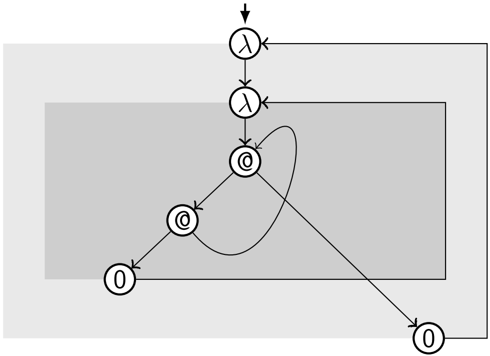
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

graph interpretation (example 1)

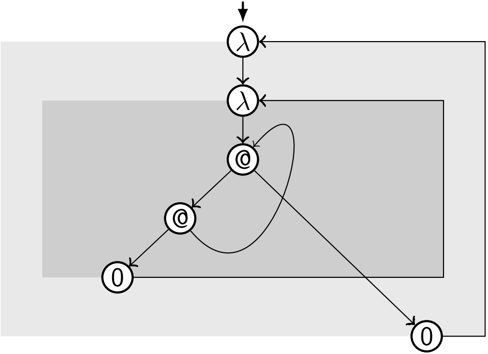
$L_0 = \lambda x. \lambda f. \text{let } r = f\,r\,x \text{ in } r$



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

graph interpretation (example 1)

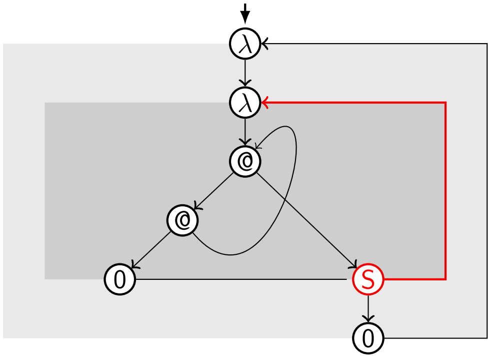
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

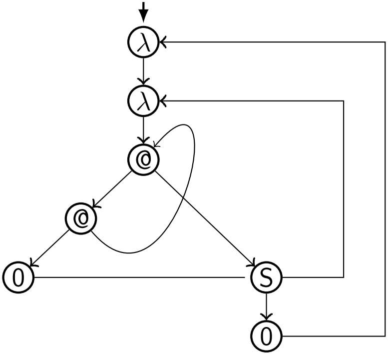
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

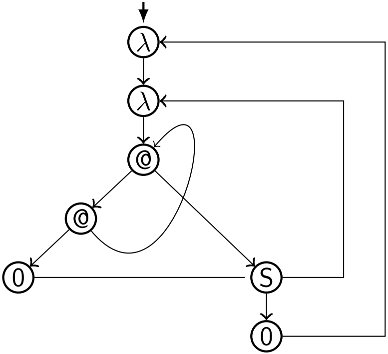
$L_0 = \lambda x.\, \lambda f.\, \mathrm{let}\ r = f\ r\ x\ \mathrm{in}\ r$



first-order term graph with scope vertices with backlinks

graph interpretation (example 1)

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



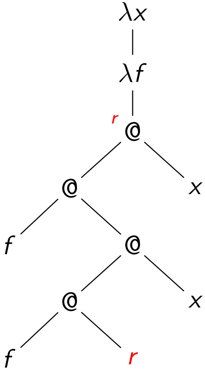
λ-term-graph $[\![L_0]\!]_{\mathcal{T}}$

graph interpretation (example 2)

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

graph interpretation (example 2)

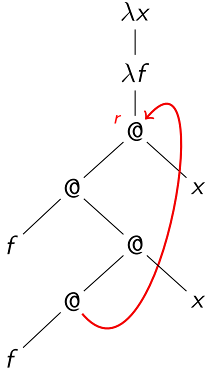
$L = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\ (f\ r\ x)\ x\ \mathsf{in}\ r$



syntax tree

graph interpretation (example 2)

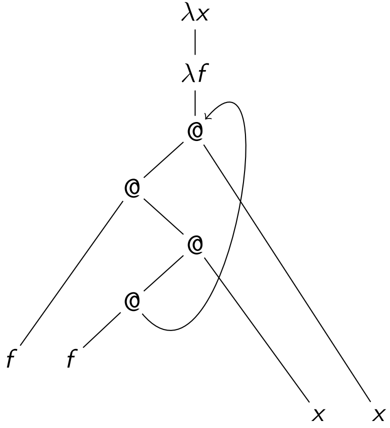
$L = \lambda x.\, \lambda f.\, \mathrm{let}\ r = f\,(f\,r\,x)\,x\ \mathrm{in}\ r$



syntax tree (+ recursive backlink)

graph interpretation (example 2)

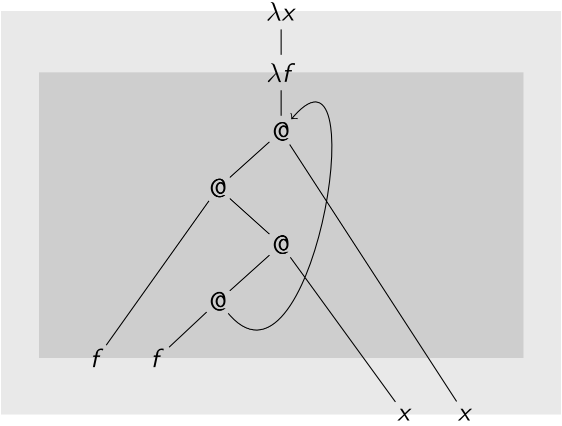
$L = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\,(f\,r\,x)\,x\ \mathsf{in}\ r$



syntax tree (+ recursive backlink)

graph interpretation (example 2)

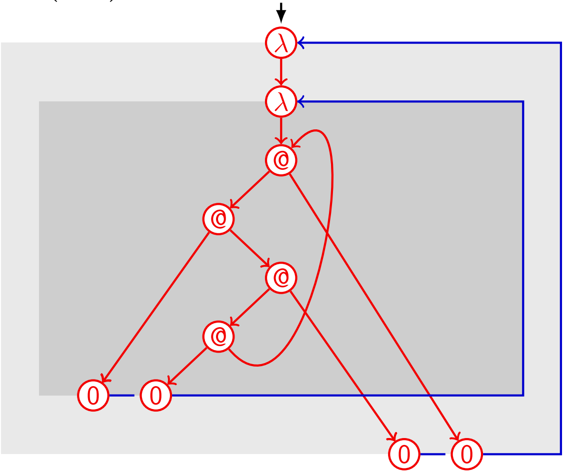
$L = \lambda x.\, \lambda f.\, \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

graph interpretation (example 2)

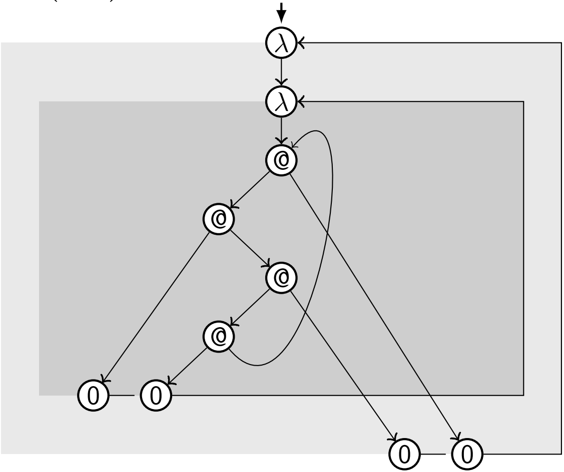
$L = \lambda x.\, \lambda f.\, \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



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

graph interpretation (example 2)

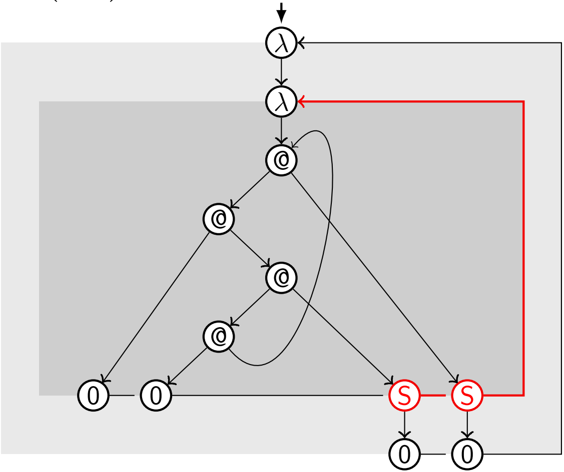
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

graph interpretation (example 2)

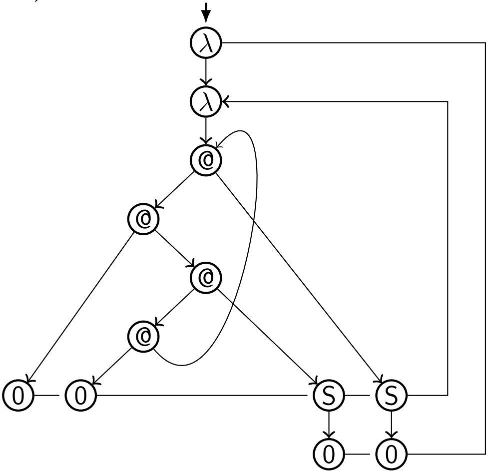
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

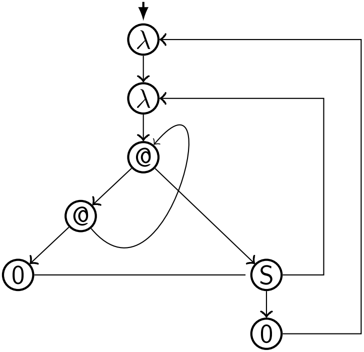
graph interpretation (example 2)

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

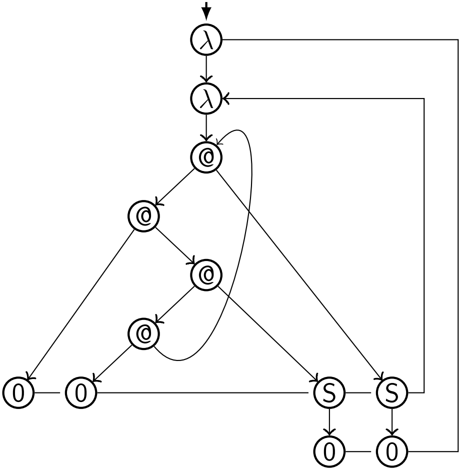


λ-term-graph $[\![L]\!]_{\mathcal{T}}$

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:

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

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

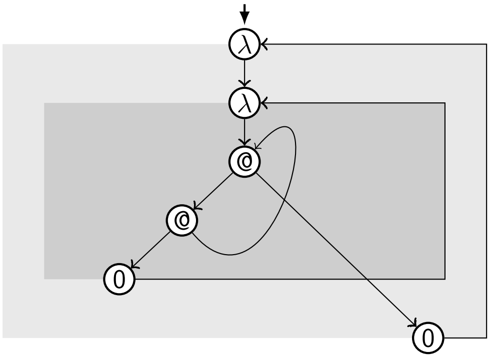
Theorem

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:

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

higher-order term graphs (scope sets/abstraction prefixes)

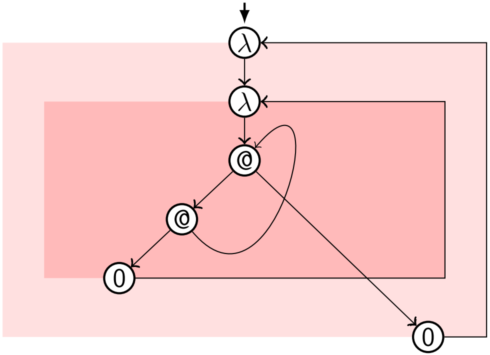
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph (+ scope sets)

higher-order term graphs (scope sets/abstraction prefixes)

