Avoiding Repetitive Reduction Patterns in Lambda Calculus with `letrec`

(Work In Progress)

Jan Rochel and Clemens Grabmayer

Dept. of Computer Science, and Dept. of Philosophy
NWO-project *Realising Optimal Sharing*
Utrecht University

TERMGRAPH 2011
Saarbrücken, April 2nd

In this talk

We report on:

- an optimising transformation for λ-calculus with `letrec`
- by which i.p. the cyclic passing on of unchanged arguments during evaluation can often be prevented

In this talk

We report on:

- an optimising transformation for λ-calculus with `letrec`
- by which i.p. the cyclic passing on of unchanged arguments during evaluation can often be prevented

Examples:

- Haskell functions *repeat*, *replicate*, $++$, *map*, *until*
- a specification of the Thue–Morse sequence

In this talk

We report on:

- an optimising transformation for λ-calculus with `letrec`
- by which i.p. the cyclic passing on of unchanged arguments during evaluation can often be prevented

Examples:

- Haskell functions *repeat*, *replicate*, $++$, *map*, *until*
- a specification of the Thue–Morse sequence

Concepts used:

- visible/concealed redexes
- generalised β-reduction
- domination in digraphs
- static analysis of cyclically reappearing redexes

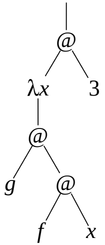
λ-Terms and λ-Trees

$$
\begin{array}{rcll}
T & ::= & V & \text{(variable)} \\
 & | & T\ T & \text{(application)} \\
 & | & \lambda V . T & \text{(abstraction)}
\end{array}
$$

λ-Terms and λ-Trees

$$
\begin{array}{lcll}
T & ::= & V & \text{(variable)} \\
 & \mid & T\ T & \text{(application)} \\
 & \mid & \lambda V . T & \text{(abstraction)}
\end{array}
$$

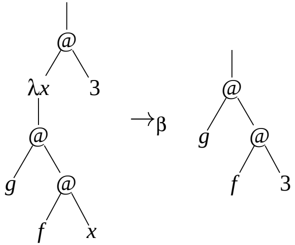
$(\lambda x . g\ (f\ x))\ 3$



β-Reduction

$$(\lambda x.M)\ N \quad \to_\beta \quad M[x := N]$$

$$(\lambda x.g\ (f\ x))\ 3 \quad \to_\beta \quad g\ (f\ 3)$$



letrec-Terms and λ-Graphs

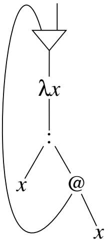
T	$::=$	V	(variable)
	$\mid$	$T\ T$	(application)
	$\mid$	$\lambda V.T$	(abstraction)
	$\mid$	$f(T,\ldots,T)$	(primitive functions)
	$\mid$	$\textbf{let}\ \mathit{Defs}\ \textbf{in}\ T$	(letrec)
Defs	$::=$	$v_1 = T\ \ldots\ v_n = T$	(equations)
		($v_1,\ldots,v_n$ distinct variables)	

$$\begin{array}{l}\textbf{let}\ \mathit{repeat} = \lambda x.x : \mathit{repeat}\ x\\ \textbf{in}\ \mathit{repeat}\end{array}$$

letrec-Terms and λ-Graphs

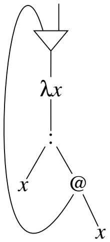
$$
\begin{array}{llll}
T & ::= & V & \text{(variable)} \\
 & \mid & T\ \ T & \text{(application)} \\
 & \mid & \lambda V.T & \text{(abstraction)} \\
 & \mid & f(T,\ldots,T) & \text{(primitive functions)} \\
 & \mid & \textbf{let}\ \mathit{Defs}\ \textbf{in}\ T & \text{(letrec)} \\
\mathit{Defs} & ::= & v_1 = T\ \ \ldots\ v_n = T & \text{(equations)} \\
 & & \quad (v_1,\ldots,v_n\ \text{distinct variables}) &
\end{array}
$$

$$
\begin{array}{l}
\textbf{let}\ \mathit{repeat} = \lambda x.x : \mathit{repeat}\ x \\
\textbf{in}\ \mathit{repeat}
\end{array}
$$



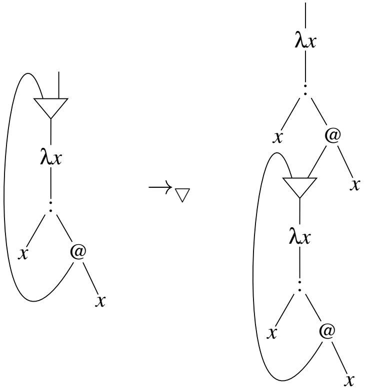
repeat

$$\textbf{let } repeat = \lambda x.x : repeat\ x$$
$$\textbf{in } repeat$$



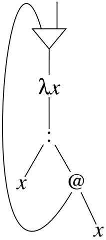
repeat

$$\textbf{let } \textit{repeat} = \lambda x.x : \textit{repeat}\ x$$
$$\textbf{in } \textit{repeat}$$

