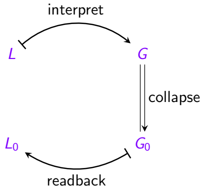
Maximal Sharing in the Lambda Calculus with letrec



Clemens Grabmayer

VU University Amsterdam (Dept. of CS)

Jan Rochel

Utrecht University (Dept. of CS)

ICFP 2014

September 1–3, 2014

motivation, questions, and results

motivation

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

questions

(1): how to maximize sharing in programs?

(2): how to check for unfolding equivalence?

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

- as abstraction & syntactical core of functional languages

our results:

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

outline

- methods consist of the steps:

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

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

  bisimilarity & bisimulation collapse of λ-term-graphs

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

- implementation
- complexity
- extensions and applications

contribution

conceptually

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

maximal sharing method

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

maximal sharing: example (fix)

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

L

maximal sharing: example (fix)

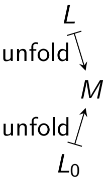
$$\lambda f.\,\text{let } r = f\,(f\ r) \text{ in } r$$

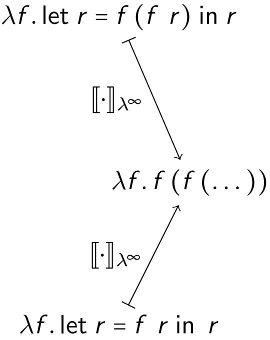
L

L_0

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

maximal sharing: the method





maximal sharing: the method

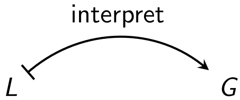
$$\lambda f.\,\text{let } r = f\,(f\ r) \text{ in } r$$

L

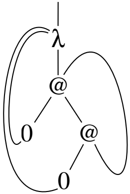
L_0

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

maximal sharing: the method



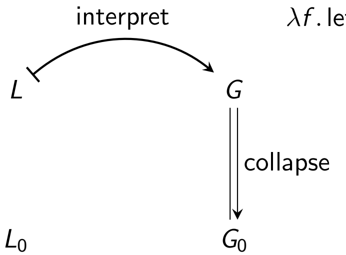
$\lambda f.\,\text{let } r = f\ (f\ \ r) \text{ in } r \quad \xrightarrow{[\![\cdot]\!]_{\mathcal{T}}}$



L_0

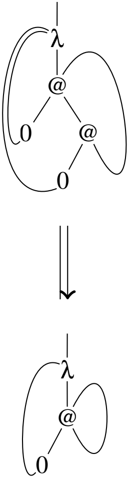
$\lambda f.\,\text{let } r = f\ \ r \text{ in } \ r$

maximal sharing: the method

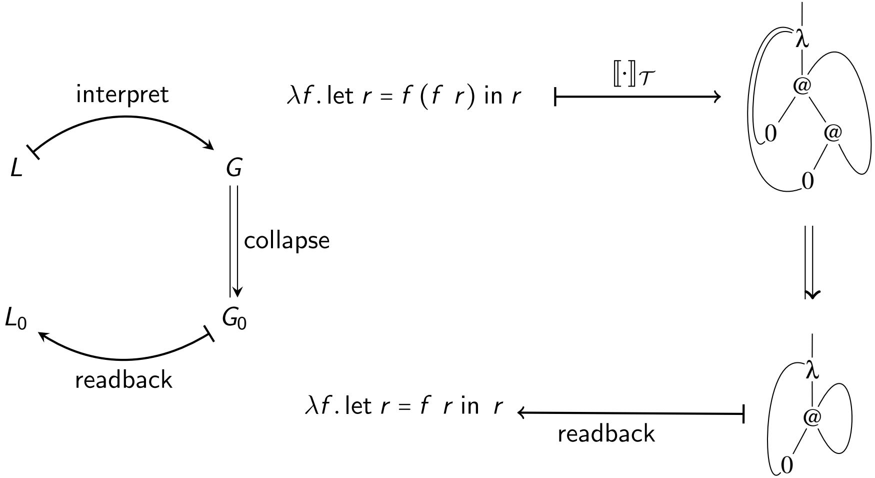


$\lambda f.\, \text{let } r = f\,(f\ r) \text{ in } r \quad \overset{[\![\cdot]\!]_{\mathcal{T}}}{\longmapsto}$

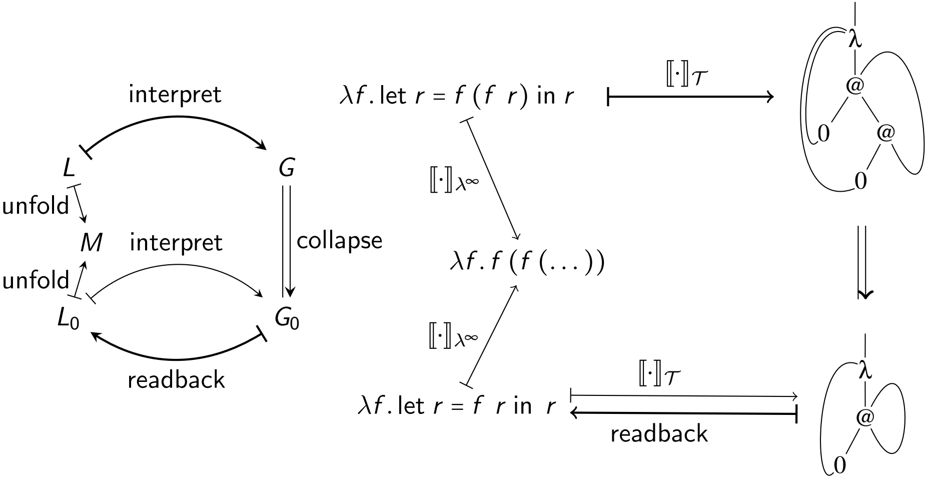
$\lambda f.\, \text{let } r = f\ r \text{ in } r$



maximal sharing: the method



maximal sharing: the method



maximal sharing: the method

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

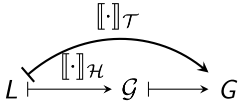
1. term graph interpretation $[\![\cdot]\!]$. of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$

maximal sharing: the method

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

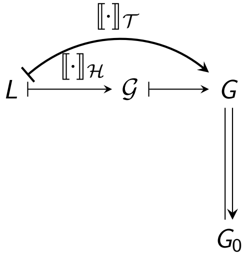
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$

maximal sharing: the method



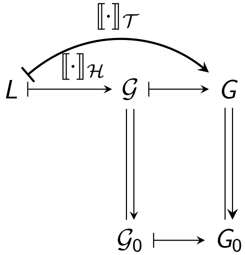
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$

maximal sharing: the method



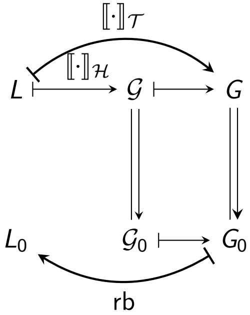
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G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

maximal sharing: the method



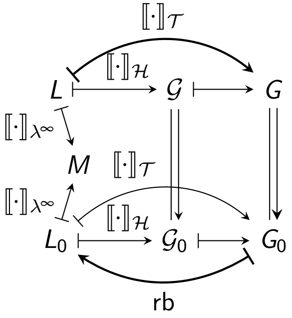
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\!\downarrow$
   of f-o term graph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

maximal sharing: the method



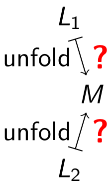
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G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

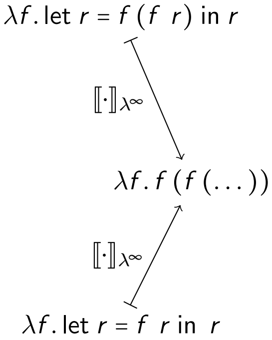
maximal sharing: the method



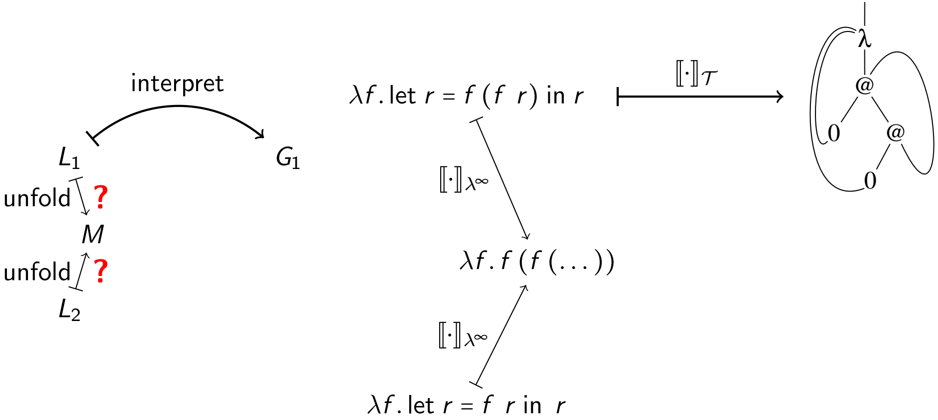
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L as:
   a. higher-order term graph
      $\mathcal{G} = [\![L]\!]_{\mathcal{H}}$
   b. first-order term graph $G = [\![L]\!]_{\mathcal{T}}$
2. bisimulation collapse $|\!\downarrow$
   of f-o term graph G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

