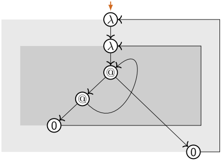
Modeling Terms in the λ-Calculus with letrec

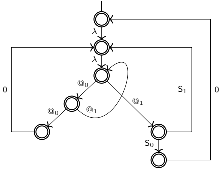
(by Term Graphs and Finite-State Automata)

Clemens Grabmayer

Gran Sasso Science Institute
L'Aquila, Italy



Computational Logic & Applications
Université de Versailles
July 1–2, 2019



Aim

Explain graph representations for (abstracted) functional programs (λ-terms with recursive bindings) that:

- are faithful to the unfolding semantics,
- facilitate use of standard methods for term graphs and DFAs,
- stay close to the term notation:

Aim

Explain graph representations for (abstracted) functional programs (λ-terms with recursive bindings) that:

- are faithful to the unfolding semantics,
- facilitate use of standard methods for term graphs and DFAs,
- stay close to the term notation:
  - use scope sharing,

Aim

Explain graph representations for (abstracted) functional programs (λ-terms with recursive bindings) that:

- are faithful to the unfolding semantics,
- facilitate use of standard methods for term graphs and DFAs,
- stay close to the term notation:
  - use extended-scope sharing,

Aim

Explain graph representations for (abstracted) functional programs (λ-terms with recursive bindings) that:

- are faithful to the unfolding semantics,
- facilitate use of standard methods for term graphs and DFAs,
- stay close to the term notation:
  - use extended-scope sharing,
  - not context sharing from optimal λ-reduction.

Aim

Explain graph representations for (abstracted) functional programs (λ-terms with recursive bindings) that:

- are faithful to the unfolding semantics,
- facilitate use of standard methods for term graphs and DFAs,
- stay close to the term notation:
  - use extended-scope sharing,
  - not context sharing from optimal λ-reduction.

Results from the interdisciplinary research project ROS (Realising Optimal Sharing, Utrecht University, 2009–2014/16), which brought together:

- term rewriters and logicians (philosophy department, UU)
  - Vincent van Oostrom, CG
- Haskell implementors (CS department, UU)
  - Doaitse Swierstra, Atze Dijkstra, Jan Rochel

Overview

- λ-calculus with letrec ($\boldsymbol{\lambda}_{\mathsf{letrec}}$)
- Expressibility of $\boldsymbol{\lambda}_{\mathsf{letrec}}$ via unfolding
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\mathsf{letrec}}$
- Nested term graphs

Overview

- λ-calculus with letrec ($\boldsymbol{\lambda}_{\text{letrec}}$)
- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Which infinite λ-terms are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms?
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - How can $\boldsymbol{\lambda}_{\text{letrec}}$-terms be compressed maximally while preserving their nested scope-structure?
- Nested term graphs
  - How to get a general framework for terms with nested scopes?

Overview

- λ-calculus with letrec ($\boldsymbol{\lambda}_{\text{letrec}}$)
- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Which infinite λ-terms are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms?
    - strongly regular λ^{∞}-terms
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - How can $\boldsymbol{\lambda}_{\text{letrec}}$-terms be compressed maximally while preserving their nested scope-structure?
- Nested term graphs
  - How to get a general framework for terms with nested scopes?

Overview

- λ-calculus with letrec ($\boldsymbol{\lambda}_{\text{letrec}}$)
- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Which infinite λ-terms are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms?
    - strongly regular λ^{∞}-terms
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - How can $\boldsymbol{\lambda}_{\text{letrec}}$-terms be compressed maximally while preserving their nested scope-structure?
    - formalization as (higher-/first-order) term graphs and DFAs
    - minimization / readback / efficiency / Haskell implementation
- Nested term graphs
  - How to get a general framework for terms with nested scopes?

Overview

- λ-calculus with letrec ($\boldsymbol{\lambda}_{\text{letrec}}$)
- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Which infinite λ-terms are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms?
    - strongly regular λ^{∞}-terms
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - How can $\boldsymbol{\lambda}_{\text{letrec}}$-terms be compressed maximally while preserving their nested scope-structure?
    - formalization as (higher-/first-order) term graphs and DFAs
    - minimization / readback / efficiency / Haskell implementation
- Nested term graphs
  - How to get a general framework for terms with nested scopes?
    - term graphs with inbuilt nesting

The λ-Calculus with letrec

$$(\lambda f.\,\mathsf{letrec}\; r = f\, r\; \mathsf{in}\; r)\, M$$

The λ-Calculus with letrec

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

The λ-Calculus

Terms in the λ-calculus (over set Var of variables):

(term)	M	$::=$	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)

The λ-Calculus

Terms in the λ-calculus (over set Var of variables):

(term)	M	$::=$	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)

Rewriting in $\boldsymbol{\lambda}$:

$$(\lambda x.\, M)\, N \;\to_\beta\; M[x := N] \qquad (\beta\text{-reduction step})$$

The λ-Calculus

Terms in the λ-calculus (over set Var of variables):

(term)	M	::=	x	(variable, $x \in Var$)
		|	$M_1\, M_2$	(application)
		|	$\lambda x.\, M$	(abstraction)

Rewriting in $\boldsymbol{\lambda}$:

$$(\lambda x.\, M)\, N \to_\beta M[x := N] \qquad (\beta\text{-reduction step})$$

$$\lambda x.\, M \to_\alpha \lambda y.\, M[x := y] \qquad (\alpha\text{-conversion step})$$

The λ-Calculus with letrec

Terms in the λ-calculus ($\boldsymbol{\lambda}_{\mathsf{letrec}}$) with letrec (over set Var of variables):

(term)	M	$::=$	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)
		$\mid$	letrec B in M	(letrec)

Rewriting in $\boldsymbol{\lambda}$:

$(\lambda x.\, M)\, N$	$\to_\beta$	$M[x := N]$	(β-reduction step)
$\lambda x.\, M$	$\to_\alpha$	$\lambda y.\, M[x := y]$	(α-conversion step)

The λ-Calculus with letrec

Terms in the λ-calculus ($\boldsymbol{\lambda}_{\text{letrec}}$) with letrec (over set Var of variables):

(term)	M	::=	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)
		$\mid$	letrec B in M	(letrec)
(binding group)	B	::=	$f_1 = M_1, \ldots, f_n = M_n$	(bindings, $f_1, \ldots, f_n \in Var$)

Rewriting in $\boldsymbol{\lambda}$:

$(\lambda x.\, M)\, N$	$\rightarrow_\beta$	$M[x := N]$	(β-reduction step)
$\lambda x.\, M$	$\rightarrow_\alpha$	$\lambda y.\, M[x := y]$	(α-conversion step)

The λ-Calculus with letrec

Terms in the λ-calculus ($\boldsymbol{\lambda}_{\mathsf{letrec}}$) with letrec (over set Var of variables):

(term)	M	$::=$	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)
		$\mid$	$\mathsf{let}\ B\ \mathsf{in}\ M$	(letrec)
(binding group)	B	$::=$	$f_1 = M_1, \ldots, f_n = M_n$	(bindings, $f_1, \ldots, f_n \in Var$)

Notation: letrec = let (like in Haskell).

Rewriting in $\boldsymbol{\lambda}$:

$$(\lambda x.\, M)\, N \;\to_\beta\; M[x := N] \qquad (\beta\text{-reduction step})$$

$$\lambda x.\, M \;\to_\alpha\; \lambda y.\, M[x := y] \qquad (\alpha\text{-conversion step})$$

The λ-Calculus with letrec

Terms in the λ-calculus ($\boldsymbol{\lambda}_{\text{letrec}}$) with letrec (over set Var of variables):

(term)	M	::=	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)
		$\mid$	let B in M	(letrec)
(binding group)	B	::=	$f_1 = M_1, \ldots, f_n = M_n$	(bindings, $f_1, \ldots, f_n \in Var$)

Notation: letrec = let (like in Haskell).

Rewriting in $\boldsymbol{\lambda}$:

$$(\lambda x.\, M)\, N \;\to_\beta\; M[x := N] \qquad (\beta\text{-reduction step})$$

$$\lambda x.\, M \;\to_\alpha\; \lambda y.\, M[x := y] \qquad (\alpha\text{-conversion step})$$

The λ-Calculus with letrec

Terms in the λ-calculus ($\boldsymbol{\lambda}_{\text{letrec}}$) with letrec (over set Var of variables):

(term)	M	$::=$	x	(variable, $x \in Var$)
		$\mid$	$M_1\, M_2$	(application)
		$\mid$	$\lambda x.\, M$	(abstraction)
		$\mid$	let B in M	(letrec)
(binding group)	B	$::=$	$f_1 = M_1, \ldots, f_n = M_n$	(bindings, $f_1, \ldots, f_n \in Var$)

Notation: letrec = let (like in Haskell).

Rewriting in $\boldsymbol{\lambda}_{\text{letrec}}$:

$(\lambda x.\, M)\, N$	$\to_\beta$	$M[x := N]$	(β-reduction step)
$\lambda x.\, M$	$\to_\alpha$	$\lambda y.\, M[x := y]$	(α-conversion step)
let B in M	$\to_\triangledown$	$\ldots$	(unfolding steps)

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$

For $\quad \mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \quad$ we find:

$$\mathsf{fix}$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$ (infinite unfolding)

For $\quad \mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \quad$ we find:

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

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$\begin{aligned} \mathsf{fix} &= \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r \\ &\to_{\triangledown} \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ f\, r \end{aligned}$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\; r = f\,r\;\mathsf{in}\; r$ we find:

$$
\begin{aligned}
\mathsf{fix} \;=\;& \lambda f.\,\mathsf{let}\; r = f\,r\;\mathsf{in}\; r \\
\to_{\triangledown}\;& \lambda f.\,\mathsf{let}\; r = f\,r\;\mathsf{in}\; f\,r \\
\to_{\triangledown}\;& \lambda f.\,(\mathsf{let}\; r = f\,r\;\mathsf{in}\; f)\,(\mathsf{let}\; r = f\,r\;\mathsf{in}\; r)
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix} \ = \ & \ \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \\
\to_{\triangledown} \ & \ \lambda f.\ \mathsf{let}\ r = f\,r\ \mathsf{in}\ f\,r \\
\to_{\triangledown} \ & \ \lambda f.\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ f)\,(\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r) \\
\to_{\triangledown} \ & \ \lambda f.\ f\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix} \; &= \; \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \\
&\to_{\triangledown} \; \lambda f.\,\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ f\,r} \\
&\to_{\triangledown} \; \lambda f.\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ f)\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown} \; \lambda f.\,f\,(\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ r})
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix} \;=\;& \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r \\
\to_{\triangledown}\;& \lambda f.\, \boxed{\mathsf{let}\ r = f\, r\ \mathsf{in}\ f\, r} \\
\to_{\triangledown}\;& \lambda f.\, (\mathsf{let}\ r = f\, r\ \mathsf{in}\ f)\,(\mathsf{let}\ r = f\, r\ \mathsf{in}\ r) \\
\to_{\triangledown}\;& \lambda f.\, f\,(\boxed{\mathsf{let}\ r = f\, r\ \mathsf{in}\ r}) \\
\twoheadrightarrow_{\triangledown}\;& \lambda f.\, f\,(f\,(\boxed{\mathsf{let}\ r = f\, r\ \mathsf{in}\ r}))
\end{aligned}
$$

Fixed-point combinator in λ_{letrec} (infinite unfolding)

For $\text{fix} := \lambda f.\,\text{let } r = f\,r \text{ in } r$ we find:

$$
\begin{aligned}
\text{fix} \;=\; & \lambda f.\,\text{let } r = f\,r \text{ in } r \\
\to_{\triangledown}\; & \lambda f.\,\boxed{\text{let } r = f\,r \text{ in } f\,r} \\
\to_{\triangledown}\; & \lambda f.\,(\text{let } r = f\,r \text{ in } f)\,(\text{let } r = f\,r \text{ in } r) \\
\to_{\triangledown}\; & \lambda f.\,f\,(\boxed{\text{let } r = f\,r \text{ in } r}) \\
\twoheadrightarrow_{\triangledown}\; & \lambda f.\,f\,(f\,(\boxed{\text{let } r = f\,r \text{ in } r})) \\
\twoheadrightarrow_{\triangledown}\; & \lambda f.\,f\,(f\,(\ldots f\,(\boxed{\text{let } r = f\,r \text{ in } r})))
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix} \;=\; & \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \\
\to_{\triangledown}\; & \lambda f.\,\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ f\,r} \\
\to_{\triangledown}\; & \lambda f.\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ f)\,(\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r) \\
\to_{\triangledown}\; & \lambda f.\,f\,(\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ r}) \\
\twoheadrightarrow_{\triangledown}\; & \lambda f.\,f\,(f\,(\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ r})) \\
\twoheadrightarrow_{\triangledown}\; & \lambda f.\,f\,(f\,(\ldots f\,(\boxed{\mathsf{let}\ r = f\,r\ \mathsf{in}\ r}))) \\
\twoheadrightarrow\!\!\!\!\rightarrow_{\triangledown}\; & \lambda f.\,f\,(f\,(\ldots f\,(\ldots)))
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$ (infinite unfolding)

For $\text{fix} := \lambda f.\,\text{let } r = f\,r \text{ in } r$ we find:

$$
\begin{aligned}
\text{fix} \;=\;& \lambda f.\,\text{let } r = f\,r \text{ in } r\\
\to_{\triangledown}\;& \lambda f.\,\text{let } r = f\,r \text{ in } f\,r\\
\to_{\triangledown}\;& \lambda f.\,(\text{let } r = f\,r \text{ in } f)\,(\text{let } r = f\,r \text{ in } r)\\
\to_{\triangledown}\;& \lambda f.\,f\,(\text{let } r = f\,r \text{ in } r)\\
\twoheadrightarrow_{\triangledown}\;& \lambda f.\,f\,(f\,(\text{let } r = f\,r \text{ in } r))\\
\twoheadrightarrow_{\triangledown}\;& \lambda f.\,f\,(f\,(\ldots f\,(\text{let } r = f\,r \text{ in } r)))\\
\twoheadrightarrow\!\!\!\rightarrow_{\triangledown}\;& \lambda f.\,f\,(f\,(\ldots f\,(\ldots)))
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$ (infinite unfolding)

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix} \;&=\; \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r \\
&\to_{\triangledown}\; \lambda f.\ \mathsf{let}\ r = f\,r\ \mathsf{in}\ f\,r \\
&\to_{\triangledown}\; \lambda f.\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ f)\,(\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown}\; \lambda f.\ f\,(\mathsf{let}\ r = f\,r\ \mathsf{in}\ r) \\
&\twoheadrightarrow_{\triangledown}\; \lambda f.\,f\,(f\,(\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r\,)) \\
&\twoheadrightarrow_{\triangledown}\; \lambda f.\,f\,(f\,(\ldots f\,(\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r\,))) \\
&\twoheadrightarrow\!\!\!\!\rightarrow_{\triangledown}\; \lambda f.\,f\,(f\,(\ldots f\,(\ldots))) \\
&=\; [\![\mathsf{fix}]\!]_{\lambda^\infty}
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\mathsf{letrec}}$

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$\mathsf{fix}\, M$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$\mathsf{fix}\, M$$

$$M\, (\mathsf{fix}\, M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

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

$$M\,(\mathsf{fix}\,M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\, M &= (\lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r)\, M \\
&\to_{\beta} \mathsf{let}\ r = M\, r\ \mathsf{in}\ r
\end{aligned}
$$

$$M\, (\mathsf{fix}\, M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\, M \quad &= \quad (\lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r)\, M \\
&\to_{\beta} \quad \mathsf{let}\ r = M\, r\ \mathsf{in}\ r \\
&\to_{\triangledown} \quad \mathsf{let}\ r = M\, r\ \mathsf{in}\ M\, r
\end{aligned}
$$

$$M\, (\mathsf{fix}\, M)$$

Fixed-point combinator in λ_{letrec}

For $\quad \mathsf{fix} := \lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r \quad$ we find:

$$
\begin{aligned}
\mathsf{fix}\, M \;&=\; (\lambda f.\, \mathsf{let}\ r = f\, r\ \mathsf{in}\ r)\, M \\
&\to_{\beta}\; \mathsf{let}\ r = M\, r\ \mathsf{in}\ r \\
&\to_{\triangledown}\; \mathsf{let}\ r = M\, r\ \mathsf{in}\ M\, r \\
&\to_{\triangledown}\; (\mathsf{let}\ r = M\, r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\, r\ \mathsf{in}\ r)
\end{aligned}
$$

$$M\,(\mathsf{fix}\, M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\,M \;&=\; (\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M \\
&\to_{\beta}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ r \\
&\to_{\triangledown}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ M\,r \\
&\to_{\triangledown}\; (\mathsf{let}\ r = M\,r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown}\; M\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r)
\end{aligned}
$$

$$M\,(\mathsf{fix}\,M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\,M \;&=\; (\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M \\
&\to_{\beta}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ r \\
&\to_{\triangledown}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ M\,r \\
&\to_{\triangledown}\; (\mathsf{let}\ r = M\,r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown}\; M\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\leftarrow_{\beta}\; M\,((\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M) \\
&\;\; M\,(\mathsf{fix}\,M)
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\,M \quad &= \quad (\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M \\
&\to_{\beta} \quad \mathsf{let}\ r = M\,r\ \mathsf{in}\ r \\
&\to_{\triangledown} \quad \mathsf{let}\ r = M\,r\ \mathsf{in}\ M\,r \\
&\to_{\triangledown} \quad (\mathsf{let}\ r = M\,r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown} \quad M\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\leftarrow_{\beta} \quad M\,((\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M) \\
&= \quad M\,(\mathsf{fix}\,M)
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\text{fix} := \lambda f.\, \text{let } r = f\, r \text{ in } r$ we find:

$$
\begin{aligned}
\text{fix}\, M &= && (\lambda f.\, \text{let } r = f\, r \text{ in } r)\, M \\
&\to_{\beta} && \text{let } r = M\, r \text{ in } r \\
&\to_{\triangledown} && \text{let } r = M\, r \text{ in } M\, r \\
&\to_{\triangledown} && (\text{let } r = M\, r \text{ in } M)\, (\text{let } r = M\, r \text{ in } r) \\
&\to_{\triangledown} && M\, (\text{let } r = M\, r \text{ in } r) \\
&\leftarrow_{\beta} && M\, ((\lambda f.\, \text{let } r = f\, r \text{ in } r)\, M) \\
&= && M\, (\text{fix}\, M)
\end{aligned}
$$

$$\text{fix}\, M \leftrightarrow^{*}_{\beta\triangledown} M\, (\text{fix}\, M)$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

For $\;\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r\;$ we find:

$$
\begin{aligned}
\mathsf{fix}\,M \;&=\; && (\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M \\
&\to_{\beta} && \mathsf{let}\ r = M\,r\ \mathsf{in}\ r \\
&\to_{\triangledown} && \mathsf{let}\ r = M\,r\ \mathsf{in}\ M\,r \\
&\to_{\triangledown} && (\mathsf{let}\ r = M\,r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown} && M\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\leftarrow_{\beta} && M\,((\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M) \\
&= && M\,(\mathsf{fix}\,M)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{fix}\,M \;&\leftrightarrow^{*}_{\beta\triangledown} && M\,(\mathsf{fix}\,M) \\
&\leftrightarrow^{*}_{\beta\triangledown} && M\,(M\,(\ldots(M\,(\mathsf{fix}\,M))\ldots))
\end{aligned}
$$

Fixed-point combinator in $\boldsymbol{\lambda}_{\text{letrec}}$

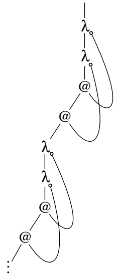
For $\mathsf{fix} := \lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r$ we find:

$$
\begin{aligned}
\mathsf{fix}\,M \;&=\; (\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M \\
&\to_{\beta}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ r \\
&\to_{\triangledown}\; \mathsf{let}\ r = M\,r\ \mathsf{in}\ M\,r \\
&\to_{\triangledown}\; (\mathsf{let}\ r = M\,r\ \mathsf{in}\ M)\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\to_{\triangledown}\; M\,(\mathsf{let}\ r = M\,r\ \mathsf{in}\ r) \\
&\leftarrow_{\beta}\; M\,((\lambda f.\,\mathsf{let}\ r = f\,r\ \mathsf{in}\ r)\,M) \\
&=\; M\,(\mathsf{fix}\,M)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{fix}\,M \;&\leftrightarrow^{*}_{\beta\triangledown}\; M\,(\mathsf{fix}\,M) \\
&\leftrightarrow^{*}_{\beta\triangledown}\; M\,(M\,(\ldots(M\,(\mathsf{fix}\,M))\ldots)) \\
&(\to^{+}_{\beta\triangledown}\cdot\leftarrow_{\beta})^{\omega}\; M\,(M\,(\ldots(M\,(\ldots))\ldots))\ .
\end{aligned}
$$

Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding

(joint work with Jan Rochel)



Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\text{letrec}}$?

Example

let $f = \lambda x.\, \lambda y.\, f\, y\, x$ in f

Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\text{letrec}}$?

Example

let $f = \lambda x.\, \lambda y.\, f\, y\, x$ in f

Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\text{letrec}}$?

Example

let $f = \lambda x.\, \lambda y.\, f\, y\, x$ in f $\quad \twoheadrightarrow_{\triangledown} \quad$ $\lambda xy.\, (\lambda xy.\, (\lambda xy.\, (\ldots)\, y\, x)\, y\, x)\, y\, x$

$\twoheadrightarrow_{\triangledown}$

Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\mathsf{letrec}}$?

Example

let $f = \lambda x.\, \lambda y.\, f\, y\, x$ in f $\quad \twoheadrightarrow_{\triangledown} \quad \lambda xy.\, (\lambda xy.\, (\lambda xy.\, (\ldots)\, y\, x)\, y\, x)\, y\, x$

$\twoheadrightarrow_{\triangledown} \not\to_{\beta}$

Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\text{letrec}}$?

Example

let $f = \lambda x.\,\lambda y.\,f\,y\,x$ in f $\quad \twoheadrightarrow_{\triangledown} \quad \lambda xy.\,(\lambda xy.\,(\lambda xy.\,(\ldots)\,y\,x)\,y\,x)\,y\,x$

$\twoheadrightarrow_{\triangledown}$ $\multimap_{\beta}$ $\quad \multimap_{\beta}$

Which infinite λ-terms are expressible finitely in $\boldsymbol{\lambda}_{\text{letrec}}$?