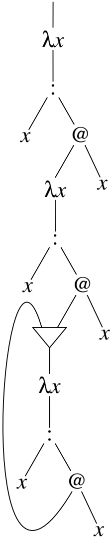


repeat

let *repeat* $= \lambda x.x : repeat\ x$
in *repeat*

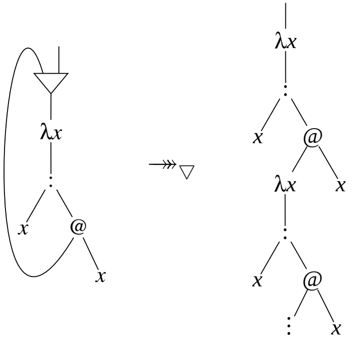


$\twoheadrightarrow_{\triangledown}$

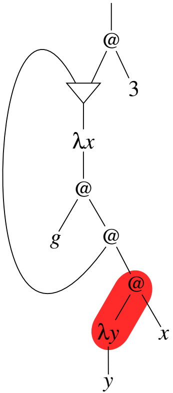


repeat

$$\textbf{let } repeat = \lambda x.x : repeat\ x$$
$$\textbf{in } repeat$$



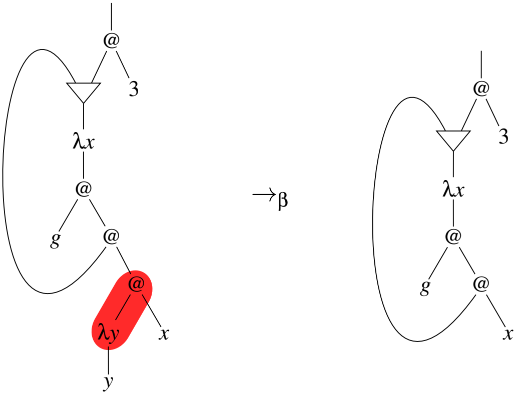
Visible and concealed redexes



Common practice in existing compilers:

- Exhaustive reduction of visible redexes

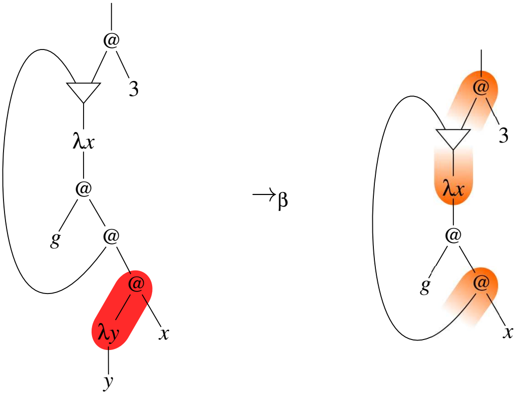
Visible and concealed redexes



Common practice in existing compilers:

- Exhaustive reduction of visible redexes

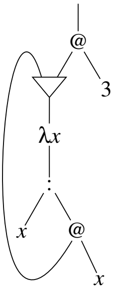
Visible and concealed redexes



Common practice in existing compilers:

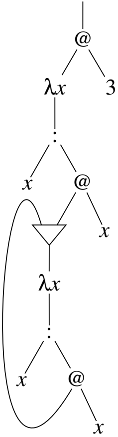
- Exhaustive reduction of visible redexes
- This is in general not possible for concealed redexes

repeat 3

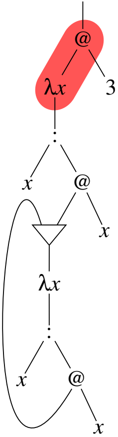


$$\textbf{let } \mathit{repeat} = \lambda x.x : \mathit{repeat}\ x$$
$$\textbf{in } \mathit{repeat}\ 3$$

$\rightarrow_{\triangledown}$

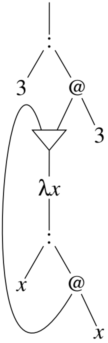


$$\textbf{let}\ repeat = \lambda x.x : repeat\ x$$
$$\textbf{in}\ (\lambda x.x : repeat\ x)\ 3$$



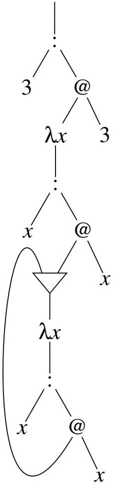
$$\mathbf{let}\ repeat = \lambda x.x : repeat\ x$$
$$\mathbf{in}\ (\lambda x.x : repeat\ x)\ 3$$

$\rightarrow_\beta$

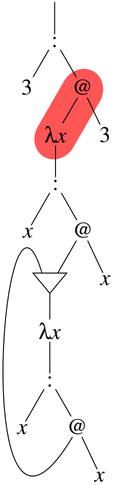


$\textbf{let}\ repeat = \lambda x.x : repeat\ x$
$\textbf{in}\ 3 : repeat\ 3$

$\rightarrow_{\triangledown}$

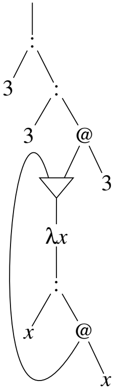


$\mathbf{let}\ repeat = \lambda x.x : repeat\ x$
$\mathbf{in}\ 3 : (\lambda x.x : repeat\ x)\ 3$



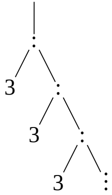
$$\mathbf{let}\ repeat = \lambda x.x : repeat\ x$$
$$\mathbf{in}\ 3 : (\lambda x.x : repeat\ x)\ 3$$

$\rightarrow_\beta$



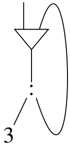
$$\textbf{let}\ \mathit{repeat} = \lambda x.x : \mathit{repeat}\ x$$
$$\textbf{in}\ 3 : 3 : \mathit{repeat}\ 3$$

$\twoheadrightarrow_{\triangledown,\beta}$



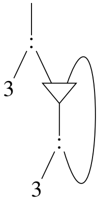
$$3 : 3 : \ldots$$

$\twoheadleftarrow_{\triangledown}$



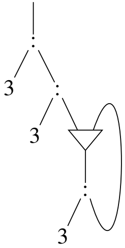
$$\textbf{let } rec = 3 : rec \textbf{ in } rec$$

$\rightarrow_{\triangledown}$



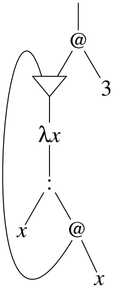
$$\textbf{let } rec = 3 : rec \textbf{ in } 3 : rec$$