unfolding equivalence: example

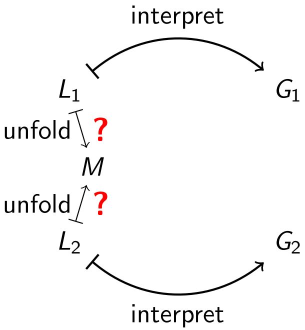


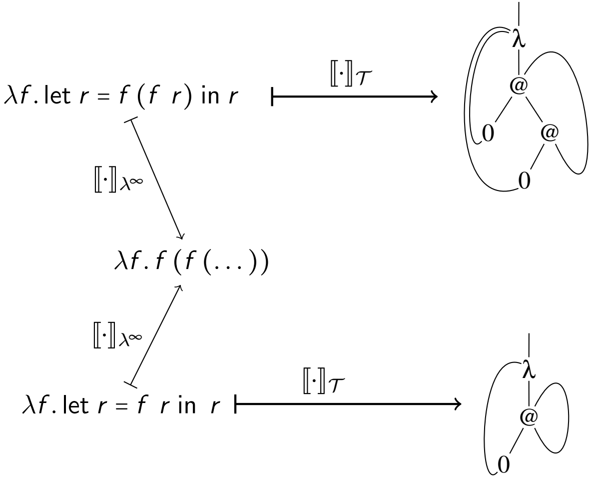


unfolding equivalence: example

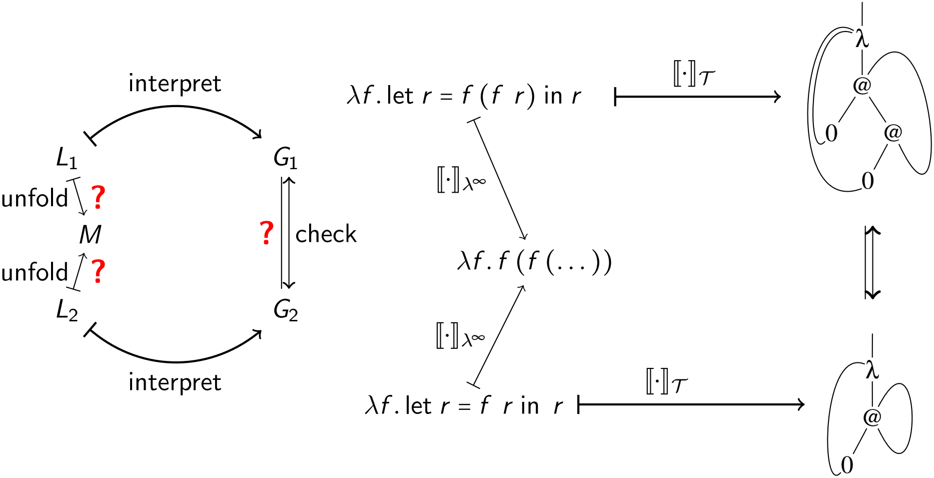


unfolding equivalence: the method

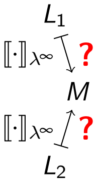




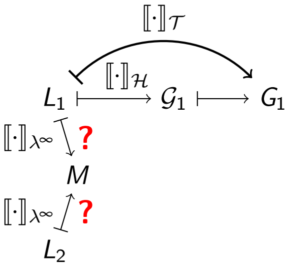
unfolding equivalence: the method



unfolding equivalence: the method

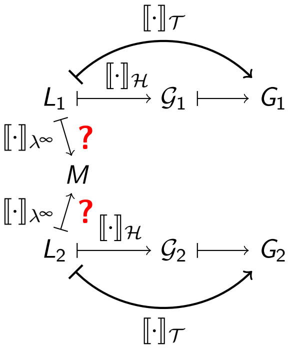


unfolding equivalence: the method



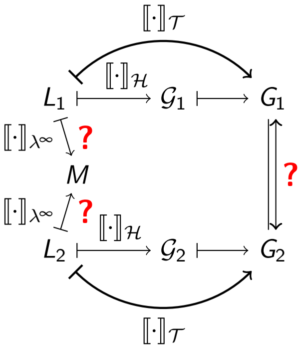
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:
   a. higher-order term graphs
      $\mathcal{G}_1 = [\![L_1]\!]_{\mathcal{H}}$
   b. first-order term graphs
      $G_1 = [\![L_1]\!]_{\mathcal{T}}$

unfolding equivalence: the method



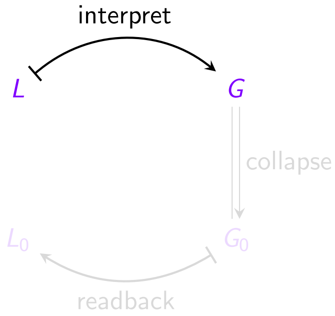
1. term graph interpretation $[\![\cdot]\!]$.
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1 and L_2 as:
   a. higher-order term graphs
      $\mathcal{G}_1 = [\![L_1]\!]_{\mathcal{H}}$ and $\mathcal{G}_2 = [\![L_2]\!]_{\mathcal{H}}$
   b. first-order term graphs
      $G_1 = [\![L_1]\!]_{\mathcal{T}}$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

unfolding equivalence: the method



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

interpretation



running example

instead of:

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

we use:

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

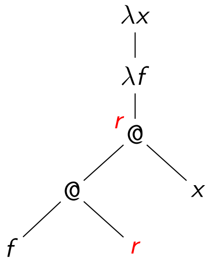
$L \quad \longmapsto_{\text{max-sharing}} \quad L_0$

graph interpretation (example 1)

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

graph interpretation (example 1)

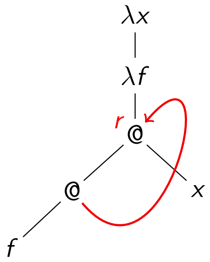
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree

graph interpretation (example 1)

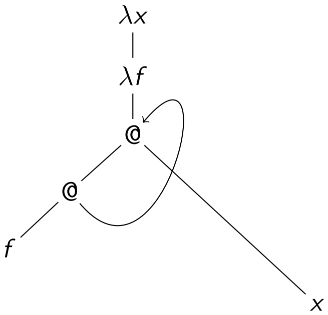
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 1)

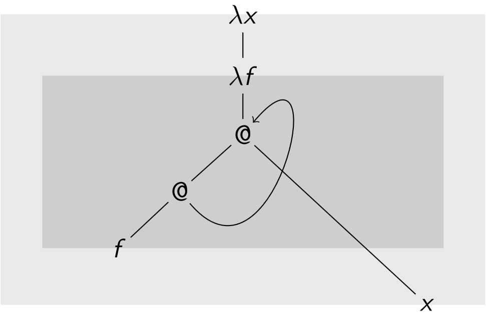
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 1)

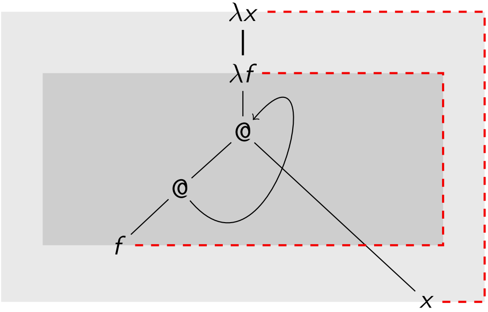
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

graph interpretation (example 1)

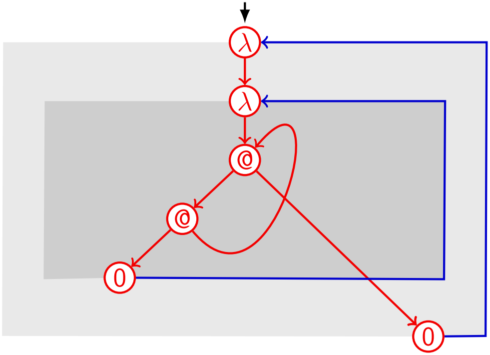
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\, r\, x \text{ in } r$



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

graph interpretation (example 1)

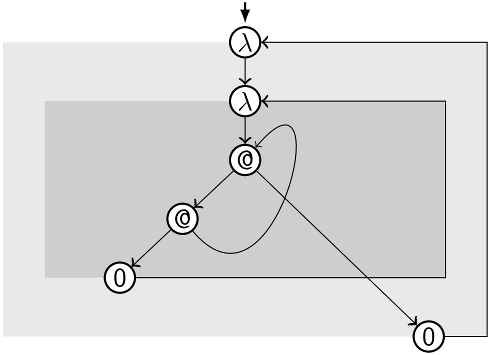
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

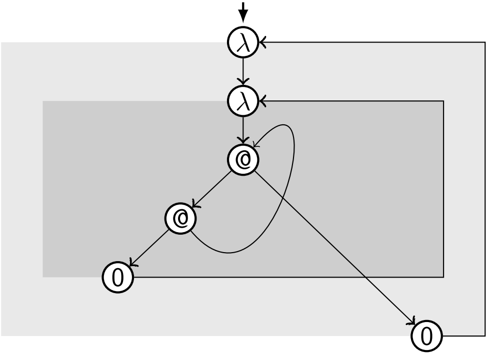
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

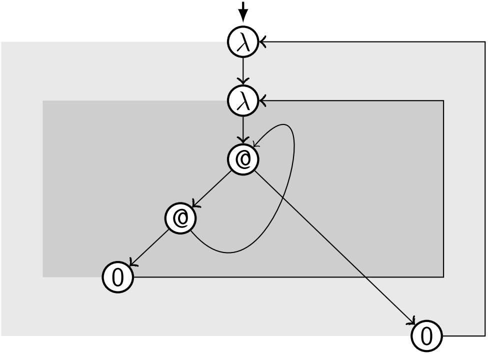
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

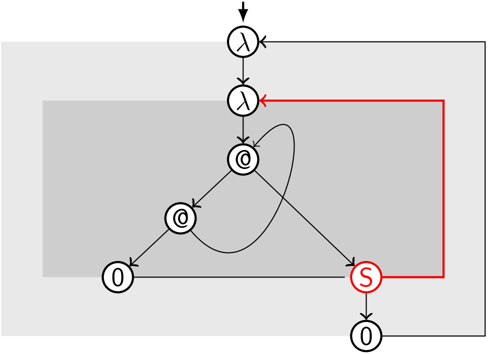
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

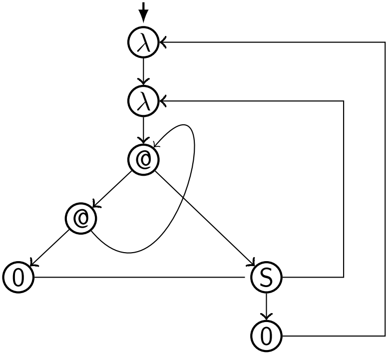
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\ r\ x \text{ in } r$



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

graph interpretation (example 1)

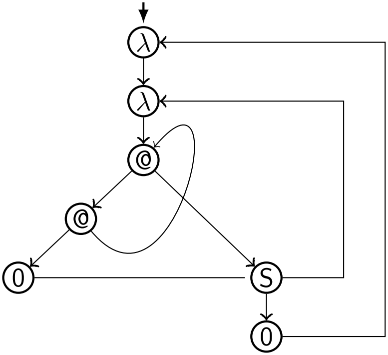
$L_0 = \lambda x.\, \lambda f.\, \text{let } r = f\; r\; x \text{ in } r$



first-order term graph with scope vertices with backlinks

graph interpretation (example 1)

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



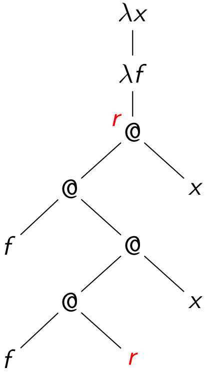
λ-term-graph $[\![L_0]\!]_{\mathcal{T}}$

graph interpretation (example 2)

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

graph interpretation (example 2)

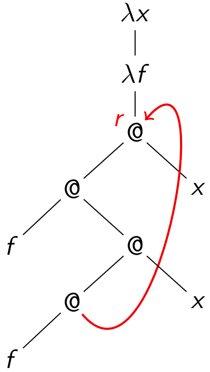
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree

graph interpretation (example 2)

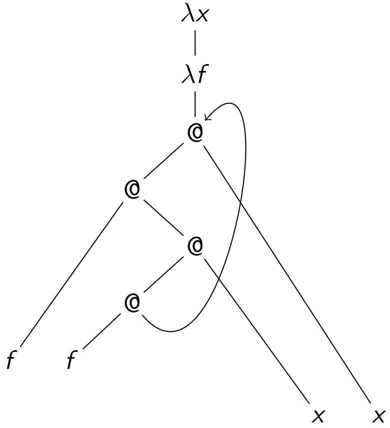
$L = \lambda x. \lambda f. \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 2)

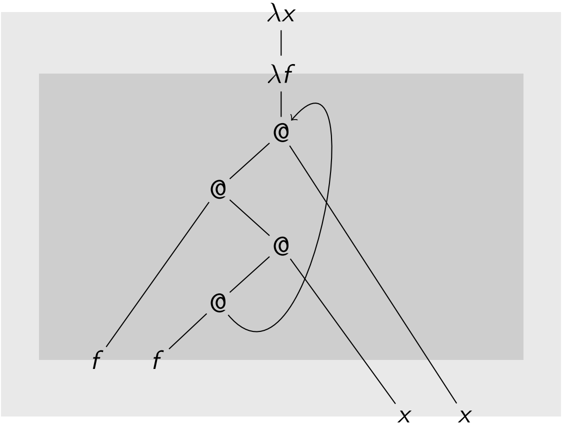
$L = \lambda x. \lambda f. \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



syntax tree (+ recursive backlink)

graph interpretation (example 2)