Example

let $f = \lambda x.\, \lambda y.\, f\, y\, x$ in f $\twoheadrightarrow_{\triangledown}$ $\lambda xy.\, (\lambda xy.\, (\lambda xy.\, (\ldots)\, y\, x)\, y\, x)\, y\, x$

$\lambda x.\, \lambda y.$ let $f = f$ in f

$\twoheadrightarrow_{\triangledown}$ $\multimap_{\beta}$ $\multimap_{\beta}$ $\twoheadleftarrow_{\triangledown}$

$\boldsymbol{\lambda}_{\text{letrec}}$-Expressible 'regular' λ^{∞}-term

let $f = \lambda xy.\, f\, y\, x$ in f

term graph syntax tree

$\boldsymbol{\lambda}_{\text{letrec}}$-Expressible 'regular' λ^∞-term

let $f = \lambda xy.\, f\, y\, x$ in f

term graph syntax tree bindings

$\boldsymbol{\lambda}_{\text{letrec}}$-Expressible 'regular' λ^{∞}-term

let $f = \lambda x y.\, f\, y\, x$ in f

term graph syntax tree bindings

finite

entanglement

λ_{letrec}-Expressible 'regular' λ^{∞}-term

let $f = \lambda xy.\, f\, y\, x$ in f

term graph syntax tree bindings scopes

finite
entanglement

$\boldsymbol{\lambda}_{\text{letrec}}$-Expressible 'regular' λ^{∞}-term

let $f = \lambda x y.\, f\, y\, x$ in f

term graph syntax tree bindings scopes scope^{+}s

finite entanglement

λ_{letrec}-Expressible 'regular' λ^∞-term

let f = $\lambda xy.\ f\,y\,x$ in f

term graph	syntax tree	bindings	scopes	scope$^+$s
		finite entanglement		finite nesting

Not $\boldsymbol{\lambda}_{\text{letrec}}$-expressible 'regular' $\boldsymbol{\lambda}^{\infty}$-term

syntax tree

Not λ_{letrec}-expressible 'regular' λ^{∞}-term

syntax tree bindings

Not λ_{letrec}-expressible 'regular' λ^{∞}-term

syntax tree bindings

infinitely entangled

Not $\boldsymbol{\lambda}_{\text{letrec}}$-expressible 'regular' $\boldsymbol{\lambda}^{\infty}$-term

syntax tree bindings scopes

infinitely entangled

Not λ_{letrec}-expressible 'regular' λ^{∞}-term

syntax tree	bindings	scopes	scope^{+}s
	infinitely entangled		infinite nesting

Deconstructing/observing λ^∞-terms

$$()\,\lambda x.\,\lambda y.\,x\,x\,y$$

Deconstructing/observing λ^{∞}-terms

$$( )\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda}$$
$$(x)\,\lambda y.\,x\,x\,y$$

$$(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 \;\to_{\lambda}\; (x_1 \ldots x_{n+1})\,M_0$$

Deconstructing/observing λ^{∞}-terms

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
&(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
&(xy)\,x\,x\,y
\end{aligned}
$$

$$(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 \;\to_{\lambda}\; (x_1 \ldots x_{n+1})\,M_0$$

Deconstructing/observing λ^∞-terms

$$\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_\lambda \\
&(x)\,\lambda y.\,x\,x\,y \rightarrow_\lambda \\
&(xy)\,x\,x\,y \rightarrow_{@_0} \\
&(xy)\,x\,x
\end{aligned}$$

$$\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 \;&\rightarrow_{@_i}\; (x_1 \ldots x_n)\,M_i \qquad (i \in \{0,1\}) \\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 \;&\rightarrow_\lambda\; (x_1 \ldots x_{n+1})\,M_0
\end{aligned}$$

Deconstructing/observing λ^∞-terms

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda\\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x
\end{aligned}
$$

$$
\begin{array}{rcll}
(x_1 \ldots x_n)\,M_0\,M_1 & \to_{@_i} & (x_1 \ldots x_n)\,M_i & (i \in \{0,1\})\\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 & \to_\lambda & (x_1 \ldots x_{n+1})\,M_0 & \\
(x_1 \ldots x_n x_{n+1})\,M_0 & \to_{\mathsf{S}} & (x_1 \ldots x_n)\,M_0 & (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{array}
$$

Deconstructing/observing λ^∞-terms

$$
\begin{aligned}
&()\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda \\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda \\
&(xy)\,x\,x\,y \to_{@_0} \\
&(xy)\,x\,x \to_{\mathsf{S}} \\
&(x)\,x\,x \to_{@_0} \\
&(x)\,x
\end{aligned}
$$

$$
\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 &\to_{@_i} (x_1 \ldots x_n)\,M_i && (i \in \{0,1\}) \\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 &\to_\lambda (x_1 \ldots x_{n+1})\,M_0 && \\
(x_1 \ldots x_n x_{n+1})\,M_0 &\to_{\mathsf{S}} (x_1 \ldots x_n)\,M_0 && (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{aligned}
$$

Deconstructing/observing λ^{∞}-terms

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(x)\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x \to_{@_0}\\
&(x)\,x
\end{aligned}
$$

$\to_{\mathsf{reg}^+}$-generated subterms of $\lambda x.\,\lambda y.\,x\,x\,y$ w.r.t. rewrite relation $\to_{\mathsf{reg}^+}$:

$$
\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 &\to_{@_i} (x_1 \ldots x_n)\,M_i && (i \in \{0,1\})\\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 &\to_{\lambda} (x_1 \ldots x_{n+1})\,M_0 &&\\
(x_1 \ldots x_n x_{n+1})\,M_0 &\to_{\mathsf{S}} (x_1 \ldots x_n)\,M_0 && (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{aligned}
$$

Deconstructing/observing λ^∞-terms

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda \\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda \\
&(xy)\,x\,x\,y \to_{@_0} \\
&(xy)\,x\,x \to_{\mathsf{S}} \\
&(x)\,x\,x \to_{@_0} \\
&(x)\,x
\end{aligned}
$$

$\to_{\mathsf{reg}^+}$-generated subterms of $\lambda x.\,\lambda y.\,x\,x\,y$ w.r.t. rewrite relation $\to_{\mathsf{reg}^+}$:

$$
\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 &\to_{@_i} (x_1 \ldots x_n)\,M_i && (i \in \{0,1\}) \\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 &\to_\lambda (x_1 \ldots x_{n+1})\,M_0 && \\
(x_1 \ldots x_n x_{n+1})\,M_0 &\to_{\mathsf{S}} (x_1 \ldots x_n)\,M_0 && (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{aligned}
$$

formalized as a CRS, e.g. rule:

$$
\mathsf{pre}_n([x_1 \ldots x_n]\mathsf{abs}([x_{n+1}]Z(\vec{x}))) \to \mathsf{pre}_{n+1}([x_1 \ldots x_{n+1}]Z(\vec{x}))
$$

Deconstructing/observing λ^∞-terms

$$
\begin{aligned}
&()\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda\\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda\\
&(xy)\,x\,x\,y \to_{@_1}\\
&(xy)\,y
\end{aligned}
$$

$$
\begin{aligned}
&()\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda\\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x \to_{@_0}\\
&(x)\,x
\end{aligned}
$$

$$
\begin{aligned}
&()\,\lambda x.\,\lambda y.\,x\,x\,y \to_\lambda\\
&(x)\,\lambda y.\,x\,x\,y \to_\lambda\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x \to_{@_1}\\
&(x)\,x
\end{aligned}
$$

$\to_{\text{reg}^+}$-generated subterms of $\lambda x.\,\lambda y.\,x\,x\,y$ w.r.t. rewrite relation $\to_{\text{reg}^+}$:

$$
\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 &\to_{@_i} (x_1 \ldots x_n)\,M_i && (i \in \{0,1\})\\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 &\to_\lambda (x_1 \ldots x_{n+1})\,M_0 &&\\
(x_1 \ldots x_n x_{n+1})\,M_0 &\to_{\mathsf{S}} (x_1 \ldots x_n)\,M_0 && (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{aligned}
$$

formalized as a CRS, e.g. rule:

$$\mathsf{pre}_n([x_1 \ldots x_n]\mathsf{abs}([x_{n+1}]Z(\vec{x}))) \to \mathsf{pre}_{n+1}([x_1 \ldots x_{n+1}]Z(\vec{x}))$$

Generated subterms

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(x)\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(xy)\,x\,x\,y \to_{@_1}\\
&(xy)\,y
\end{aligned}
$$

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(x)\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x \to_{@_0}\\
&(x)\,x
\end{aligned}
$$

$$
\begin{aligned}
&(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(x)\,\lambda y.\,x\,x\,y \to_{\lambda}\\
&(xy)\,x\,x\,y \to_{@_0}\\
&(xy)\,x\,x \to_{\mathsf{S}}\\
&(x)\,x\,x \to_{@_1}\\
&(x)\,x
\end{aligned}
$$

$$
\begin{aligned}
(x_1 \ldots x_n)\,M_0\,M_1 &\to_{@_i} (x_1 \ldots x_n)\,M_i && (i \in \{0,1\})\\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 &\to_{\lambda} (x_1 \ldots x_{n+1})\,M_0\\
(x_1 \ldots x_n x_{n+1})\,M_0 &\to_{\mathsf{S}} (x_1 \ldots x_n)\,M_0 && (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{aligned}
$$

$\to_{\mathsf{reg}}$-generated subterms w.r.t. rewrite relation $\to_{\mathsf{reg}}$, rules above plus:

$$(x_1 \ldots x_i \ldots x_{n+1})\,M_0 \to_{\mathsf{del}} (x_1 \ldots x_{i-1} x_{i+1} \ldots x_{n+1})\,M_0 \quad (\text{if } \lambda x_i \text{ is vacuous})$$

Generated subterms

$$\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_1} \\
(xy)\,y
\end{array}$$

$$\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_0} \\
(xy)\,x\,x \to_{\mathsf{S}} \\
(x)\,x\,x \to_{@_0} \\
(x)\,x
\end{array}$$

$$\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_0} \\
(xy)\,x\,x \to_{\mathsf{S}} \\
(x)\,x\,x \to_{@_1} \\
(x)\,x
\end{array}$$

$$(x_1 \dots x_n)\,M_0\,M_1 \;\to_{@_i}\; (x_1 \dots x_n)\,M_i \qquad (i \in \{0,1\})$$
$$(x_1 \dots x_n)\,\lambda x_{n+1}.\,M_0 \;\to_{\lambda}\; (x_1 \dots x_{n+1})\,M_0$$

$\to_{\text{reg}}$-generated subterms w.r.t. rewrite relation $\to_{\text{reg}}$, rules above plus:

$$(x_1 \dots x_i \dots x_{n+1})\,M_0 \to_{\mathsf{del}} (x_1 \dots x_{i-1} x_{i+1} \dots x_{n+1})\,M_0 \quad (\text{if } \lambda x_i \text{ is vacuous})$$

Generated subterms

$$
\begin{array}{l}
(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_1} \\
(xy)\,y \to_{\mathsf{del}} \\
(y)\,y
\end{array}
\qquad
\begin{array}{l}
(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_0} \\
(xy)\,x\,x \to_{\mathsf{S}} \\
(x)\,x\,x \to_{@_0} \\
(x)\,x
\end{array}
\qquad
\begin{array}{l}
(\,)\,\lambda x.\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \to_{\lambda} \\
(xy)\,x\,x\,y \to_{@_0} \\
(xy)\,x\,x \to_{\mathsf{S}} \\
(x)\,x\,x \to_{@_1} \\
(x)\,x
\end{array}
$$

$$
\begin{array}{rcll}
(x_1 \dots x_n)\,M_0\,M_1 & \to_{@_i} & (x_1 \dots x_n)\,M_i & (i \in \{0,1\}) \\
(x_1 \dots x_n)\,\lambda x_{n+1}.\,M_0 & \to_{\lambda} & (x_1 \dots x_{n+1})\,M_0 &
\end{array}
$$

$\to_{\mathsf{reg}}$-generated subterms w.r.t. rewrite relation $\to_{\mathsf{reg}}$, rules above plus:

$$(x_1 \dots x_i \dots x_{n+1})\,M_0 \to_{\mathsf{del}} (x_1 \dots x_{i-1} x_{i+1} \dots x_{n+1})\,M_0 \quad (\text{if } \lambda x_i \text{ is vacuous})$$

Generated subterms

$$
\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(xy)\,x\,x\,y \rightarrow_{@_1} \\
(xy)\,y \rightarrow_{\mathsf{del}} \\
(y)\,y
\end{array}
\qquad
\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(xy)\,x\,x\,y \rightarrow_{@_0} \\
(xy)\,x\,x \rightarrow_{\mathsf{S}} \\
(x)\,x\,x \rightarrow_{@_0} \\
(x)\,x
\end{array}
\qquad
\begin{array}{l}
()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\
(xy)\,x\,x\,y \rightarrow_{@_0} \\
(xy)\,x\,x \rightarrow_{\mathsf{S}} \\
(x)\,x\,x \rightarrow_{@_1} \\
(x)\,x
\end{array}
$$

$\rightarrow_{\mathsf{reg}^+}$-generated subterms of $\lambda x.\,\lambda y.\,x\,x\,y$ w.r.t. rewrite relation $\rightarrow_{\mathsf{reg}^+}$:

$$
\begin{array}{rcll}
(x_1 \ldots x_n)\,M_0\,M_1 & \rightarrow_{@_i} & (x_1 \ldots x_n)\,M_i & (i \in \{0,1\}) \\
(x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 & \rightarrow_{\lambda} & (x_1 \ldots x_{n+1})\,M_0 & \\
(x_1 \ldots x_n x_{n+1})\,M_0 & \rightarrow_{\mathsf{S}} & (x_1 \ldots x_n)\,M_0 & (\text{if } \lambda x_{n+1} \text{ is vacuous})
\end{array}
$$

$\rightarrow_{\mathsf{reg}}$-generated subterms w.r.t. rewrite relation $\rightarrow_{\mathsf{reg}}$, rules above plus:

$$(x_1 \ldots x_i \ldots x_{n+1})\,M_0 \rightarrow_{\mathsf{del}} (x_1 \ldots x_{i-1} x_{i+1} \ldots x_{n+1})\,M_0 \quad (\text{if } \lambda x_i \text{ is vacuous})$$

Generated subterms

$$\begin{array}{l} ()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (xy)\,x\,x\,y \rightarrow_{@_1} \\ (xy)\,y \rightarrow_{\mathsf{del}} \\ (y)\,y \end{array}
\qquad
\begin{array}{l} ()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (xy)\,x\,x\,y \rightarrow_{@_0} \\ (xy)\,x\,x \rightarrow_{\mathsf{S}} \\ (x)\,x\,x \rightarrow_{@_0} \\ (x)\,x \end{array}
\qquad
\begin{array}{l} ()\,\lambda x.\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (x)\,\lambda y.\,x\,x\,y \rightarrow_{\lambda} \\ (xy)\,x\,x\,y \rightarrow_{@_0} \\ (xy)\,x\,x \rightarrow_{\mathsf{S}} \\ (x)\,x\,x \rightarrow_{@_1} \\ (x)\,x \end{array}$$

$\rightarrow_{\mathsf{reg}^+}$-generated subterms of $\lambda x.\,\lambda y.\,x\,x\,y$ w.r.t. rewrite relation $\rightarrow_{\mathsf{reg}^+}$:

$$\begin{array}{rcll} (x_1 \ldots x_n)\,M_0\,M_1 & \rightarrow_{@_i} & (x_1 \ldots x_n)\,M_i & (i \in \{0,1\}) \\ (x_1 \ldots x_n)\,\lambda x_{n+1}.\,M_0 & \rightarrow_{\lambda} & (x_1 \ldots x_{n+1})\,M_0 & \\ (x_1 \ldots x_n x_{n+1})\,M_0 & \rightarrow_{\mathsf{S}} & (x_1 \ldots x_n)\,M_0 & (\text{if } \lambda x_{n+1} \text{ is vacuous}) \end{array}$$

$\rightarrow_{\mathsf{reg}}$-generated subterms w.r.t. rewrite relation $\rightarrow_{\mathsf{reg}}$, rules above plus:

$$(x_1 \ldots x_i \ldots x_{n+1})\,M_0 \rightarrow_{\mathsf{del}} (x_1 \ldots x_{i-1} x_{i+1} \ldots x_{n+1})\,M_0 \quad (\text{if } \lambda x_i \text{ is vacuous})$$

We use eager application of scope-closure rules for $\rightarrow_{\mathsf{reg}^+}$ and $\rightarrow_{\mathsf{reg}}$.

Regularity and strong regularity

An infinite first-order term t is regular if:

t has only finitely many subterms.

Definition

1. A λ^∞-term M is strongly regular if:

   $()M$ has only finitely many $\to_{\text{reg}^+}$-generated subterms.

Regularity and strong regularity

An infinite first-order term t is regular if:

t has only finitely many subterms.

Definition

1. A λ^∞-term M is strongly regular if:

   $()M$ has only finitely many $\rightarrow_{\text{reg}^+}$-generated subterms.

2. A λ^∞-term N is regular if:

   $()N$ has only finitely many $\rightarrow_{\text{reg}}$-generated subterms.

Strongly regular λ^∞-term

$$()\,M \quad = \quad ()\,\lambda xy.\,M\,y\,x$$

$M = \lambda xy.\,M\,y\,x$

$\rightarrow_{\mathsf{reg}^+}$-generated subterms

Strongly regular λ^∞-term

$$
\begin{aligned}
()\,M &= ()\,\lambda xy.\,M\,y\,x \\
&\to_\lambda (x)\,\lambda y.\,M\,y\,x
\end{aligned}
$$

$M = \lambda xy.\,M\,y\,x$

$\to_{\text{reg}^+}$-generated subterms

Strongly regular λ^∞-term

$$\begin{aligned} ()\,M &= ()\,\lambda xy.\,M\,y\,x \\ &\to_\lambda (x)\,\lambda y.\,M\,y\,x \\ &\to_\lambda (xy)\,M\,y\,x \end{aligned}$$

$M = \lambda xy.\,M\,y\,x$

$\to_{\mathsf{reg}^+}$-generated subterms

Strongly regular λ^∞-term

$$
\begin{aligned}
(\,)\, M &= (\,)\, \lambda xy.\, M\, y\, x \\
&\to_{\lambda} (x)\, \lambda y.\, M\, y\, x \\
&\to_{\lambda} (xy)\, M\, y\, x \\
&\to_{@_0} (xy)\, M\, y
\end{aligned}
$$

$M = \lambda xy.\, M\, y\, x$

$\to_{\mathsf{reg}^+}$-generated subterms

Strongly regular λ^{∞}-term

$$
\begin{array}{rcl}
(\,)\,M & = & (\,)\,\lambda xy.\,M\,y\,x \\
 & \to_{\lambda} & (x)\,\lambda y.\,M\,y\,x \\
 & \to_{\lambda} & (xy)\,M\,y\,x \\
 & \to_{@_0} & (xy)\,M\,y \\
 & \to_{@_0} & (xy)\,M
\end{array}
$$

$M = \lambda xy.\,M\,y\,x$

$\to_{\text{reg}^+}$-generated subterms

Strongly regular λ^∞-term

$$
\begin{aligned}
()\,M \quad &= \quad ()\,\lambda xy.\,M\,y\,x \\
&\to_\lambda \quad (x)\,\lambda y.\,M\,y\,x \\
&\to_\lambda \quad (xy)\,M\,y\,x \\
&\to_{@_0} \quad (xy)\,M\,y \\
&\to_{@_0} \quad (xy)\,M \\
&\to_{\mathsf{S}} \quad (x)\,M
\end{aligned}
$$

$M = \lambda xy.\,M\,y\,x$ $\qquad$ $\to_{\text{reg}^+}$-generated subterms

Strongly regular λ^{∞}-term

$$
\begin{array}{rcl}
()\,M & = & ()\,\lambda xy.\,M\,y\,x \\
 & \to_{\lambda} & (x)\,\lambda y.\,M\,y\,x \\
 & \to_{\lambda} & (xy)\,M\,y\,x \\
 & \to_{@_0} & (xy)\,M\,y \\
 & \to_{@_0} & (xy)\,M \\
 & \to_{\mathsf{S}} & (x)\,M \\
 & \to_{\mathsf{S}} & ()\,M
\end{array}
$$

$M = \lambda xy.\,M\,y\,x$

$\to_{\text{reg}^+}$-generated subterms

Strongly regular λ^∞-term

$$
\begin{array}{lcl}
()\,M & = & ()\,\lambda xy.\,M\,y\,x \\
 & \to_{\lambda} & (x)\,\lambda y.\,M\,y\,x \\
 & \to_{\lambda} & (xy)\,M\,y\,x \\
 & \to_{@_0} & (xy)\,M\,y \\
 & \to_{@_0} & (xy)\,M \\
 & \to_{\mathsf{S}} & (x)\,M \\
 & \to_{\mathsf{S}} & ()\,M \\
 & \dots &
\end{array}
$$

$M = \lambda xy.\,M\,y\,x$

$\to_{\text{reg}^+}$-generated subterms

Strongly regular λ^{∞}-term

$M = \lambda xy.\, M\, y\, x$

finitely many $\rightarrow_{\text{reg}^+}$-generated subterms

$\Longrightarrow$ M is strongly regular

Not strongly regular λ^∞-term

$$N \quad = \quad ()\,\lambda a.\,\lambda b.\,(\ldots)\,a$$

λ^∞-term N $\rightarrow_{\mathsf{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$\begin{aligned} N &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\ &\rightarrow_\lambda \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \end{aligned}$$

λ^∞-term N $\rightarrow_{\mathsf{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$
\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_\lambda \quad (a)\,\lambda b.\,(\lambda c.\;\ldots)\,a \\
&\to_\lambda \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a
\end{aligned}
$$

λ^∞-term N

$\to_{\mathsf{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_{\lambda} \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_{\lambda} \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} \quad (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b
\end{aligned}$$

λ^∞-term N $\quad$ $\to_{\mathsf{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$
\begin{array}{rcl}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
 & \to_\lambda & (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
 & \to_\lambda & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
 & \to_{@_0} & (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
 & \to_\lambda & (abc)\,(\lambda d.\,(\ldots)\,c)\,b
\end{array}
$$

λ^∞-term N $\to_{\text{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$
\begin{array}{lll}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
 & \to_\lambda & (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
 & \to_\lambda & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
 & \to_{@_0} & (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
 & \to_\lambda & (abc)\,(\lambda d.\,(\ldots)\,c)\,b \\
 & \to_{@_0} & (abc)\,\lambda d.\,(\lambda e.\ \ldots)\,c
\end{array}
$$

λ^∞-term N $\to_{\text{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$
\begin{array}{lll}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
 & \to_\lambda & (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
 & \to_\lambda & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
 & \to_{@_0} & (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
 & \to_\lambda & (abc)\,(\lambda d.\,(\ldots)\,c)\,b \\
 & \to_{@_0} & (abc)\,\lambda d.\,(\lambda e.\ \ldots)\,c \\
 & \to_\lambda & (abcd)\,(\lambda e.\,(\ldots)\,d)\,c
\end{array}
$$

λ^∞-term N $\to_{\text{reg}^+}$-generated subterms

Not strongly regular λ^∞-term

$$
\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_\lambda \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_\lambda \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} \quad (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_\lambda \quad (abc)\,(\lambda d.\,(\ldots)\,c)\,b \\
&\to_{@_0} \quad (abc)\,\lambda d.\,(\lambda e.\ \ldots)\,c \\
&\to_\lambda \quad (abcd)\,(\lambda e.\,(\ldots)\,d)\,c \\
&\to_{@_0} \quad (abcd)\,\lambda e.\,(\lambda f.\ \ldots)\,d \\
&\ldots
\end{aligned}
$$

λ^∞-term N infinitely many $\to_{\text{reg}^+}$-generated subterms
$\Longrightarrow$ N is not strongly regular

Regular λ^∞-term

$$N \quad = \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a$$

λ^∞-term N

$\rightarrow_{\text{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_\lambda \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a
\end{aligned}
$$

λ^∞-term N

$\rightarrow_{\text{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{aligned}
N \quad &= \quad ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_\lambda \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_\lambda \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a
\end{aligned}
$$

λ^∞-term N

$\to_{\text{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{aligned}
N \quad &= &&(\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_\lambda &&(a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_\lambda &&(ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} &&(ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b
\end{aligned}
$$

λ^∞-term N

$\to_{\text{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{array}{lll}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
 & \to_\lambda & (a)\,\lambda b.\,(\lambda c.\;\ldots)\,a \\
 & \to_\lambda & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
 & \to_{@_0} & (ab)\,\lambda c.\,(\lambda d.\;\ldots)\,b \\
 & \to_{\text{del}} & (b)\,\lambda c.\,(\lambda d.\;\ldots)\,b
\end{array}
$$

λ^∞-term N

$\to_{\text{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_{\lambda} \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_{\lambda} \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} \quad (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\mathsf{del}} \quad (b)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\lambda} \quad (bc)\,(\lambda d.\,(\ldots)\,c)\,b
\end{aligned}
$$

λ^∞-term N

$\to_{\mathsf{reg}}$-generated subterms

Regular λ^{∞}-term

$$
\begin{aligned}
N &= && ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_{\lambda} && (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_{\lambda} && (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} && (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\mathsf{del}} && (b)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\lambda} && (bc)\,(\lambda d.\,(\ldots)\,c)\,b \\
&\to_{@_0} && (bc)\,\lambda d.\,(\lambda d.\ \ldots)\,c
\end{aligned}
$$

λ^{∞}-term N

$\to_{\mathsf{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{array}{lll}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
& \rightarrow_{\lambda} & (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
& \rightarrow_{\lambda} & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
& \rightarrow_{@_0} & (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
& \rightarrow_{\mathsf{del}} & (b)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
& \rightarrow_{\lambda} & (bc)\,(\lambda d.\,(\ldots)\,c)\,b \\
& \rightarrow_{@_0} & (bc)\,\lambda d.\,(\lambda d.\ \ldots)\,c \\
& \rightarrow_{\mathsf{del}} & (c)\,\lambda d.\,(\lambda e.\ \ldots)\,d
\end{array}
$$

λ^∞-term N

$\rightarrow_{\mathsf{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{aligned}
N \quad & = && (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
& \to_{\lambda} && (a)\,\lambda b.\,(\lambda c.\;\ldots)\,a \\
& \to_{\lambda} && (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
& \to_{@_0} && (ab)\,\lambda c.\,(\lambda d.\;\ldots)\,b \\
& \to_{\mathsf{del}} && (b)\,\lambda c.\,(\lambda d.\;\ldots)\,b \\
& \to_{\lambda} && (bc)\,(\lambda d.\,(\ldots)\,c)\,b \\
& \to_{@_0} && (bc)\,\lambda d.\,(\lambda d.\;\ldots)\,c \\
& \to_{\mathsf{del}} && (c)\,\lambda d.\,(\lambda e.\;\ldots)\,d \\
& \to_{\lambda} && (cd)\,(\lambda e.\,(\ldots)\,d)\,c
\end{aligned}
$$

λ^∞-term N

$\to_{\mathsf{reg}}$-generated subterms

Regular λ^∞-term

$$
\begin{array}{lll}
N & = & ()\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
 & \rightarrow_{\lambda} & (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
 & \rightarrow_{\lambda} & (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
 & \rightarrow_{@_0} & (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
 & \rightarrow_{\mathsf{del}} & (b)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
 & \rightarrow_{\lambda} & (bc)\,(\lambda d.\,(\ldots)\,c)\,b \\
 & \rightarrow_{@_0} & (bc)\,\lambda d.\,(\lambda d.\ \ldots)\,c \\
 & \rightarrow_{\mathsf{del}} & (c)\,\lambda d.\,(\lambda e.\ \ldots)\,d \\
 & \rightarrow_{\lambda} & (cd)\,(\lambda e.\,(\ldots)\,d)\,c \\
 & \rightarrow_{@_0} & (cd)\,\lambda e.\,(\lambda f.\ \ldots)\,d
\end{array}
$$

λ^∞-term N

$\rightarrow_{\mathsf{reg}}$-generated subterms

Regular λ^{∞}-term

$$
\begin{aligned}
N \quad &= \quad (\,)\,\lambda a.\,\lambda b.\,(\ldots)\,a \\
&\to_{\lambda} \quad (a)\,\lambda b.\,(\lambda c.\ \ldots)\,a \\
&\to_{\lambda} \quad (ab)\,(\lambda c.\,(\ldots)\,b)\,a \\
&\to_{@_0} \quad (ab)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\mathsf{del}} \quad (b)\,\lambda c.\,(\lambda d.\ \ldots)\,b \\
&\to_{\lambda} \quad (bc)\,(\lambda d.\,(\ldots)\,c)\,b \\
&\to_{@_0} \quad (bc)\,\lambda d.\,(\lambda d.\ \ldots)\,c \\
&\to_{\mathsf{del}} \quad (c)\,\lambda d.\,(\lambda e.\ \ldots)\,d \\
&\to_{\lambda} \quad (cd)\,(\lambda e.\,(\ldots)\,d)\,c \\
&\to_{@_0} \quad (cd)\,\lambda e.\,(\lambda f.\ \ldots)\,d \\
&\to_{\mathsf{del}} \quad (d)\,\lambda e.\,(\lambda f.\ \ldots)\,d
\end{aligned}
$$

λ^{∞}-term N $\quad$ $\to_{\text{reg}}$-generated subterms

Regular λ^∞-term

λ^∞-term N	finitely many $\rightarrow_{\text{reg}}$-generated subterms
$\{N = \lambda xy.\, R(y)\, x,$ $R(z) = \lambda u.\, R(u)\, z\}$	$\Longrightarrow$ M is regular

Strongly regular ⇒ regular

Proposition

- Every strongly regular λ^{∞}-term is also regular.
- Finite λ-terms are both regular and strongly regular.