$\rightarrow_{\bigtriangledown}$

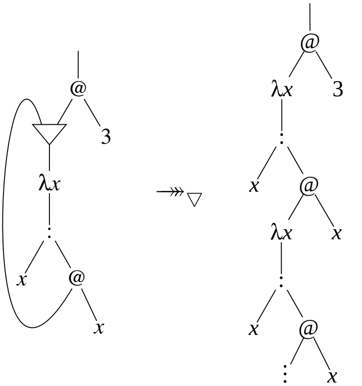


$$\textbf{let } rec = 3 : rec \textbf{ in } 3 : 3 : rec$$

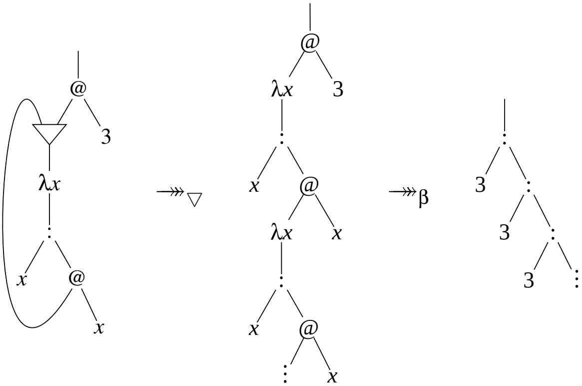
repeat 3



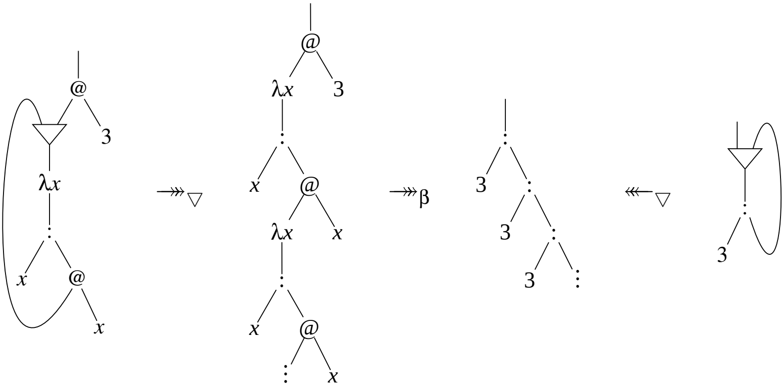
repeat 3



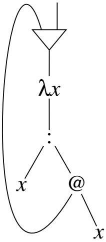
repeat 3



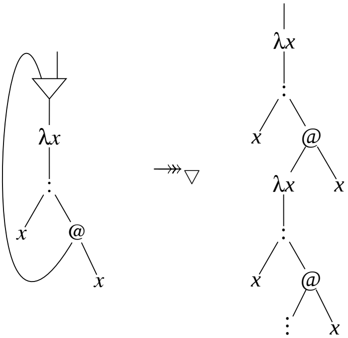
repeat 3



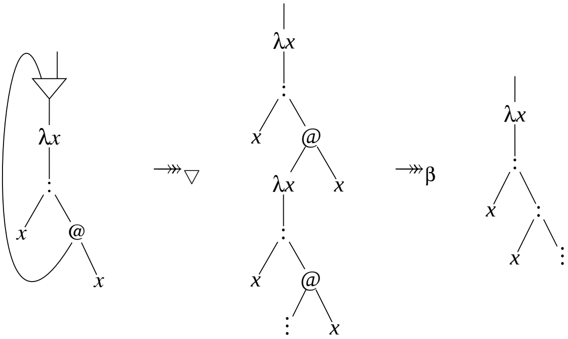
repeat



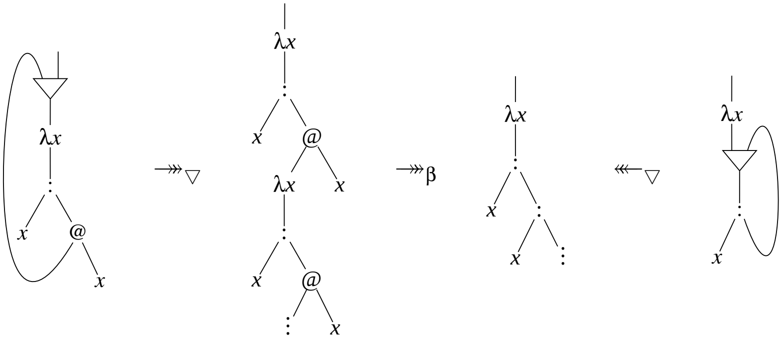
repeat



repeat

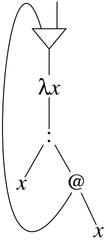


repeat

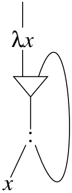


Optimising *repeat*

let *repeat* $= \lambda x.x : repeat\ x$
in *repeat*



$=^{\infty}_{\triangledown,\beta}$



let *repeat* $= \lambda x.$**let** $xs = x : xs$ **in** xs
in *repeat*

Operational equivalence I

Used here:

$$=^{\infty}_{\triangledown,\beta} \;=\; \left(\twoheadleftarrow\!\!\!\!\!\leftarrow_{\triangledown} \cup \twoheadleftarrow\!\!\!\!\!\leftarrow_{\beta} \cup \twoheadrightarrow\!\!\!\!\!\rightarrow_{\beta} \cup \twoheadrightarrow\!\!\!\!\!\rightarrow_{\triangledown}\right)^{*}$$

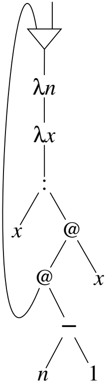
as notion of operational equivalence.