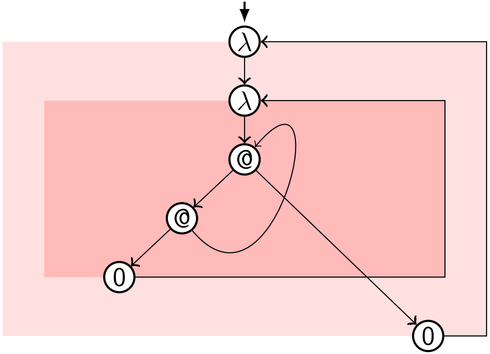
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\; r\; x \text{ in } r$



first-order term graph (+ scope sets)

higher-order term graphs (scope sets/abstraction prefixes)

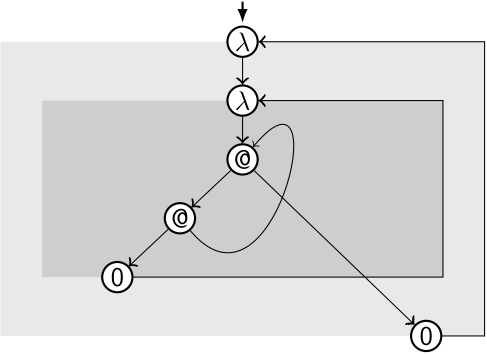
$L_0 = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\ r\ x\ \mathsf{in}\ r$



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

higher-order term graphs (scope sets/abstraction prefixes)

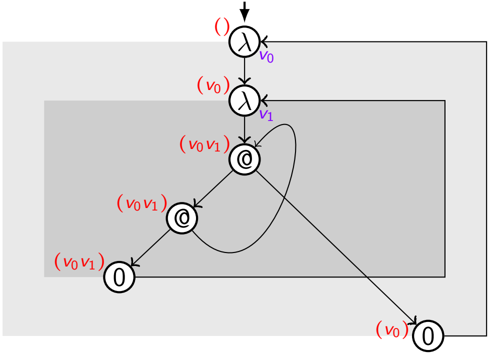
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

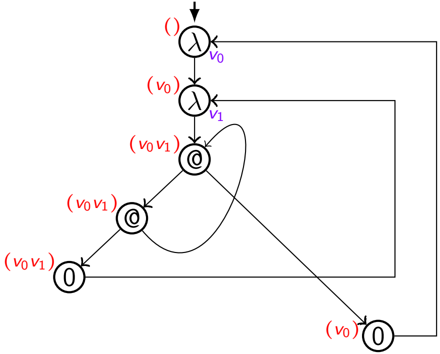
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

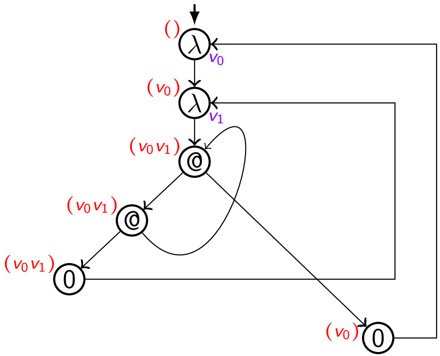
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

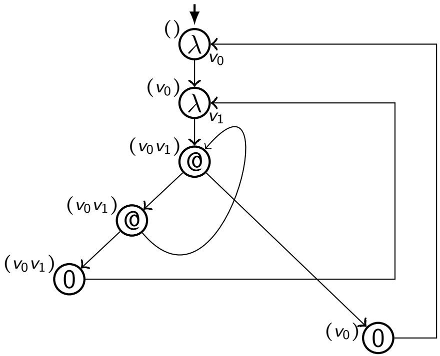
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\; r\; x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

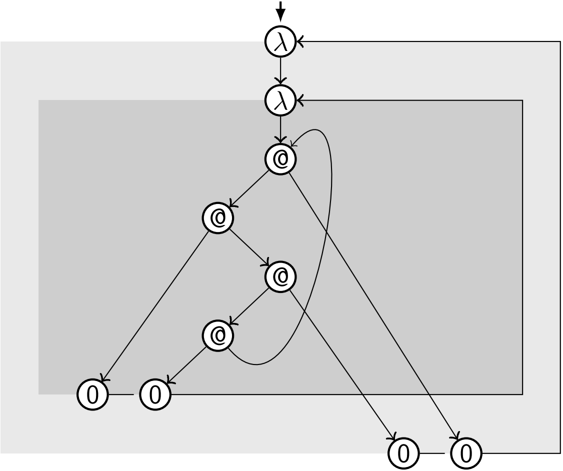
$L_0 = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\ r\ x\ \mathsf{in}\ r$



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

higher-order term graphs (scope sets/abstraction prefixes)

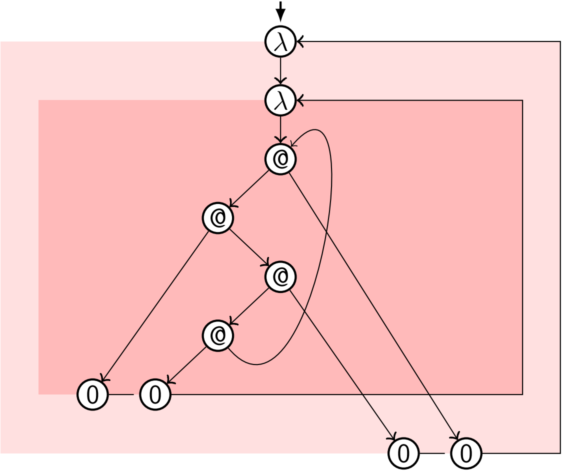
$L = \lambda x.\,\lambda f.\,\text{let } r = f\;(f\;r\;x)\;x \text{ in } r$



first-order term graph (+ scope sets)

higher-order term graphs (scope sets/abstraction prefixes)

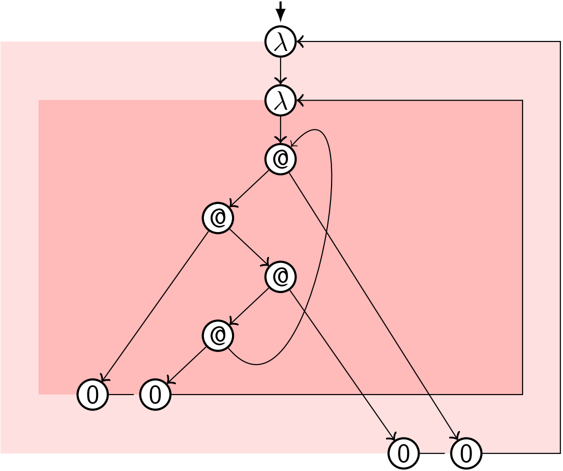
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\; r\; x)\; x \text{ in } r$



first-order term graph (+ scope sets)

higher-order term graphs (scope sets/abstraction prefixes)

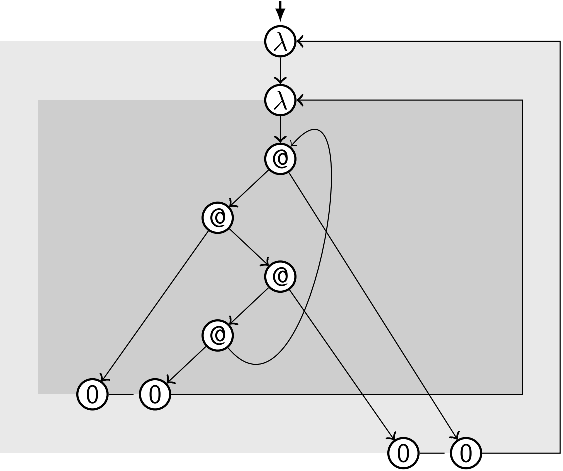
$L = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\ (f\ r\ x)\ x\ \mathsf{in}\ r$



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

higher-order term graphs (scope sets/abstraction prefixes)

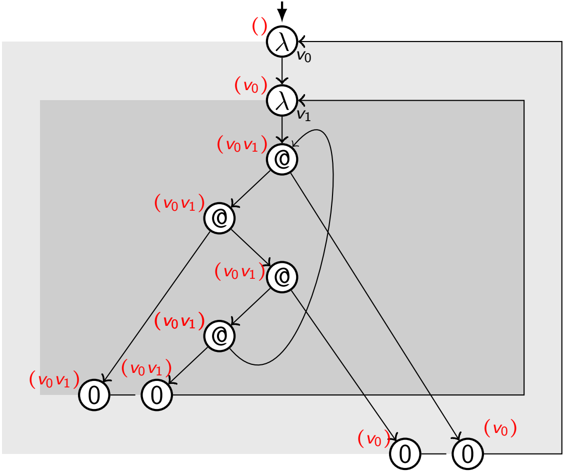
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

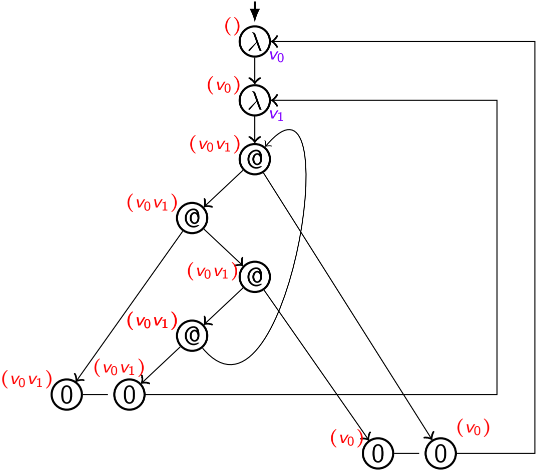
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

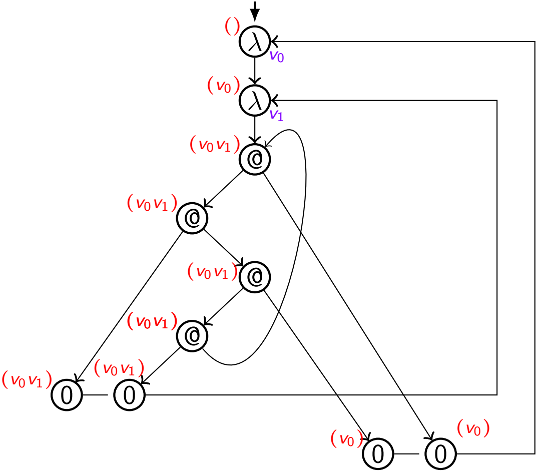
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

higher-order term graphs (scope sets/abstraction prefixes)

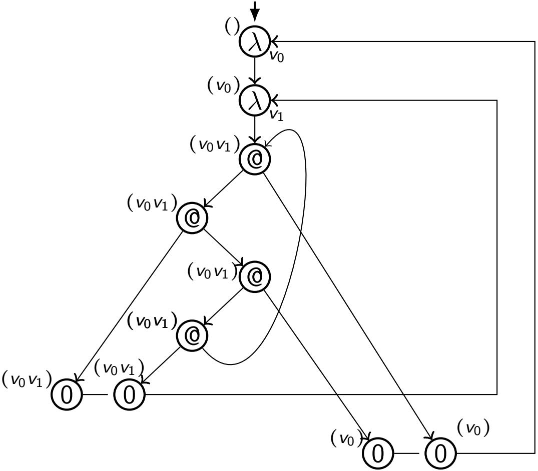
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

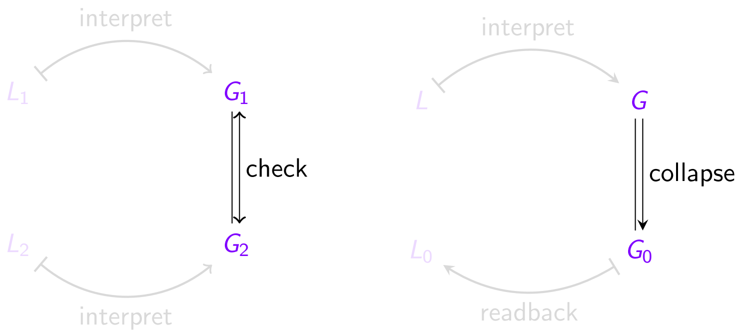
higher-order term graphs (scope sets/abstraction prefixes)