$\boldsymbol{\lambda}_{\mathsf{letrec}}$-Expressibility

Proposition

- Every strongly regular λ^{∞}-term is also regular.
- Finite λ-terms are both regular and strongly regular.

Theorem ($\lambda_{\mathsf{letrec}}$-expressibility)

An λ^{∞}-term is $\boldsymbol{\lambda}_{\mathsf{letrec}}$-expressible if and only if it is strongly regular.

Binding–capturing chains

Definition (Melliés, van Oostrom)

For positions p, q, r, s:

$$p \multimap q \;:\Longleftrightarrow\; \text{binder at } p \text{ binds variable occurrence at position } q$$

$$r \dashrightarrow s \;:\Longleftrightarrow\; \text{variable occurrence at } r \text{ is captured by binding at } s$$

Binding–capturing chains: $p_0 \multimap p_1 \dashrightarrow p_2 \multimap p_3 \dashrightarrow p_4 \multimap \ldots$

Binding–capturing chains

Definition (Melliés, van Oostrom)

For positions p, q, r, s:

$$p \multimap q \;:\Longleftrightarrow\; \text{binder at } p \text{ binds variable occurrence at position } q$$

$$r \dashrightarrow s \;:\Longleftrightarrow\; \text{variable occurrence at } r \text{ is captured by binding at } s$$

Binding–capturing chains: $p_0 \multimap p_1 \dashrightarrow p_2 \multimap p_3 \dashrightarrow p_4 \multimap \ldots$

Main theorem (extended)

Theorem (binding–capturing chains)

For all λ^∞-term M:

M *is strongly regular* $\Longleftrightarrow$ M *is regular,* <u>*and*</u>
M *has only finite binding–capturing chains.*

Main theorem (extended)

Theorem (binding–capturing chains)

For all λ^∞-term M:

M *is strongly regular* $\iff$ M *is regular, and*

M *has only finite binding–capturing chains.*

Theorem ($\boldsymbol{\lambda}_{\text{letrec}}$-expressibility, extended)

For all λ^∞-terms M the following are equivalent:

(i) M *is $\boldsymbol{\lambda}_{\text{letrec}}$-expressible.*

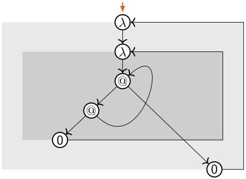
(ii) M *is strongly regular.*

(iii) M *is regular, and it only contains finite binding–capturing chains.*

Maximal sharing of functional programs

(joint work with Jan Rochel)

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



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

Results:

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

The method

- Methods consist of the steps:
  1. interpretation of $\boldsymbol{\lambda}_{\text{letrec}}$-terms as term graphs
     - higher-order term graphs: λ-ho-term-graphs
     - first-order term graphs : λ-term-graphs
     - deterministic finite-state automata: λ-DFAs
  2. bisimilarity & bisimulation collapse of λ-term-graphs
     - implemented as: DFA-minimization of λ-DFAs
  3. readback of λ-term-graphs as $\boldsymbol{\lambda}_{\text{letrec}}$-terms
- Haskell implementation
- Complexity

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: the method

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

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

Maximal sharing: the method

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

L

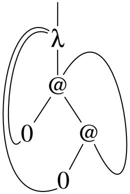
L_0

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

Maximal sharing: the method

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

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



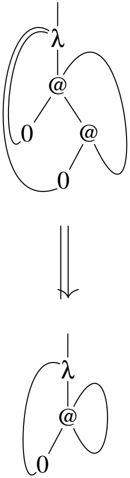
L_0

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

Maximal sharing: the method

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

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

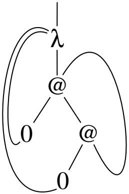


$\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r$

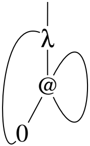
Maximal sharing: the method

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

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



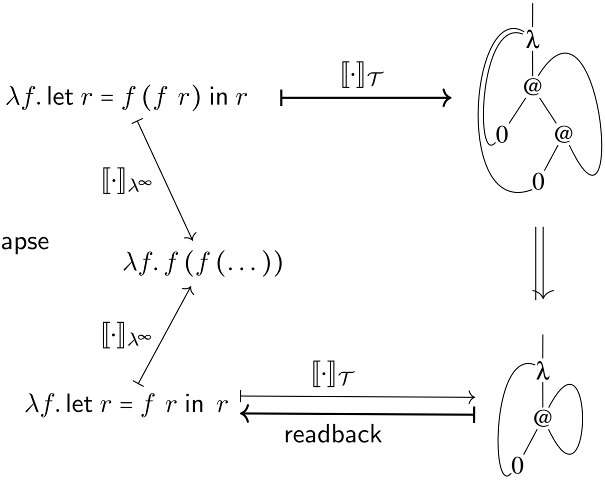
$\Downarrow$



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

Maximal sharing: the method

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



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 $\boldsymbol{\lambda}_{\mathsf{letrec}}$-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

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

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

Maximal sharing: the method

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

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

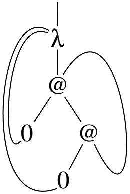
Unfolding equivalence: example

$$
\begin{tikzcd}
L_1 \arrow[d, mapsto, "\text{unfold}"', "?"] \\
M \\
L_2 \arrow[u, mapsto, "\text{unfold}", "?"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\lambda f.\,\mathsf{let}\ r = f\,(f\ r)\ \mathsf{in}\ r \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"'] \\
\lambda f.\,f\,(f\,(\ldots)) \\
\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"]
\end{tikzcd}
$$

Unfolding equivalence: example

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

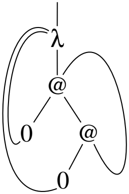
$$
\begin{tikzcd}
\lambda f.\,\mathsf{let}\ r = f\,(f\ r)\ \mathsf{in}\ r \arrow[r, mapsto, "{[\![\cdot]\!]_{\mathcal{T}}}"] \arrow[d, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"'] & \text{} \\
\lambda f.\,f\,(f\,(\ldots)) & \\
\lambda f.\,\mathsf{let}\ r = f\ r\ \mathsf{in}\ r \arrow[u, mapsto, "{[\![\cdot]\!]_{\lambda^\infty}}"]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

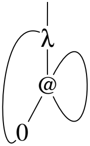


Unfolding equivalence: the method

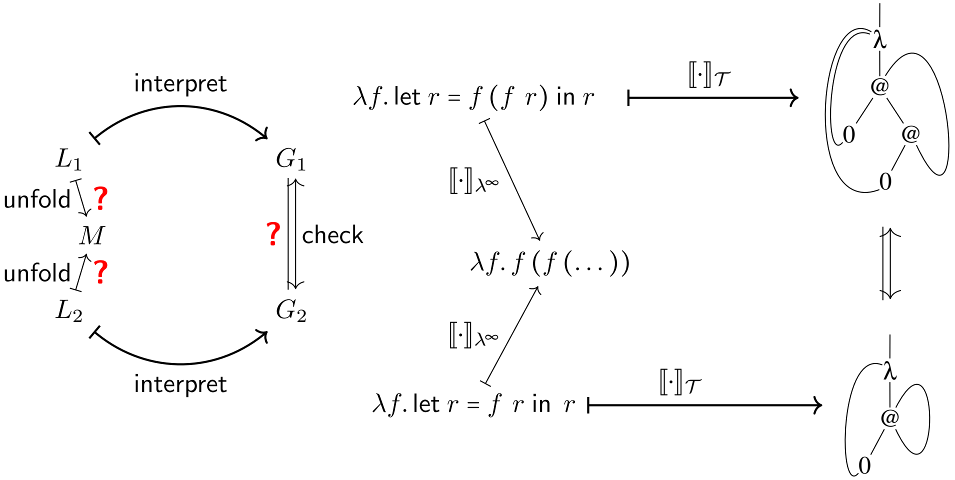
$$\begin{tikzcd} L_1 \arrow[rr, mapsto, bend left, "\text{interpret}"] \arrow[d, mapsto, "\text{unfold}"', "\color{red}?"] & & G_1 \\ M & & \\ L_2 \arrow[u, mapsto, "\text{unfold}", "\color{red}?"'] \arrow[rr, mapsto, bend right, "\text{interpret}"'] & & G_2 \end{tikzcd}$$

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





Unfolding equivalence: the method



Unfolding equivalence: the method

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

Unfolding equivalence: the method

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

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

Unfolding equivalence: the method

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

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 $G_2 = [\![L_2]\!]_{\mathcal{T}}$

Unfolding equivalence: the method

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

1. term graph interpretation $\llbracket\cdot\rrbracket$.

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:

   a. higher-order term graphs

      $\mathcal{G}_1 = \llbracket L_1 \rrbracket_{\mathcal{H}}$ and $\mathcal{G}_2 = \llbracket L_2 \rrbracket_{\mathcal{H}}$

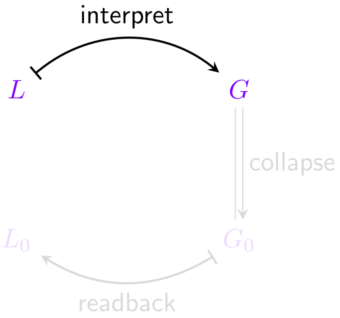
   b. first-order term graphs

      $G_1 = \llbracket L_1 \rrbracket_{\mathcal{T}}$ and $G_2 = \llbracket L_2 \rrbracket_{\mathcal{T}}$

2. check bisimilarity

   of f-o term graphs G_1 and G_2

Interpretation



Running example

instead of:

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

we use:

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

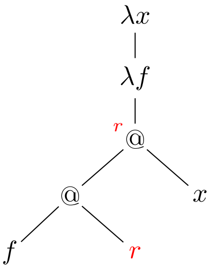
$L \quad \longmapsto_{\text{max-sharing}} \quad L_0$

Graph interpretation (example 1)

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

Graph interpretation (example 1)

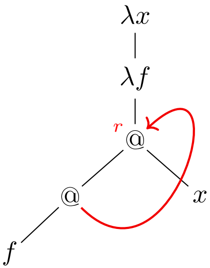
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



syntax tree

Graph interpretation (example 1)

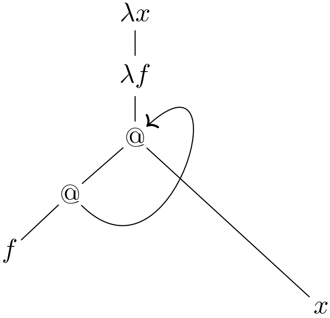
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 1)

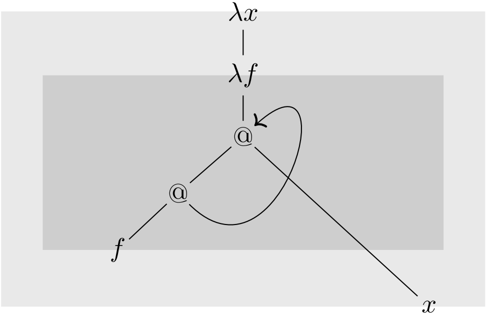
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 1)

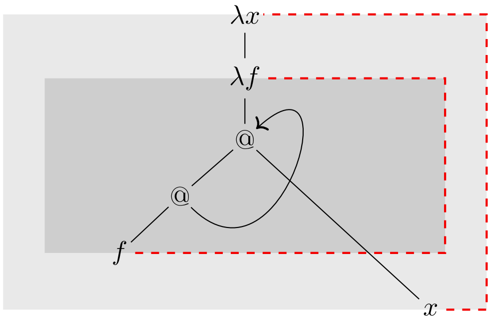
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

Graph interpretation (example 1)

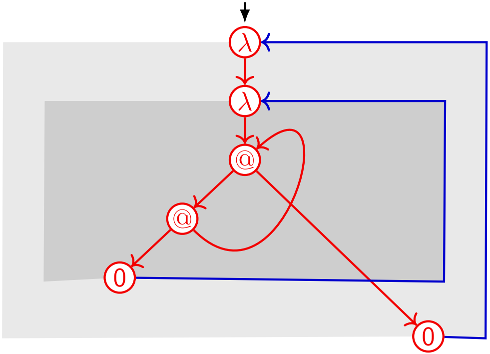
$L_0 = \lambda x. \lambda f. \,\text{let}\; r = f\, r\, x \;\text{in}\; r$



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

Graph interpretation (example 1)

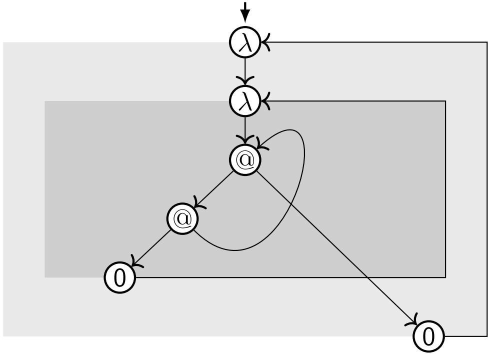
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

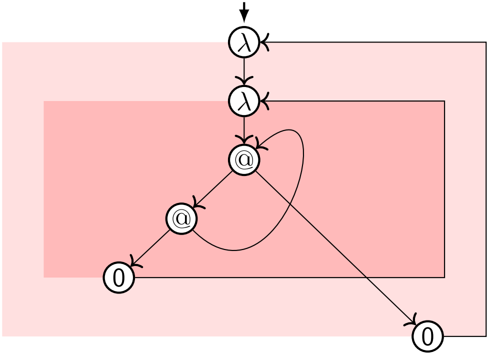
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

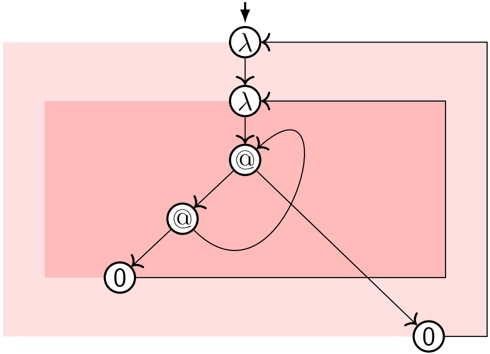
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



first-order term graph (+ scope sets)

Graph interpretation (example 1)

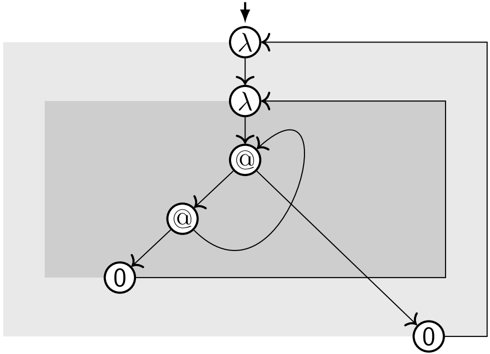
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

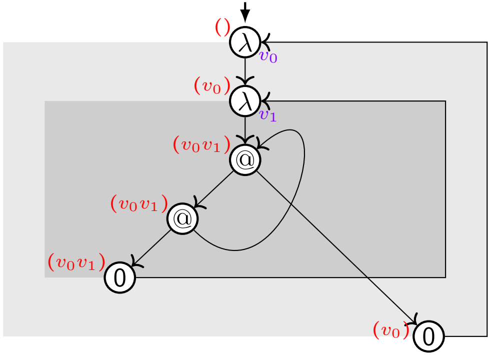
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

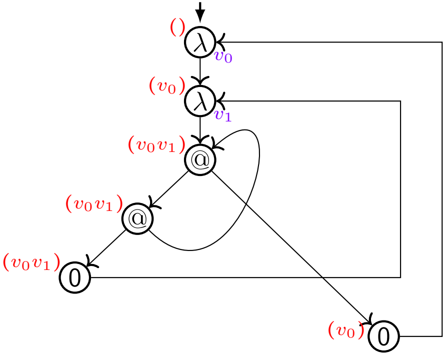
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

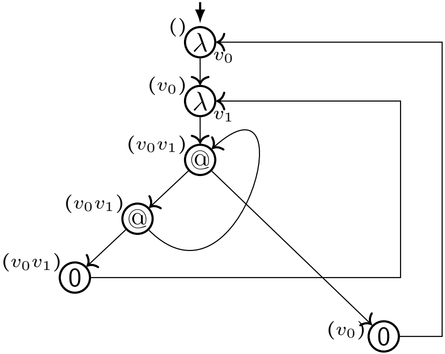
$L_0 = \lambda x. \lambda f. \,\text{let}\; r = f\,r\,x \;\text{in}\; r$



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

Graph interpretation (example 1)

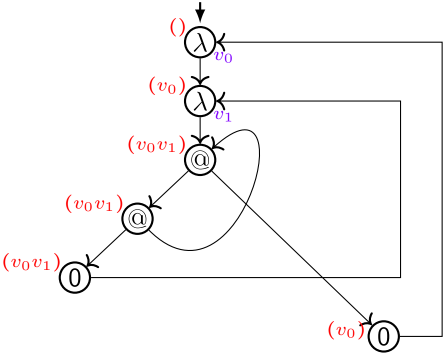
$L_0 = \lambda x.\, \lambda f.\, \mathsf{let}\ r = f\, r\, x\ \mathsf{in}\ r$



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

Graph interpretation (example 1)

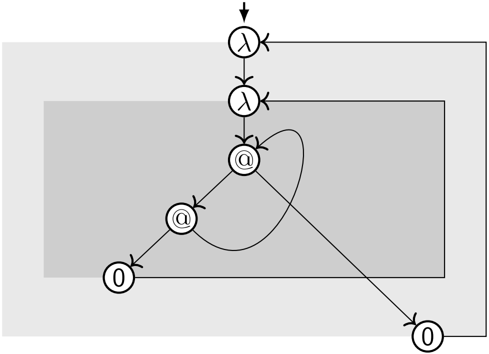
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

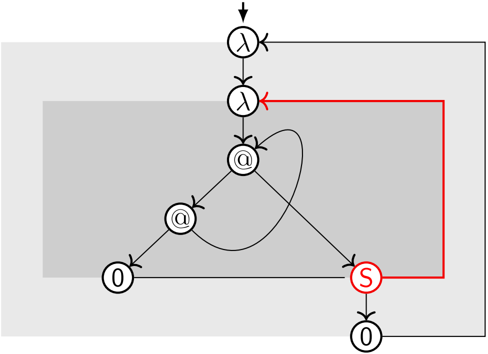
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



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

Graph interpretation (example 1)

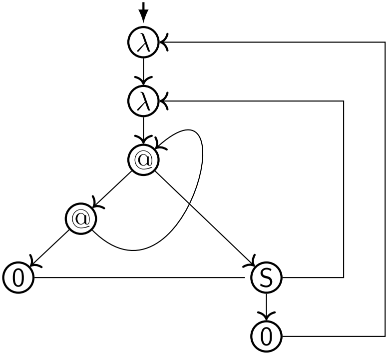
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



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

Graph interpretation (example 1)

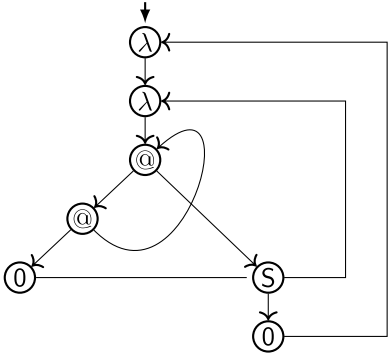
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



first-order term graph with scope vertices with backlinks

Graph interpretation (example 1)

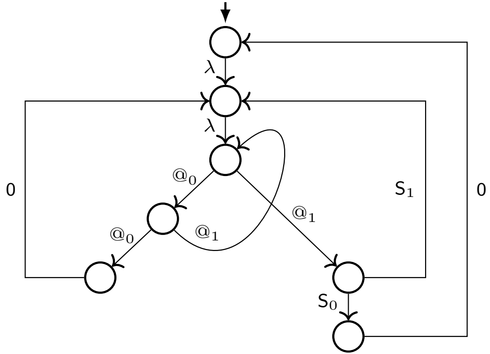
$L_0 = \lambda x. \lambda f. \, \text{let } r = f \, r \, x \text{ in } r$



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

Graph interpretation (example 1)

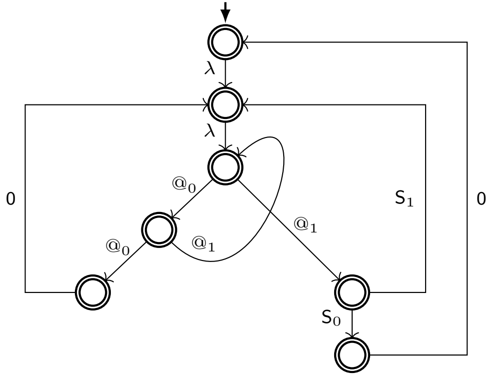
$L_0 = \lambda x.\,\lambda f.\,\text{let } r = f\,r\,x \text{ in } r$



incomplete DFA

Graph interpretation (example 1)

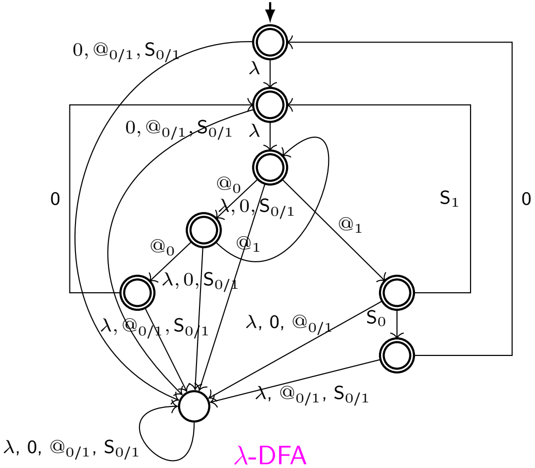
$L_0 = \lambda x. \lambda f. \text{let } r = f\, r\, x \text{ in } r$



incomplete λ-DFA

Graph interpretation (example 1)

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

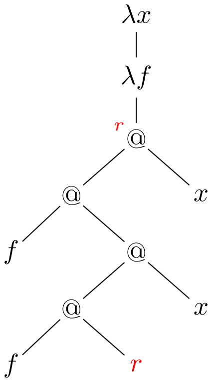


Graph interpretation (example 2)

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

Graph interpretation (example 2)

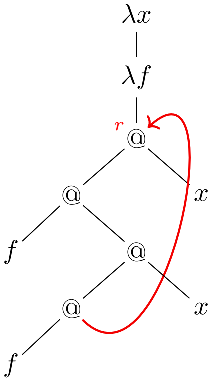
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree

Graph interpretation (example 2)

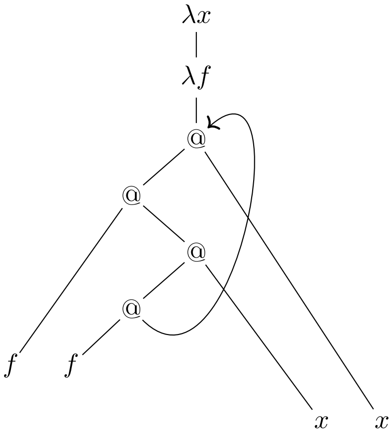
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 2)