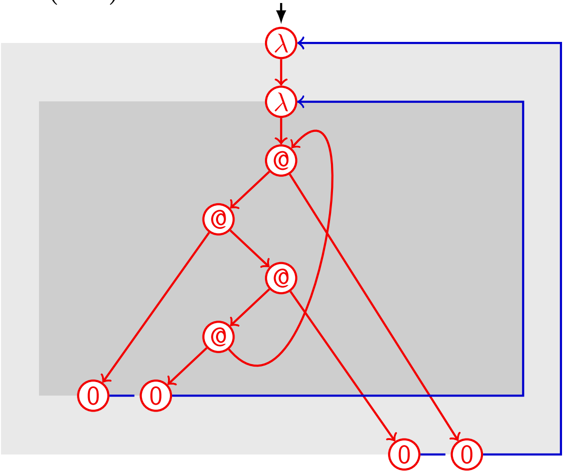
$L = \lambda x.\, \lambda f.\, \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

graph interpretation (example 2)

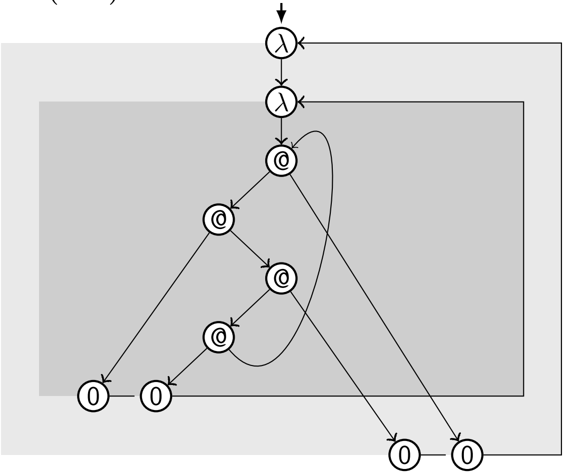
$L = \lambda x. \lambda f. \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



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

graph interpretation (example 2)

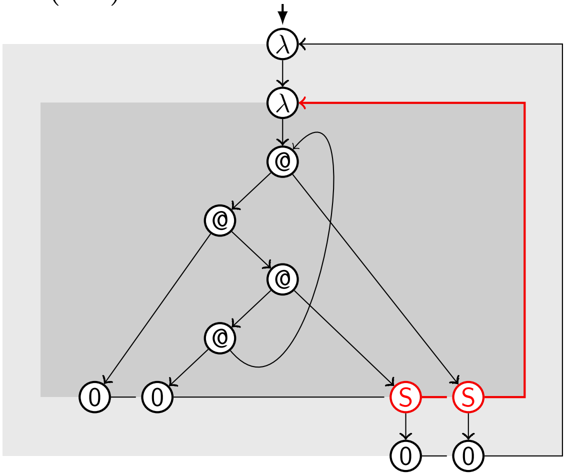
$L = \lambda x. \lambda f. \text{let } r = f\,(f\,r\,x)\,x \text{ in } r$



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

graph interpretation (example 2)

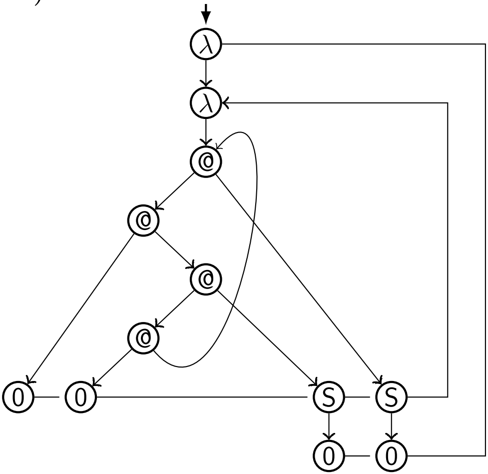
$L = \lambda x. \lambda f. \text{let } r = f\ (f\ r\ x)\ x \text{ in } r$



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

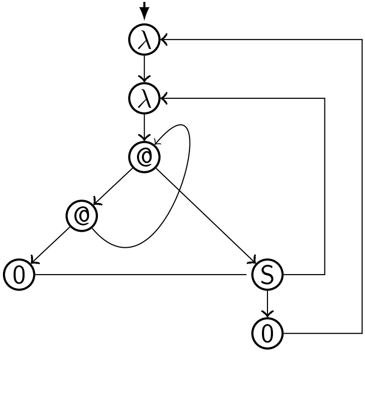
graph interpretation (example 2)

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

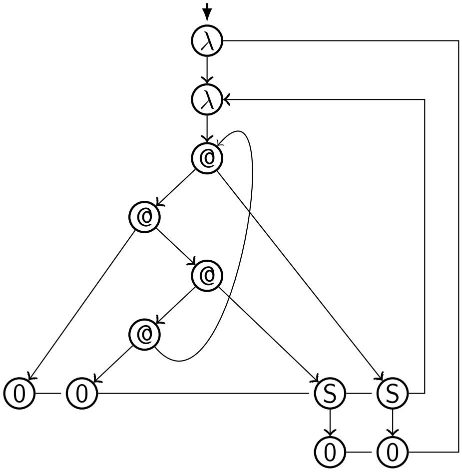


λ-term-graph $[\![L]\!]_{\mathcal{T}}$

graph interpretation (examples 1 and 2)



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



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

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

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

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

Theorem

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

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

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

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

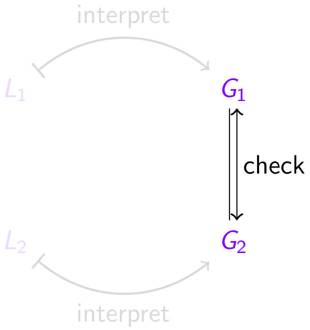
- defined by induction on structure of L
- similar analysis as fully-lazy lambda-lifting
- yields eager-scope λ-term-graphs: $\sim$ minimal scopes

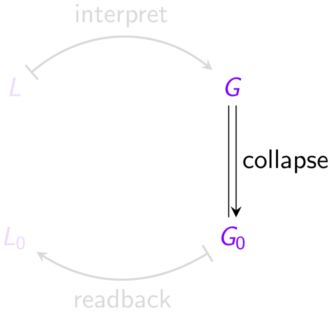
Theorem

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

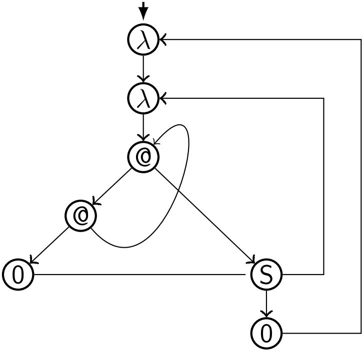
$$[\![L_1]\!]_{\lambda^\infty} = [\![L_2]\!]_{\lambda^\infty} \iff [\![L_1]\!]_{\mathcal{T}} \leftrightarrows [\![L_2]\!]_{\mathcal{T}}$$

bisimulation check and collapse

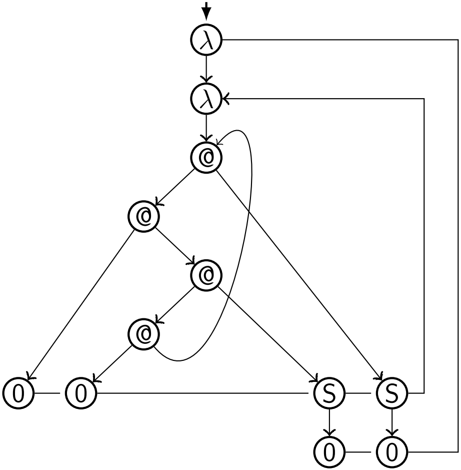




bisimulation check between λ-term-graphs

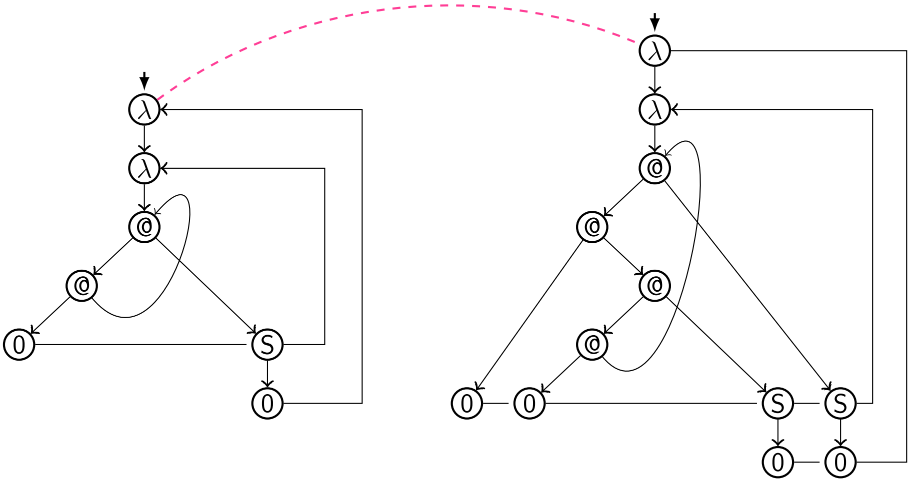


$[\![L_0]\!]_{\mathcal{T}}$



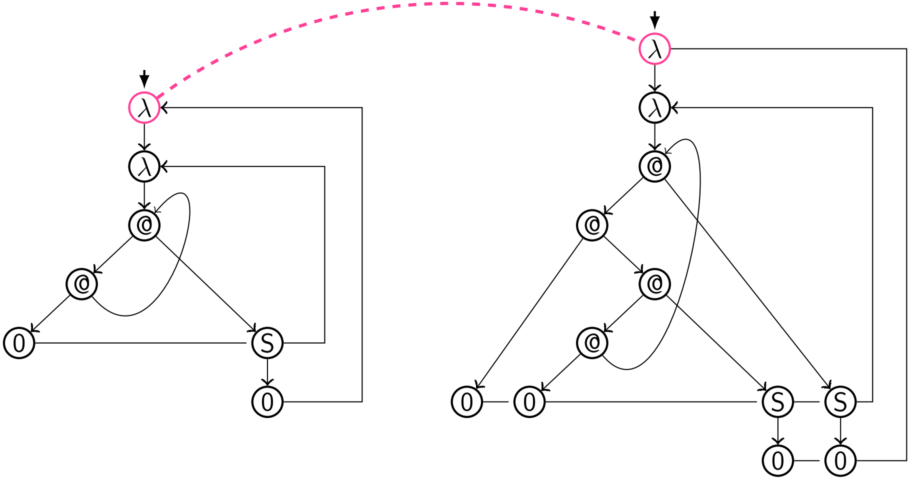
$[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



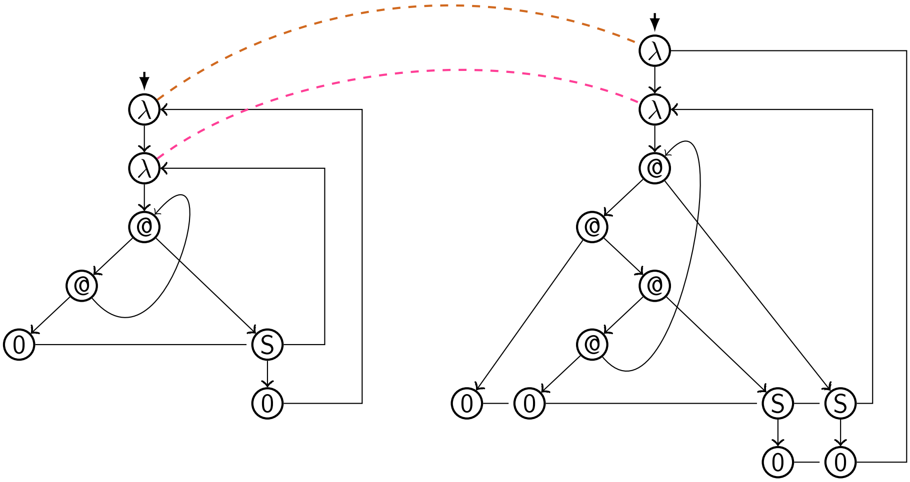
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