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

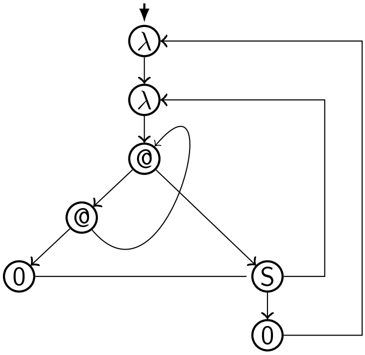


λ-higher-order-term-graph $[\![L_0]\!]_{\mathcal{H}}$

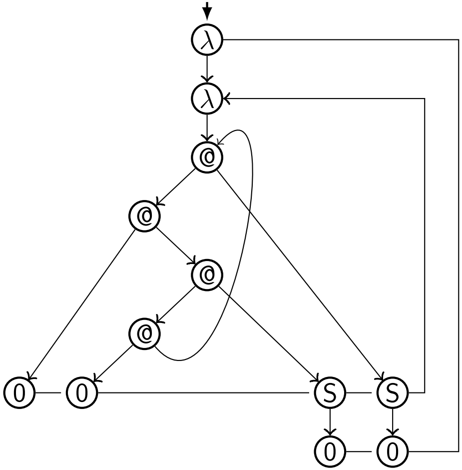
bisimulation check and collapse



bisimulation check between λ-term-graphs

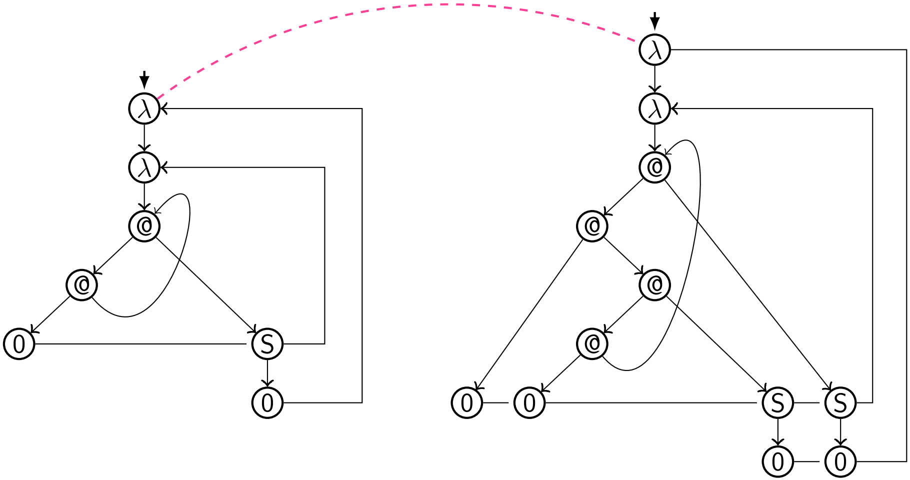


$[\![L_0]\!]_{\mathcal{T}}$



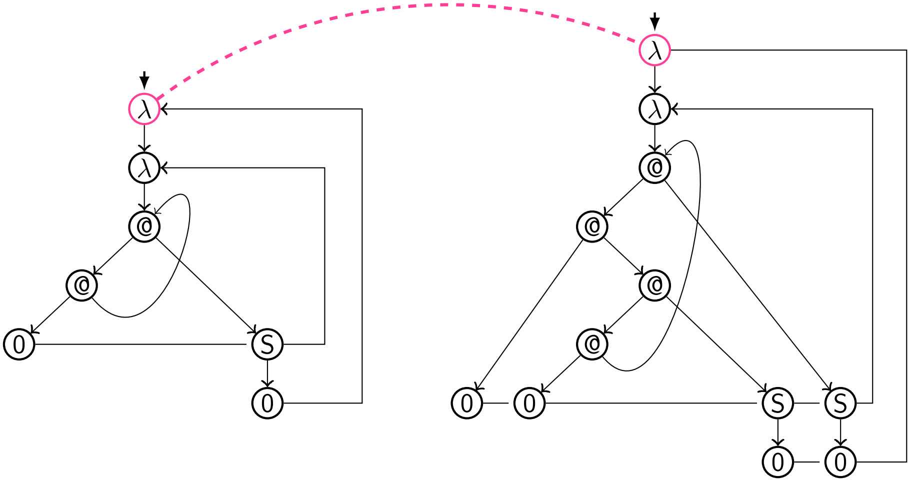
$[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



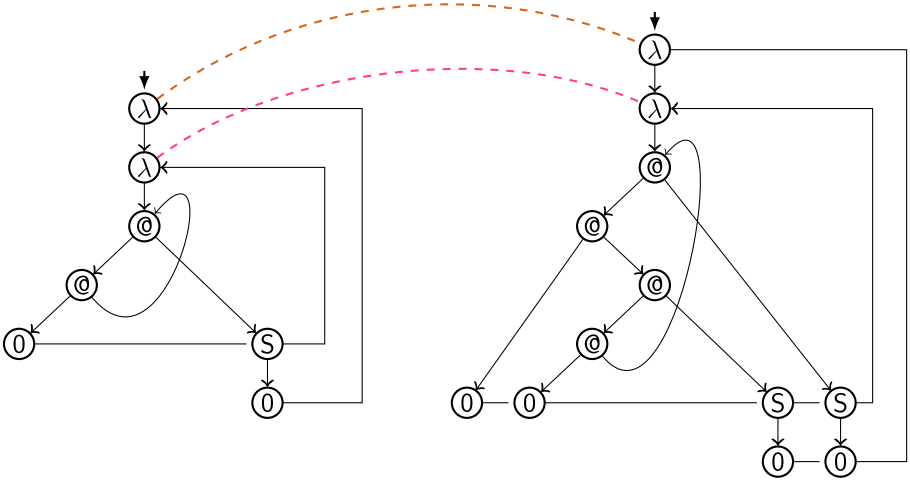
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



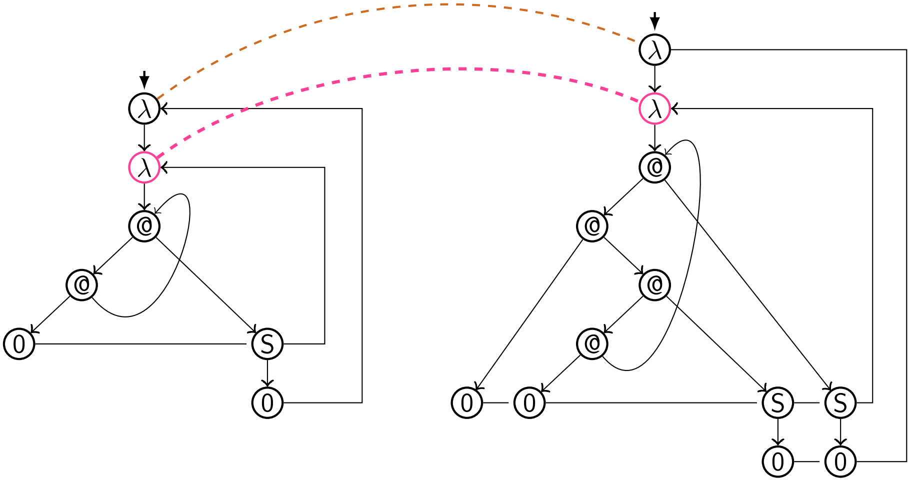
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



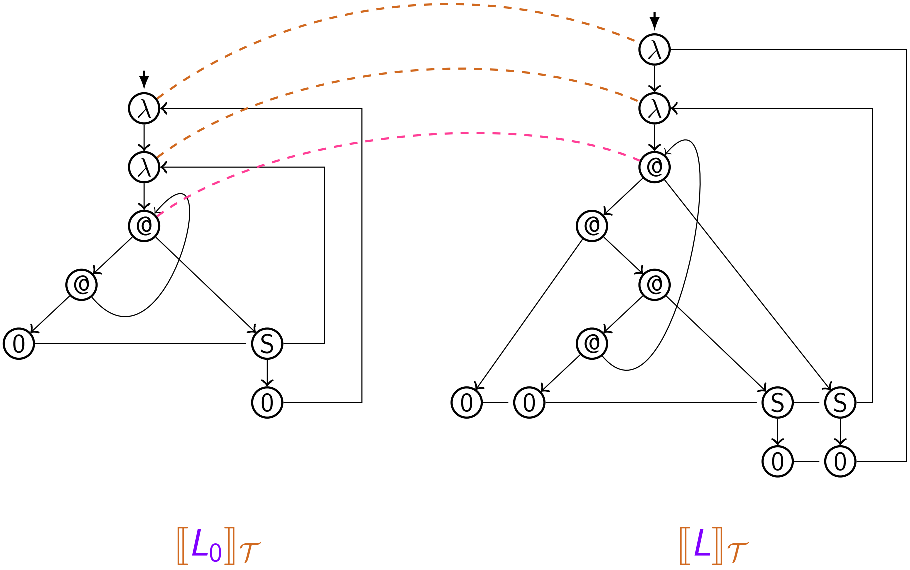
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

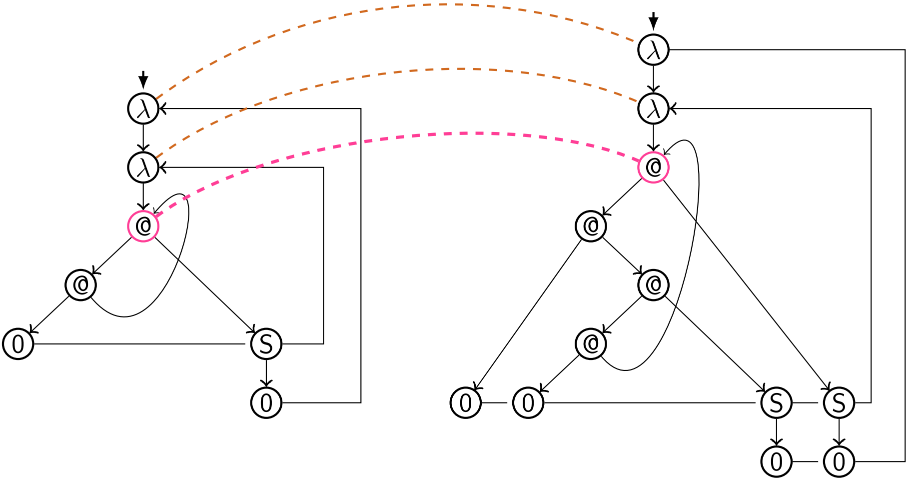


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

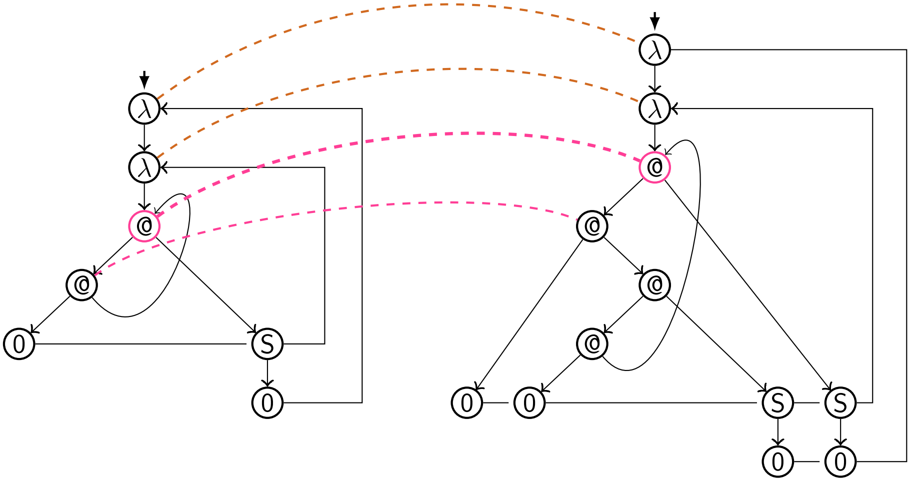


bisimulation check between λ-term-graphs



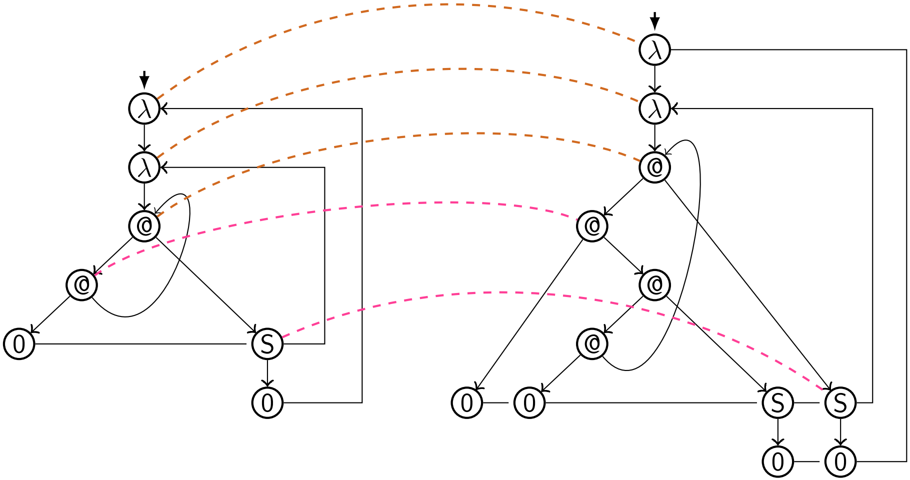
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