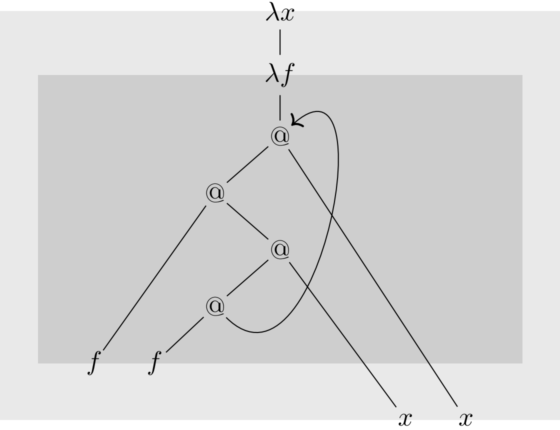
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink)

Graph interpretation (example 2)

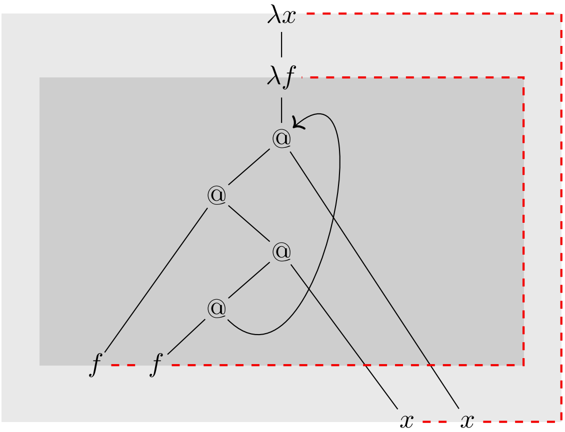
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

Graph interpretation (example 2)

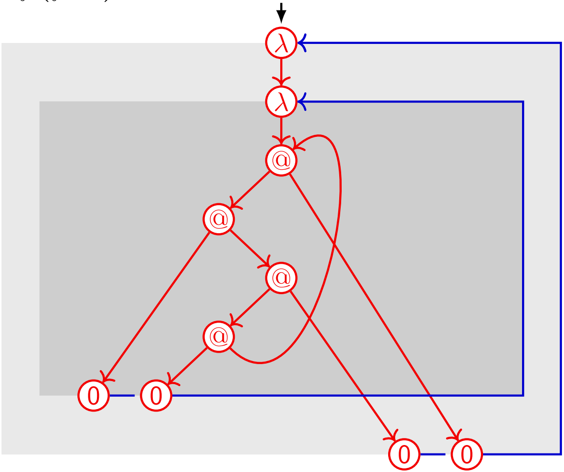
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

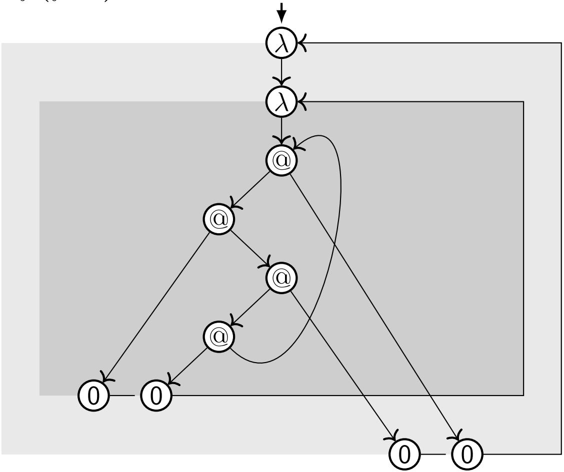
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

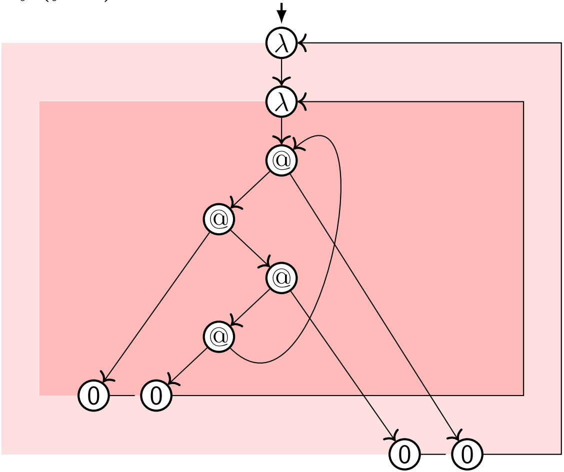
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

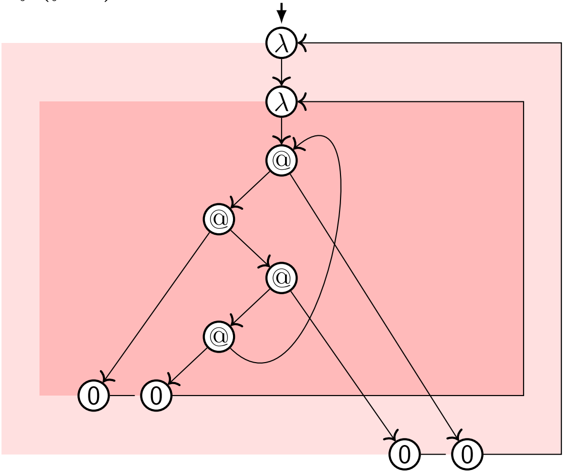
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



first-order term graph (+ scope sets)

Graph interpretation (example 2)

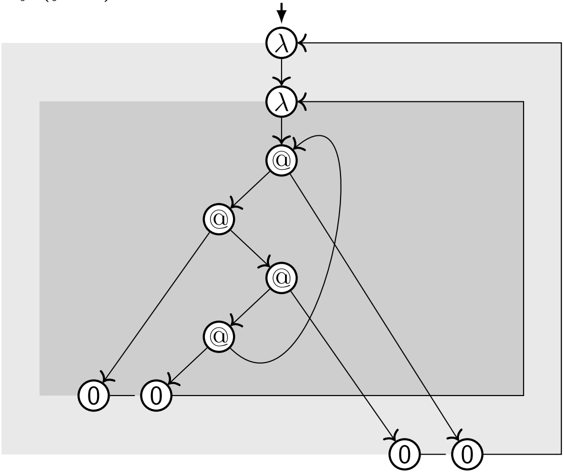
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

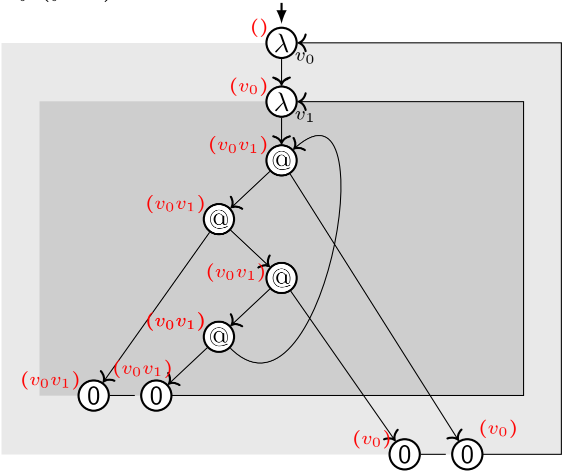
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

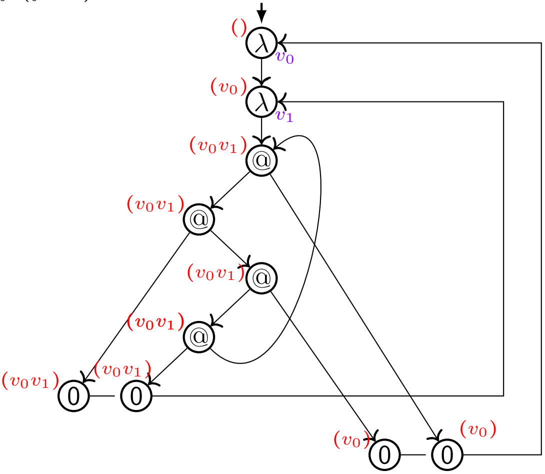
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

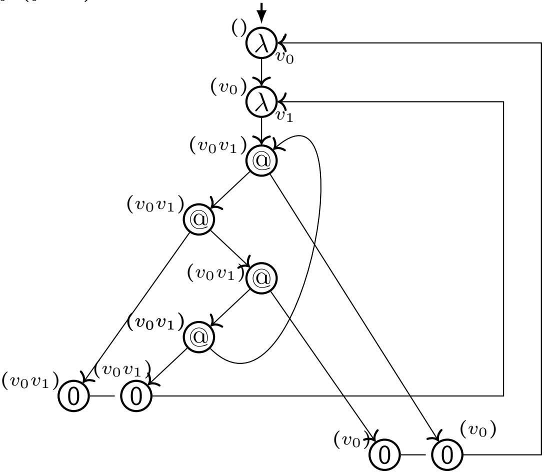
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

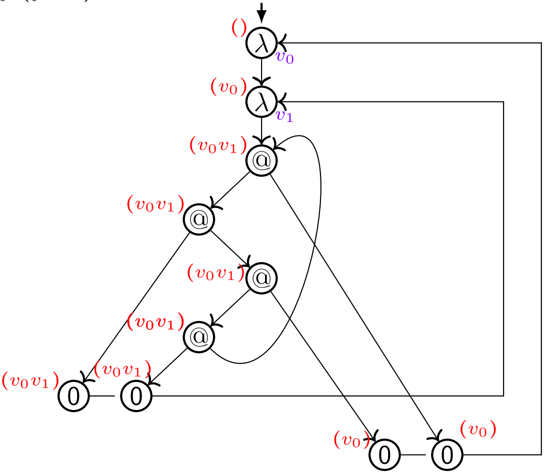
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

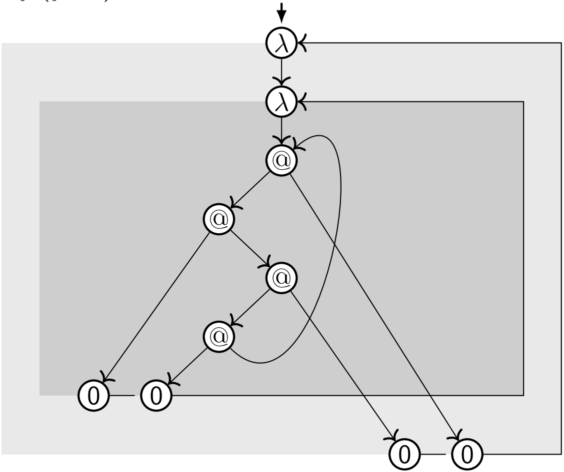
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

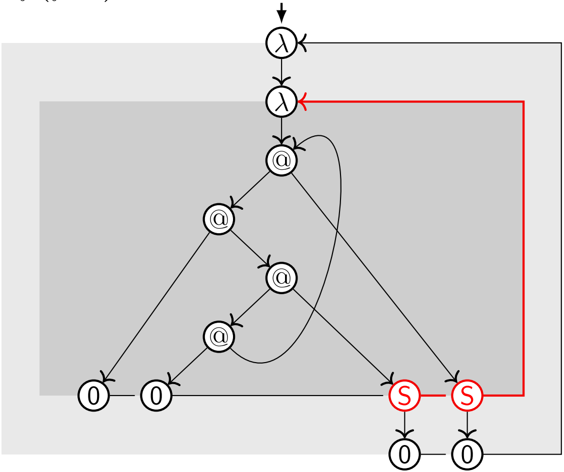
$L = \lambda x.\,\lambda f.\,\mathsf{let}\ r = f\,(f\,r\,x)\,x\ \mathsf{in}\ r$



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

Graph interpretation (example 2)

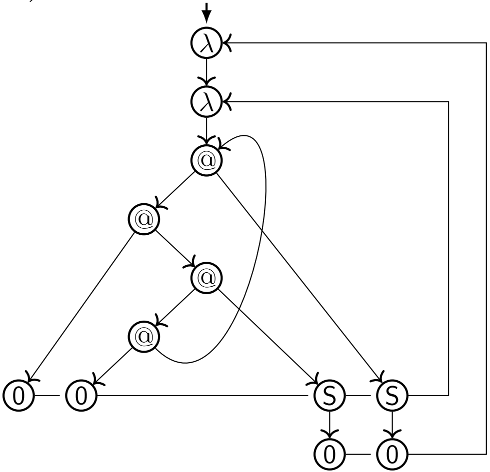
$L = \lambda x.\,\lambda f.\,\text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

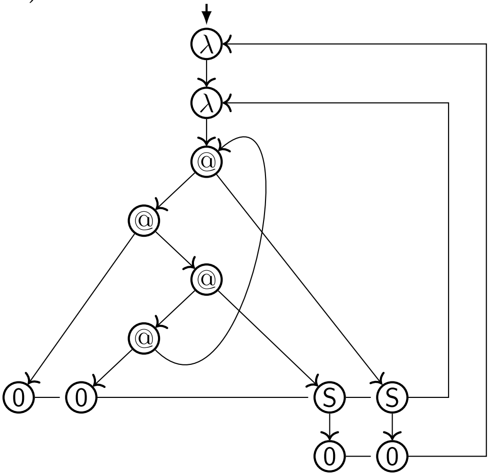
$L = \lambda x. \lambda f. \mathsf{let}\ r = f\,(f\,r\,x)\,x\ \mathsf{in}\ r$



first-order term graph with scope vertices with backlinks

Graph interpretation (example 2)

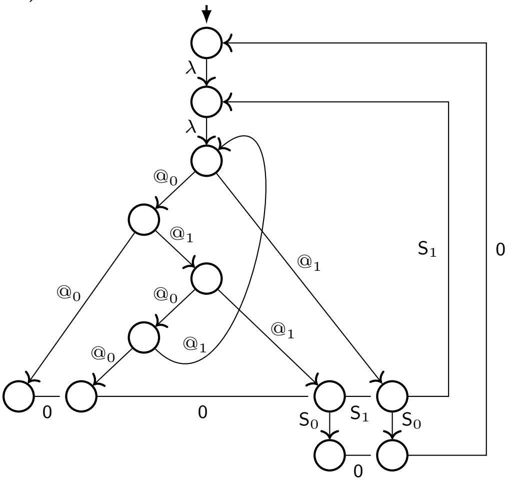
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

Graph interpretation (example 2)

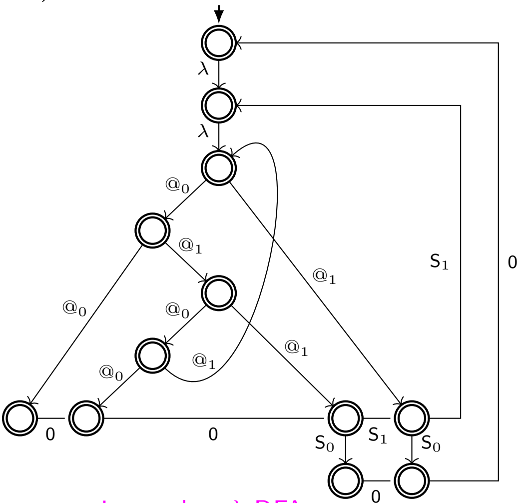
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



incomplete DFA

Graph interpretation (example 2)

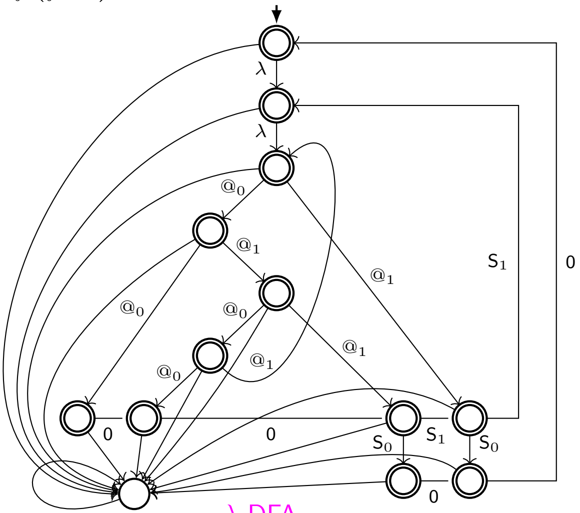
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



incomplete λ-DFA

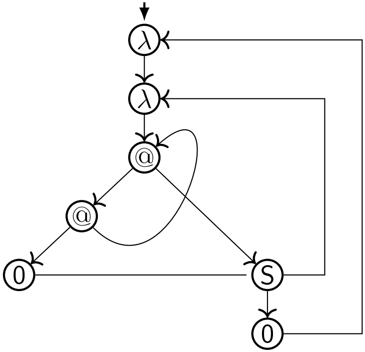
Graph interpretation (example 2)

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

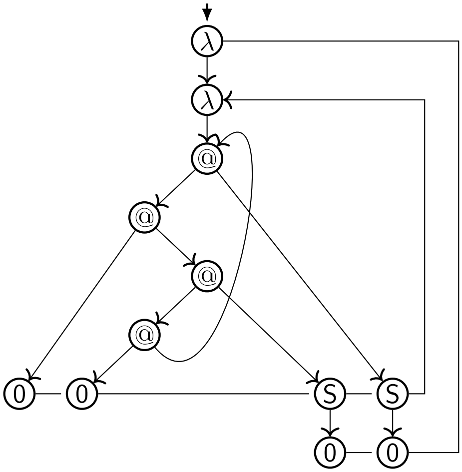


λ-DFA

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 $L \longmapsto$ λ-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 $L \longmapsto$ λ-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

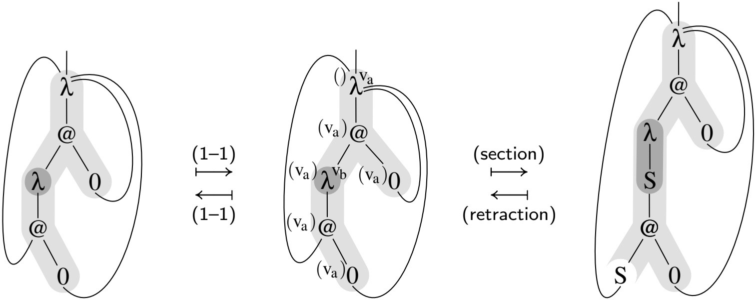
For $\boldsymbol{\lambda}_{\text{letrec}}$-terms L_1 and L_2 it holds: Equality of infinite unfolding coincides with bisimilarity of λ-term-graph interpretations:

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

structure constraints (L'Aquila)

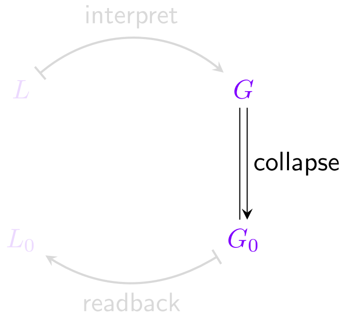
higher-order as first-order term graphs

let $f = \lambda x.\,(\lambda y.\,f\;x)\,x$ in f

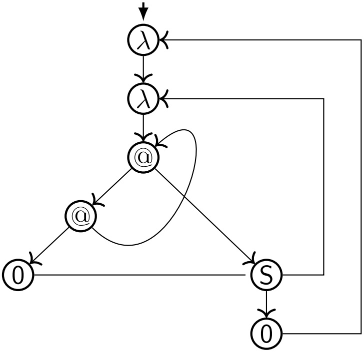


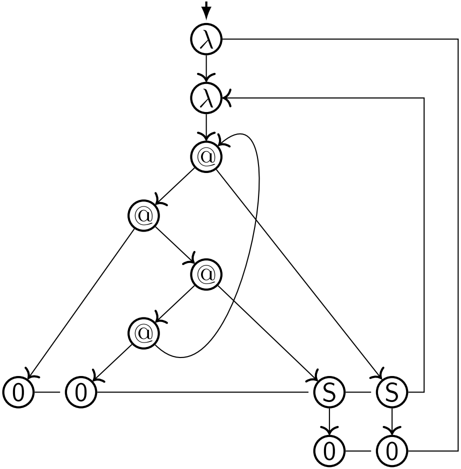
higher-order term graph [Blom '03] | higher-order term graph (abstraction-prefix funct.) | first-order term graph

Collapse



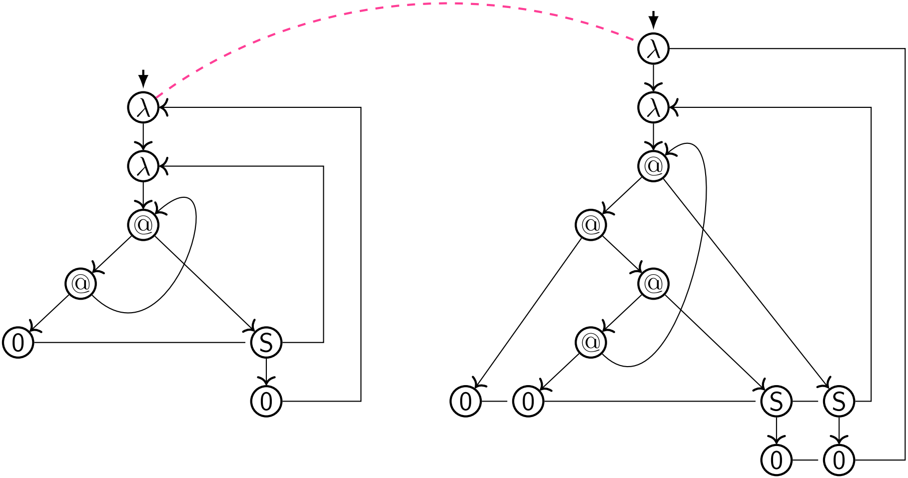
Bisimulation check between λ-term-graphs





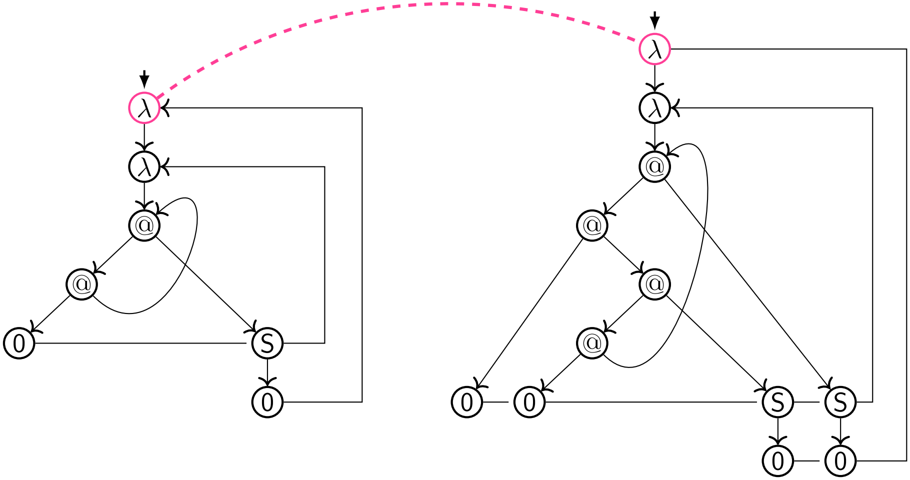
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



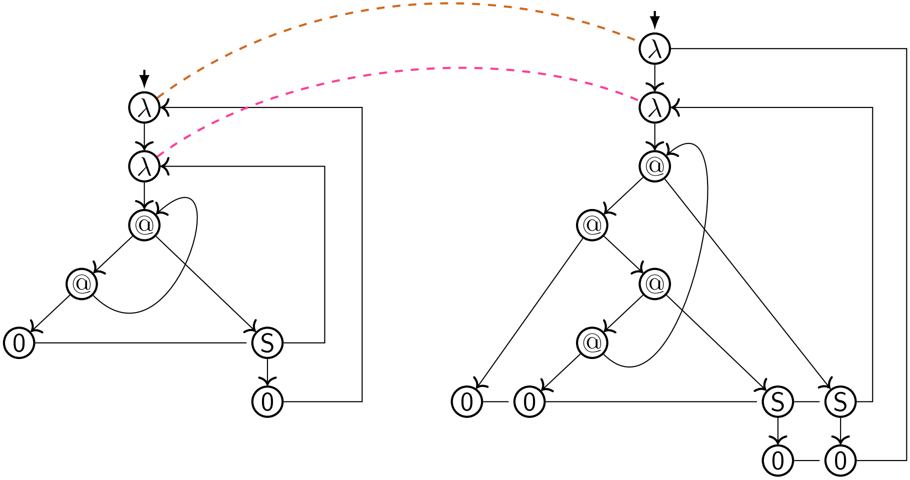
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