replicate

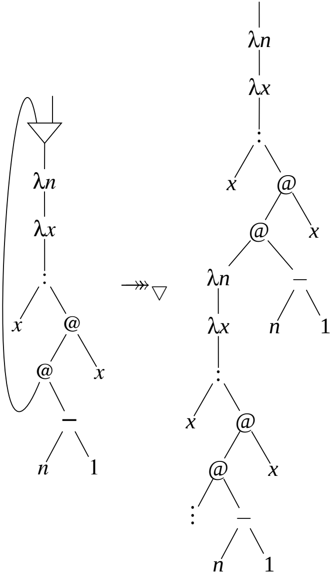
$$
\begin{aligned}
&\mathit{replicate}\; 0\; x = [\,]\\
&\mathit{replicate}\; n\; x = x : \mathit{replicate}\; (n-1)\; x
\end{aligned}
$$

$$
\begin{aligned}
\mathit{replicate}\; n\; x = \mathbf{let}\; &\mathit{rec}\; 0 = [\,]\\
&\mathit{rec}\; n = x : \mathit{rec}\; (n-1)\\
\mathbf{in}\; &\mathit{rec}\; n
\end{aligned}
$$

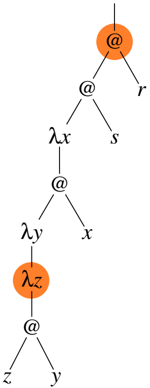
replicate – generalised β-reduction



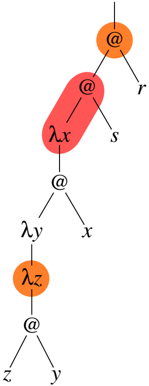
replicate – generalised β-reduction



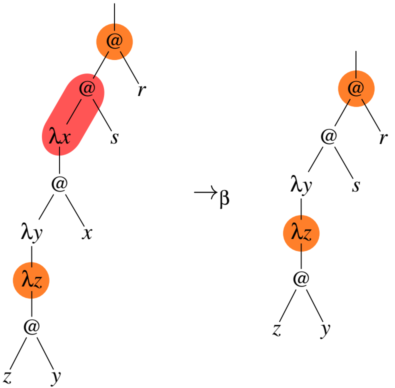
Generalised β-Reduction



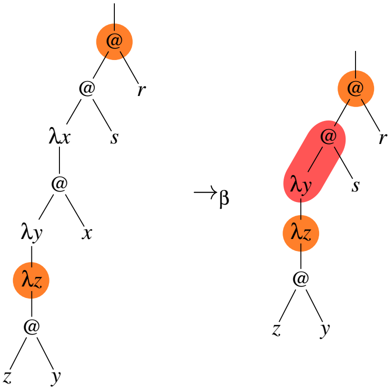
Generalised β-Reduction



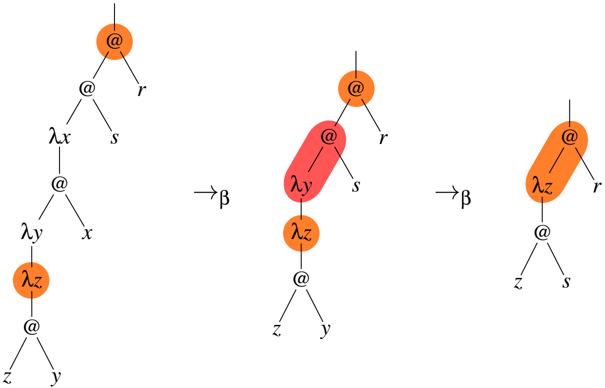
Generalised β-Reduction



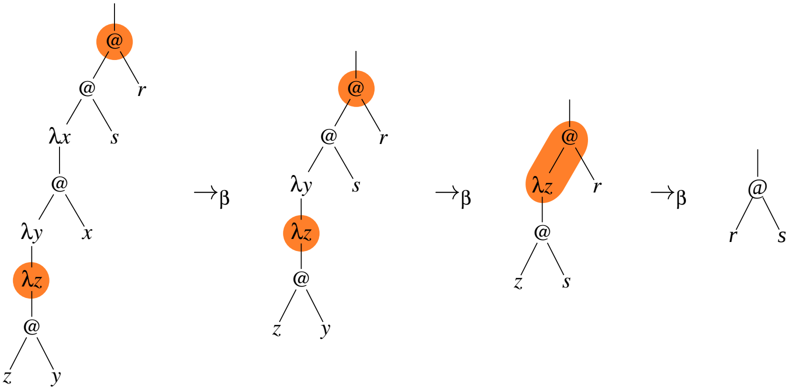
Generalised β-Reduction



