Term Graph Representations for Cyclic Lambda-Terms

Clemens Grabmayer   Jan Rochel

Department of Philosophy
Utrecht University
The Netherlands
clemens@phil.uu.nl

Department of Computing Sciences
Utrecht University
The Netherlands
jan@rochel.info

23rd March 2013

Area of Research

- functional programming languages
- untyped λ-calculus with **letrec** ($\boldsymbol{\lambda}_{\mathsf{letrec}}$)
- sharing in $\boldsymbol{\lambda}_{\mathsf{letrec}}$

Motivation

Example

$\lambda x.\textbf{letrec}\, f = x\, f \,\textbf{in}\, f \;\twoheadrightarrow_{\triangledown}\; \lambda x.x\,(x\,(x \ldots))$

Example

$\lambda x.\textbf{letrec}\, f = x\,(x\, f) \,\textbf{in}\, f \;\twoheadrightarrow_{\triangledown}\; \lambda x.x\,(x\,(x \ldots))$

Efficient methods for determining

- whether two $\boldsymbol{\lambda}_{\text{letrec}}$-terms have the same unfolding
- the maximally shared form of a $\boldsymbol{\lambda}_{\text{letrec}}$-term

On the theoretical side:

- a notion of maximal sharing
- a sharing preorder

Example

$\lambda x.\textbf{letrec}\, f = x\, f \,\textbf{in}\, f \quad \geq \quad \lambda x.\textbf{letrec}\, f = x\,(x\, f) \,\textbf{in}\, f$

To reason about unfolding equivalence and sharing we want to abstract over:

- order of **letrec**-bindings
- position of binding groups
- names of recursion- and λ-variables

$\Longrightarrow$ work with graph representations of $\boldsymbol{\lambda}_{\mathsf{letrec}}$-terms that faithfully represent the sharing that occurs in a $\boldsymbol{\lambda}_{\mathsf{letrec}}$-term.

Bisimulation ~ unfolding equivalence.
Functional bisimulation ~ compactification.

Computing the maximally shared form of a term

G computes the graph representation of a term. R is a ‘readback’.
G is a left inverse of R: $G \circ R = id$

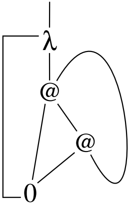
$$\begin{tikzcd}
L_1 \arrow[d, "G"'] & & L_2 \\
G_1 \arrow[rr, Rightarrow, "\text{bisimulation}", "\text{collapse}"'] & & G_2 \arrow[u, "R"']
\end{tikzcd}$$

Computing the maximally shared form of a term

$$\lambda x.\textbf{letrec}\ f = x\,(x\,f)\ \textbf{in}\ f \qquad \leq \qquad \lambda x.\textbf{letrec}\ f = x\,f\ \textbf{in}\ f$$

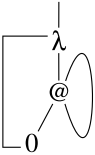
G

R



bisimulation

collapse



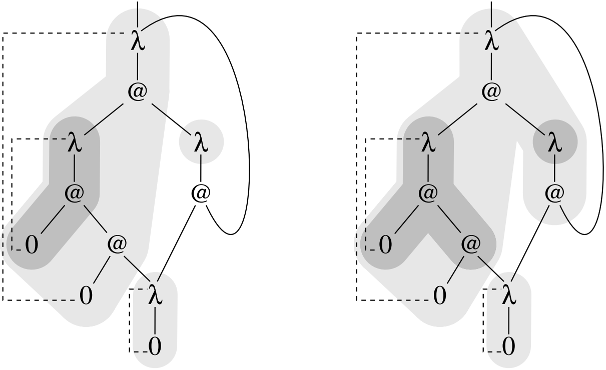
Overview

We study various graph formalisms and show how they relate:

- λ-higher-order-term-graphs: first-order term graphs + a scope function (based on 'higher-order term graphs' [Blom, 2001])
- abstraction-prefix based λ-higher-order-term-graphs first-order term graphs + an abstraction prefix function (motivated by [G&R, 2012])
- λ-term-graphs with scope delimiters: plain first-order term graphs with scope delimiter vertices

We want to establish correspondences between the formalisms to show that one can be implemented in terms of the other.