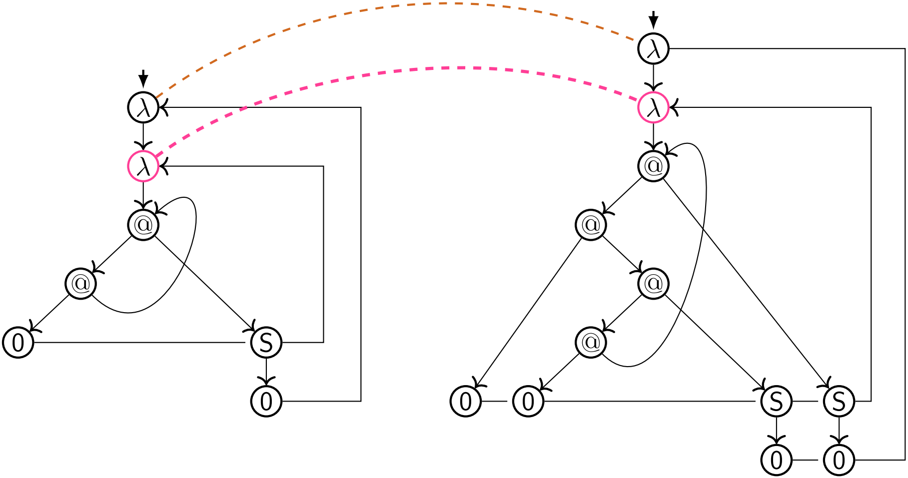
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



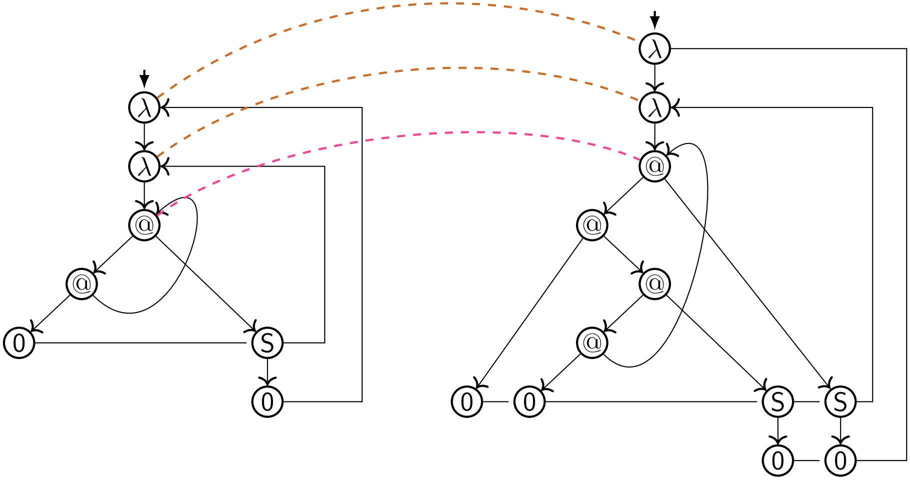
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



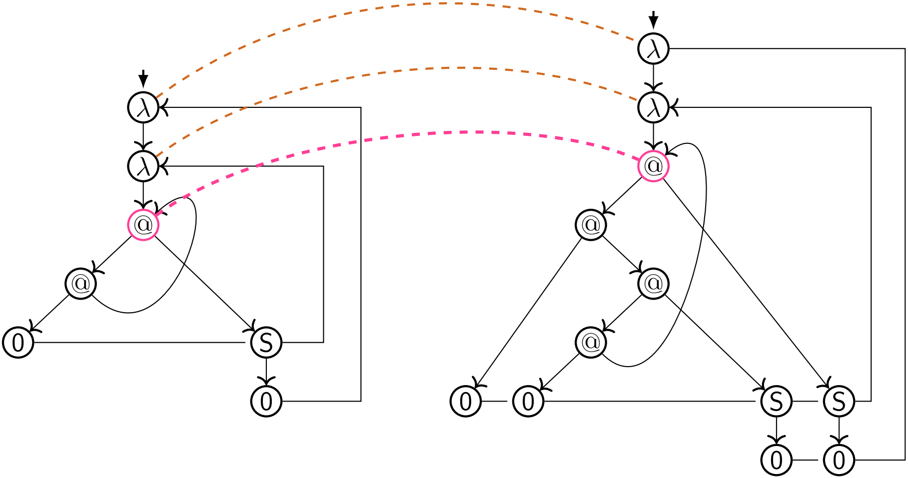
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



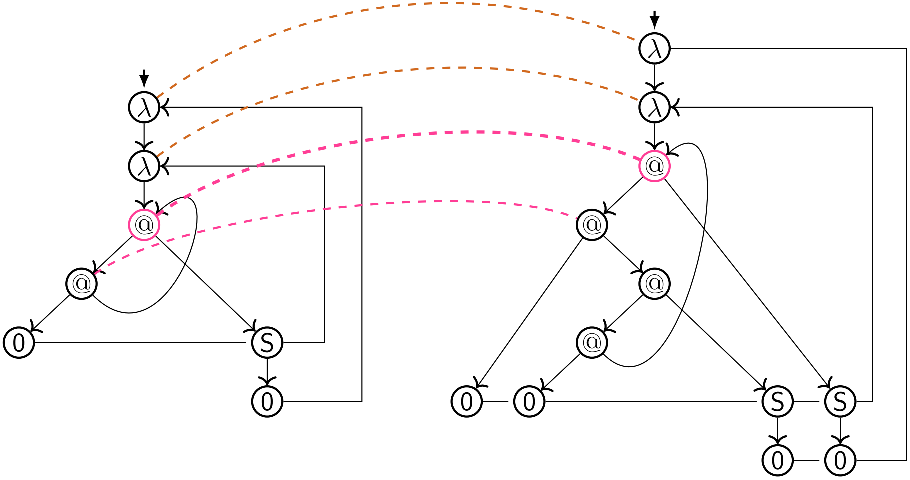
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



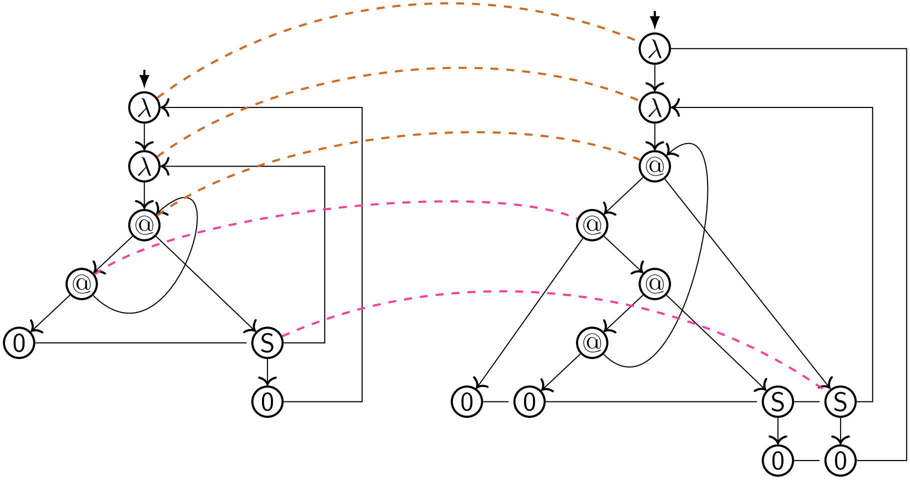
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



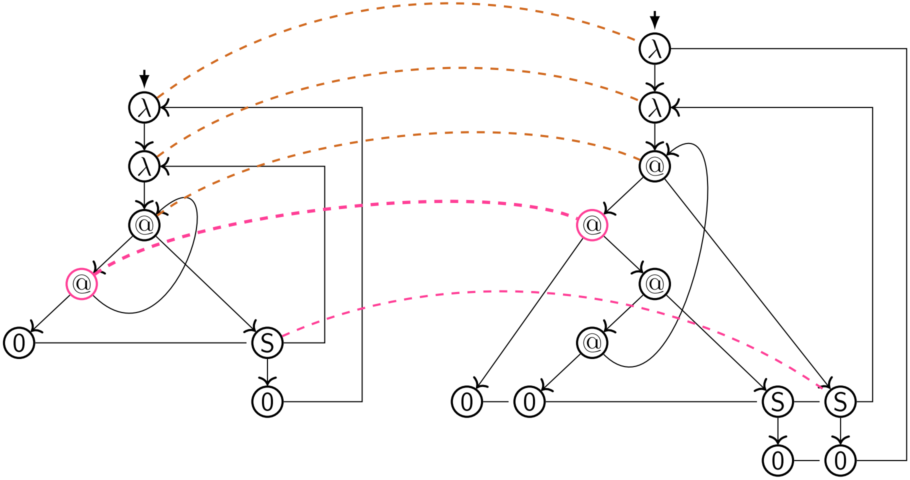
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



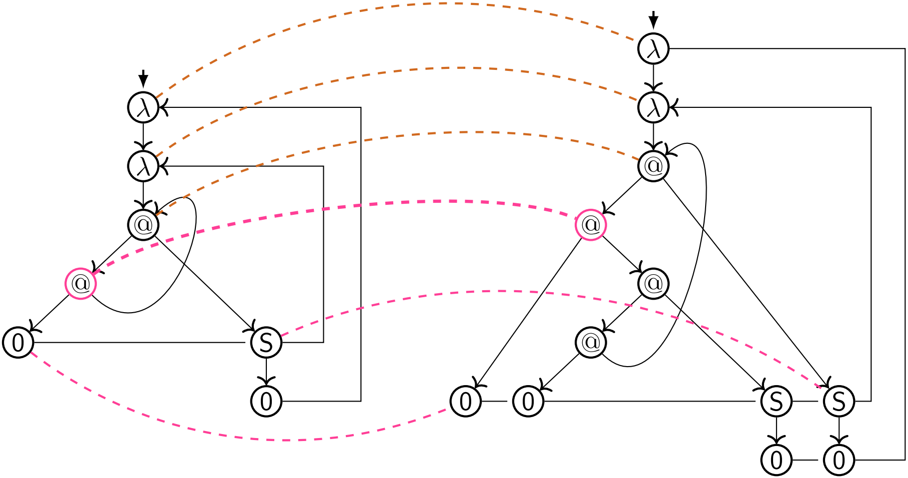
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



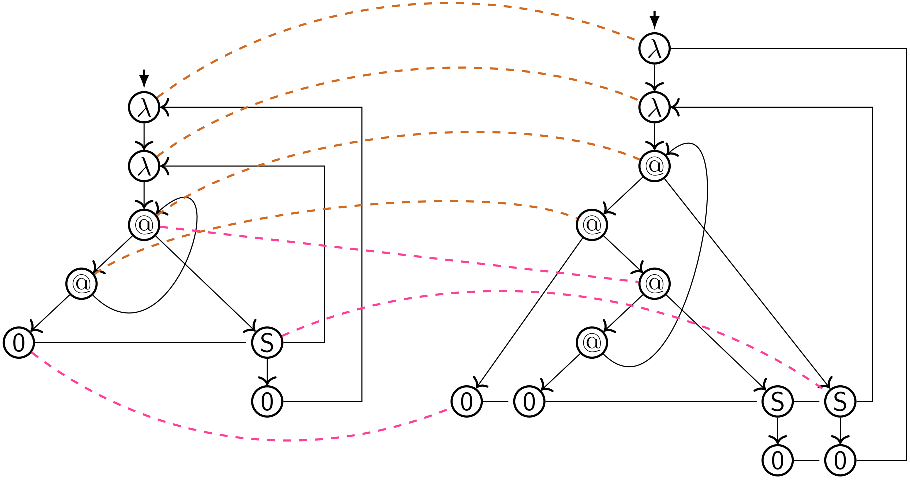
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



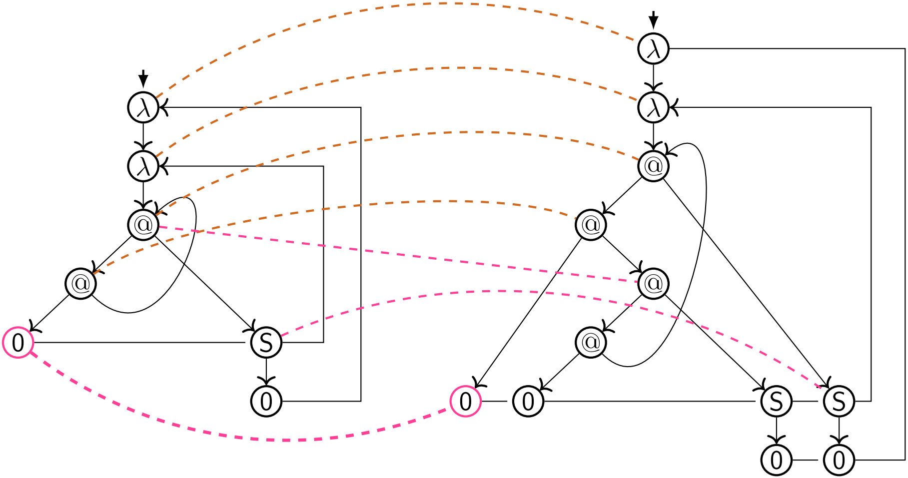
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



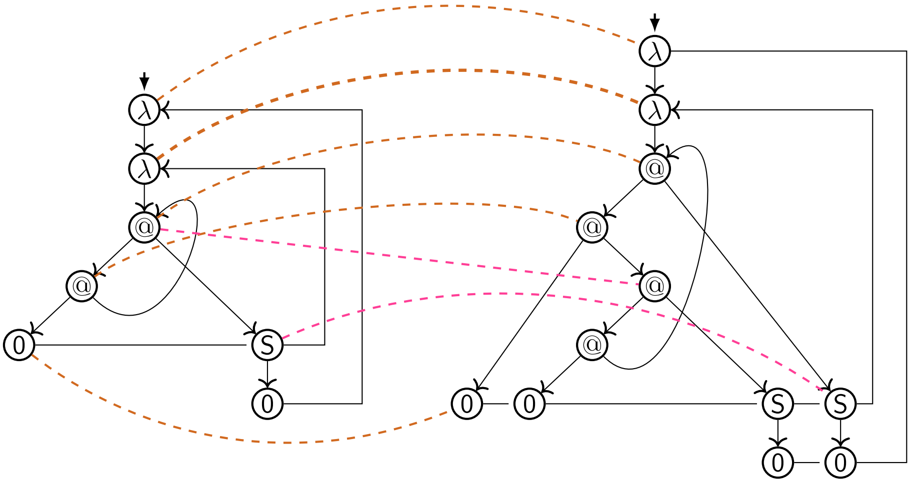
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



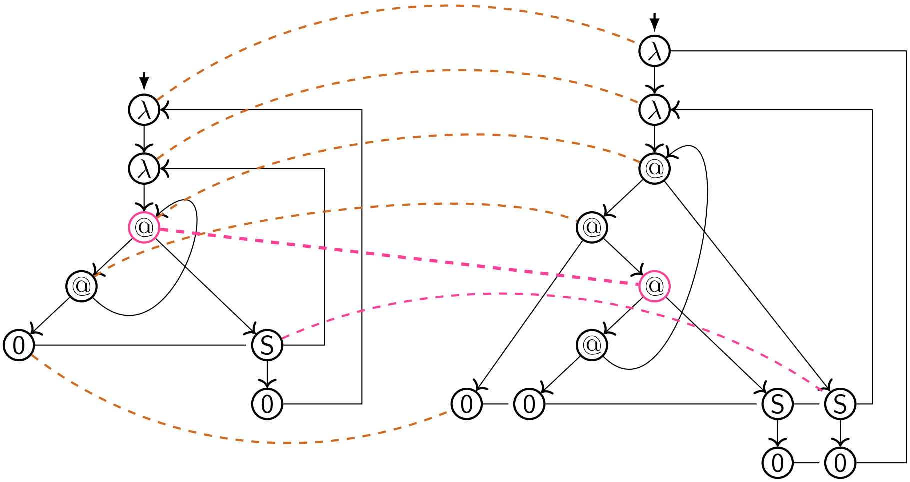
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



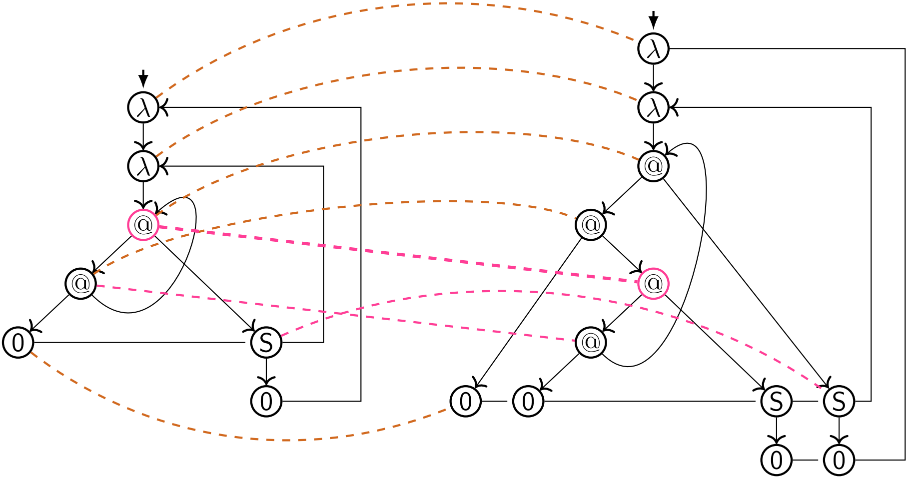
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



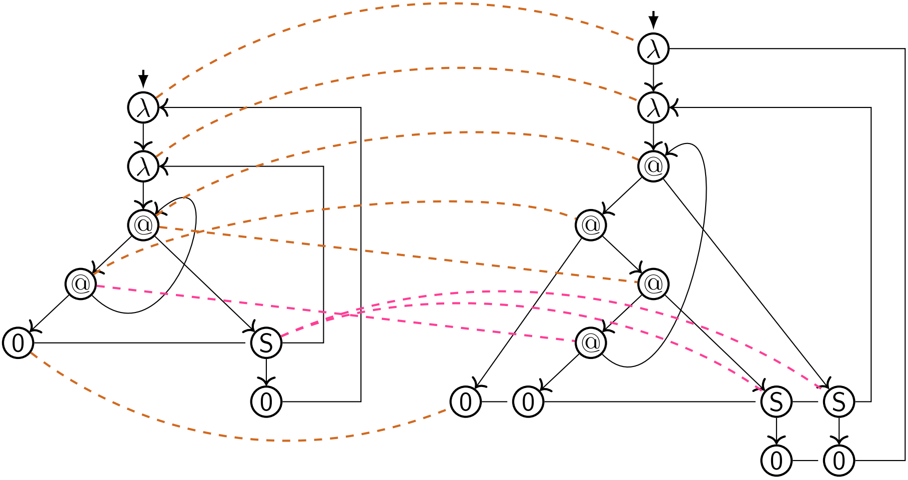
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



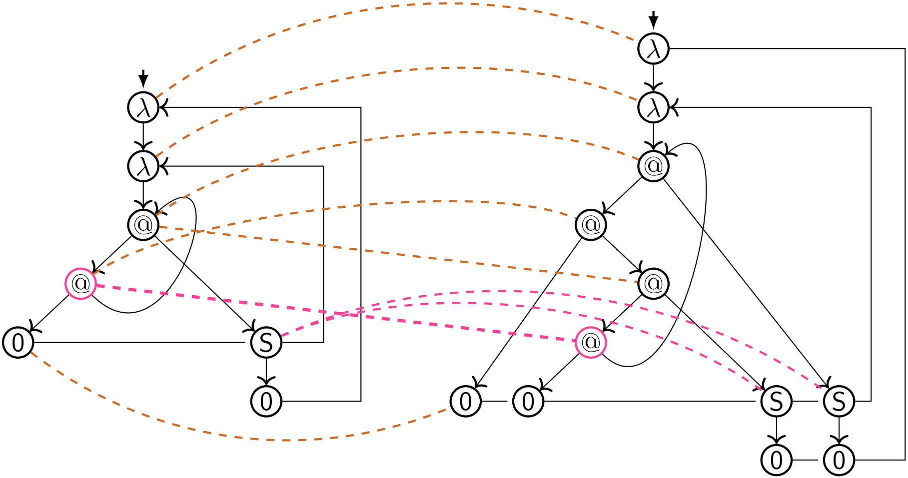
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



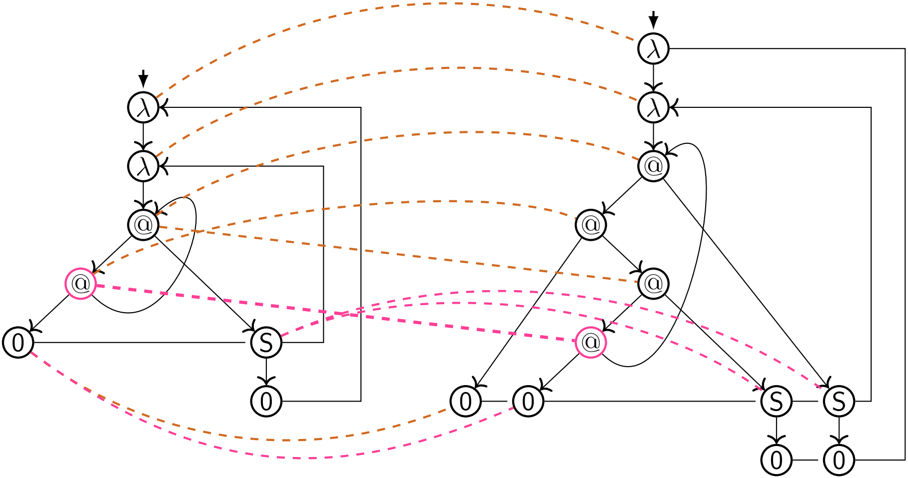
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



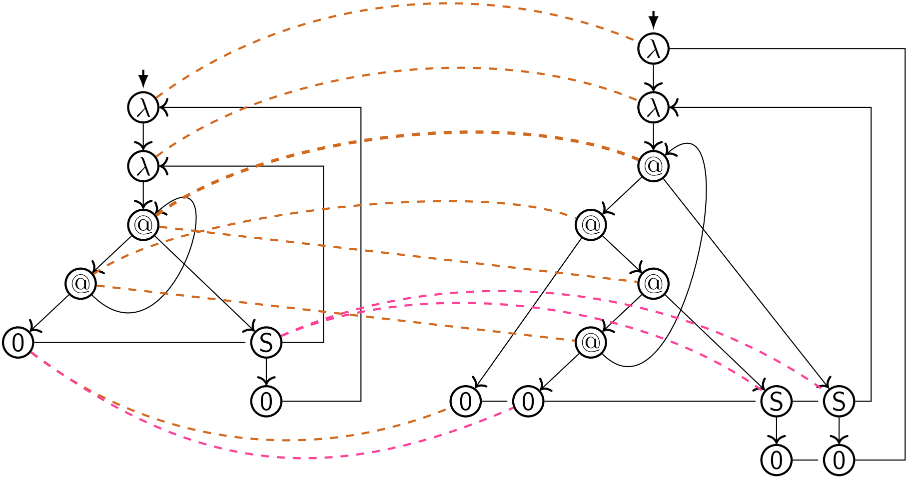
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



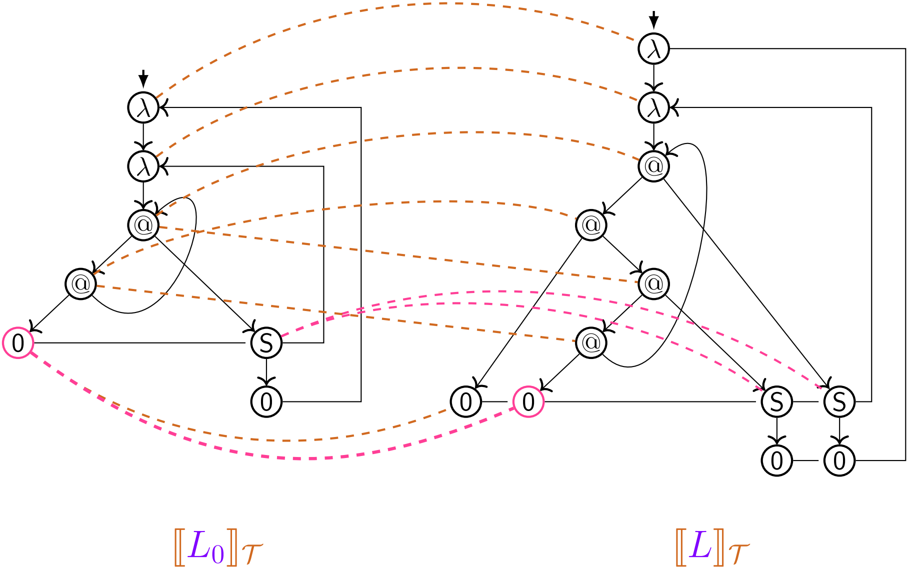
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