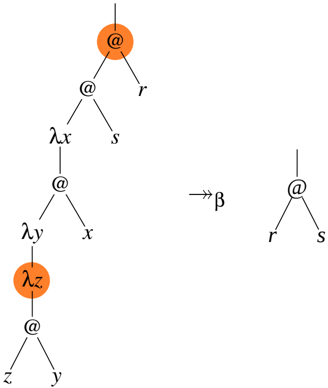
Generalised β-Reduction



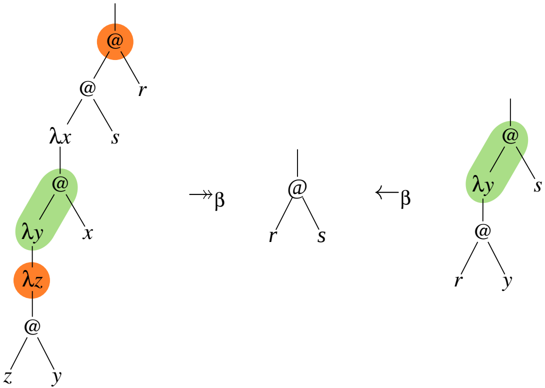
Generalised β-Reduction



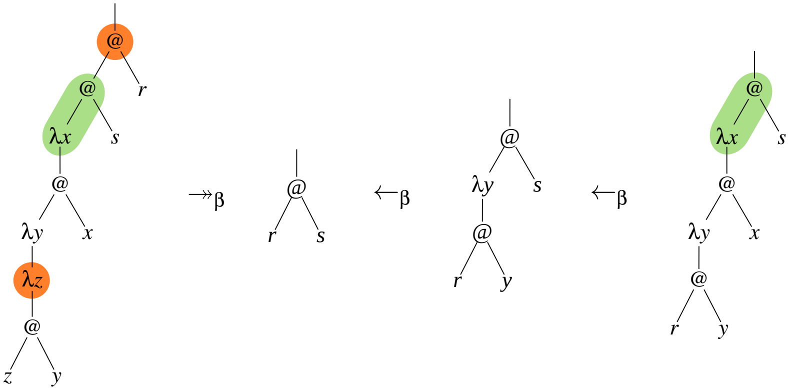
Generalised β-Reduction



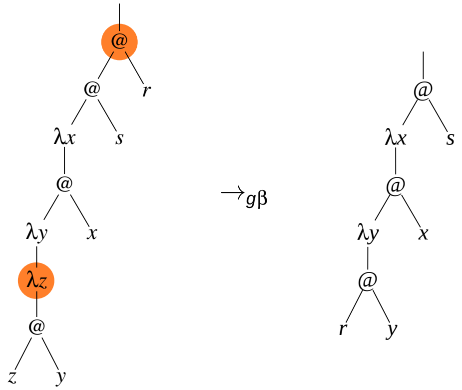
Generalised β-Reduction



Generalised β-Reduction

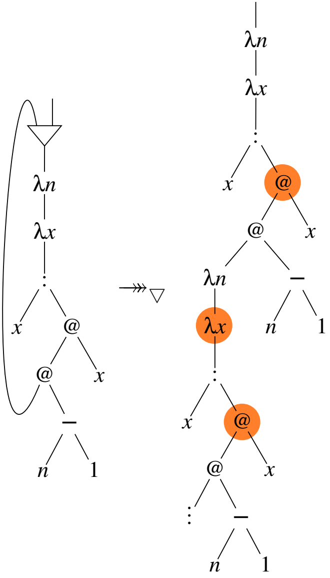


Generalised β-Reduction

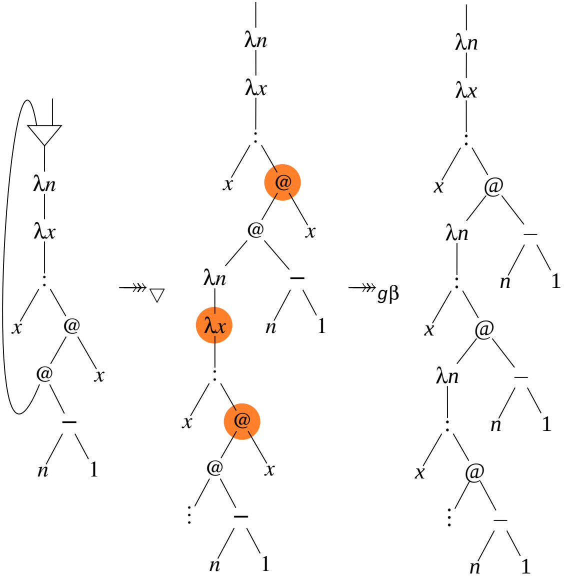


$$\to_{g\beta} \quad \subseteq \quad \twoheadrightarrow_\beta \cdot \to_\beta \cdot \twoheadleftarrow_\beta \quad \subseteq \quad \leftrightarrow^*_\beta$$

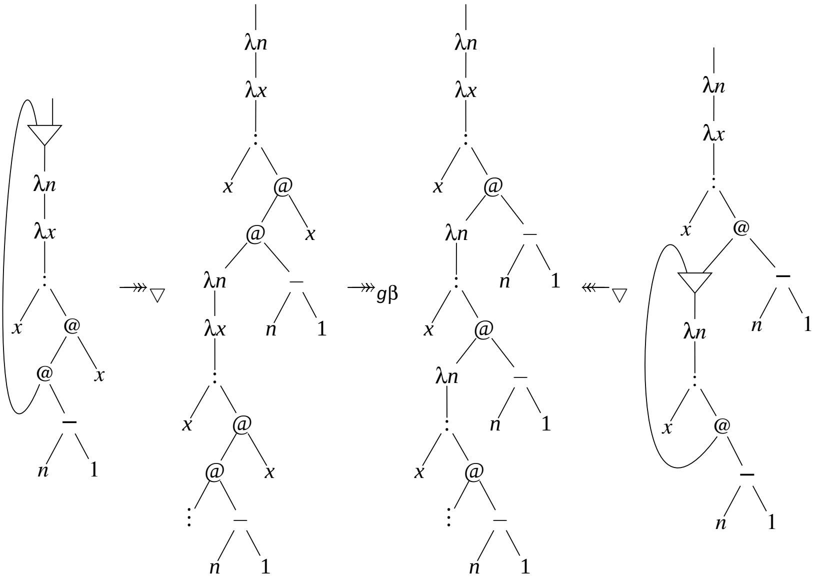
replicate – duplication of the function body



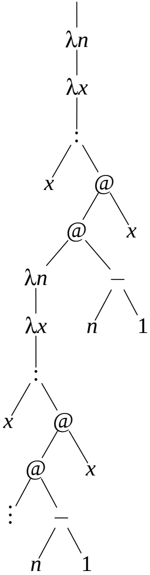
replicate – duplication of the function body



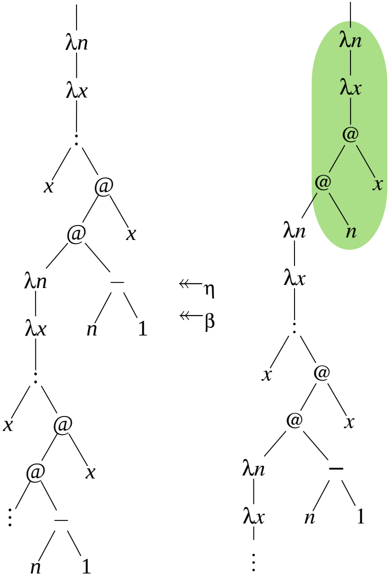
replicate – duplication of the function body