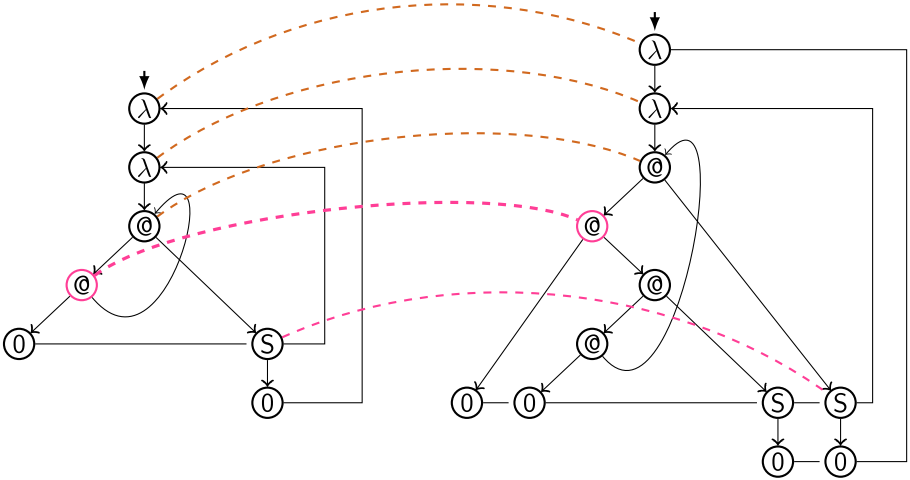
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



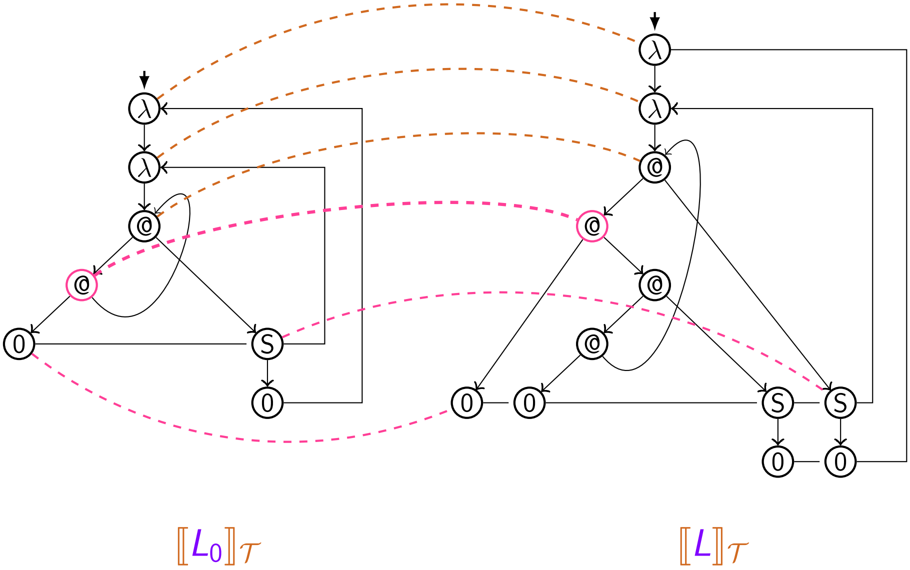
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

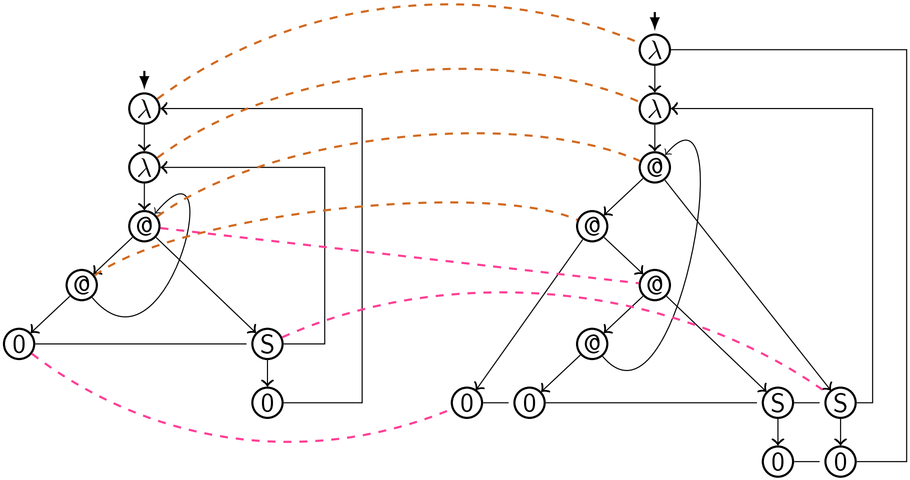


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

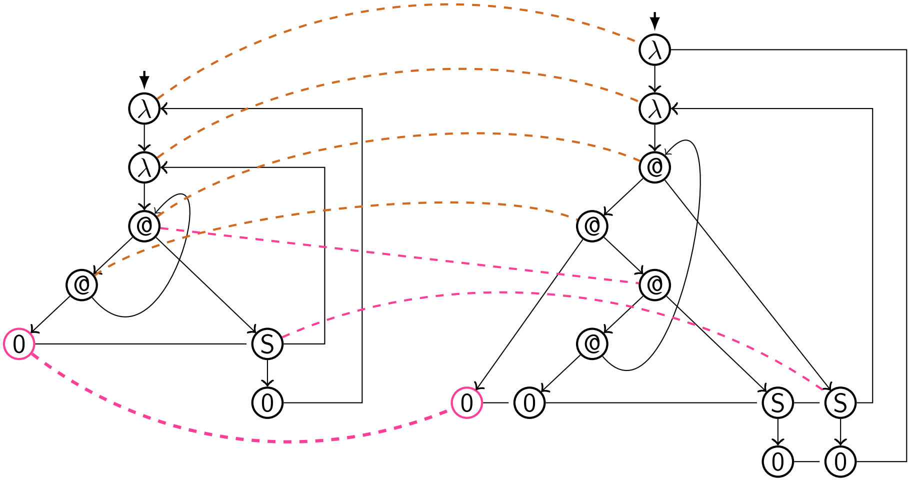


bisimulation check between λ-term-graphs



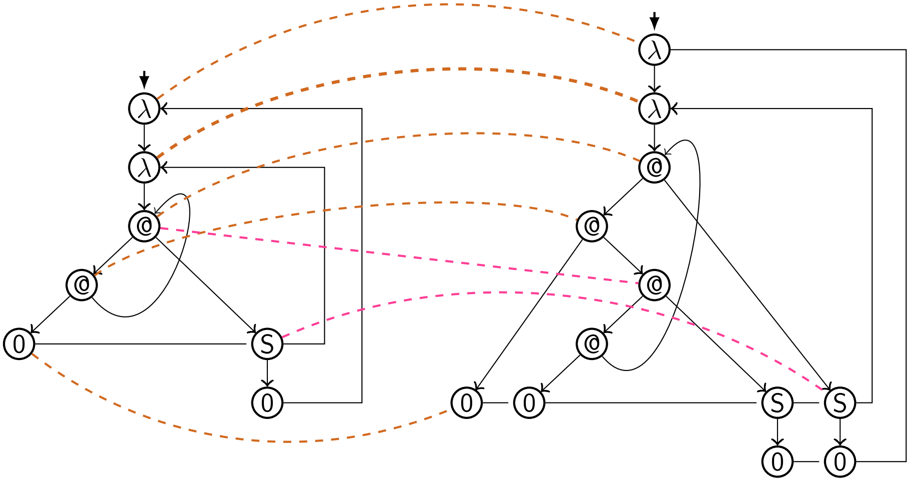
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



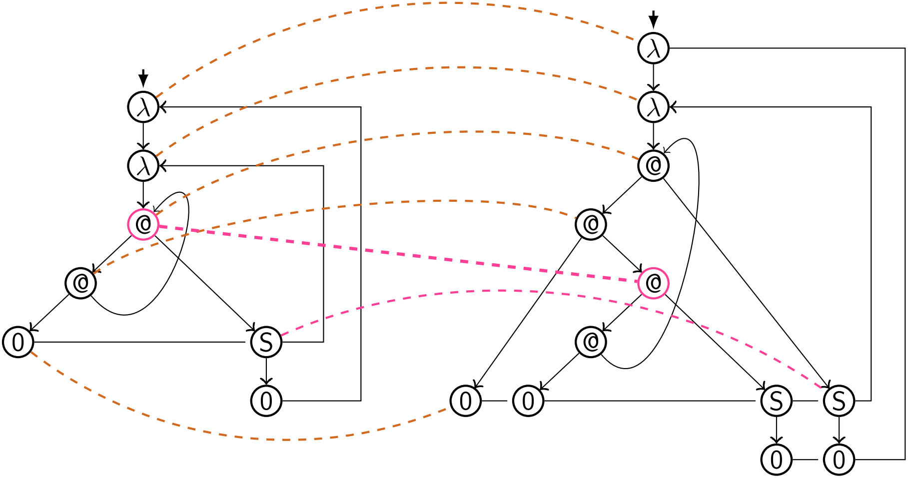
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



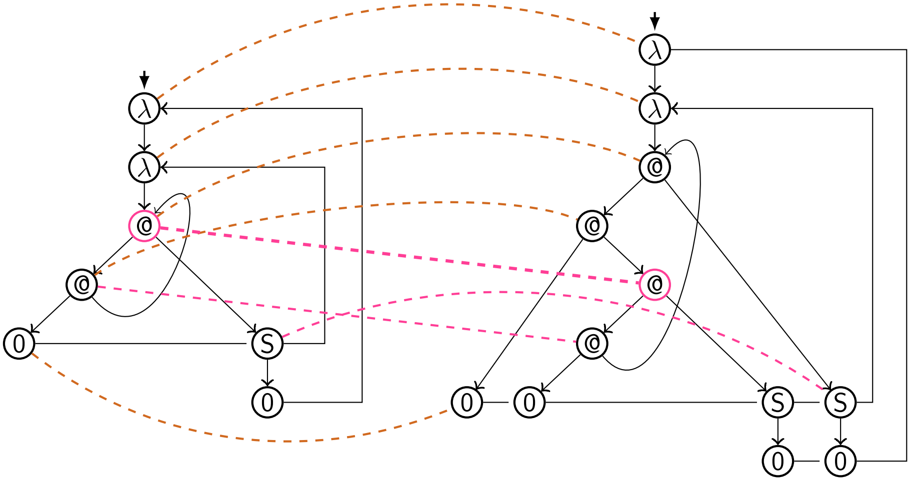
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