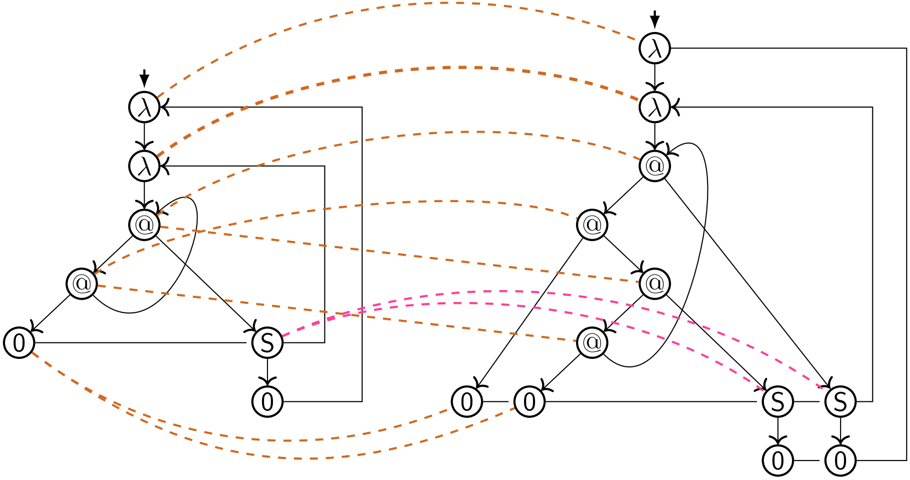


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

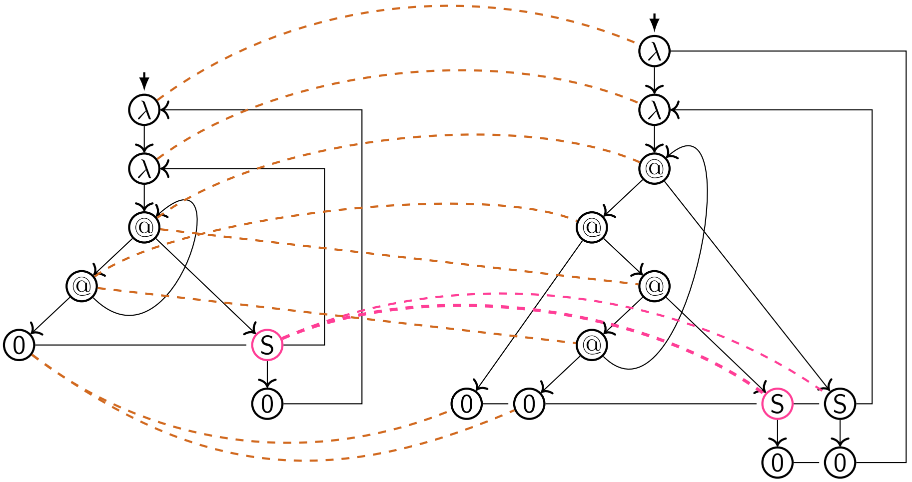


Bisimulation check between λ-term-graphs



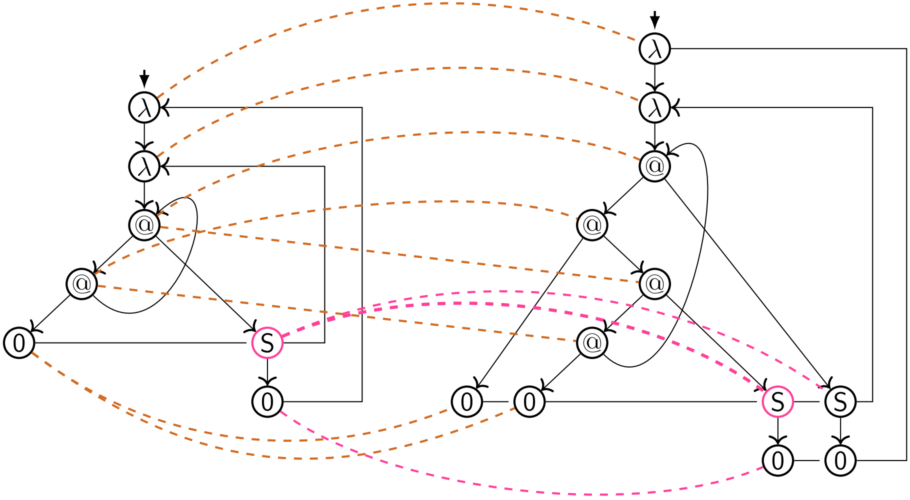
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



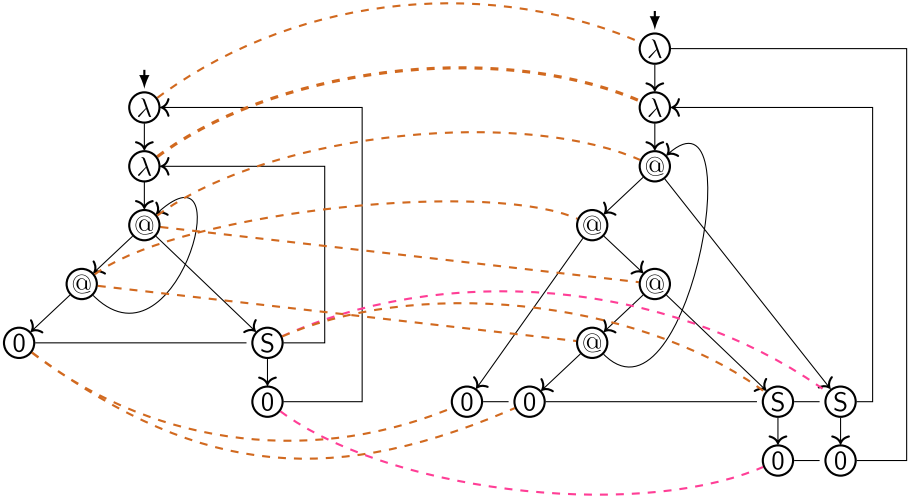
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



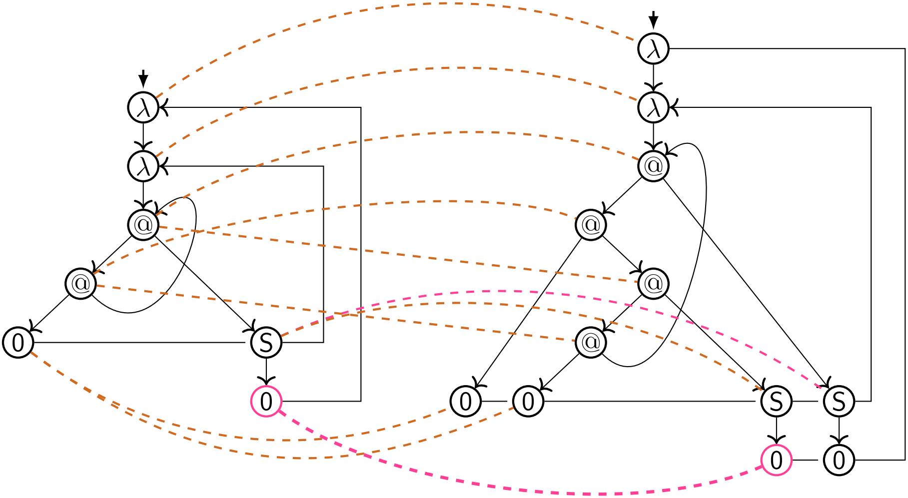
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



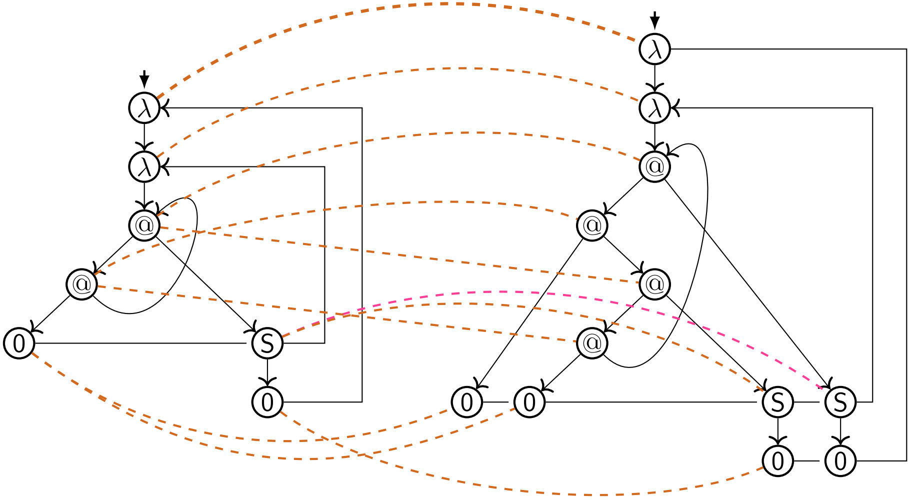
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs



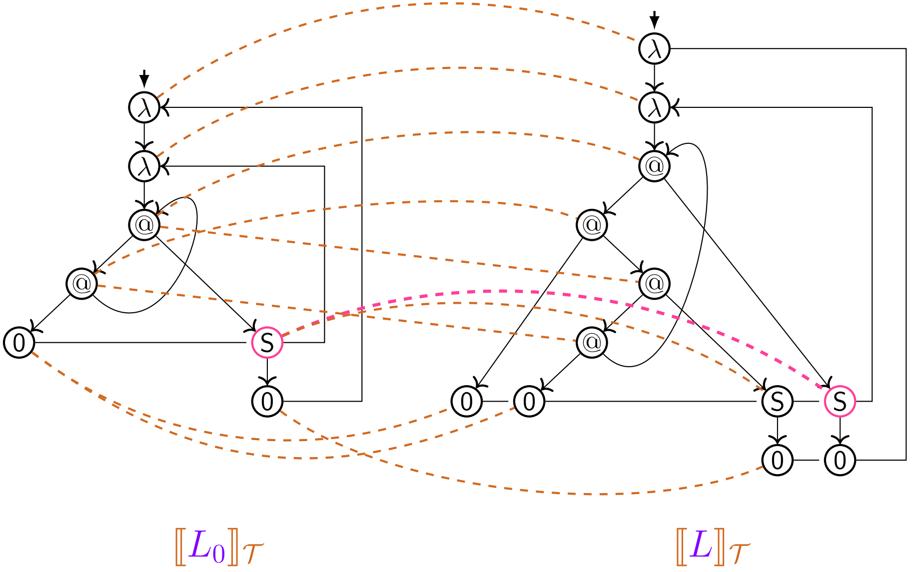
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

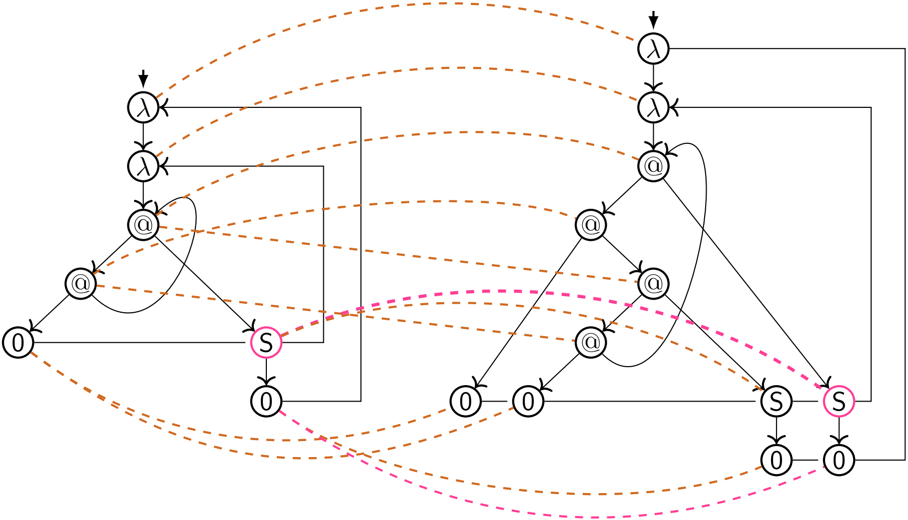


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

Bisimulation check between λ-term-graphs

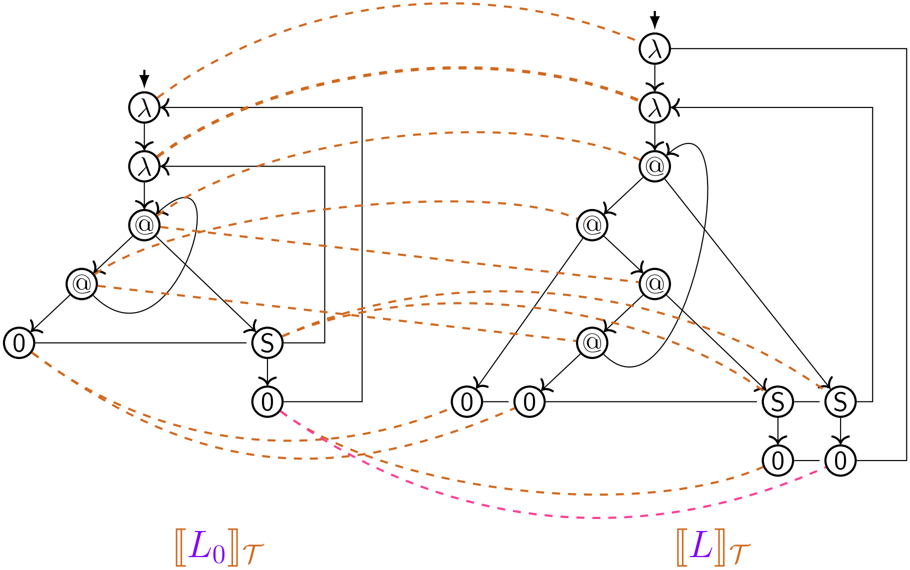


Bisimulation check between λ-term-graphs

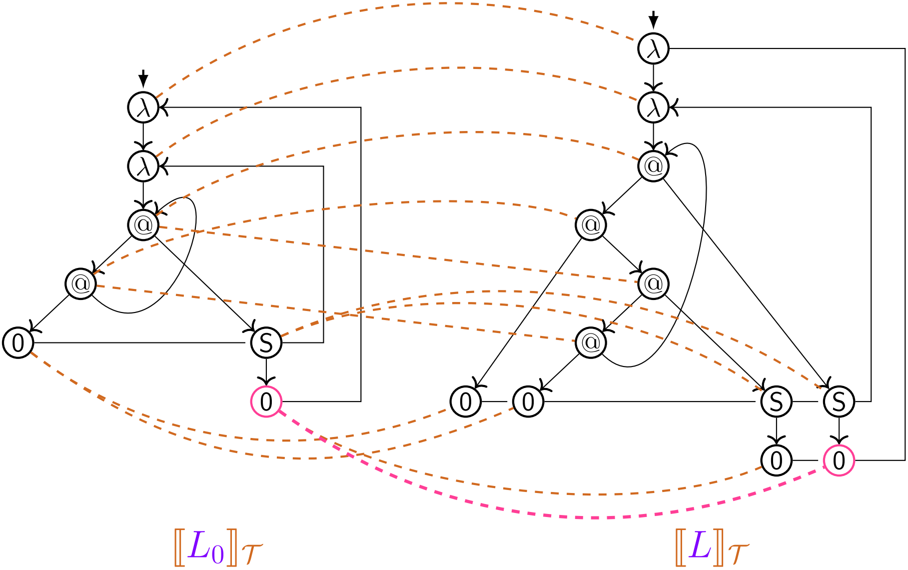


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

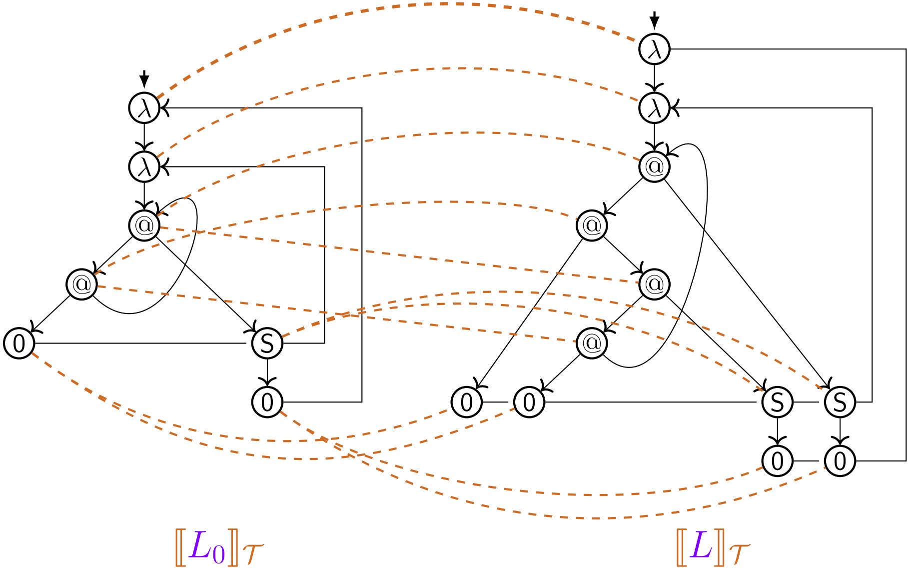
Bisimulation check between λ-term-graphs



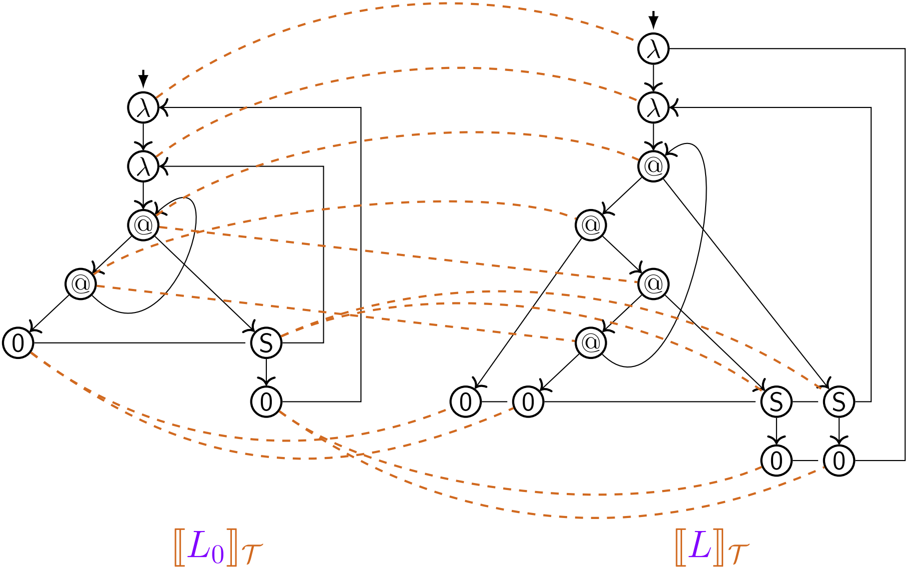
Bisimulation check between λ-term-graphs



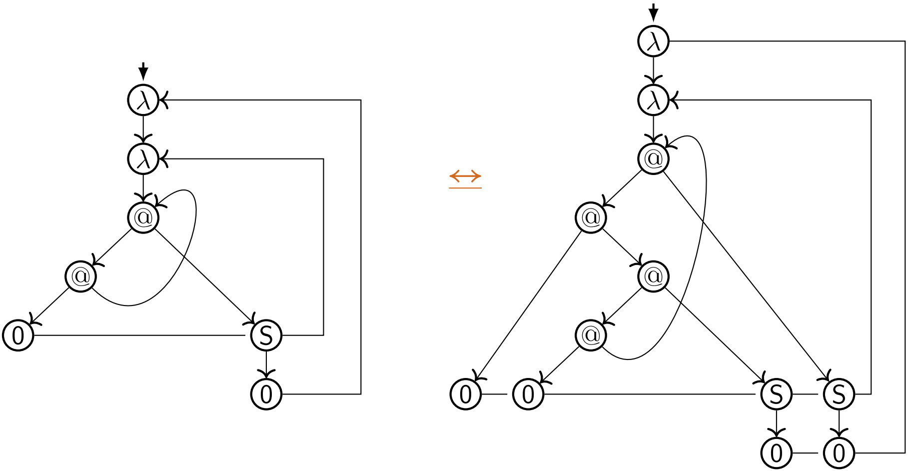
Bisimulation check between λ-term-graphs



Bisimulation between λ-term-graphs

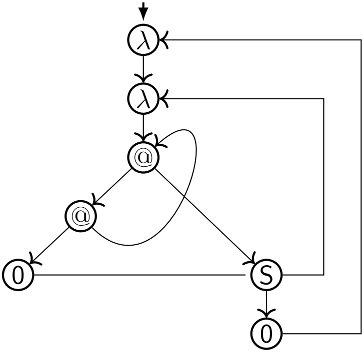


Bisimilarity between λ-term-graphs

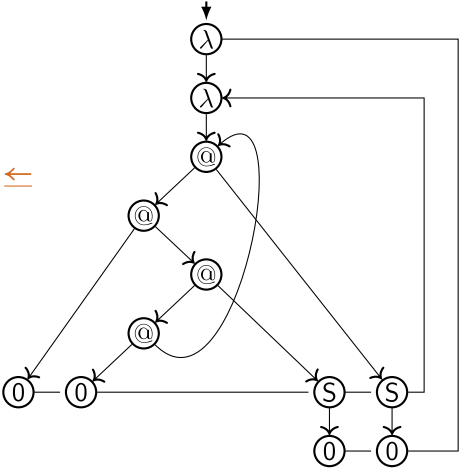


$$[\![L_0]\!]_{\mathcal{T}} \quad \underline{\leftrightarrow} \quad [\![L]\!]_{\mathcal{T}}$$

Functional bisimilarity and bisimulation collapse



$\Leftarrow$



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

Bisimulation collapse: property

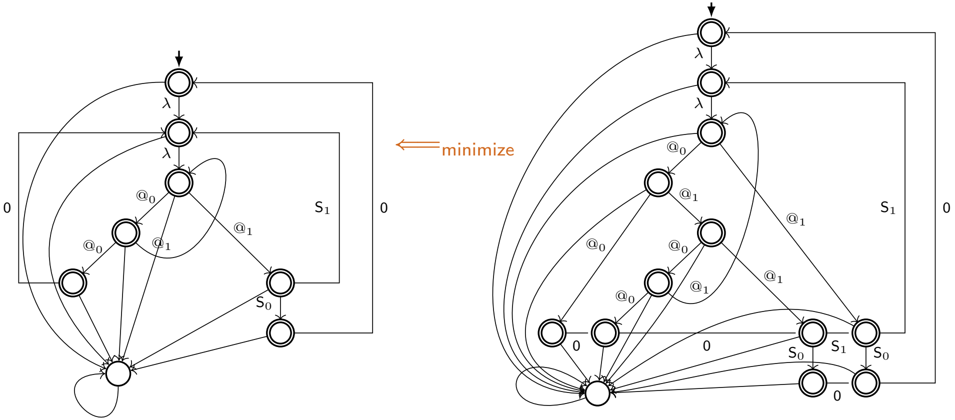
Theorem

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

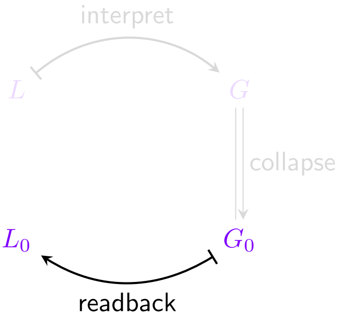
$\Longrightarrow$ For a $\boldsymbol{\lambda}_{\text{letrec}}$-term L

the bisimulation collapse of $[\![L]\!]_{\mathcal{T}}$ is again an eager-scope λ-term-graph.