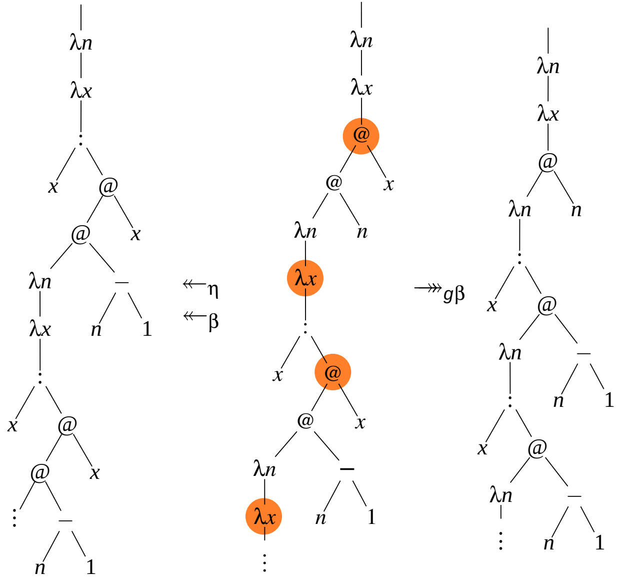
replicate – header trick



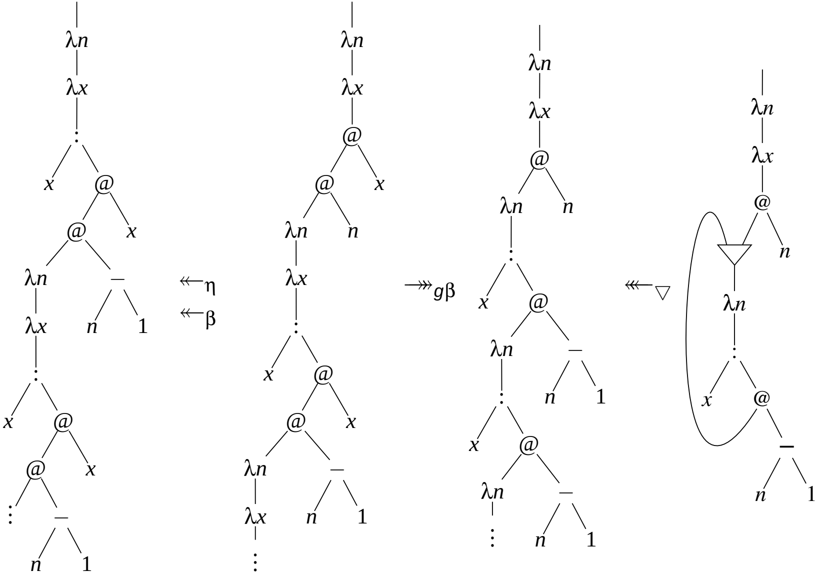
replicate – header trick



replicate – header trick



replicate – header trick



Operational equivalence II

- $g\beta$-Convertibility:

$$=^{\infty}_{\triangledown, g\beta} \quad := \quad ({}_{\triangledown}\!\twoheadleftarrow\!\!\twoheadleftarrow \cup {}_{g\beta}\!\twoheadleftarrow\!\!\twoheadleftarrow \cup \twoheadrightarrow\!\!\twoheadrightarrow_{g\beta} \cup \twoheadrightarrow\!\!\twoheadrightarrow_{\triangledown})^{*}$$

Rewrite Rule Formulation

$$f = \lambda x_1. \dots \lambda x_n.\lambda y.C\,[f\ t_1 \dots t_n\ y]$$

$$\to$$

$$\begin{aligned} f = {} & \lambda x_1. \dots \lambda x_n.\lambda y. \\ & \textbf{let } f' = \lambda x_1. \dots \lambda x_n.C\,[f'\ t_1 \dots t_n] \\ & \textbf{in } f'\ x_1 \dots x_n \end{aligned}$$

Rewriting *repeat*

$$\begin{aligned}&\textbf{let } \mathit{repeat} = \lambda x.x : \mathit{repeat}\ x\\&\textbf{in } \mathit{repeat}\end{aligned}$$

$$\rightarrow$$

$$\begin{aligned}&\textbf{let } \mathit{repeat} = \lambda x.\textbf{let } \mathit{xs} = x : \mathit{xs} \textbf{ in } \mathit{xs}\\&\textbf{in } \mathit{repeat}\end{aligned}$$

Rewriting *replicate*

$$\begin{aligned} \mathit{replicate}\ 0\ x &= [\,] \\ \mathit{replicate}\ n\ x &= x : \mathit{replicate}\ (n-1)\ x \end{aligned}$$

$$\rightarrow$$

$$\begin{aligned} \mathit{replicate}\ n\ x = \mathbf{let}\ & \mathit{rec}\ 0 = [\,] \\ & \mathit{rec}\ n = x : \mathit{rec}\ (n-1) \\ \mathbf{in}\ & \mathit{rec}\ n \end{aligned}$$

Rewriting *append*

$$
\begin{aligned}
(+\!\!+)\ [\ ]\ ys &= ys \\
(+\!\!+)\ (x : xs)\ ys &= x : xs +\!\!+ ys
\end{aligned}
$$

$$\to$$

$$
\begin{aligned}
(+\!\!+)\ xs\ ys = \mathbf{let}\ & rec\ [\ ] = ys \\
& rec\ (x : xs) = x : rec\ xs \\
\mathbf{in}\ & rec\ xs
\end{aligned}
$$

Rewriting *map*

$$
\begin{array}{l}
\mathit{map}\ \_\ [\,] \qquad\quad = [\,] \\
\mathit{map}\ f\ \ (x : xs) = f\ x : \mathit{map}\ f\ xs
\end{array}
$$

$$\rightarrow$$

$$
\begin{array}{l}
\mathit{map}\ f = \textbf{let}\ \mathit{rec}\ [\,] \qquad\quad = [\,] \\
\qquad\qquad\qquad\ \mathit{rec}\ (x : xs) = f\ x : \mathit{rec}\ xs \\
\qquad\qquad\ \ \textbf{in}\ \mathit{rec}
\end{array}
$$