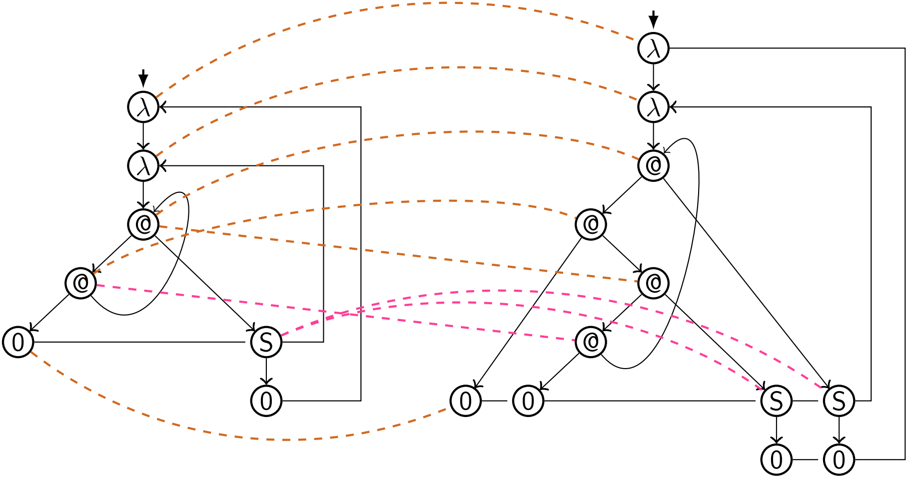
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



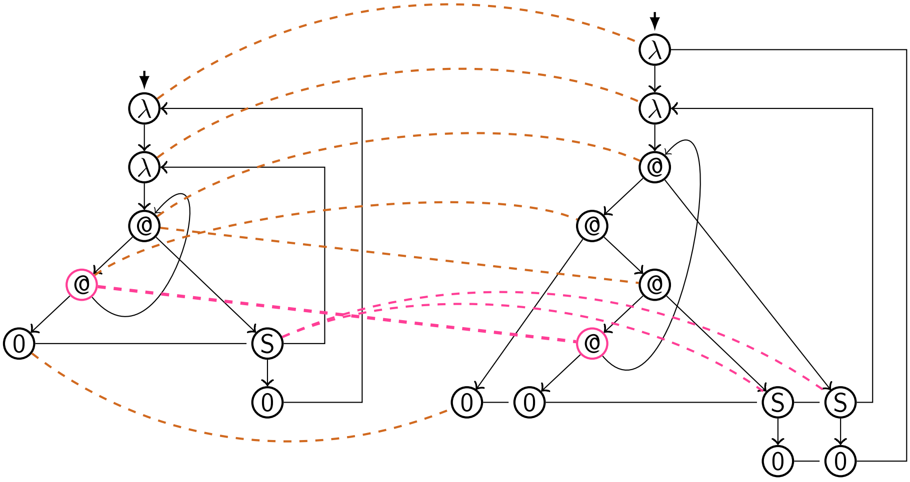
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



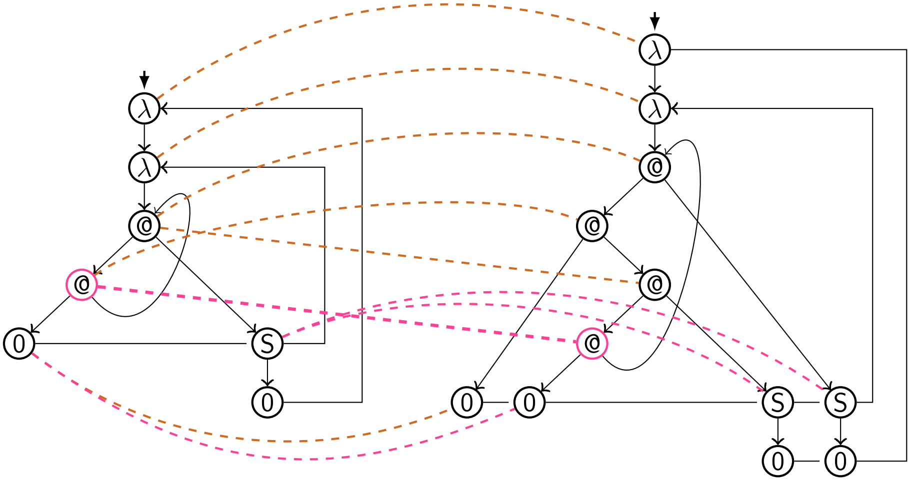
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



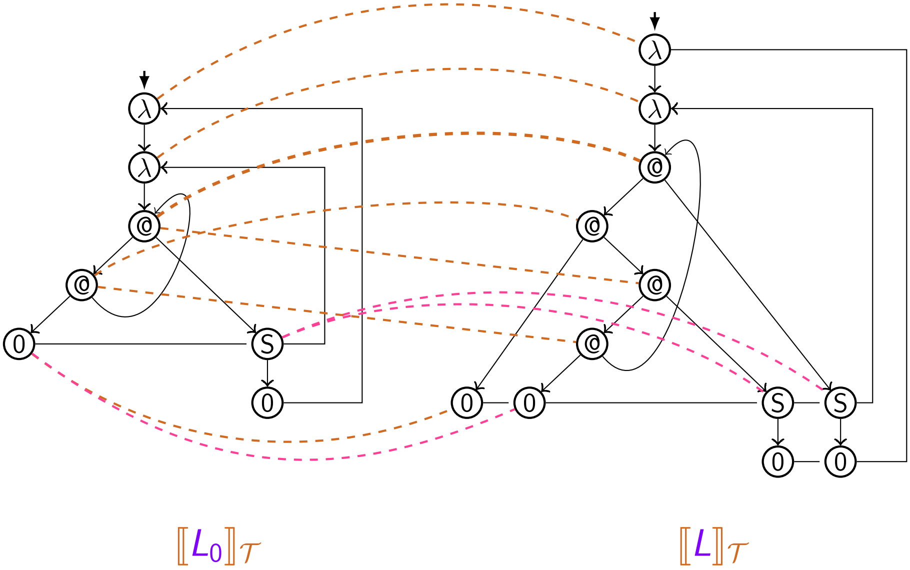
$[\![L_0]\!]_{\mathcal{T}}$ $\qquad$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

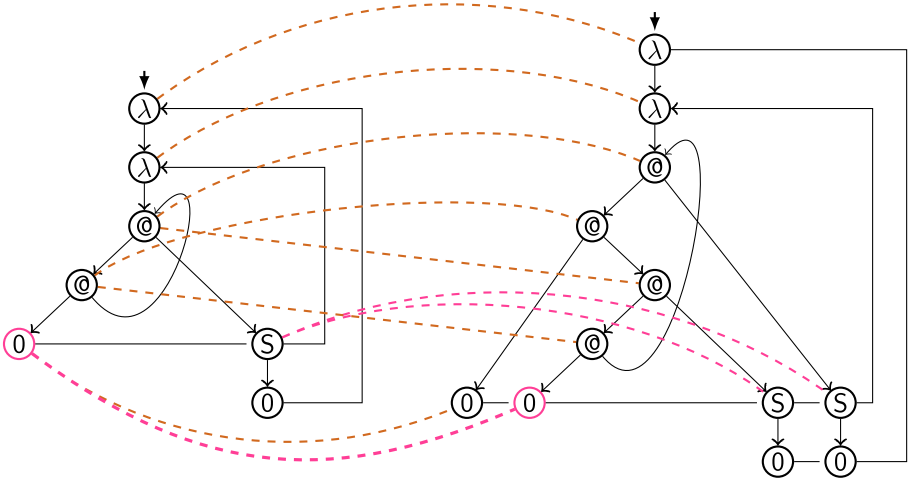


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

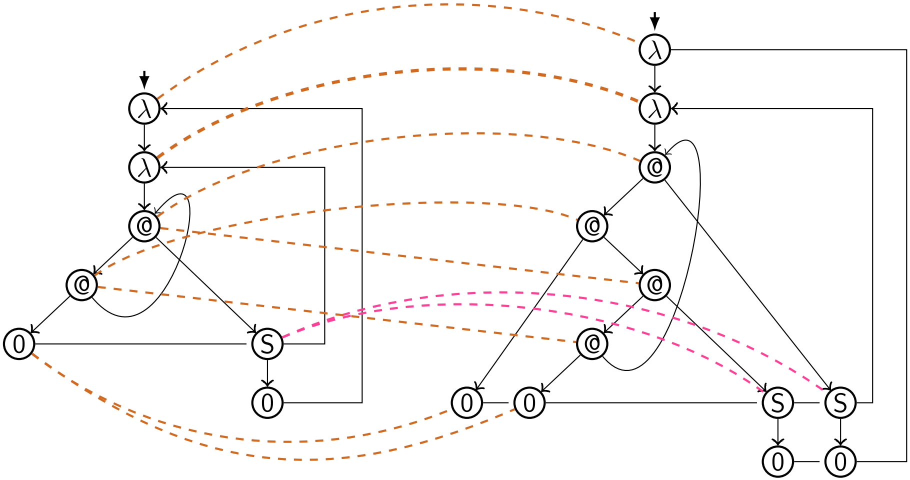


bisimulation check between λ-term-graphs



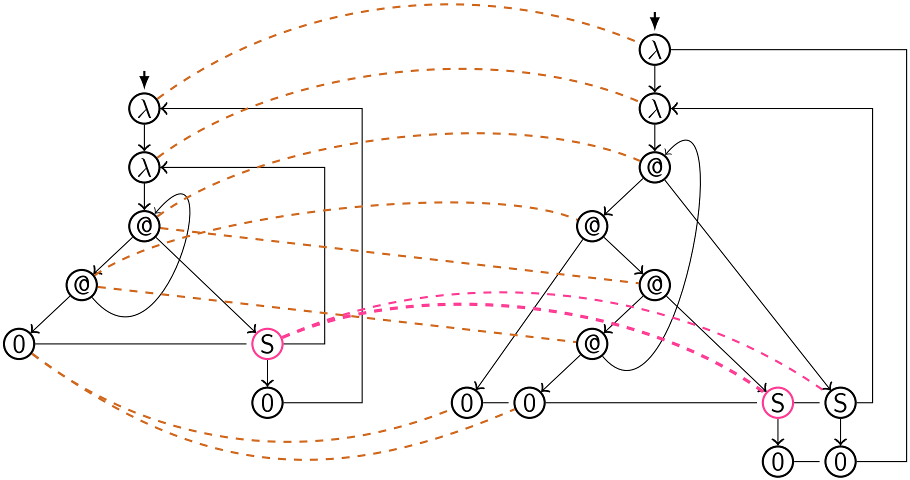
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



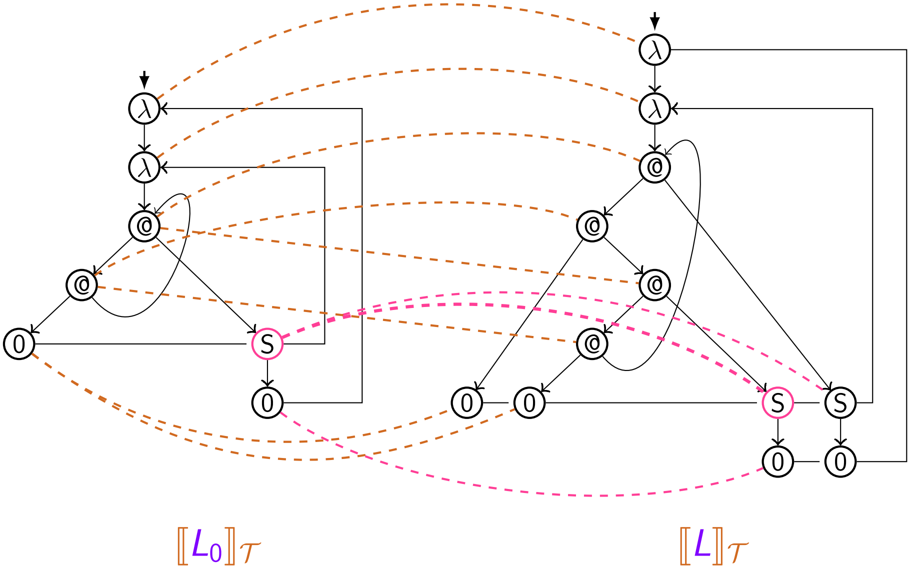
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