λ-DFA-Minimization



Readback



Readback

defined with property:

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

Readback

defined with property:

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

Readback

defined with property:

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

Theorem

For all eager-scope λ-term-graphs G:

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

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

Readback

defined with property:

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

Theorem

For all eager-scope λ-term-graphs G:

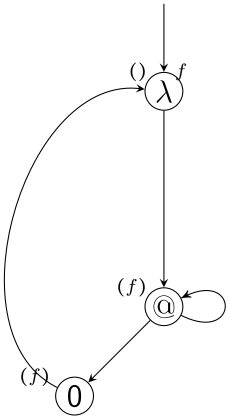
$$([\![\cdot]\!]_{\mathcal{T}} \circ \mathsf{rb})(G) \simeq G$$

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

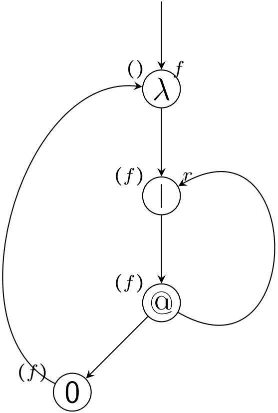
idea:

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

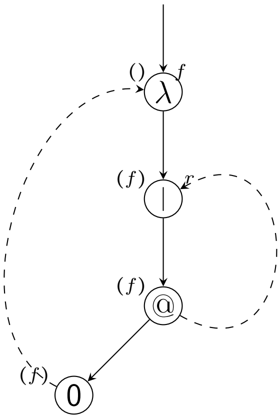
Readback: example (fix)



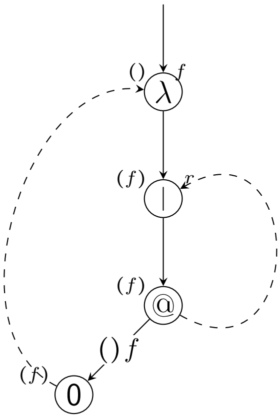
Readback: example (fix)



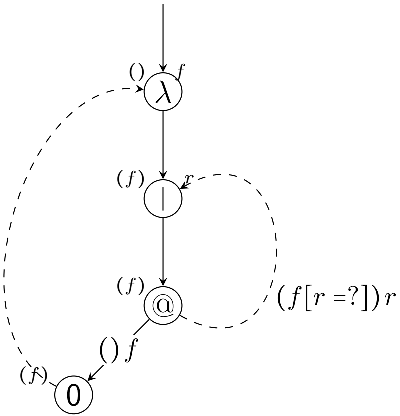
Readback: example (fix)



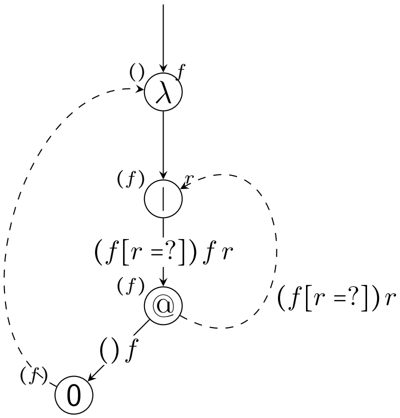
Readback: example (fix)



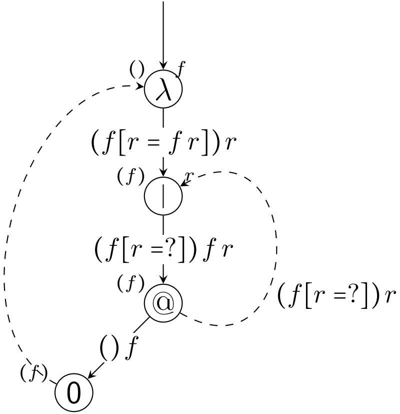
Readback: example (fix)



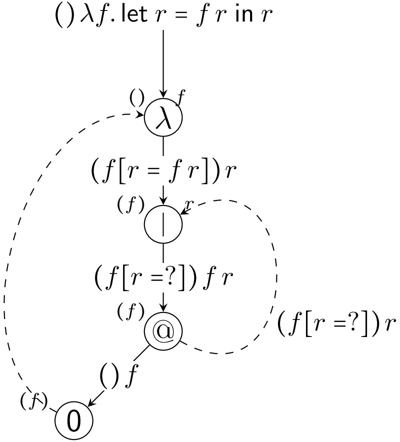
Readback: example (fix)



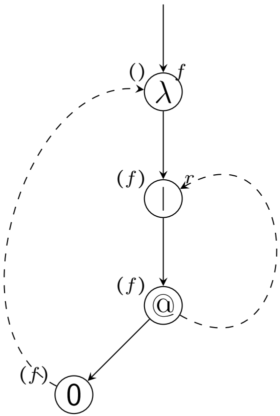
Readback: example (fix)



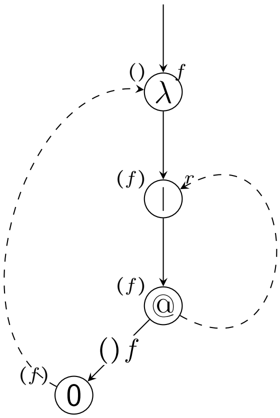
Readback: example (fix)

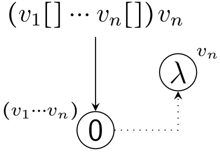


readback: example (fix)

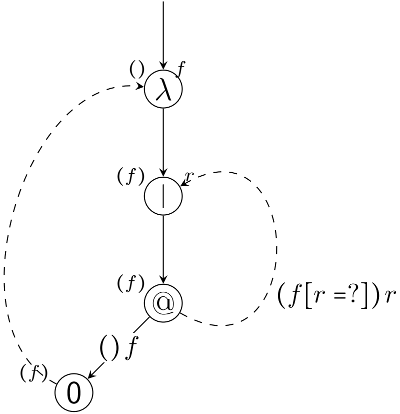


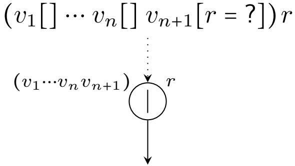
readback: example (fix)



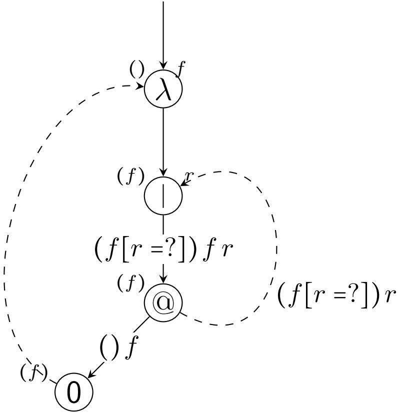


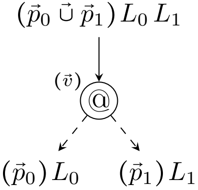
readback: example (fix)



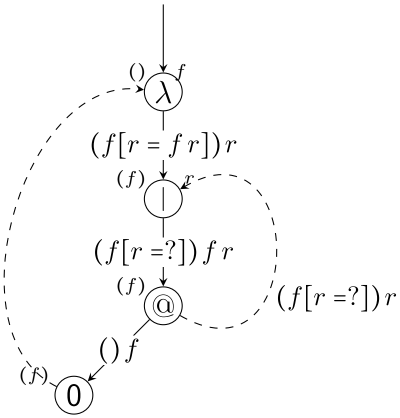


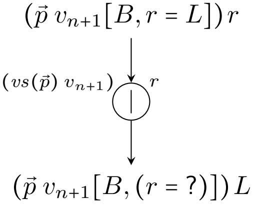
readback: example (fix)



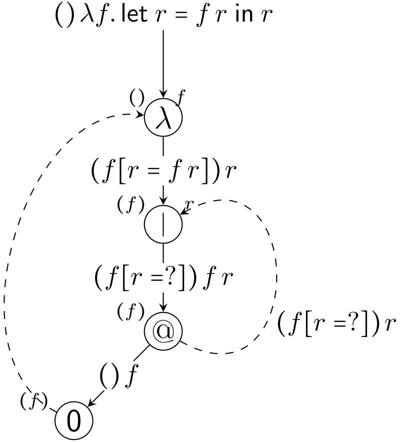


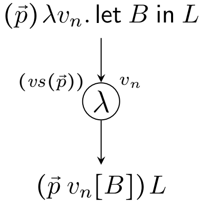
readback: example (fix)





readback: example (fix)





Maximal sharing: complexity

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

1. interpretation

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

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 $\boldsymbol{\lambda}_{\text{letrec}}$-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

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

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 $\mid\!\downarrow$

   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

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

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

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

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

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

Maximal sharing: complexity

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

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

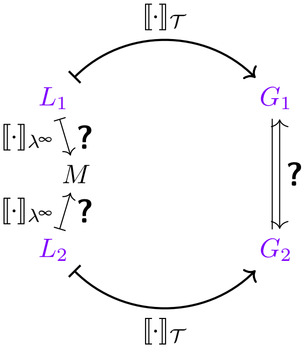
   yielding $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \mathsf{rb}(G_0)$.

   ▶ 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 $\boldsymbol{\lambda}_{\text{letrec}}$-term L requires time $O(n^2 \log n)$, where $|L| = n$.

Unfolding equivalence: complexity



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

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

1. interpretation

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$

   as λ-term-graphs $G_1 = \llbracket L_1 \rrbracket_{\mathcal{T}}$ and $G_2 = \llbracket L_2 \rrbracket_{\mathcal{T}}$

   - 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

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

1. interpretation

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$

   as λ-term-graphs $G_1 = \llbracket L_1 \rrbracket_{\mathcal{T}}$ and $G_2 = \llbracket L_2 \rrbracket_{\mathcal{T}}$

   - 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

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

1. interpretation

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$
   as λ-term-graphs $G_1 = \llbracket L_1 \rrbracket_{\mathcal{T}}$ and $G_2 = \llbracket L_2 \rrbracket_{\mathcal{T}}$

   - 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 $O(|G_i|\,\alpha(|G_i|)) = O(n^2\,\alpha(n))$

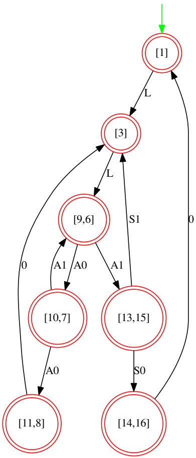
Theorem

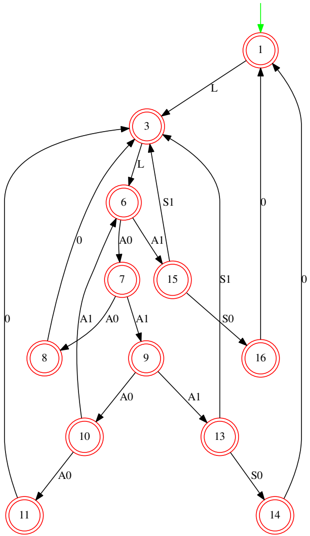
Deciding whether $\boldsymbol{\lambda}_{\text{letrec}}$-terms L_1 and L_2 are unfolding-equivalent requires almost quadratic time $O(n^2\alpha(n))$ for $n = \max\{|L_1|, |L_2|\}$.

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-DFAs





Desiderata $\rightarrow$ results: structure-constrained term graphs

$\boxed{\lambda\text{-calculus with letrec under unfolding semantics } [\![\cdot]\!]_{\lambda^\infty}}$

Not available: term graph semantics that is studied under $\underline{\leftrightarrow}$

- graph representations used by compilers were not intended for use under $\underline{\leftrightarrow}$

Desiderata $\rightarrowtriangle$ results: structure-constrained term graphs

λ-calculus with letrec under unfolding semantics $[\![\cdot]\!]_{\lambda^\infty}$

Not available: term graph semantics that is studied under $\leftrightarrow$

- graph representations used by compilers were not intended for use under $\leftrightarrow$

Desired: term graph semantics that:

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

Desiderata $\rightarrow$ results: structure-constrained term graphs

λ-calculus with letrec under unfolding semantics $[\![\cdot]\!]_{\lambda^\infty}$

Not available: term graph semantics that is studied under $\underline{\leftrightarrow}$

- graph representations used by compilers were not intended for use under $\underline{\leftrightarrow}$

Desired: term graph semantics that:

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

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

Desiderata $\multimap$ results: structure-constrained process graphs

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

Given: process graph interpretation $[\![\cdot]\!]_P$, studied under $\underline{\leftrightarrow}$

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

Desiderata $\rightharpoonup$ results: structure-constrained process graphs

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

Given: process graph interpretation $[\![\cdot]\!]_P$, studied under $\underline{\leftrightarrow}$

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

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

understand incompleteness by a structural graph property

Desiderata $\rightharpoonup$ results: structure-constrained process graphs

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

Given: process graph interpretation $[\![\cdot]\!]_{\boldsymbol{P}}$, studied under $\underline{\leftrightarrow}$

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

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

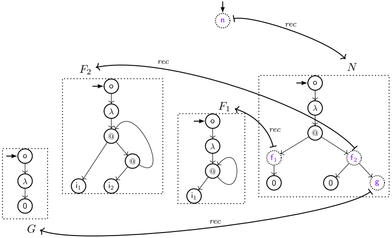
understand incompleteness by a structural graph property

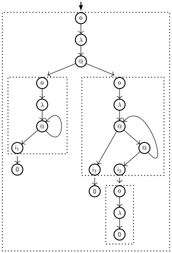
Defined: class of process graphs with LEE / (layered) LEE-witness

- 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 $[\![\cdot]\!]_{\boldsymbol{P}}^{\cancel{1r}\backslash\star}$-expressible modulo $\underline{\leftrightarrow}$

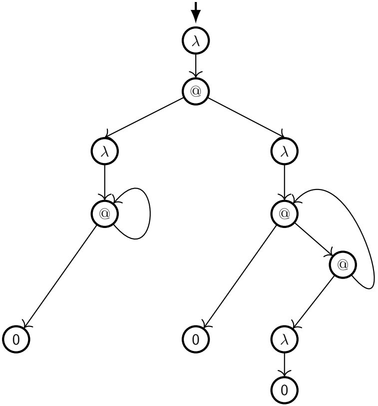
Nested Term Graphs

(joint work with Vincent van Oostrom)



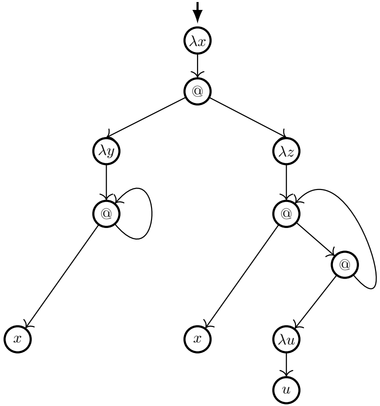


Nested scopes in λ_{letrec} terms



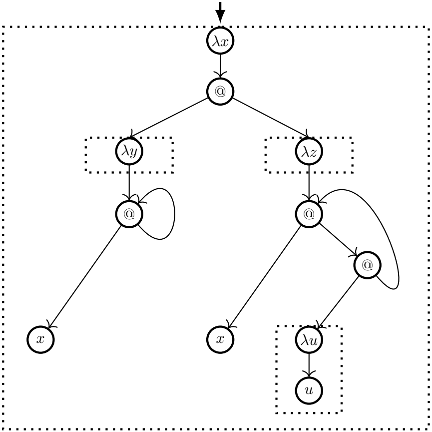
First-order term graph over $\Sigma = \{\lambda/1,\ @/2,\ 0/0\}$

Nested scopes in λ_{letrec} terms



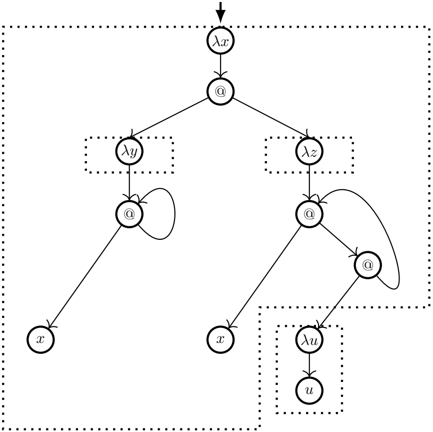
$$\lambda x.\,(\lambda y.\,\mathsf{let}\ \alpha = x\,\alpha\ \mathsf{in}\ \alpha)\,(\lambda z.\,\mathsf{let}\ \beta = x\,(\lambda u.\,u)\,\beta\ \mathsf{in}\ \beta)$$

Nested scopes in λ_{letrec} terms



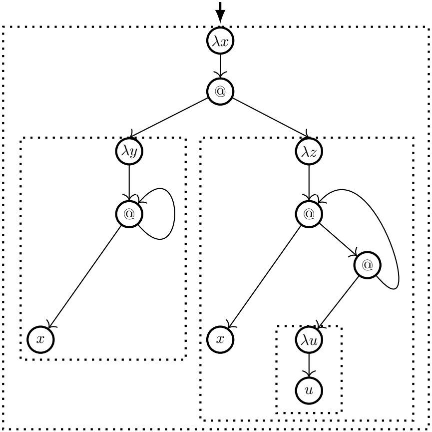
$$\lambda x.\,(\lambda y.\,\mathsf{let}\ \alpha = x\,\alpha\ \mathsf{in}\ \alpha)\,(\lambda z.\,\mathsf{let}\ \beta = x\,(\lambda u.\,u)\,\beta\ \mathsf{in}\ \beta)$$

Nested scopes in λ_{letrec} terms



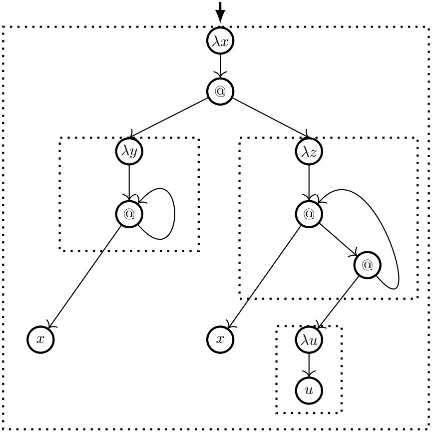
$$\lambda x.\,(\lambda y.\,\mathsf{let}\ \alpha = x\,\alpha\ \mathsf{in}\ \alpha)\,(\lambda z.\,\mathsf{let}\ \beta = x\,(\lambda u.\,u)\,\beta\ \mathsf{in}\ \beta)$$

Nested scopes in λ_{letrec} terms



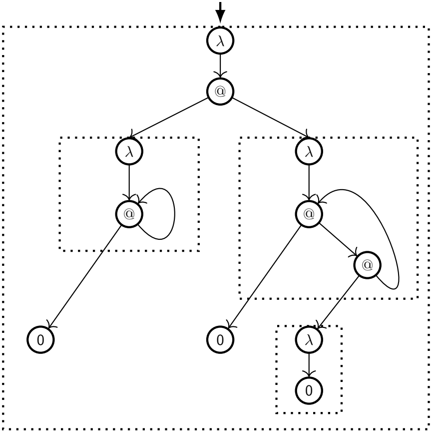
$$\lambda x.\,(\lambda y.\,\text{let } \alpha = x\,\alpha \text{ in } \alpha)\,(\lambda z.\,\text{let } \beta = x\,(\lambda u.\,u)\,\beta \text{ in } \beta)$$

Nested scopes in λ_{letrec} terms



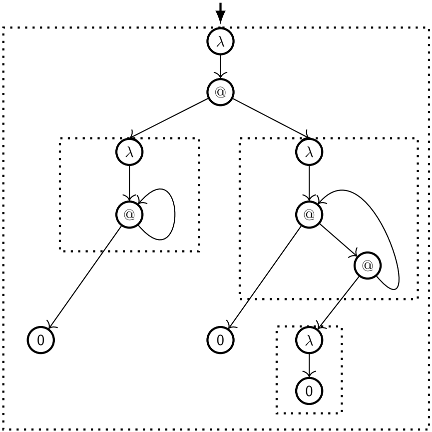
$$\lambda x.\,(\lambda y.\,\text{let } \alpha = x\,\alpha \text{ in } \alpha)\,(\lambda z.\,\text{let } \beta = x\,(\lambda u.\,u)\,\beta \text{ in } \beta)$$

Nested scopes in λ_{letrec} terms

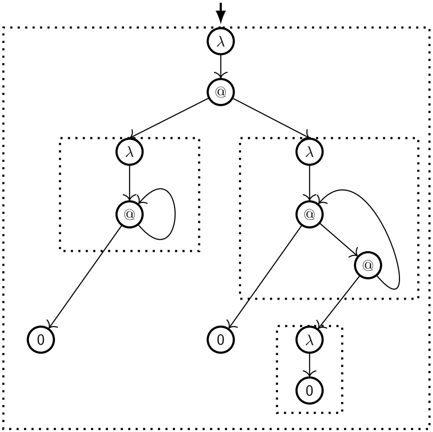


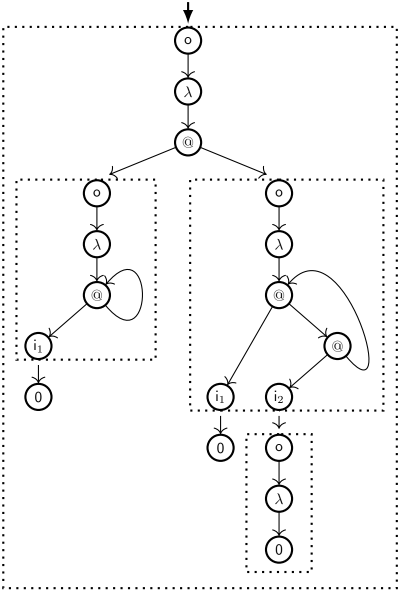
$$\lambda x.\,(\lambda y.\,\mathsf{let}\ \alpha = x\,\alpha\ \mathsf{in}\ \alpha)\,(\lambda z.\,\mathsf{let}\ \beta = x\,(\lambda u.\,u)\,\beta\ \mathsf{in}\ \beta)$$

Nested scopes in λ-terms



Nested scopes ⊸ nested term graph





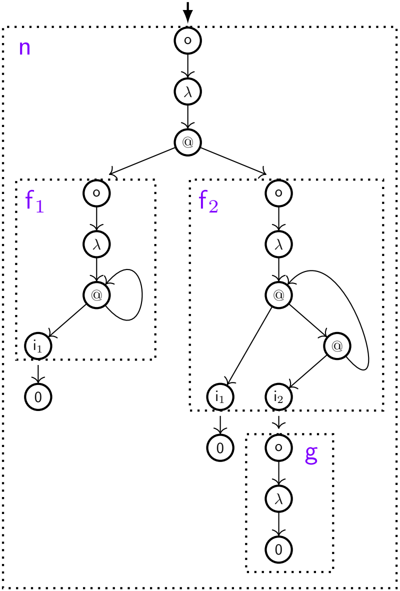
nested term graph

gletrec

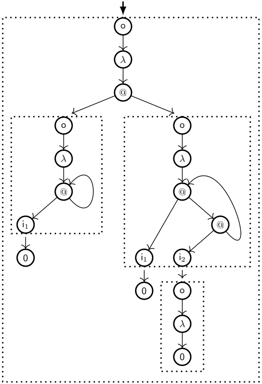
$$
\begin{aligned}
\mathsf{n}() &= \lambda x.\mathsf{f}_1(x)\mathsf{f}_2(x,\mathsf{g}())\\
\mathsf{f}_1(X_1) &= \lambda x.\text{let}\,\alpha = X_1\alpha\,\text{in}\,\alpha\\
\mathsf{f}_2(X_1,X_2) &= \lambda y.\text{let}\,\beta = X_1(X_2\beta)\,\text{in}\,\beta\\
\mathsf{g}() &= \lambda z.z
\end{aligned}
$$

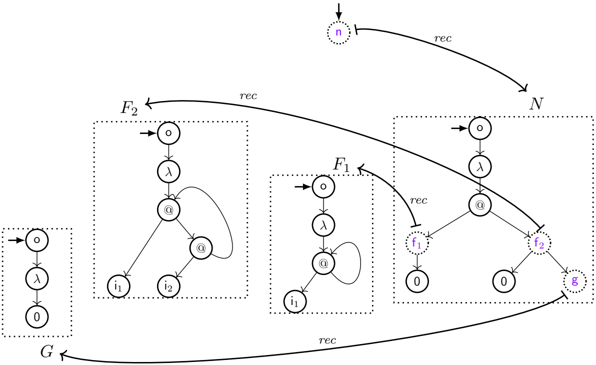
in

$$\mathsf{n}()$$



nested term graph





Signature

A *signature for nested term graphs* (*ntg-signature*) is a signature Σ that is partitioned into:

- *atomic* symbol alphabet Σ_{at}
- *nested* symbol alphabet Σ_{ne}

Additionally used:

- *interface* symbols alphabet $OI = O \cup I$
  - $O = \{\mathsf{o}\}$ with o unary
  - $I = \{\mathsf{i}_1, \mathsf{i}_2, \mathsf{i}_3, \ldots\}$ with i_j nullary

Recursive graph specification

Definition

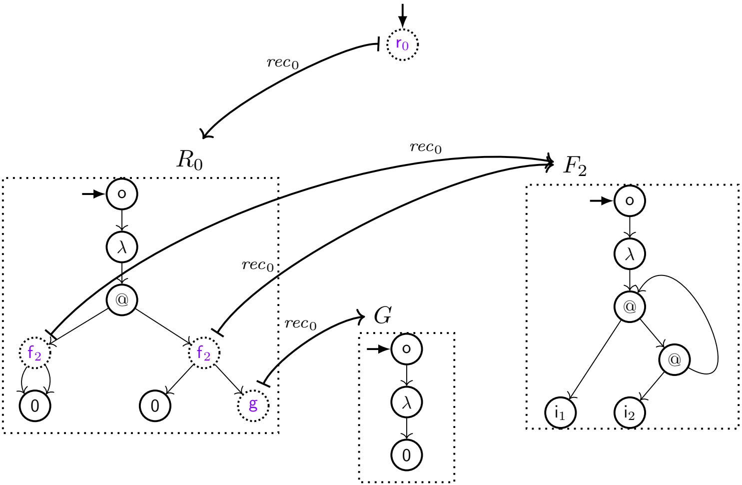
Let Σ be an ntg-signature.
A *recursive graph specification* (a *rgs*) $\mathcal{R} = \langle rec, r \rangle$ consists of:

- *specification function*

$$rec : \Sigma_{\mathsf{ne}} \longrightarrow \mathsf{TG}(\Sigma \cup OI)$$
$$f_{/k} \longmapsto rec(f) = F \in \mathsf{TG}(\Sigma \cup \{\mathsf{o}, \mathsf{i}_1, \ldots, \mathsf{i}_k\})$$

  where F contains precisely one vertex labeled by o, the root, and one vertex each labeled by i_j, for $j \in \{1, \ldots, k\}$;
- nullary *root symbol* $r \in \Sigma_{\mathsf{ne}}$.