Rewriting *until*

$$until\ p\ f\ x = \textbf{if}\ p\ x\ \textbf{then}\ x\ \textbf{else}\ until\ p\ f\ (f\ x)$$

$$\rightarrow$$

$$\begin{aligned} until\ p\ f\ x = {} & \textbf{let}\ rec\ x = \textbf{if}\ p\ x\ \textbf{then}\ x\ \textbf{else}\ rec\ (f\ x) \\ & \textbf{in}\ \ rec\ x \end{aligned}$$

Rewriting the Thue-Morse Sequence

$$\begin{array}{l}
\textbf{let}\; x\; a\; b = b : zip\; (x\; a\; b)\; (y\; a\; b) \\
\quad y\; s\; t\; = s : zip\; (y\; s\; t)\; (x\; s\; t) \\
\quad zip\; (x : xs)\; (y : ys) = x : y : zip\; xs\; ys \\
\textbf{in}\; x\; 0\; 1
\end{array}$$

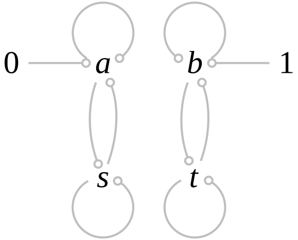
$$\rightarrow$$

$$\begin{array}{l}
\textbf{let}\; x\; a\; b = \textbf{let}\; x' = b : zip\; x'\; (y\; a\; b)\; \textbf{in}\; x' \\
\quad y\; s\; t\; = \textbf{let}\; y' = s : zip\; y'\; (x\; s\; t)\; \textbf{in}\; y' \\
\quad zip\; (x : xs)\; (y : ys) = x : y : zip\; xs\; ys \\
\textbf{in}\; x\; 0\; 1
\end{array}$$

Binding-Graph Method

$$
\begin{array}{l}
\textbf{let}\; x\,a\,b = b : zip\,(x\,a\,b)\,(y\,a\,b) \\
\quad\; y\,s\,t\; = s : zip\,(y\,s\,t)\;\;(x\,s\,t) \\
\quad\; zip\,(x : xs)\,(y : ys) = x : y : zip\,xs\,ys \\
\textbf{in}\; x\,0\,1
\end{array}
$$

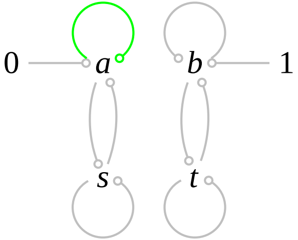
Binding relation: $\multimap \;\subseteq\; S \times S$



Binding-Graph Method

$$\begin{aligned}
&\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b)\\
&\quad y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t)\\
&\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys\\
&\textbf{in}\ x\ 0\ 1
\end{aligned}$$

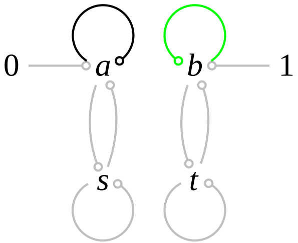
Binding relation: $\multimap\ \subseteq\ S \times S$



Binding-Graph Method

$$
\begin{array}{l}
\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b) \\
\quad y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t) \\
\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys \\
\textbf{in}\ x\ 0\ 1
\end{array}
$$

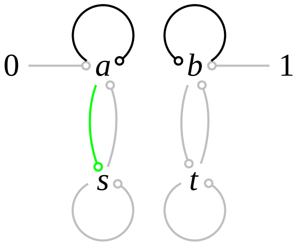
Binding relation: $\multimap \ \subseteq \ S \times S$



Binding-Graph Method

$$
\begin{aligned}
&\mathbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b)\\
&\quad y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t)\\
&\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys\\
&\mathbf{in}\ x\ 0\ 1
\end{aligned}
$$

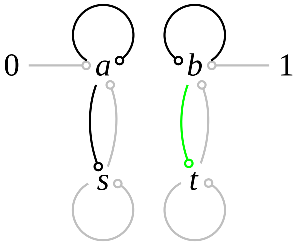
Binding relation: $\circ\!\!-\ \subseteq\ S \times S$



Binding-Graph Method

$$\begin{aligned}
&\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b) \\
&\quad y\ s\ t \ = s : zip\ (y\ s\ t)\ (x\ s\ t) \\
&\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys \\
&\textbf{in}\ x\ 0\ 1
\end{aligned}$$

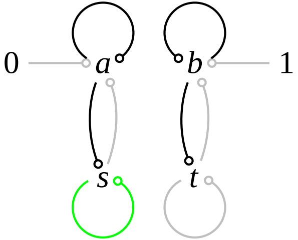
Binding relation: $\multimap\ \subseteq\ S \times S$



Binding-Graph Method

$$
\begin{aligned}
&\textbf{let}\; x\; a\; b = b : zip\; (x\; a\; b)\; (y\; a\; b)\\
&\quad y\; s\; t = s : zip\; (y\; s\; t)\; (x\; s\; t)\\
&\quad zip\; (x : xs)\; (y : ys) = x : y : zip\; xs\; ys\\
&\textbf{in}\; x\; 0\; 1
\end{aligned}
$$

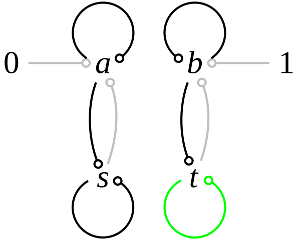
Binding relation: $\multimap \;\subseteq\; S \times S$



Binding-Graph Method

$$\begin{array}{l}
\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b) \\
\quad y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t) \\
\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys \\
\textbf{in}\ x\ 0\ 1
\end{array}$$