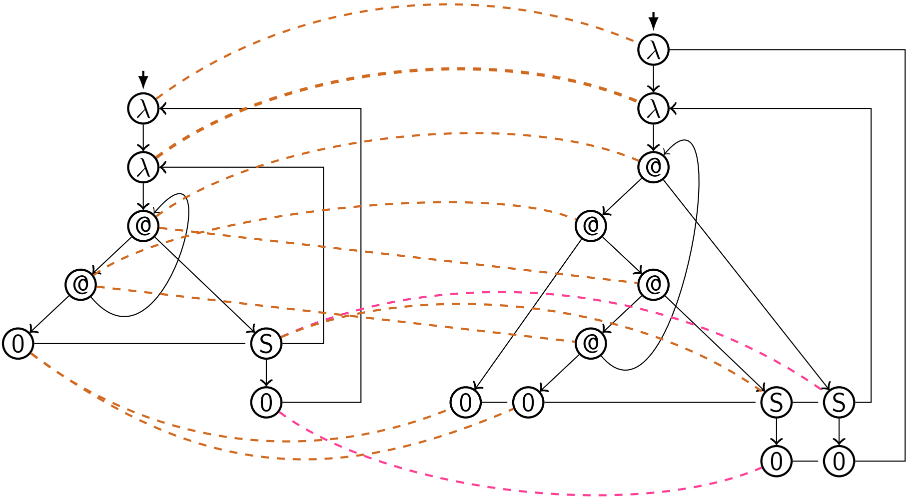


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

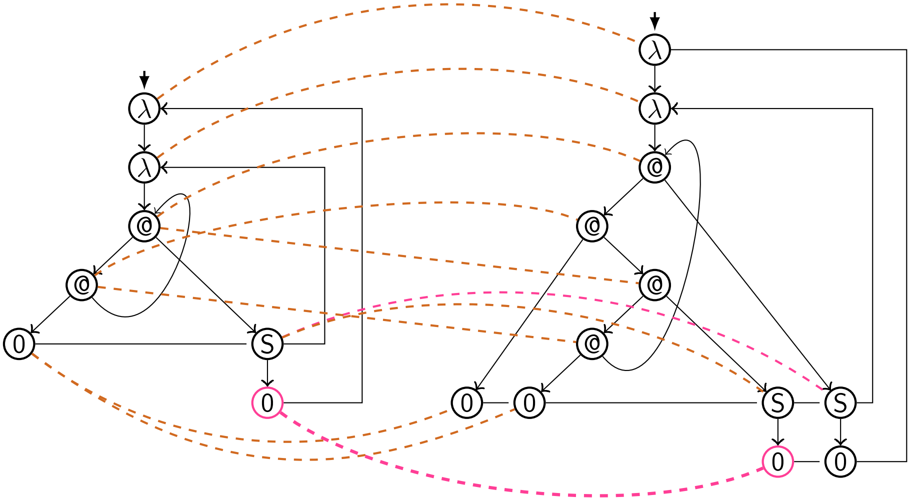


bisimulation check between λ-term-graphs



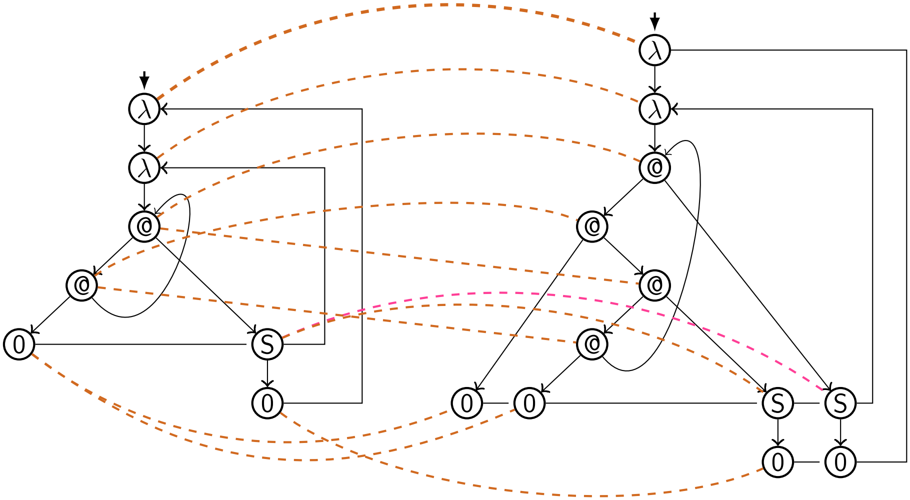
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



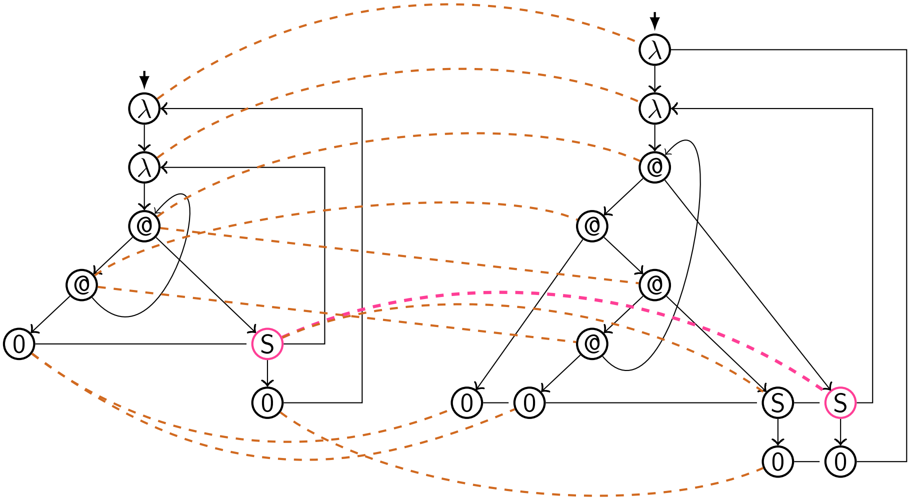
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



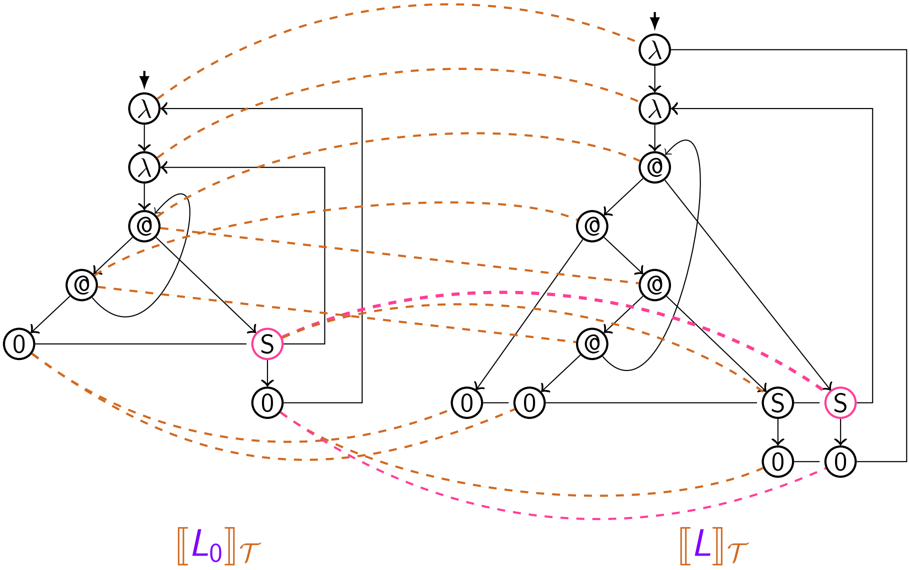
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