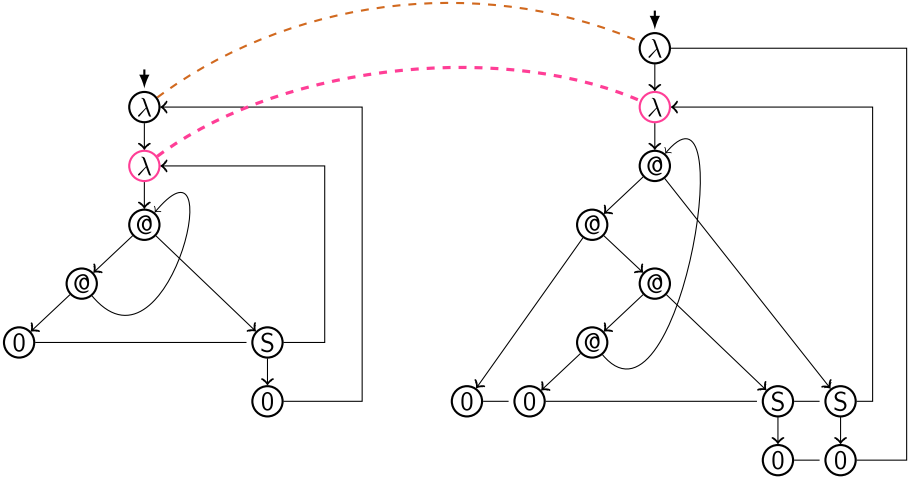
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



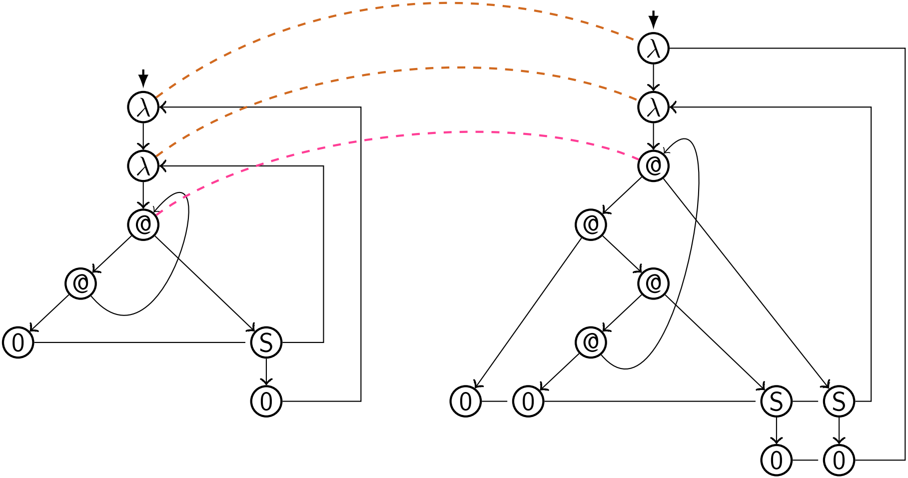
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



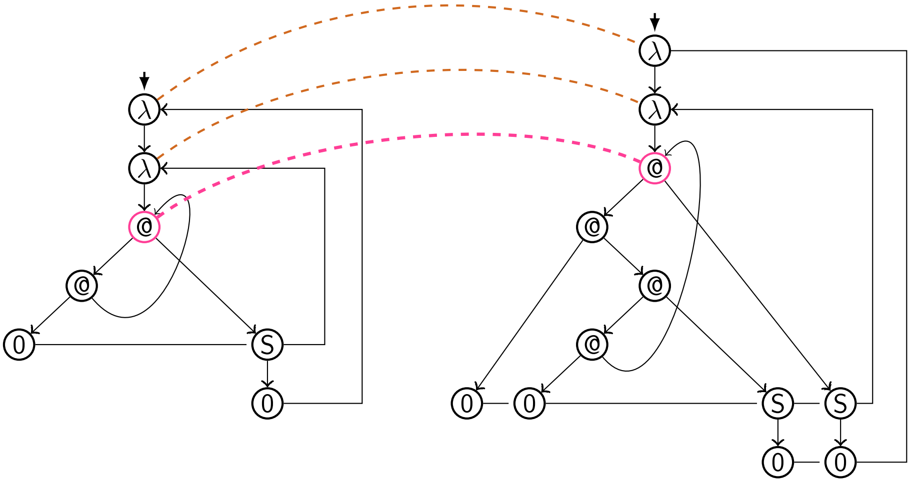
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



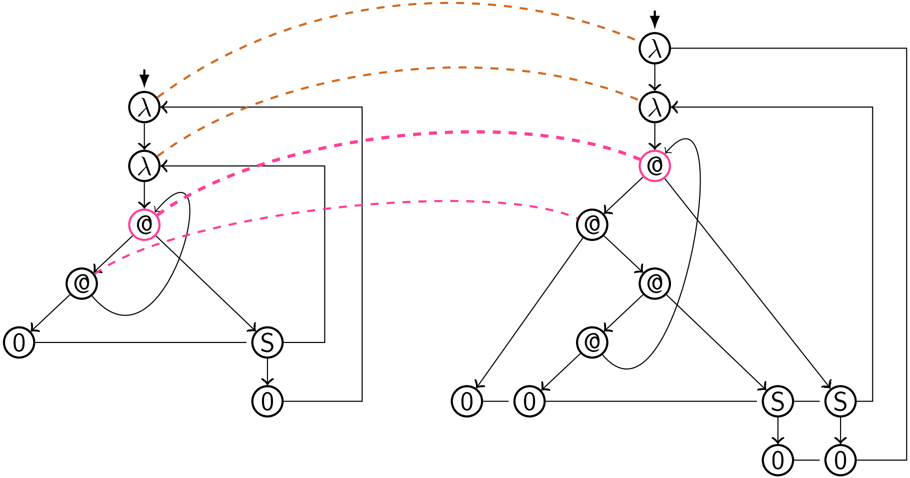
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



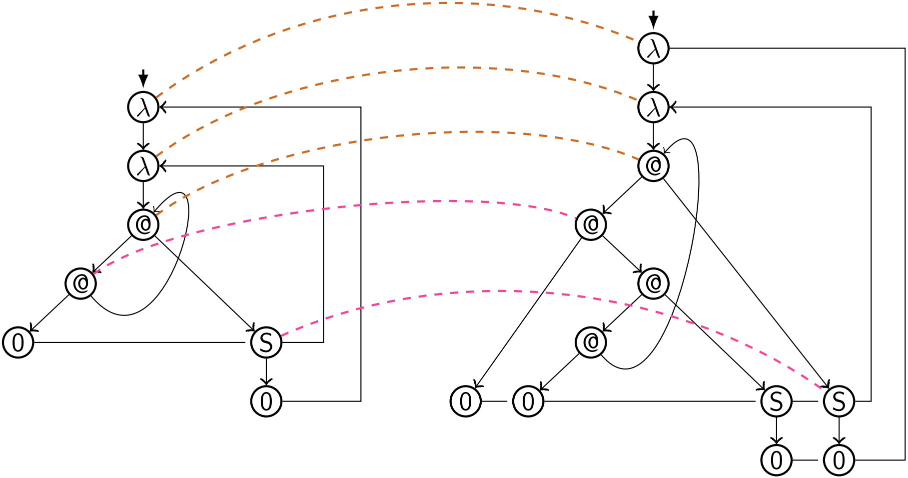
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



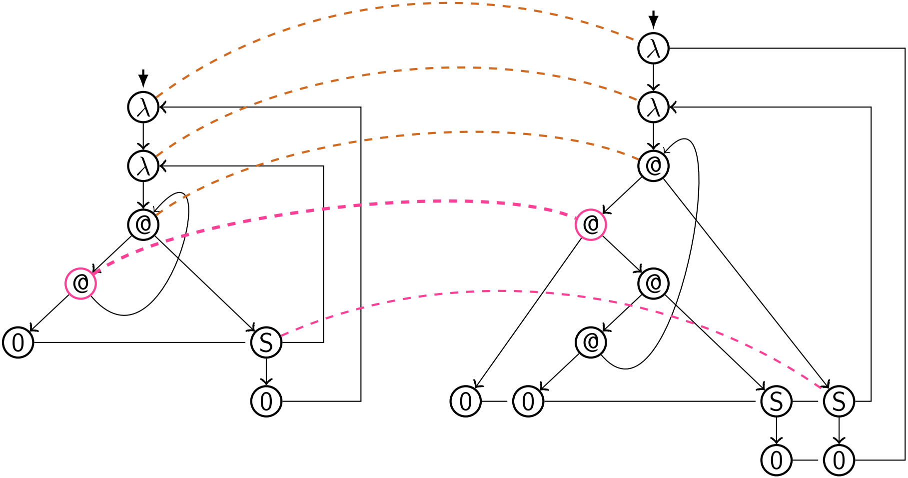
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



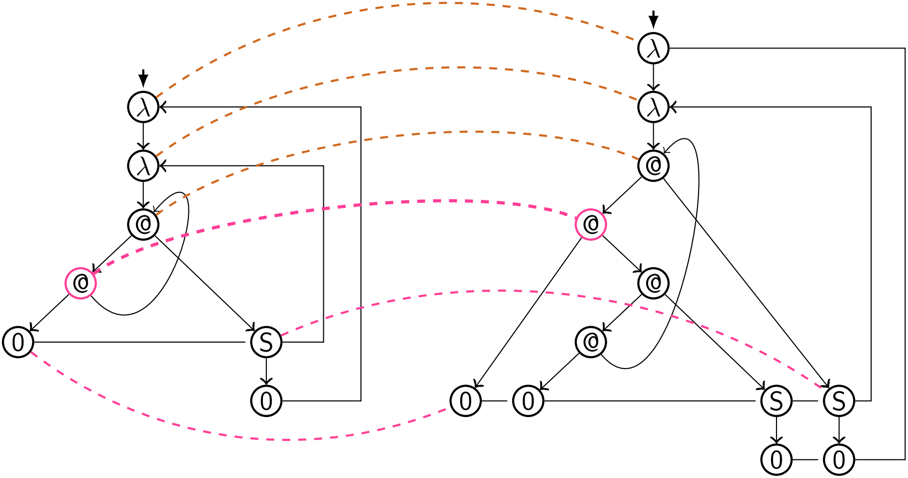
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



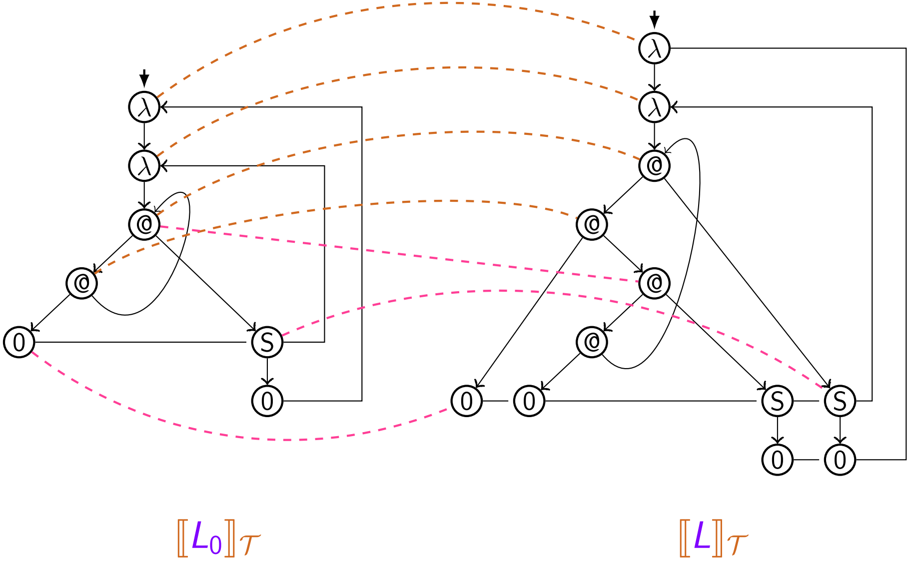
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