Recursive graph specification



$\Sigma_{\mathsf{at}} = \{\lambda/1,\ @/2,\ \mathsf{0}/0\},\ \Sigma_{\mathsf{ne}} = \{\mathsf{r}_0/0,\ \mathsf{f}_2/2,\ \mathsf{g}/0\},\ O = \{\mathsf{o}/1\},$
$I = \{\mathsf{i}_1/0, \mathsf{i}_2/0, \ldots\}.$

Recursive graph specification

Definition

Let Σ be an ntg-signature.
A *recursive graph specification* (a *rgs*) $\mathcal{R} = \langle rec, r \rangle$ consists of:

- *specification function*

$$rec : \Sigma_{\mathsf{ne}} \longrightarrow \mathsf{TG}(\Sigma \cup OI)$$
$$f/k \longmapsto rec(f) = F \in \mathsf{TG}(\Sigma \cup \{\mathsf{o}, \mathsf{i}_1, \dots, \mathsf{i}_k\})$$

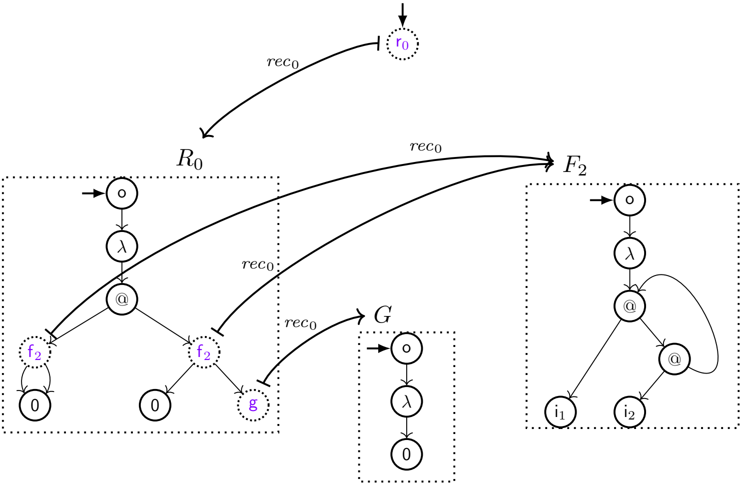
  where F contains precisely one vertex labeled by o, the root, and one vertex each labeled by i_i, for $i \in \{1, \dots, k\}$;
- nullary *root symbol* $r \in \Sigma_{\mathsf{ne}}$.

rooted *dependency* ARS $\multimap$ of $\mathcal{R}$:

- objects: nested symbols in Σ_{ne}
- steps: for all $f, g \in \Sigma_{\mathsf{ne}}$:

$$p : f \multimap g \iff g \text{ occurs in the term graph } rec(f) \text{ at position } p$$

Recursive graph specification



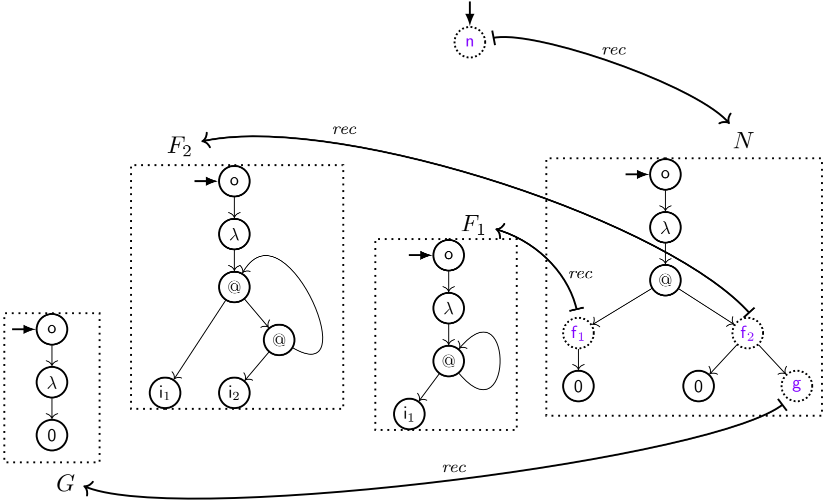
dependency ARS: $f_2 \rightrightarrows r_0 \leftarrow g$ is a dag (but not a tree).

Nested term graph: intensional definition

Definition

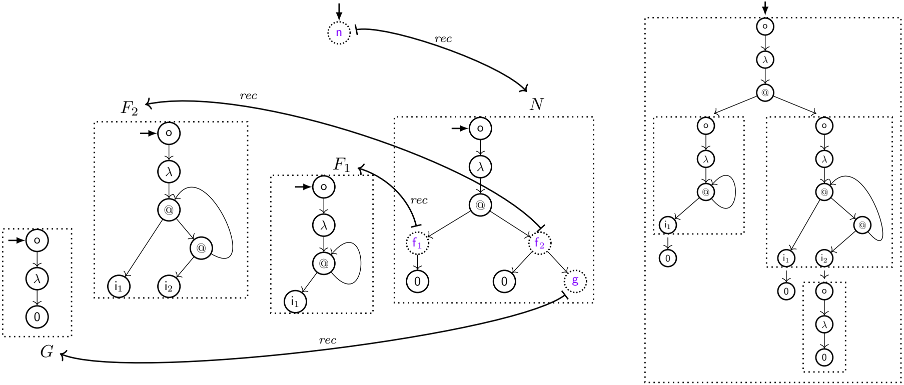
Let Σ be an ntg-signature.
A *nested term graph* over Σ is an rgs $\mathcal{N} = \langle rec, r \rangle$ such that the rooted dependency ARS $\multimapinv$ is a tree.

Nested term graph (intensionally)



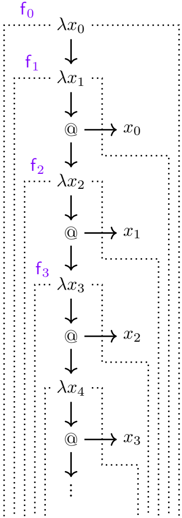
dependency ARS: $\mathsf{f}_1 \multimap \mathsf{n} \begin{array}{l} \circ\!\!- \ \mathsf{f}_2 \\ \circ\!\!- \ \mathsf{g} \end{array}$ is a tree.

Nested term graph (intensionally)



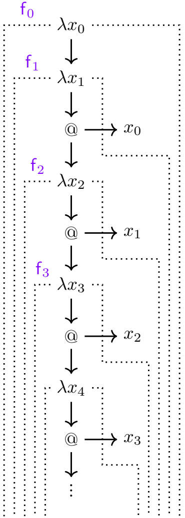
dependency ARS: $\mathsf{f}_1 \multimap \mathsf{n} \begin{array}{l} \multimapinv \mathsf{f}_2 \\ \multimapinv \mathsf{g} \end{array}$ is a tree.

Nested term graph (intensionally)

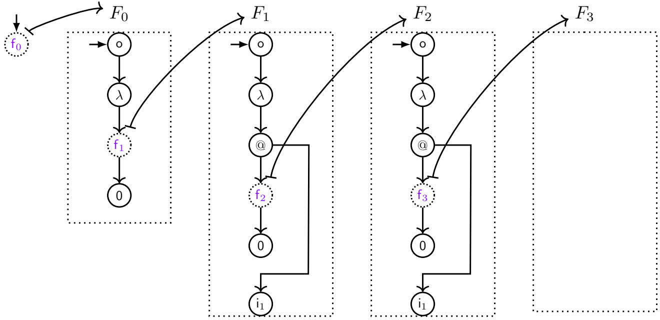


infinite λ-term
(infinitely nested scopes)

Nested term graph (intensionally)

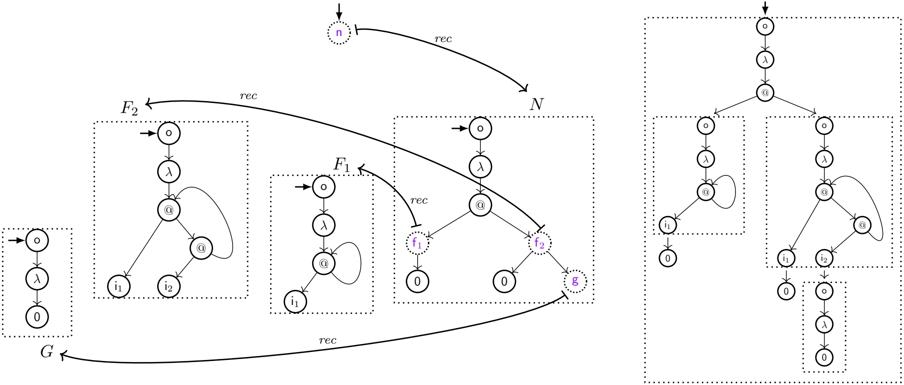


infinite λ-term
(infinitely nested scopes)

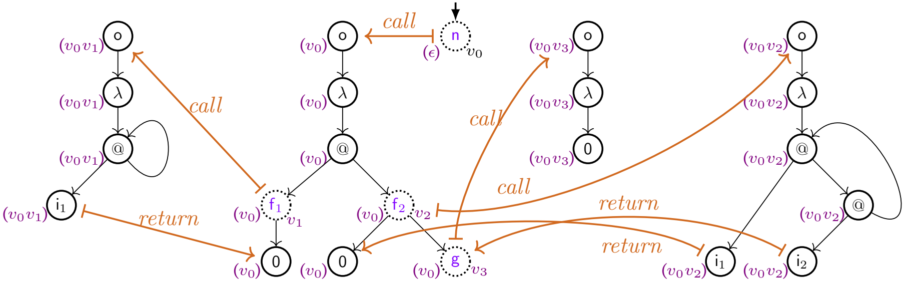


nested term graph with infinite nesting
dependency ARS: $\mathsf{f}_0 \multimapinv \mathsf{f}_1 \multimapinv \mathsf{f}_2 \multimapinv \mathsf{f}_3 \multimapinv \ldots$

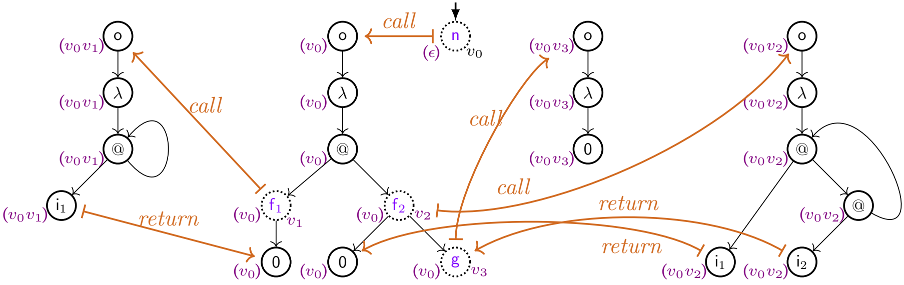
Nested term graph (intensionally)



Nested term graph: extensional definition



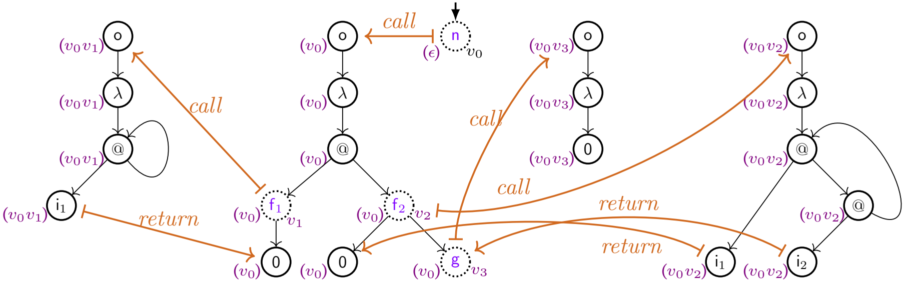
Nested term graph: extensional definition



An *extensional description* of an ntg (an *entg*) over Σ is a term graph over $\Sigma \cup OI$ (not root-connected) with vertex set V enriched by:

- *call* : $V \rightharpoonup V$, (v with nested symbol) $\mapsto$ (root of graph nested into v)

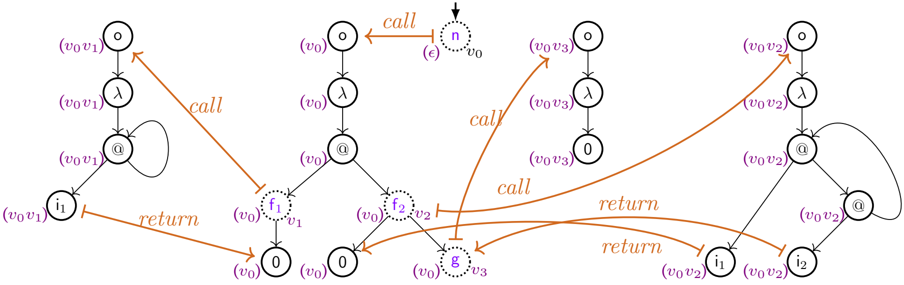
Nested term graph: extensional definition



An *extensional description* of an ntg (an *entg*) over Σ is a term graph over $\Sigma \cup OI$ (not root-connected) with vertex set V enriched by:

- $\mathit{call} : V \rightharpoonup V$, ($v$ with nested symbol) $\mapsto$ (root of graph nested into v)
- $\mathit{return} : V \rightharpoonup V$, ($v$ with output vertex i_j) $\mapsto$ (j-th successor of vertex into which the graph containing v is nested)

Nested term graph: extensional definition



An *extensional description* of an ntg (an *entg*) over Σ is a term graph over $\Sigma \cup OI$ (not root-connected) with vertex set V enriched by:

- $call : V \rightharpoonup V$, ($v$ with nested symbol) $\mapsto$ (root of graph nested into v)
- $return : V \rightharpoonup V$, ($v$ with output vertex i_j) $\mapsto$ (j-th successor of vertex into which the graph containing v is nested)
- $anc : V \to V^*$ *ancestor function*: $v \mapsto$ word $anc(v) = v_1 \cdots v_n$ of the vertices in which v is nested

Nested term graphs: intensional vs. extensional definition

Proposition

- Every nested term graph has an extensional description.
- For every entg $\mathcal{G}$ there is a nested term graph for which $\mathcal{G}$ is the extensional description.

Bisimulation

Bisimulation (for intensional ntg-definition)

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the disjoint union of the vertices of term graphs in $\mathcal{N}_1$. Similar for V_2 w.r.t. $\mathcal{N}_2$.

Bisimulation (for intensional ntg-definition)

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the disjoint union of the vertices of term graphs in $\mathcal{N}_1$. Similar for V_2 w.r.t. $\mathcal{N}_2$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are bisimilar (denoted by $\mathcal{N}_1 \leftrightarrow \mathcal{N}_2$) if there is a bisimulation between $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. a binary relation B betw. V_1 and V_2 such that:

- roots are related
- related vertices either both have nested labels, or both have interface labels, or both have the same atomic label

Bisimulation (for intensional ntg-definition)

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the disjoint union of the vertices of term graphs in $\mathcal{N}_1$. Similar for V_2 w.r.t. $\mathcal{N}_2$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are bisimilar (denoted by $\mathcal{N}_1 \leftrightarrow \mathcal{N}_2$) if there is a bisimulation between $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. a binary relation B betw. V_1 and V_2 such that:

- roots are related
- related vertices either both have nested labels, or both have interface labels, or both have the same atomic label
- progression on atomic vertices: as for f-o term graphs

Bisimulation (for intensional ntg-definition)

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the disjoint union of the vertices of term graphs in $\mathcal{N}_1$. Similar for V_2 w.r.t. $\mathcal{N}_2$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are bisimilar (denoted by $\mathcal{N}_1 \,\underline{\leftrightarrow}\, \mathcal{N}_2$) if there is a bisimulation between $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. a binary relation B betw. V_1 and V_2 such that:

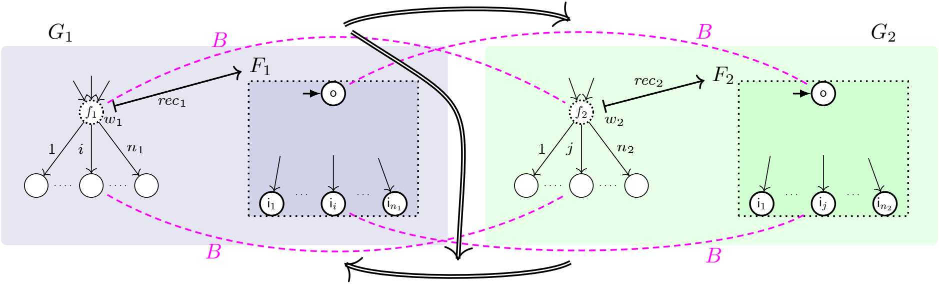
- roots are related
- related vertices either both have nested labels, or both have interface labels, or both have the same atomic label
- progression on atomic vertices: as for f-o term graphs
- progression on nested vertices: interface clause

Bisimulation (for intensional ntg-definition)

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the disjoint union of the vertices of term graphs in $\mathcal{N}_1$. Similar for V_2 w.r.t. $\mathcal{N}_2$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are bisimilar (denoted by $\mathcal{N}_1 \leftrightarrow \mathcal{N}_2$) if there is a bisimulation between $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. a binary relation B betw. V_1 and V_2 such that:

- roots are related
- related vertices either both have nested labels, or both have interface labels, or both have the same atomic label
- progression on atomic vertices: as for f-o term graphs
- progression on nested vertices: interface clause

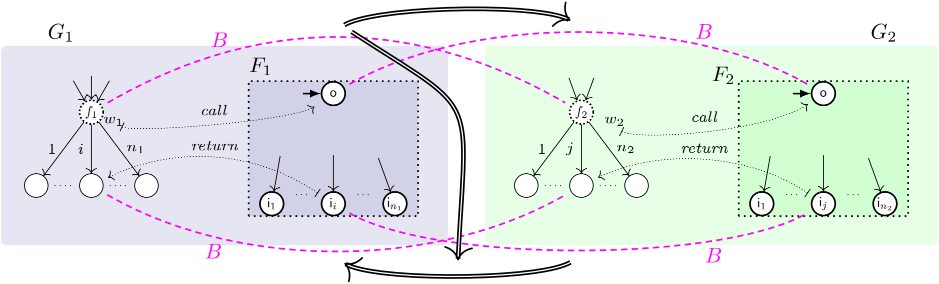


Bisimulation (for extensional ntg-definition)

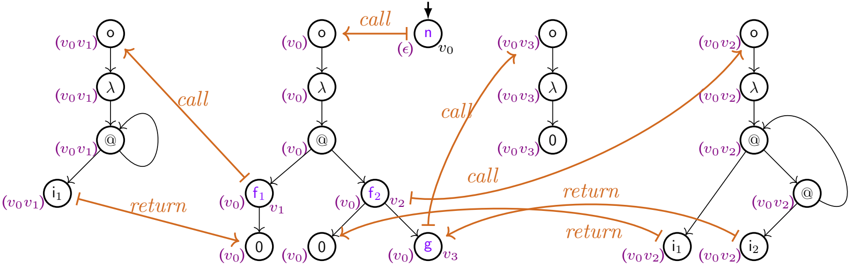
Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be nested term graphs. Let V_1 be the vertices of $\mathcal{N}_1$, and let V_2 be the vertices of $\mathcal{N}_2$.

$\mathcal{N}_1$ and $\mathcal{N}_2$ are bisimilar (denoted by $\mathcal{N}_1 \underline{\leftrightarrow} \mathcal{N}_2$) if there is a bisimulation between $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. a binary relation B betw. V_1 and V_2 such that:

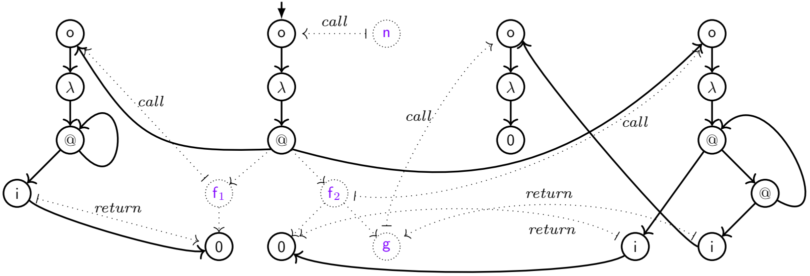
- roots are related
- related vertices either both have nested labels, or both have interface labels, or both have the same atomic label
- progression on atomic vertices: as for f-o term graphs
- progression on nested vertices: interface clause



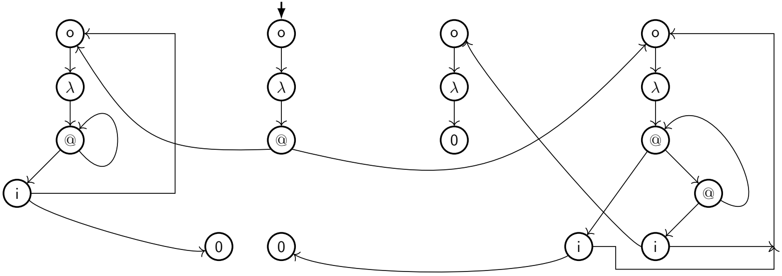
Implementation by first-order term graph (via entg)



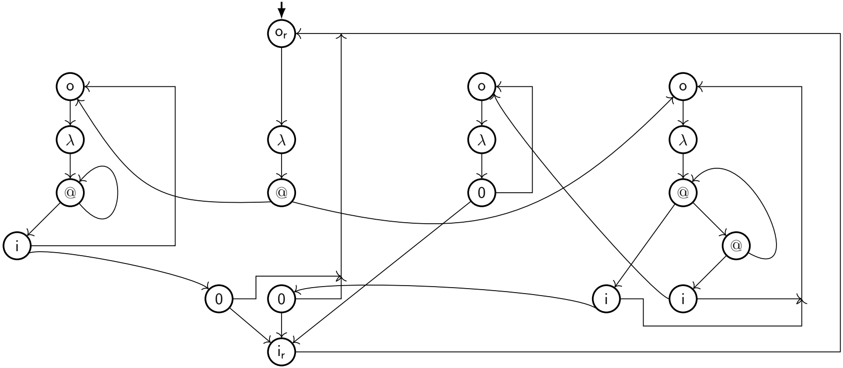
Implementation by first-order term graph (via entg)



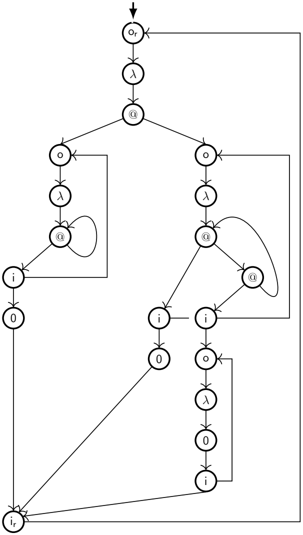
Implementation by first-order term graph (via entg)



Implementation by first-order term graph (via entg)



Implementation by first-order term graph (via entg)



Summary

- Expressibility of λ_{letrec} via unfolding
- Maximal sharing of functional programs in λ_{letrec}
- Nested term graphs

Summary

- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Characterizations of infinite λ-terms
    that are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms as:
    - strongly regular λ^{∞}-terms,
    - regular λ^{∞}-terms with finite binding–capturing chains.
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
- Nested term graphs

Summary

- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Characterizations of infinite λ-terms
    that are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms as:
    - strongly regular λ^{∞}-terms,
    - regular λ^{∞}-terms with finite binding–capturing chains.
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - Maximal compactification of $\boldsymbol{\lambda}_{\text{letrec}}$-terms
    while preserving their nested scope-structure, by:
    - formalization as (higher-/first-order) term graphs and DFAs
    - minimization / readback / complexity / Haskell implementation
- Nested term graphs

Summary

- Expressibility of $\boldsymbol{\lambda}_{\text{letrec}}$ via unfolding
  - Characterizations of infinite λ-terms
    that are unfoldings of $\boldsymbol{\lambda}_{\text{letrec}}$-terms as:
    - strongly regular λ^{∞}-terms,
    - regular λ^{∞}-terms with finite binding–capturing chains.
- Maximal sharing of functional programs in $\boldsymbol{\lambda}_{\text{letrec}}$
  - Maximal compactification of $\boldsymbol{\lambda}_{\text{letrec}}$-terms
    while preserving their nested scope-structure, by:
    - formalization as (higher-/first-order) term graphs and DFAs
    - minimization / readback / complexity / Haskell implementation
- Nested term graphs
  - Basic ideas for a general framework
    for graph representations of terms with nested scopes

Resources

- papers and reports
  - G: Modeling Terms by Graphs with Structure Constraints
    - TERMGRAPH 2018 post-proceedings in in EPTCS 288
  - G, Rochel: Maximal Sharing in the Lambda Calculus with Letrec
    - ICFP 2014 paper, extending report arXiv:1401.1460
  - G, Rochel: Term Graph Representations for Cyclic Lambda Terms
    - TERMGRAPH 2013 proceedings, report arXiv:1308.1034
  - G, Vincent van Oostrom: 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
- tools by Jan Rochel
  - maxsharing on hackage.haskell.org
  - port graph rewriting