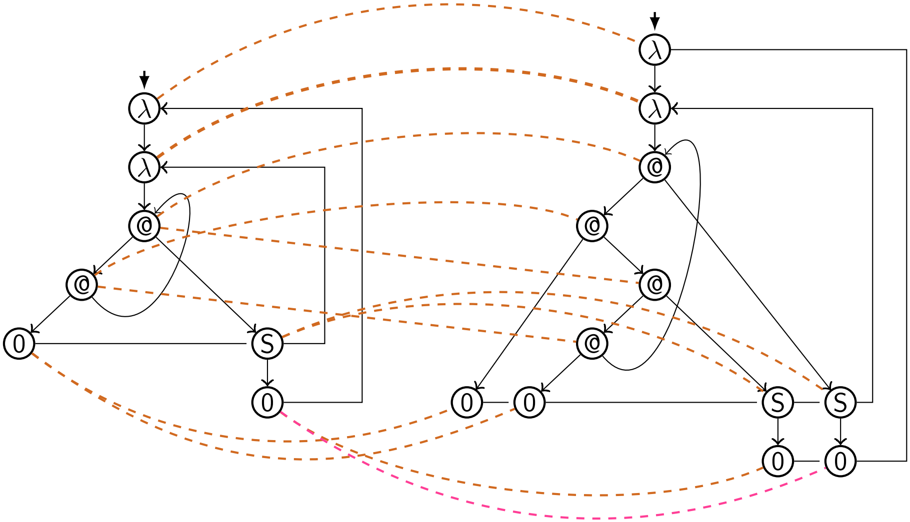


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

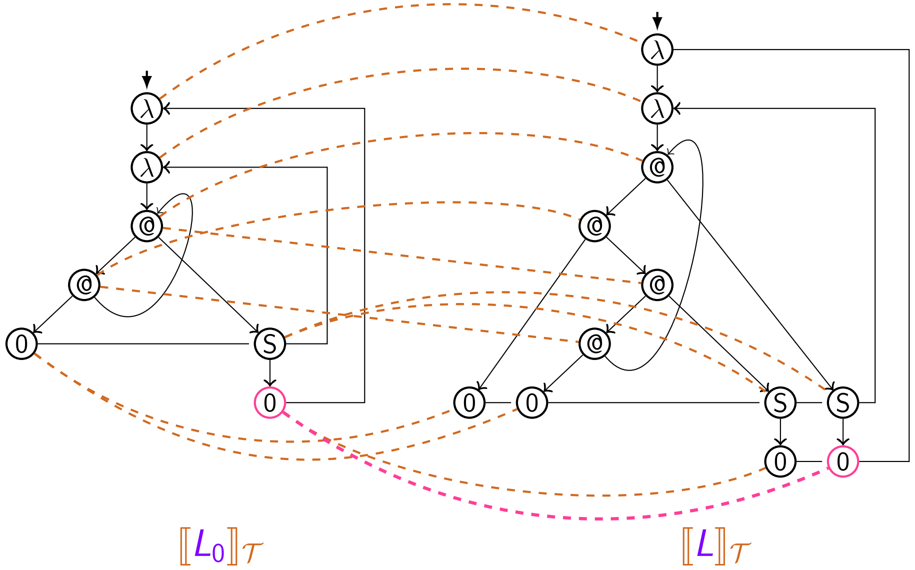


bisimulation check between λ-term-graphs

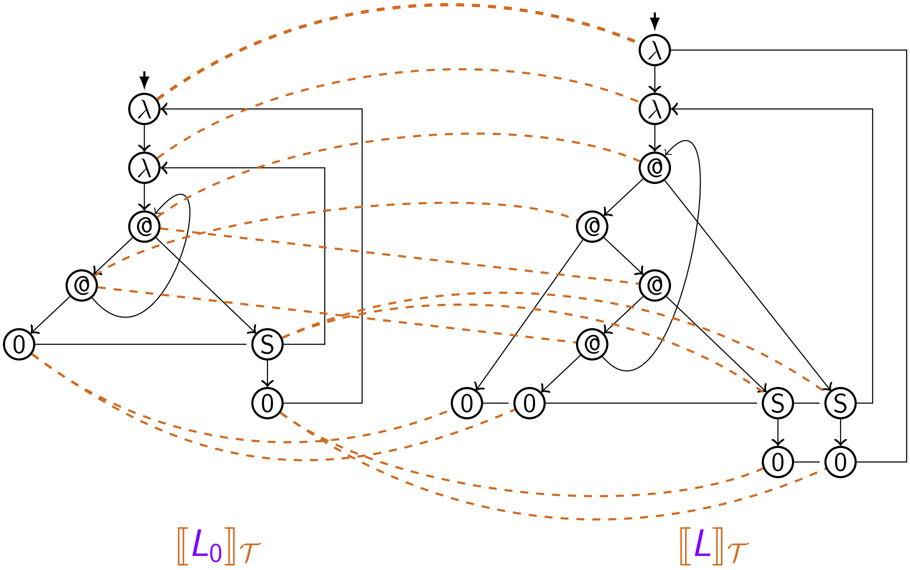


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

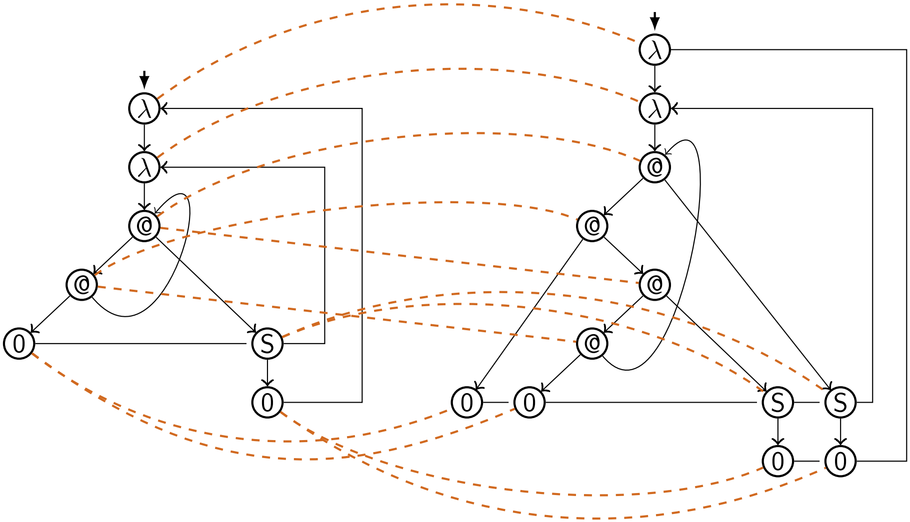
bisimulation check between λ-term-graphs



bisimulation check between λ-term-graphs

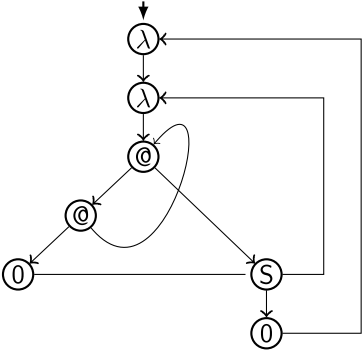


bisimulation between λ-term-graphs

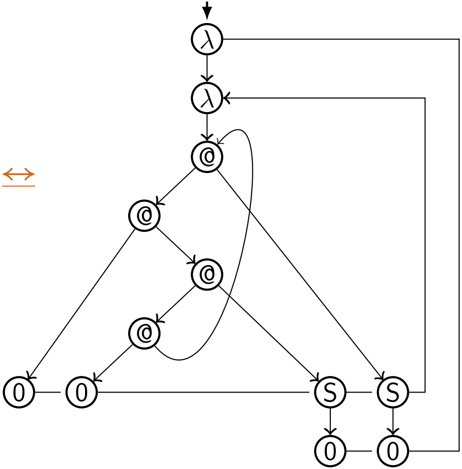


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimilarity between λ-term-graphs

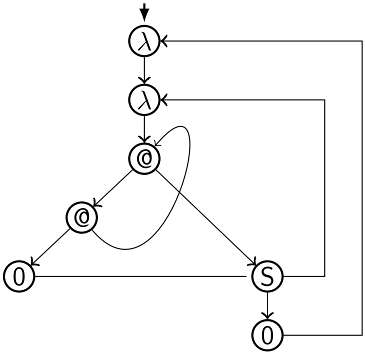


$\leftrightarrow$

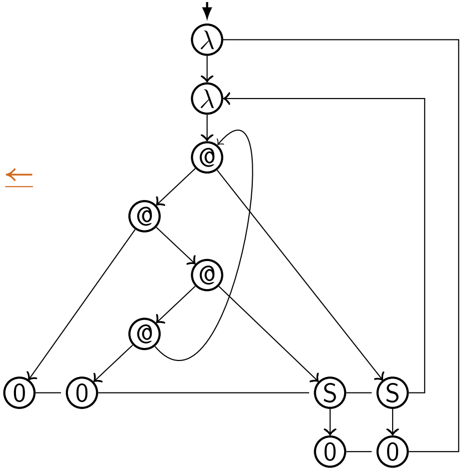


$[\![L_0]\!]_{\mathcal{T}} \quad \leftrightarrow \quad [\![L]\!]_{\mathcal{T}}$

functional bisimilarity and bisimulation collapse



$\leftharpoondown$



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

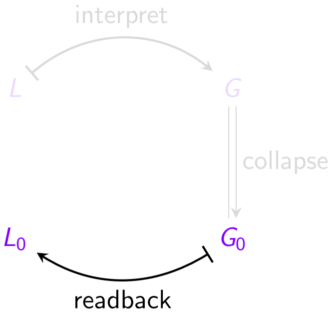
bisimulation collapse: property

Theorem

The class of eager-scope λ-term-graphs
is closed under functional bisimilarity $\stackrel{\to}{=}$.

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

readback



readback

defined with property:

$$\begin{tikzcd} L & G \arrow[l, mapsto, bend left, "\mathrm{rb}"] \end{tikzcd}$$ eager-scope

readback

defined with property:

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

readback

defined with property:

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

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 $\simeq$.

readback

defined with property:

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

Theorem

For all eager-scope λ-term-graphs G:

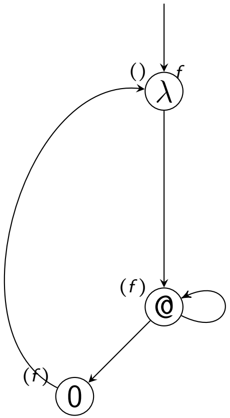
$$([\![\cdot]\!]_{\mathcal{T}} \circ \mathrm{rb})(G) \simeq G$$

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

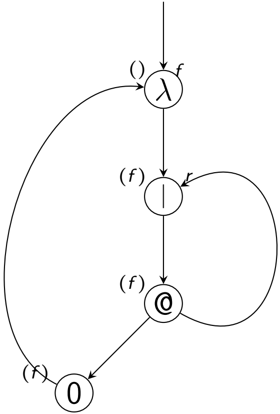
idea:

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

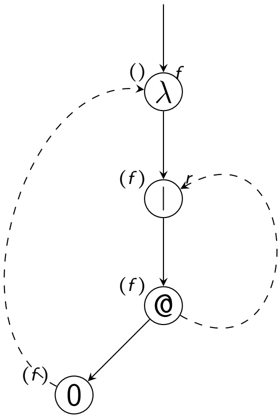
readback: example (fix)



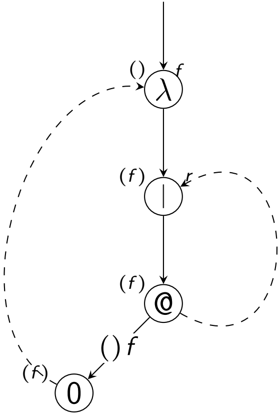
readback: example (fix)



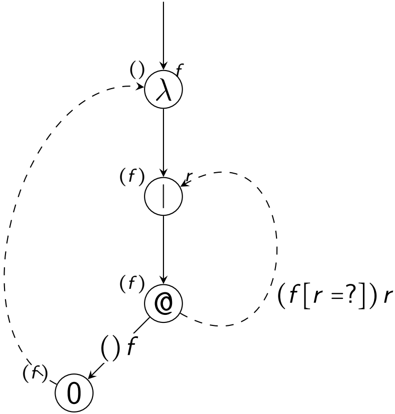
readback: example (fix)



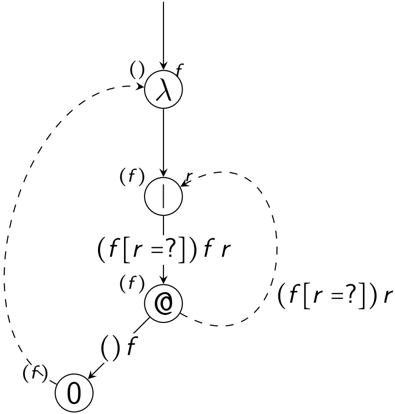
readback: example (fix)



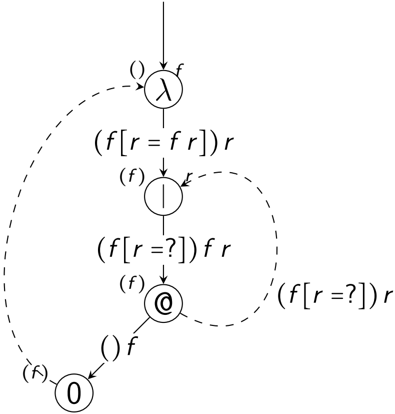
readback: example (fix)



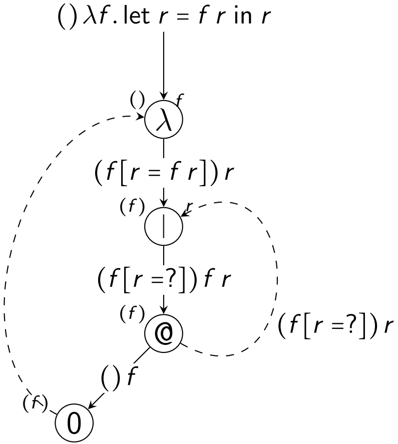
readback: example (fix)



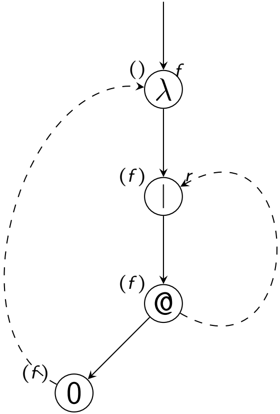
readback: example (fix)



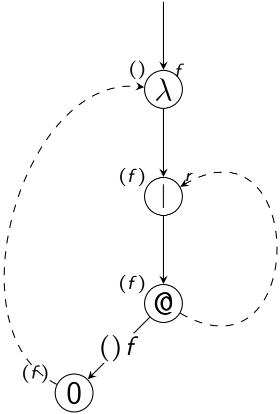
readback: example (fix)

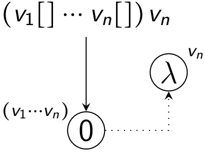


readback: example (fix)

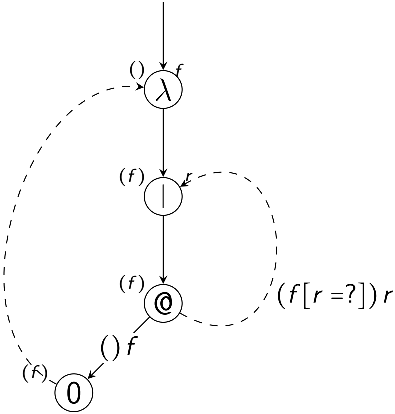


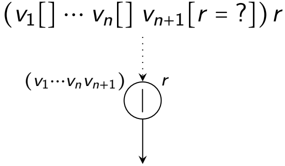
readback: example (fix)



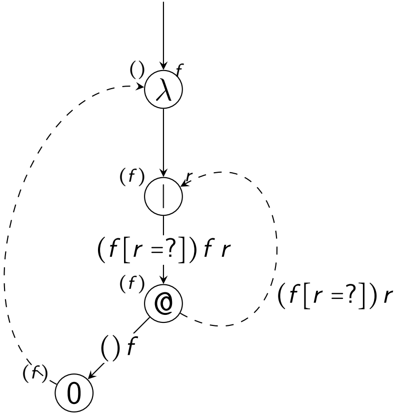


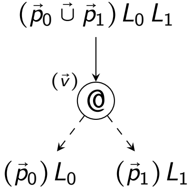
readback: example (fix)