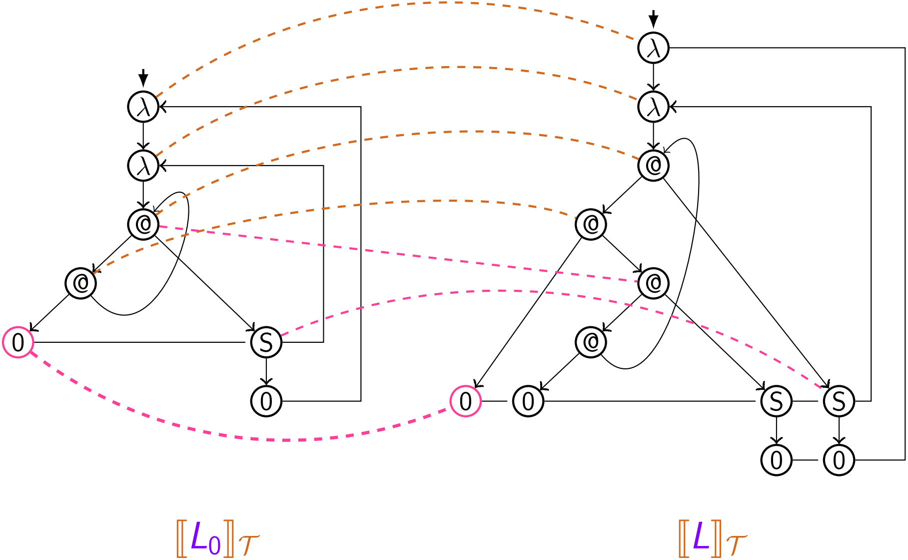


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

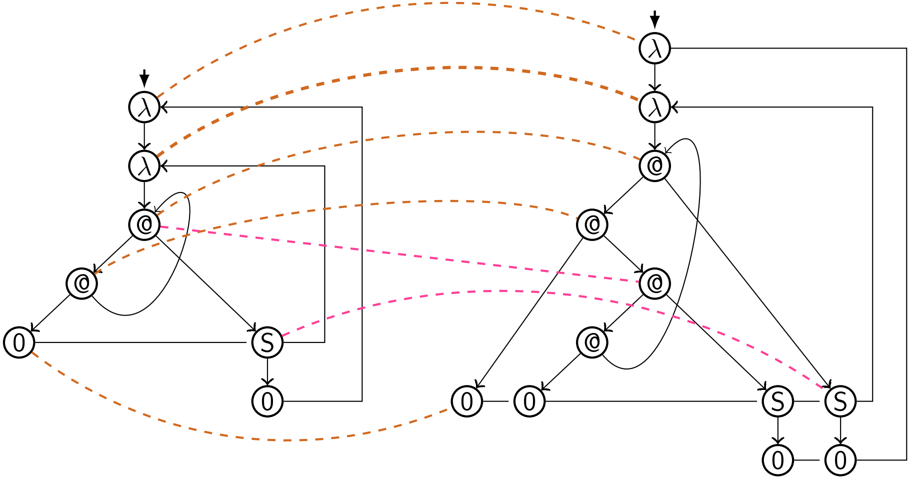
bisimulation check between λ-term-graphs