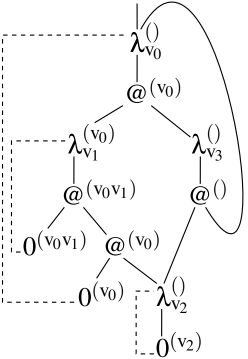
$$\textbf{letrec}\ \begin{array}{l} f = \lambda x.(\lambda y.(y\,(x\,g)))\,(\lambda z.g\,f) \\ g = \lambda i.i \end{array}\ \textbf{in}\ f$$

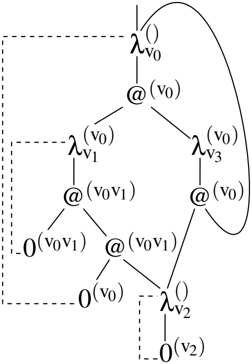


The scope function assigns to each abstraction node a set of nodes

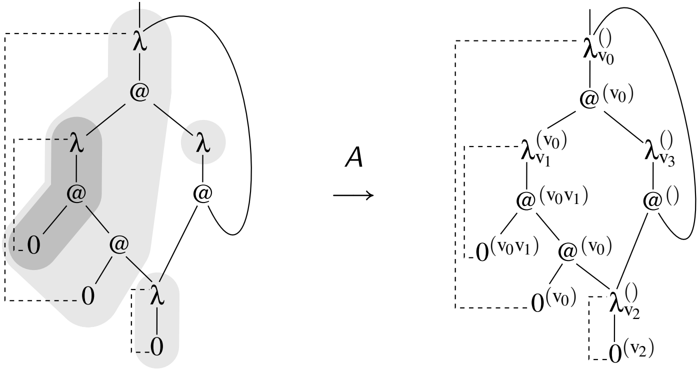
abstraction-prefix based λ-higher-order-term-graphs

$$\textbf{letrec} \begin{array}{l} f = \lambda x.(\lambda y.(y\,(x\,g)))\,(\lambda z.g\,f) \\ g = \lambda i.i \end{array} \textbf{in } f$$





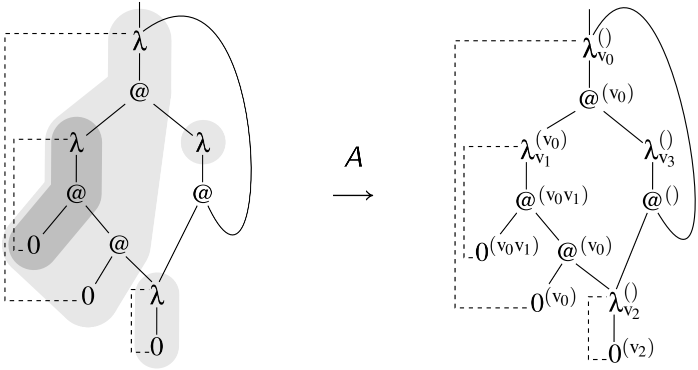
An isomorphic correspondence



A preserves the sharing order:

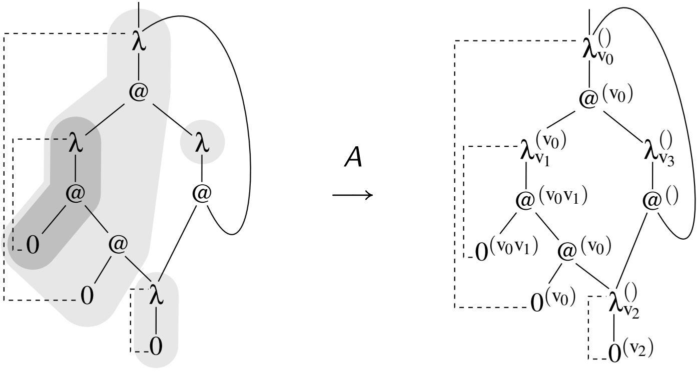
- $G_1 \mathrel{\underline{\rightarrow}} G_2 \implies A(G_1) \mathrel{\underline{\rightarrow}} A(G_2)$

An isomorphic correspondence



A preserves and reflects the sharing order:

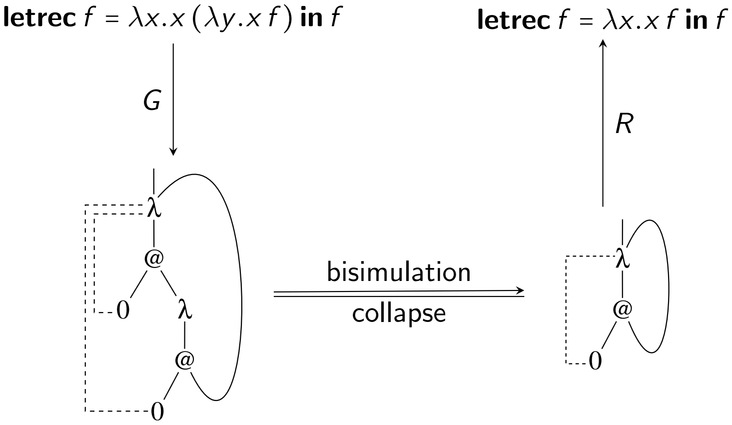
- $G_1 \mathrel{\underline{\rightarrow}} G_2 \iff A(G_1) \mathrel{\underline{\rightarrow}} A(G_2)$



A preserves and reflects the sharing order:

- $G_1 \mathrel{\underline{\rightarrow}} G_2 \iff A(G_1) \mathrel{\underline{\rightarrow}} A(G_2)$
- $A^{-1}(G_1) \mathrel{\underline{\rightarrow}} A^{-1}(G_2) \iff G_1 \mathrel{\underline{\rightarrow}} G_2$

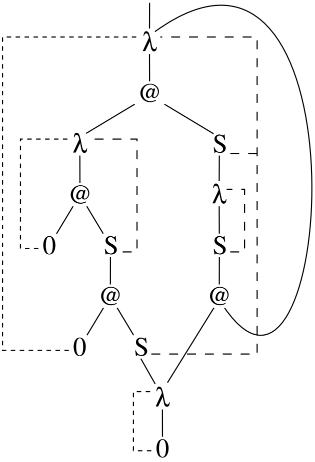
Scopes are important

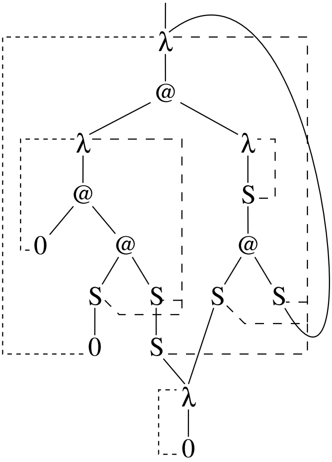


If scoping information is omitted, bisimulation would relate terms with different unfoldings.

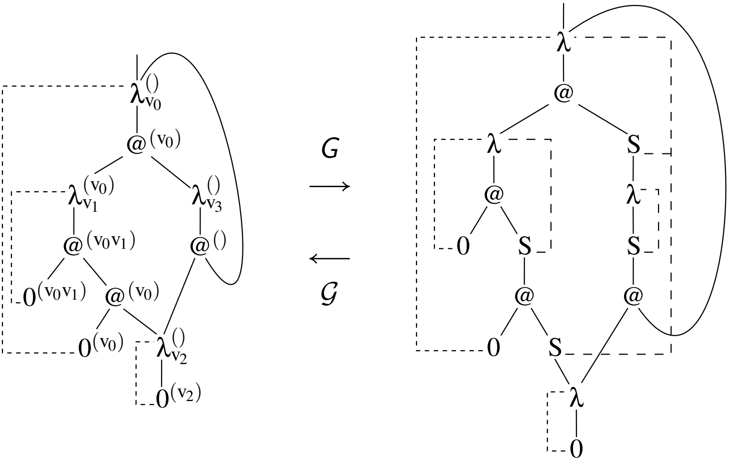
λ-term-graphs with scope delimiters

$$\textbf{letrec}\ \begin{array}{l} f = \lambda x.(\lambda y.(y\,(x\,g)))\,(\lambda z.g\,f) \\ g = \lambda i.i \end{array}\ \textbf{in}\ f$$





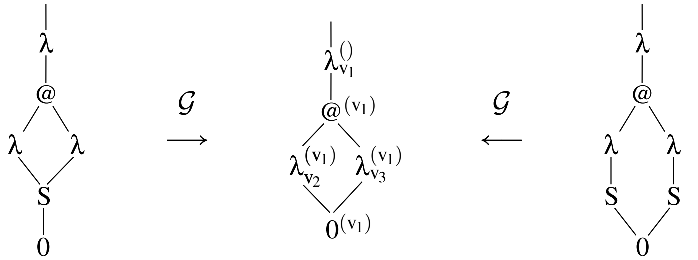
A correspondence



G and $\mathcal{G}$ also preserve and reflect the sharing order

A correspondence

The correspondence is not an isomorphism:



$\mathcal{G}$ is not injective because of S-sharing $\implies$ $\mathcal{G} \neq G^{-1}$

But:

- $\mathcal{G} \circ G = id$ ($\mathcal{G}$ is a left-inverse of G)
- $G \circ \mathcal{G}(g) \trianglerighteq^S g$ (G is a left-inverse of $\mathcal{G}$ up to S-sharing)

Correspondences yield implementations

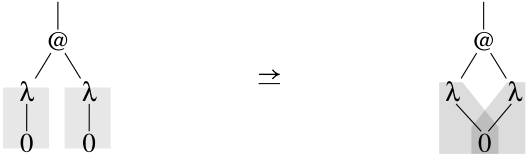
λ-higher-order-term-graphs

abstraction-prefix based graphs

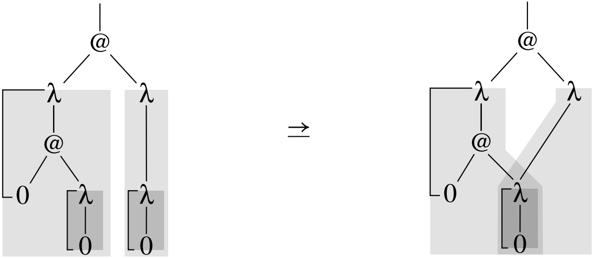
λ-term-graphs with scope delimiters

$$
\begin{tikzcd}
{} \arrow[r, Rightarrow] \arrow[d, "A"'] & {} \\
{} \arrow[r, Rightarrow] \arrow[d, "G"'] & {} \arrow[u, "A^{-1}"'] \\
{} \arrow[r, Rightarrow] & {} \arrow[u, "\mathcal{G}"']
\end{tikzcd}
$$

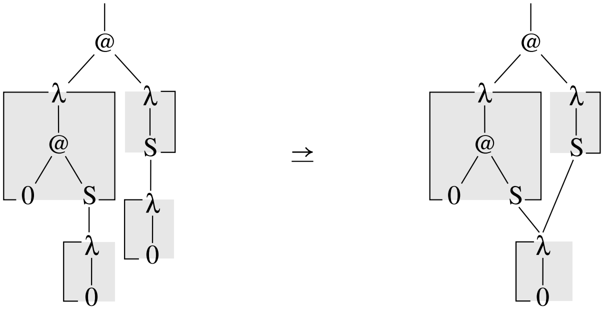
Closedness-Issues: variable backlinks



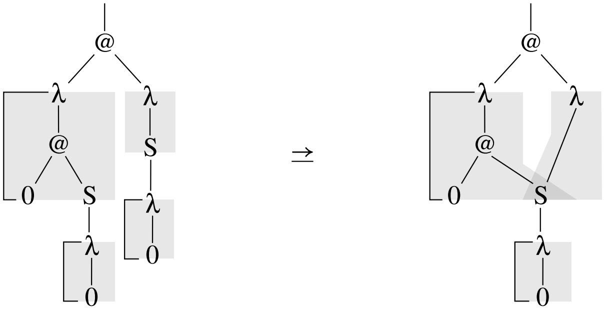
Closedness-Issues: eager scope-closure



Closedness-Issues: eager scope-closure



Closedness-Issues: *S – backlinks*

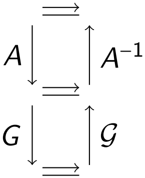


Closedness

λ-term-graphs are closed under unrestricted functional bisimulation if they have:

- scope delimiters
- delimiter backlinks
- variable backlinks
- eager placement of delimiters

What have we gained?



- *Practical:* Implementation of maximal sharing through bisimulation collapse
- *Theoretical:* Transfer of properties known for first-order term graphs to the higher-order term graphs
  - E.g. for all graphs g from the classes:
    - λ-higher-order-term-graphs
    - abstraction-prefix based λ-higher-order-term-graphs

  it holds: $\langle [g]_{\leftrightarrow}, \sqsupseteq \rangle$ is a complete lattice.
- *Easy generalisation:* e.g. to higher-order term graphs representing iCRS-terms (instead of infinite λ-terms).