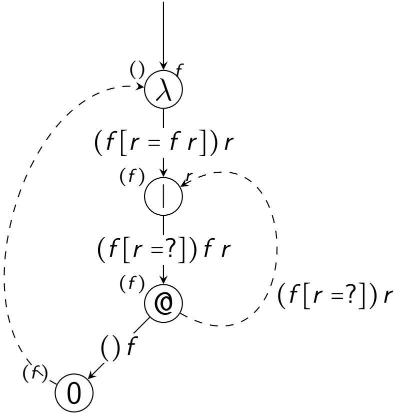


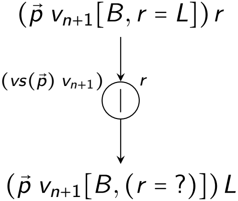
readback: example (fix)



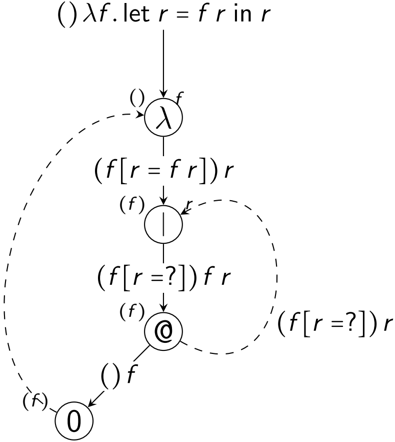


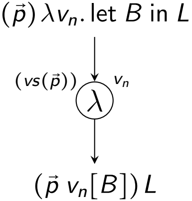
readback: example (fix)





readback: example (fix)



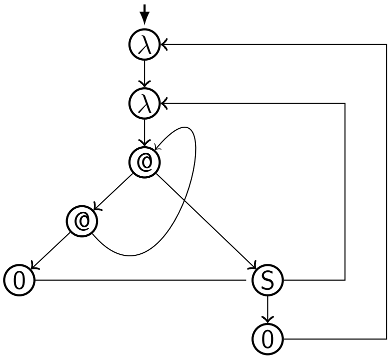


implementation

- tool maxsharing on `hackage.haskell.org`
  - uses Utrecht University Attribute Grammar Compiler (UUAGC)
- uses DFA-minimization instead of bisimulation collapse
  - reason: trace equivalence $=$ bisimilarity for deterministic LTSs
- examples and explanation
  - in accompanying report

λ-DFAs from λ-term-graphs

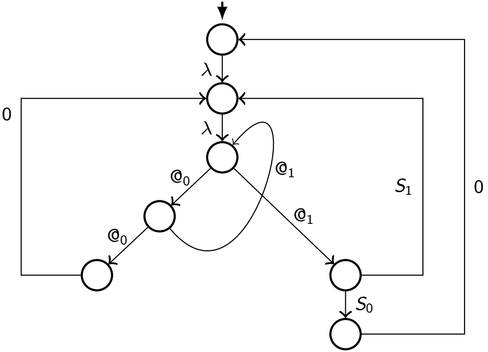
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

λ-DFAs from λ-term-graphs

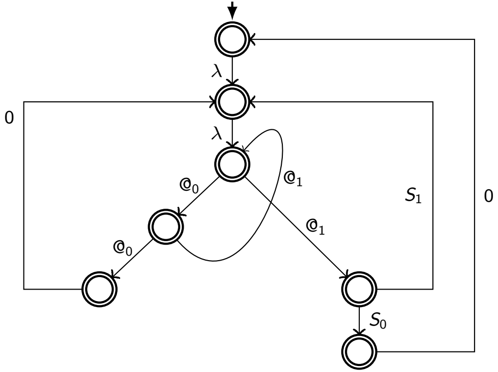
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



finite-state automaton (missing transitions)

λ-DFAs from λ-term-graphs

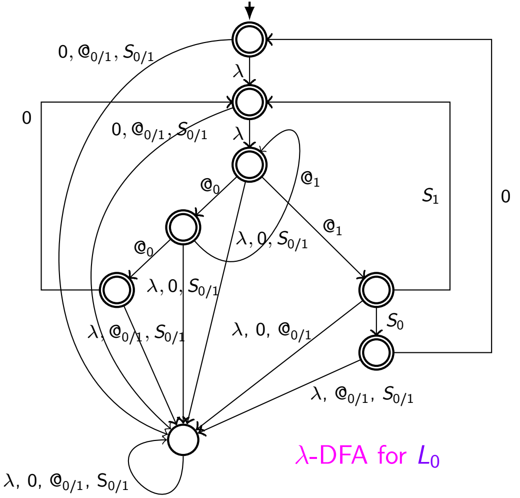
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



finite-state automaton (missing transitions)

λ-DFAs from λ-term-graphs

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

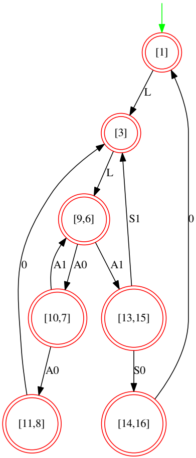


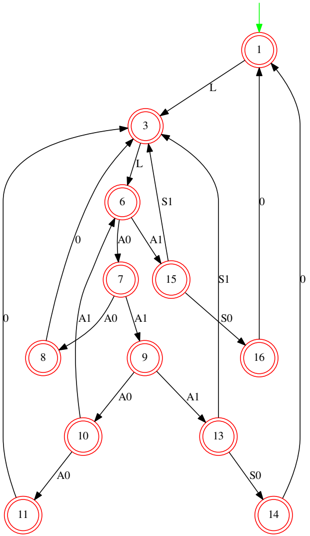
λ-DFA for L_0

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>
```

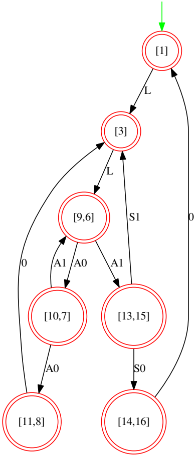
Demo: generated DFAs

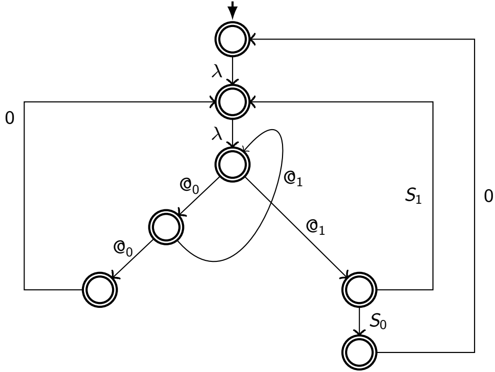




λ-DFA

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





λ-DFA for L_0 (without non-accepting transitions)

maximal sharing: complexity

$$\begin{tikzcd}[row sep=huge, column sep=huge] L \arrow[r, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & G \arrow[d, Rightarrow] \\ L_0 & G_0 \arrow[l, mapsto, bend left, "\mathrm{rb}"] \end{tikzcd}$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L
   as λ-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 $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.

maximal sharing: complexity

$$
\begin{tikzcd}
L \arrow[r, mapsto, bend left, "\llbracket\cdot\rrbracket_{\mathcal{T}}"] & G \arrow[d, "|\!\downarrow"] \\
L_0 & G_0 \arrow[l, mapsto, bend left, "\mathrm{rb}"]
\end{tikzcd}
$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L
   as λ-term-graph $G = \llbracket L \rrbracket_{\mathcal{T}}$

2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G into G_0

3. readback rb
   of f-o term graph G_0
   yielding $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.

maximal sharing: complexity

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

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
3. readback rb
   of f-o term graph G_0
   yielding $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.

maximal sharing: complexity

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

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-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 $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.

maximal sharing: complexity

$$\begin{tikzcd}
L \arrow[r, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] & G \arrow[d, Rrightarrow] \\
L_0 & G_0 \arrow[l, mapsto, bend left, "\mathrm{rb}"]
\end{tikzcd}$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-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 $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.
   - in time $O(|G| \log |G|) = O(n^2 \log n)$

maximal sharing: complexity

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

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-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 $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathrm{rb}(G_0)$.
   - in time $O(|G| \log |G|) = O(n^2 \log n)$

Theorem

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

unfolding equivalence: complexity

$$
\begin{tikzcd}
L_1 \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\textbf{?}"] & & G_1 \arrow[dd, leftrightarrow, "\textbf{?}"] \\
M & & \\
L_2 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}", "\textbf{?}"'] \arrow[rr, mapsto, bend right, "{[\![\cdot]\!]_{\mathcal{T}}}"'] & & G_2
\end{tikzcd}
$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2
   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$

2. check bisimilarity
   of λ-term-graphs G_1 and G_2

unfolding equivalence: complexity

$$\begin{tikzcd}
L_1 \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\mathbf{?}"] & & G_1 \arrow[dd, Leftrightarrow, "\mathbf{?}"] \\
M & & \\
L_2 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}", "\mathbf{?}"'] \arrow[rr, mapsto, bend right, "{[\![\cdot]\!]_{\mathcal{T}}}"'] & & G_2
\end{tikzcd}$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$
   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$
   - in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.
2. check bisimilarity
   of λ-term-graphs G_1 and G_2

unfolding equivalence: complexity

$$\begin{tikzcd}
L_1 \arrow[rr, mapsto, bend left, "[\![\cdot]\!]_{\mathcal{T}}"] \arrow[d, mapsto, "[\![\cdot]\!]_{\lambda^\infty}"', "\textbf{?}"] & & G_1 \arrow[dd, Leftrightarrow, "\textbf{?}"] \\
M & & \\
L_2 \arrow[u, mapsto, "[\![\cdot]\!]_{\lambda^\infty}", "\textbf{?}"'] \arrow[rr, mapsto, bend right, "[\![\cdot]\!]_{\mathcal{T}}"'] & & G_2
\end{tikzcd}$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$
   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$
   - in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.
2. check bisimilarity
   of λ-term-graphs G_1 and G_2
   - in time $O(|G_i|\,\alpha(|G_i|)) = O(n^2\,\alpha(n))$

unfolding equivalence: complexity

$$
\begin{tikzcd}
L_1 \arrow[rr, mapsto, bend left, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\mathbf{?}"] & & G_1 \arrow[dd, Leftrightarrow, "\mathbf{?}"] \\
M & & \\
L_2 \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"', "\mathbf{?}"] \arrow[rr, mapsto, bend right, "{[\![\cdot]\!]_{\mathcal{T}}}"'] & & G_2
\end{tikzcd}
$$

1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$
   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$
   - in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.
2. check bisimilarity
   of λ-term-graphs G_1 and G_2
   - in time $O(|G_i|\,\alpha(|G_i|)) = O(n^2\,\alpha(n))$

Theorem

Deciding whether $\boldsymbol{\lambda}_{\text{letrec}}$-terms L_1 and L_2 are unfolding-equivalent requires almost quadratic time $O(n^2\alpha(n))$ for $n = \max\{|L_1|, |L_2|\}$.

extensions

- support for full functional languages
  - work on a Core language with constructors, case statements
  - model these by enriching $\boldsymbol{\lambda}_{\text{letrec}}$ with function symbols
  - adapt our method to this $\boldsymbol{\lambda}_{\text{letrec}}$-extension
- prevent space leaks caused by disadvantageous sharing
  - identify 'sharing-unfit' positions/vertices
  - modify λ-term-graph interpretation
    in order to constrain the bisimulation collapse
- fully-lazy lambda-lifting
  - necessary analysis is similar
  - can be implemented as: $\mathrm{rb}_{LL} \circ [\![\cdot]\!]_{\mathcal{T}}$ (with modified readback rb_{LL})

applications

- maximal sharing at run-time
  - repeatedly compactify at run-time
  - possible directly on supercombinator graphs
  - can be coupled with garbage collection
- code improvement
  - detect code duplication
  - provide guidance on how to obtain a more compact form
- function equivalence
  - detecting unfolding equivalence provides partial solution
  - relevant for proof assistants, theorem provers,
    dependently-typed programming languages

resources

- tool maxsharing on `hackage.haskell.org`
- papers 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