bisimulation check between λ-term-graphs

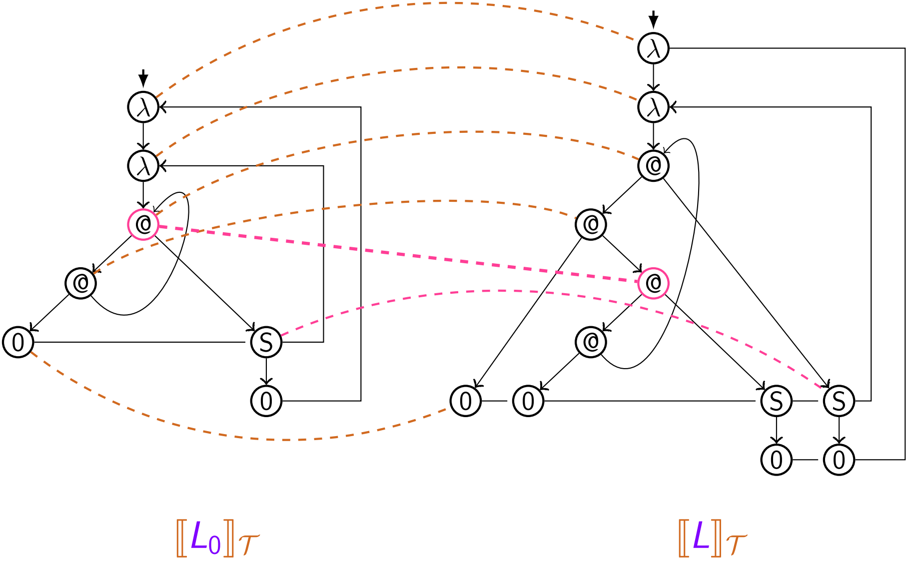


bisimulation check between λ-term-graphs

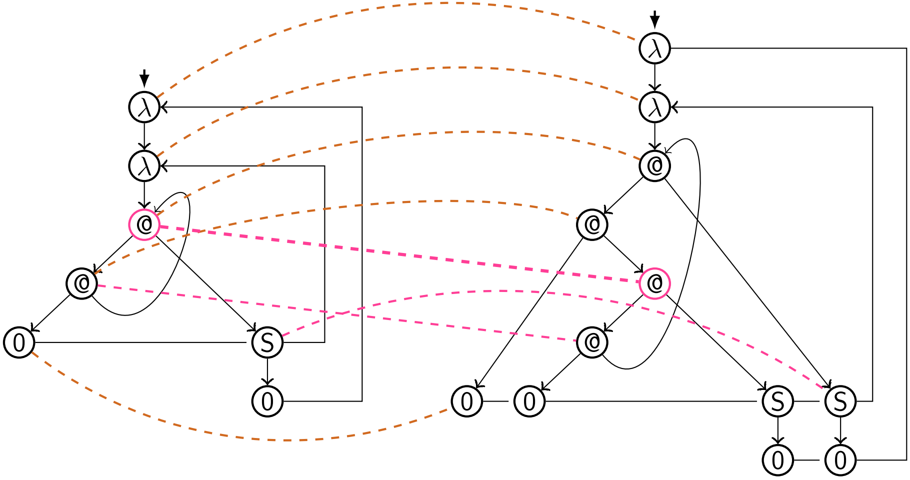


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

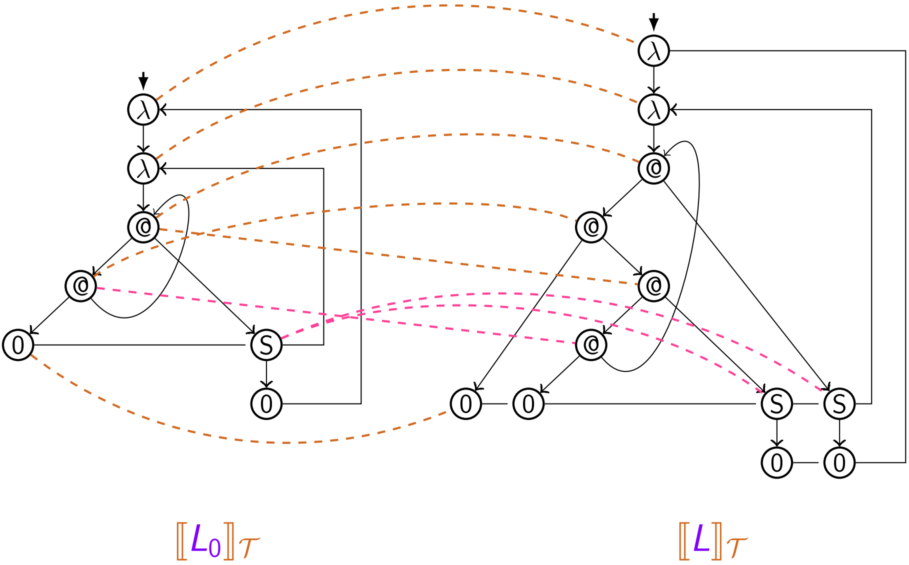


bisimulation check between λ-term-graphs

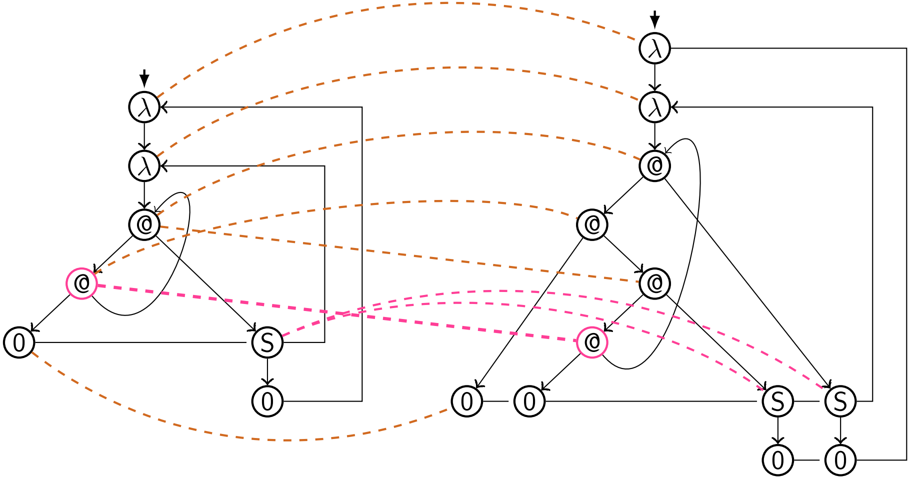


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

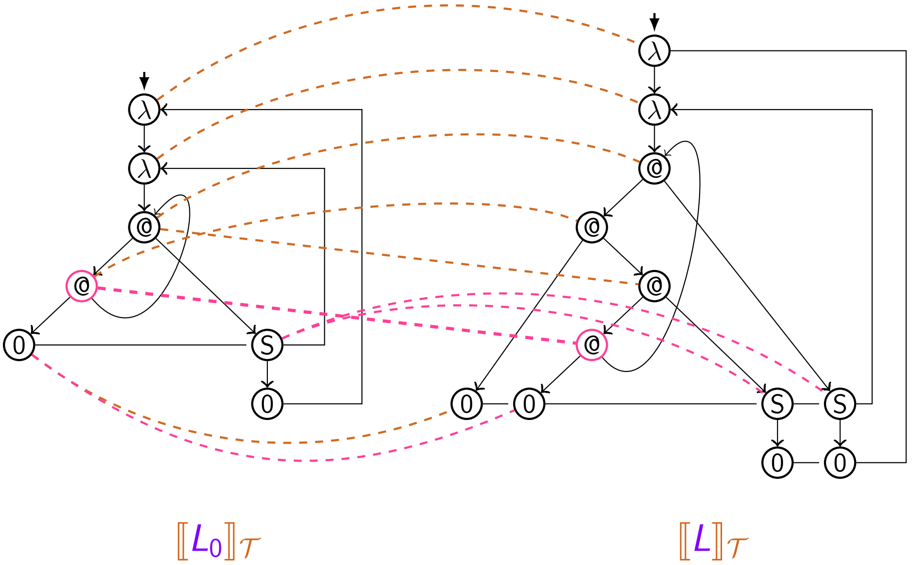


bisimulation check between λ-term-graphs

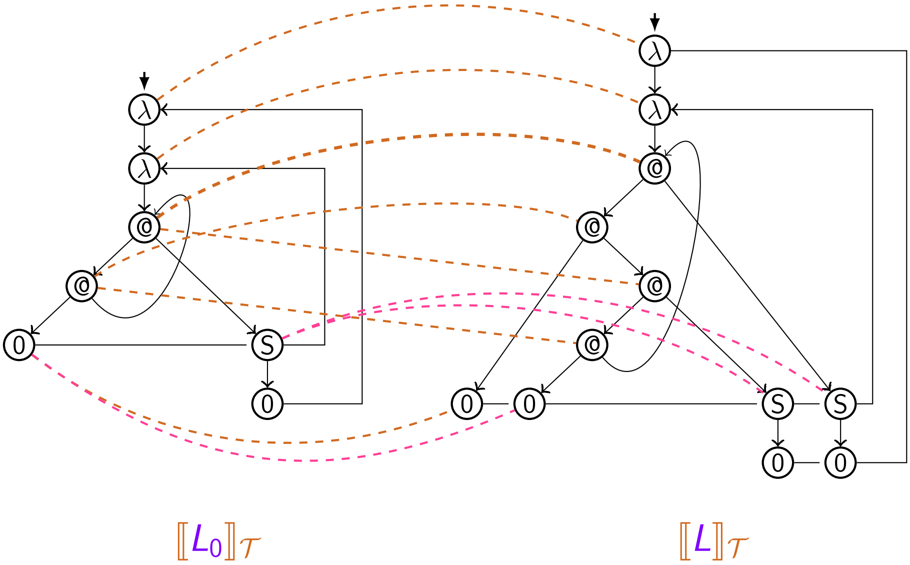


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

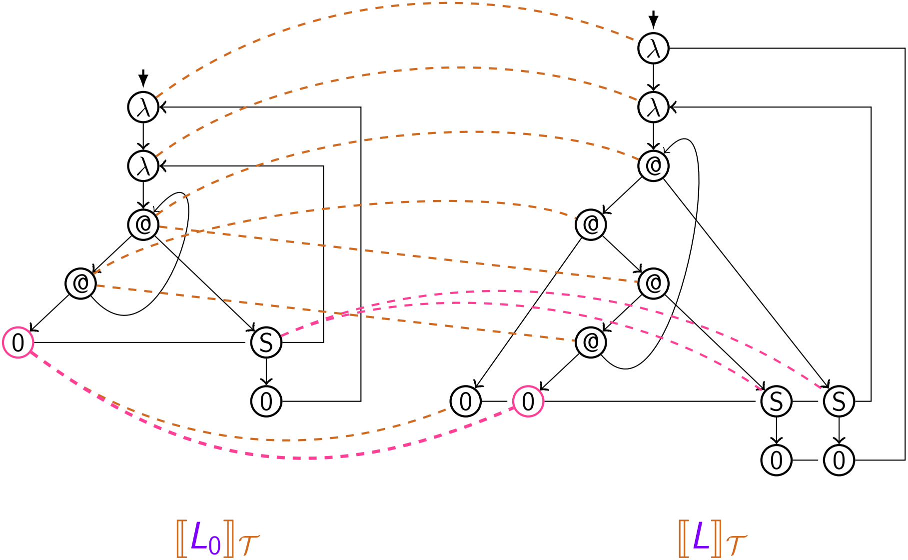
bisimulation check between λ-term-graphs



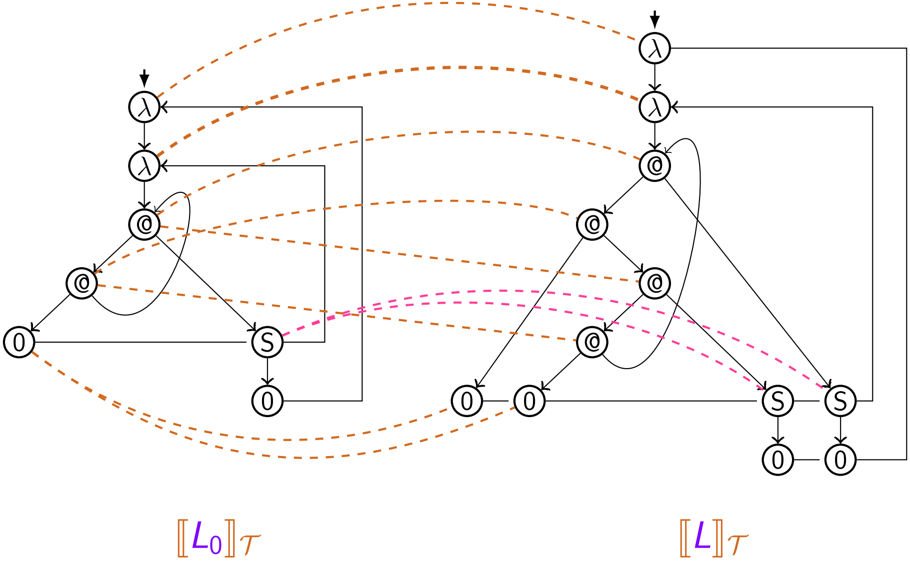
bisimulation check between λ-term-graphs



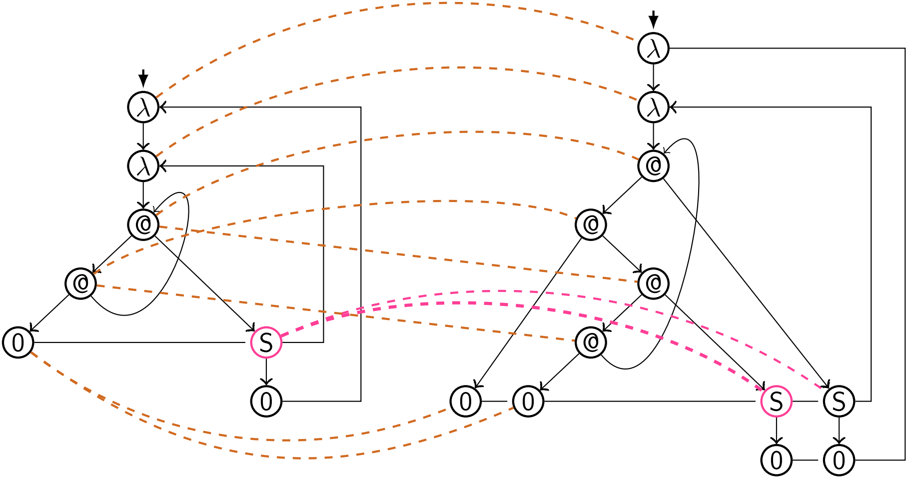
bisimulation check between λ-term-graphs



bisimulation check between λ-term-graphs

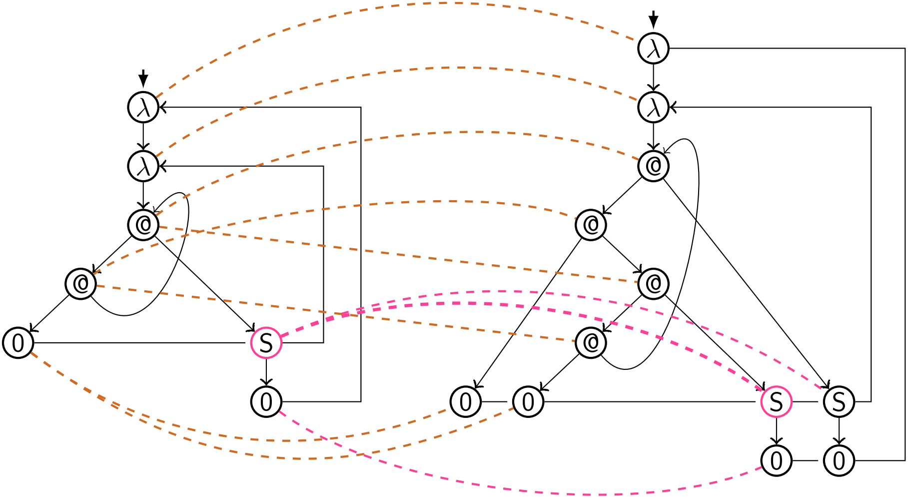


bisimulation check between λ-term-graphs



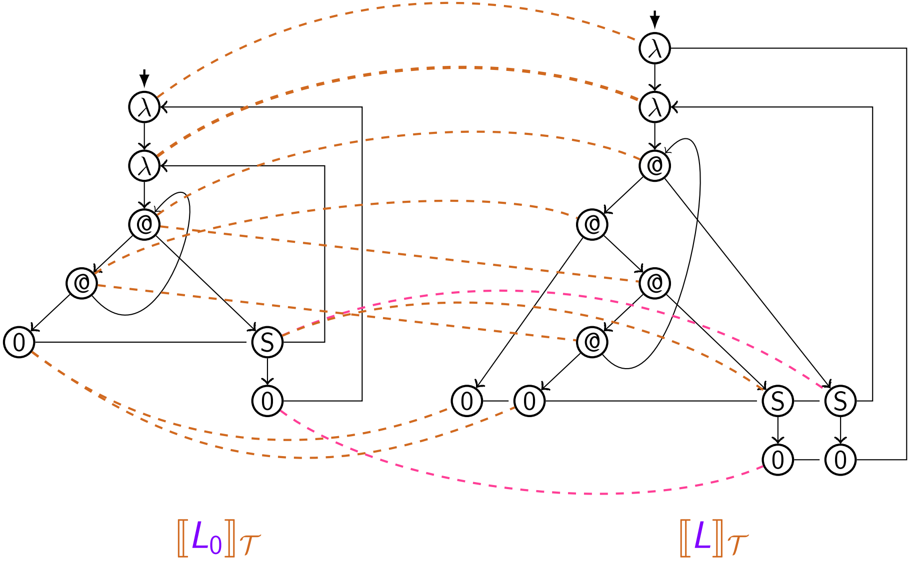
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

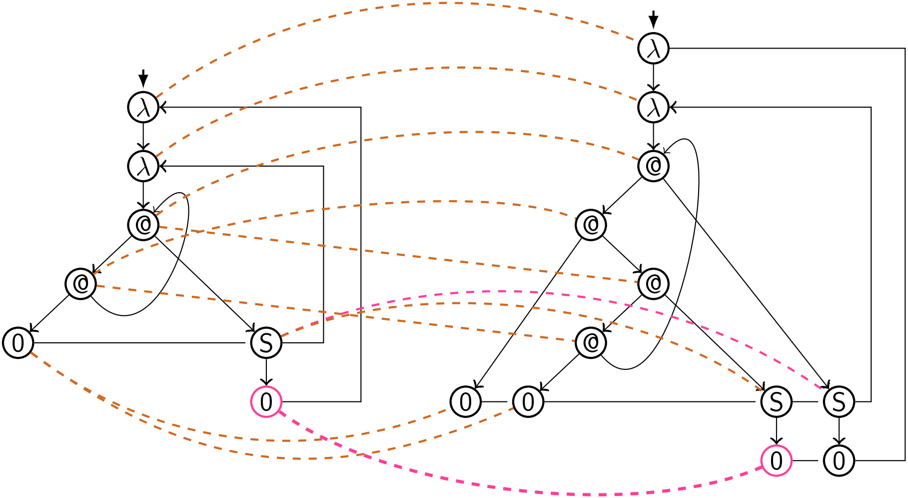


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

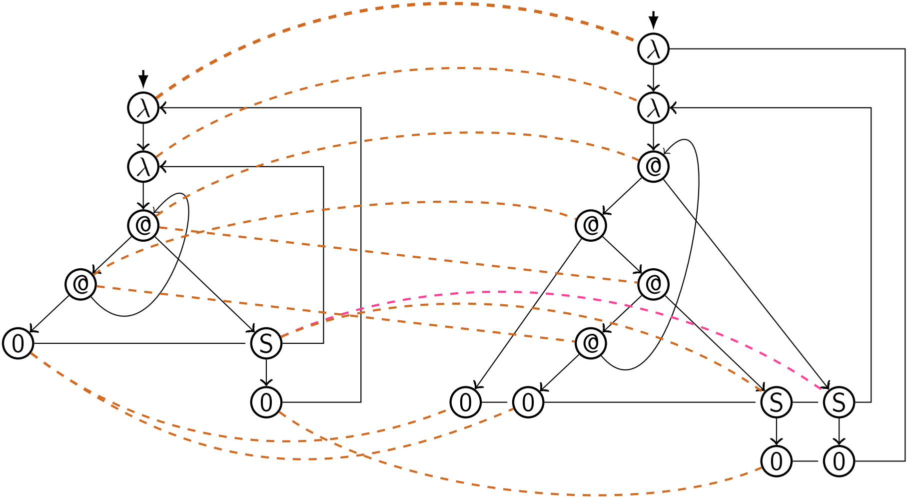


bisimulation check between λ-term-graphs



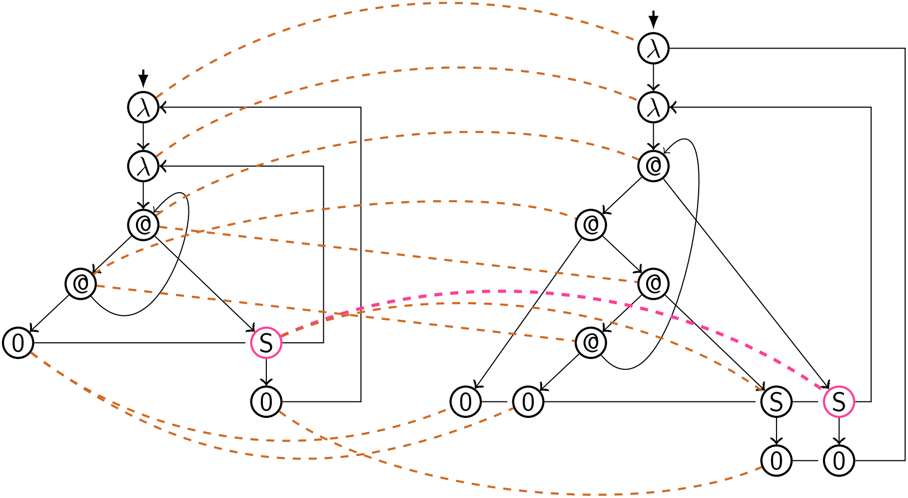
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs



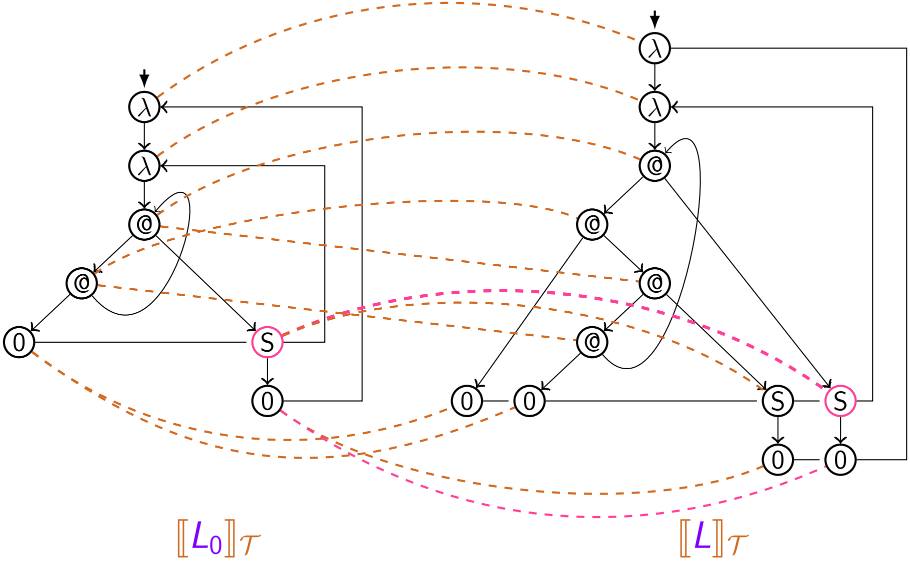
$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

bisimulation check between λ-term-graphs

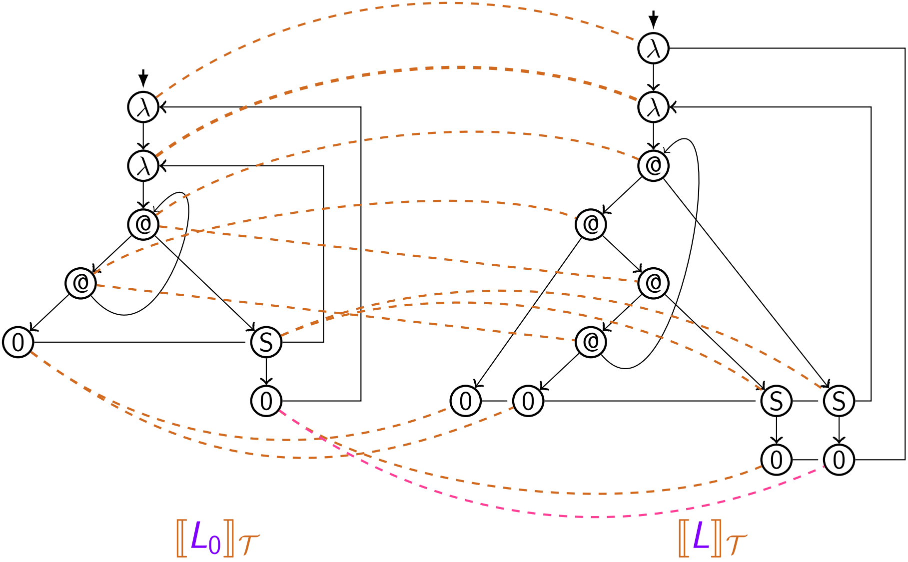


$[\![L_0]\!]_{\mathcal{T}}$ $[\![L]\!]_{\mathcal{T}}$

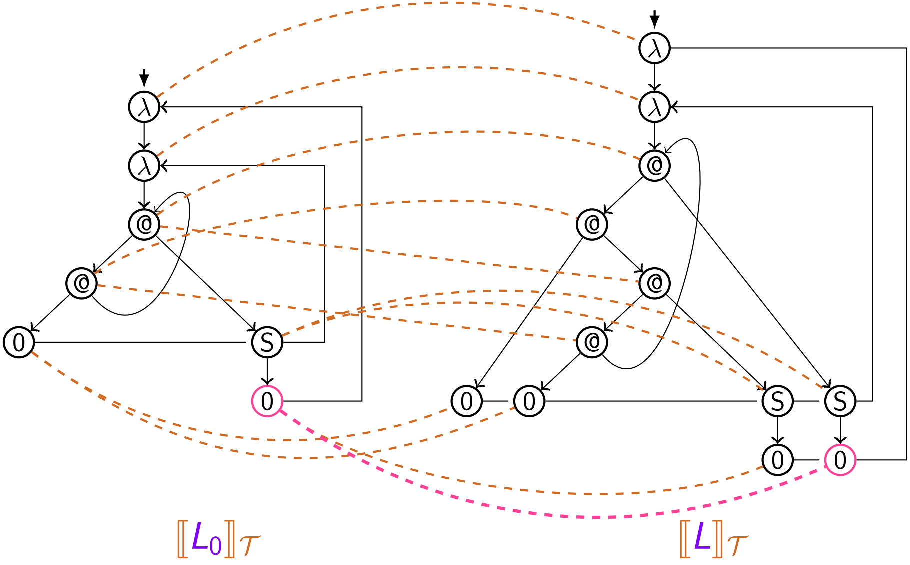
bisimulation check between λ-term-graphs



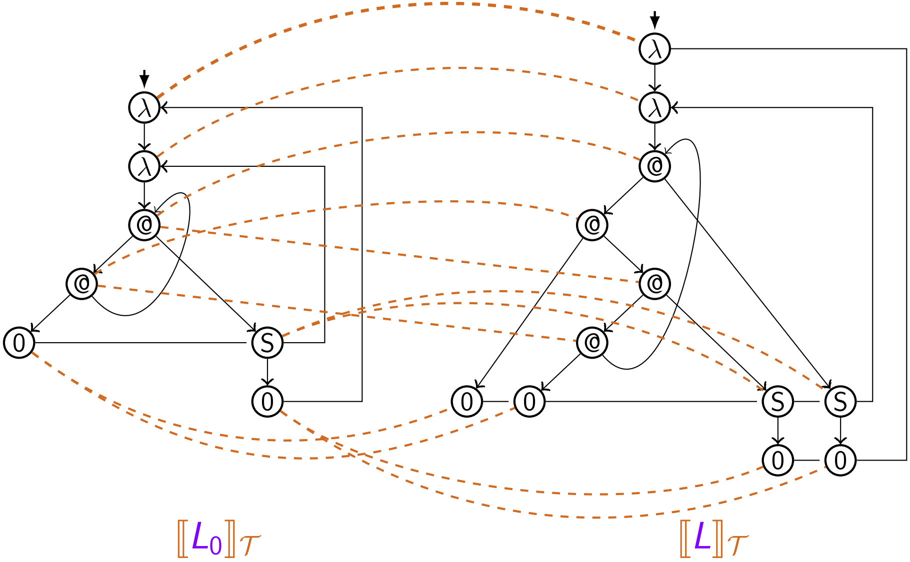
bisimulation check between λ-term-graphs



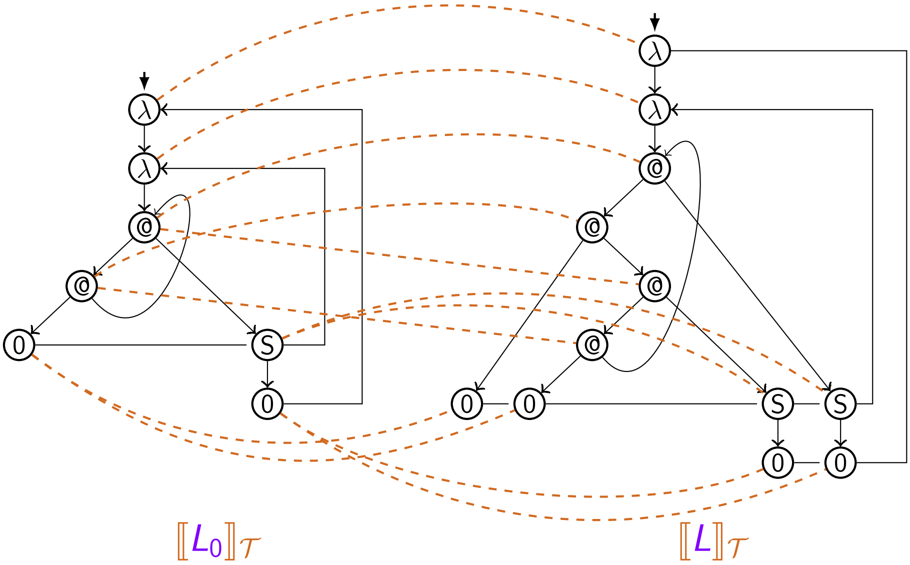
bisimulation check between λ-term-graphs



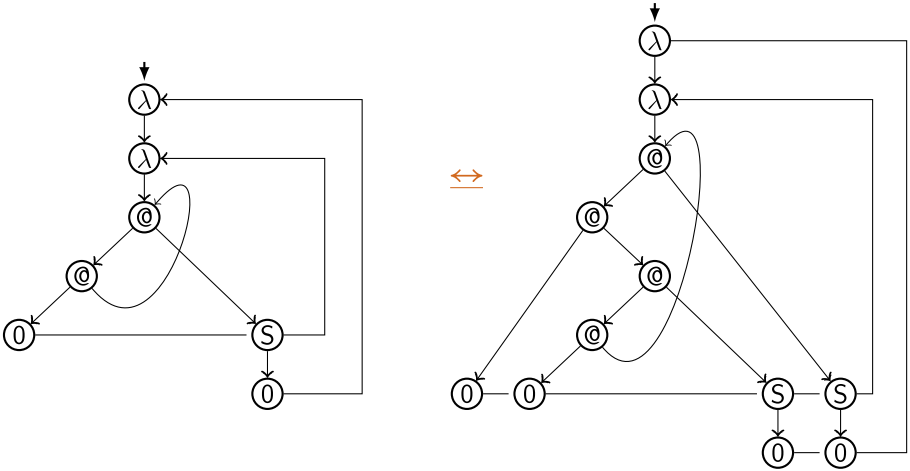
bisimulation check between λ-term-graphs



bisimulation between λ-term-graphs

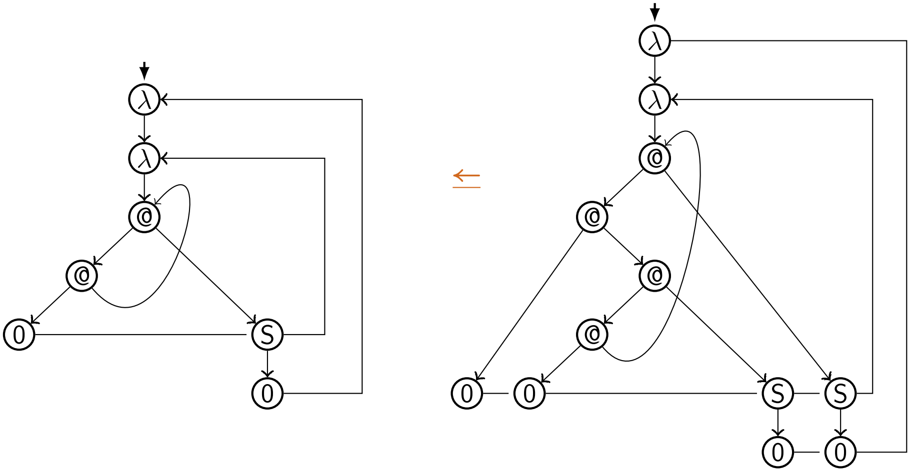


bisimilarity between λ-term-graphs



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

functional bisimilarity and bisimulation collapse



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

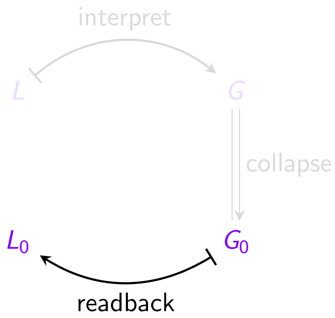
bisimulation collapse: property

Theorem

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

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

readback



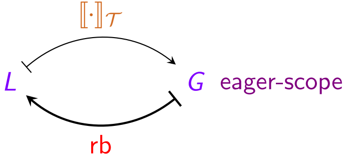
readback

defined with property:

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

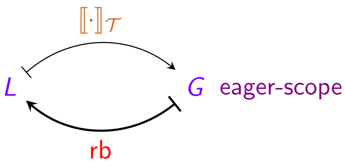
readback

defined with property:



readback

defined with property:



Theorem

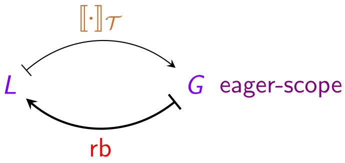
For all eager-scope λ-term-graphs G:

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

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

readback

defined with property:



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 $\simeq$.

main idea:

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

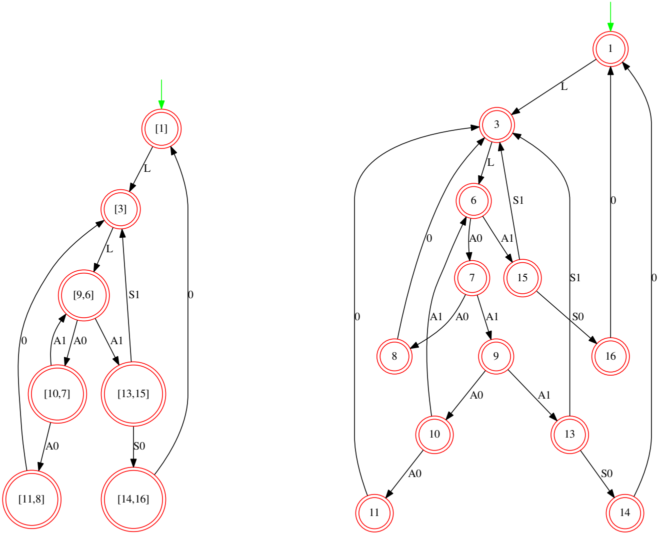
implementation

- tool maxsharing on `hackage.haskell.org`
  - uses Utrecht University Attribute Grammar Compiler (UUAGC)
- examples and explanation
  - in accompanying report

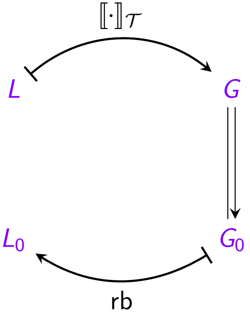
Demo: console output

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

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

writing DFA to file: running-dfa.pdf

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

writing minimised DFA to file: running-mindfa.pdf

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

Demo: generated DFAs



maximal sharing: complexity



1. interpretation

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L

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

2. bisimulation collapse $\Downarrow$

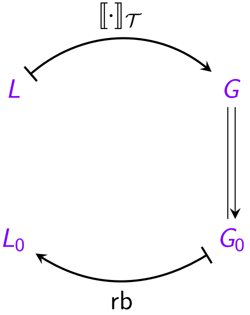
   of f-o term graph G into G_0

3. readback rb

   of f-o term graph G_0

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

maximal sharing: complexity



1. interpretation

   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L

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

2. bisimulation collapse $\downdownarrows$

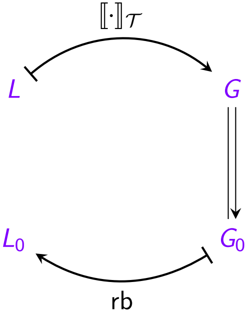
   of f-o term graph G into G_0

3. readback rb

   of f-o term graph G_0

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

maximal sharing: complexity



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 $|\!\!\downarrow$

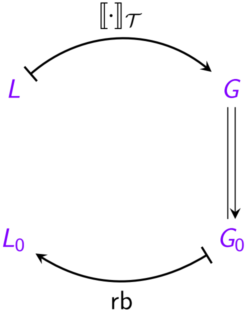
   of f-o term graph G into G_0

3. readback rb

   of f-o term graph G_0

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

maximal sharing: complexity



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 $\downarrow\!\!|$

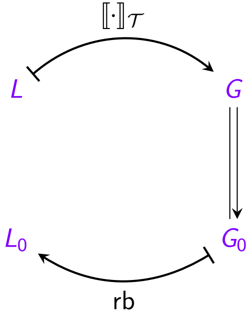
   of f-o term graph G into G_0

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

3. readback rb

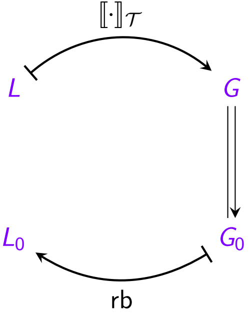
   of f-o term graph G_0
   yielding $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \text{rb}(G_0)$.

maximal sharing: complexity



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 $L_0 = \text{rb}(G_0)$.
   - in time $O(|G| \log |G|) = O(n^2 \log n)$

maximal sharing: complexity



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

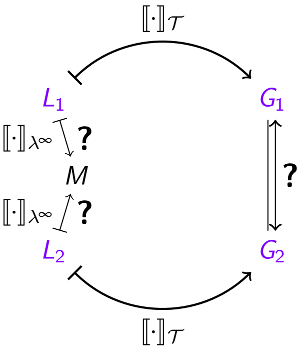
   yielding $\boldsymbol{\lambda}_{\text{letrec}}$-term $L_0 = \text{rb}(G_0)$.

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

Theorem

Computing a maximally compact form $L_0 = (\text{rb} \circ |\!\downarrow \circ [\![\cdot]\!]_{\mathcal{T}})(L)$ *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

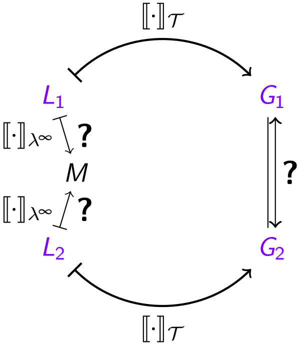
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2

   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$

2. check bisimilarity

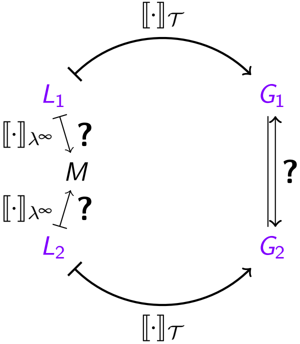
   of λ-term-graphs G_1 and G_2

unfolding equivalence: complexity



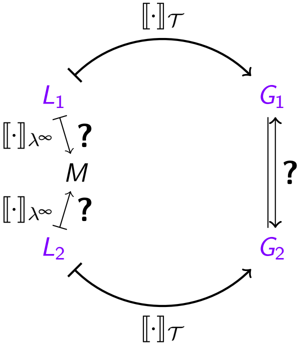
1. interpretation
   of $\boldsymbol{\lambda}_{\text{letrec}}$-term L_1, L_2 with $n = \max\{|L_1|, |L_2|\}$
   as λ-term-graphs $G_1 = [\![L_1]\!]_{\mathcal{T}}$ and $G_2 = [\![L_2]\!]_{\mathcal{T}}$
   - in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.
2. check bisimilarity
   of λ-term-graphs G_1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

unfolding equivalence: complexity



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

unfolding equivalence: complexity



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

Theorem

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

extensions

- support for full functional languages
  - work on a Core language with constructors, case statements
  - model these by enriching $\boldsymbol{\lambda}_{\mathsf{letrec}}$ with function symbols
  - adapt our method to this $\boldsymbol{\lambda}_{\mathsf{letrec}}$-extension
- prevent space leaks caused by disadvantageous sharing
  - identify 'sharing-unfit' positions/vertices
  - modify λ-term-graph interpretation
    in order to constrain the bisimulation collapse

applications

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