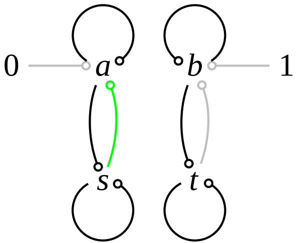
Binding relation: $\circ\!\!-\ \subseteq\ S \times S$



Binding-Graph Method

$$\begin{array}{l}
\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b) \\
\quad y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t) \\
\quad zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys \\
\textbf{in}\ x\ 0\ 1
\end{array}$$

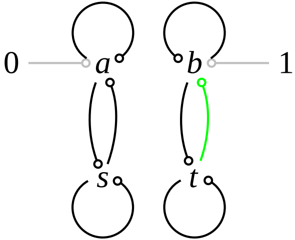
Binding relation: $\circ\!\!-\ \subseteq\ S \times S$



Binding-Graph Method

$$
\begin{aligned}
&\textbf{let}\; x\;a\;b = b : zip\;(x\;a\;b)\;(y\;a\;b)\\
&\qquad y\;s\;t = s : zip\;(y\;s\;t)\;(x\;s\;t)\\
&\qquad zip\;(x : xs)\;(y : ys) = x : y : zip\;xs\;ys\\
&\textbf{in}\; x\;0\;1
\end{aligned}
$$

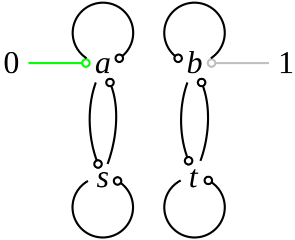
Binding relation: $\multimap \;\subseteq\; S \times S$



Binding-Graph Method

$$
\begin{aligned}
\textbf{let}\ & x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b) \\
& y\ s\ t\ = s : zip\ (y\ s\ t)\ (x\ s\ t) \\
& zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys \\
\textbf{in}\ & x\ 0\ 1
\end{aligned}
$$

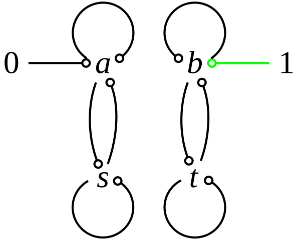
Binding relation: $\circ\!\!-\ \subseteq\ S \times S$



Binding-Graph Method

$$
\begin{array}{l}
\mathbf{let}\; x\; a\; b = b : zip\; (x\; a\; b)\; (y\; a\; b) \\
\quad y\; s\; t = s : zip\; (y\; s\; t)\; (x\; s\; t) \\
\quad zip\; (x : xs)\; (y : ys) = x : y : zip\; xs\; ys \\
\mathbf{in}\; x\; 0\; 1
\end{array}
$$

Binding relation: $\multimap \;\subseteq\; S \times S$

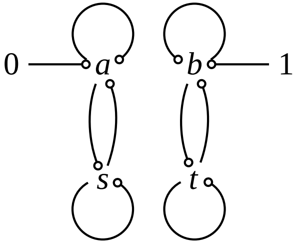


Strong domination

Strong domination:

$$sdom_G(d,w) :=$$

$$\forall p_0 \rightarrowtail \ldots \rightarrowtail p_n = v \qquad n \geq 0$$

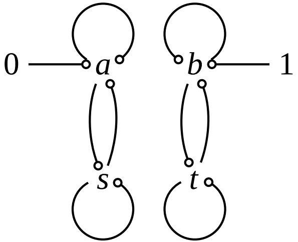


Strong domination

Strong domination:

$$sdom_G(d,w) :=$$

$$\forall p_0 \rightarrowtail \ldots \rightarrowtail p_n = v \;:\; d \in \{p_0,\ldots,p_n\} \qquad n \geq 0$$

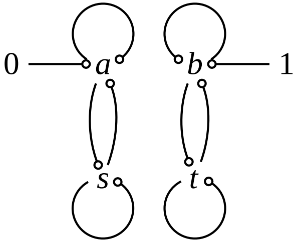


Strong domination

Strong domination:

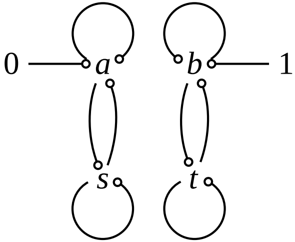
$$sdom_G(d,w) :=$$

$$\forall p_0 \rightarrowtail \ldots \rightarrowtail p_n = v \;:\; d \in \{p_0,\ldots,p_n\} \vee d \rightarrowtail^{+} p_0 \wedge p_0 \not\rightarrowtail^{+} d \quad n \geq 0$$



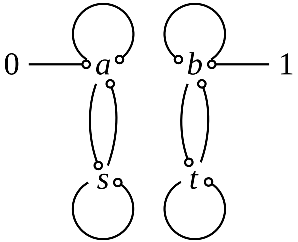
Optimising the Thue-Morse Sequence

$$
\begin{aligned}
&\textbf{let}\ x\ a\ b = b : zip\ (x\ a\ b)\ (y\ a\ b)\\
&\quad\ y\ s\ t\ = s : zip\ (y\ s\ t)\ \ (x\ s\ t)\\
&\quad\ zip\ (x : xs)\ (y : ys) = x : y : zip\ xs\ ys\\
&\textbf{in}\ x\ 0\ 1
\end{aligned}
$$



Optimising the Thue-Morse Sequence

$$
\begin{array}{l}
\textbf{let}\; x = 1 : zip\; x\; y \\
\quad\; y = 0 : zip\; y\; x \\
\quad\; zip\; (x : xs)\; (y : ys) = x : y : zip\; xs\; ys \\
\textbf{in}\; x
\end{array}
$$



Current Plans

- practical aspects
  - implementation
  - repetitive reduction patters in the wild: population census
  - benchmarks
  - analysis of effects for different run-time systems
- theoretical aspects
  - HRS formulation
  - domination after unfolding
  - efficiency measure for comparing different results of optimisation
  - interactions between optimisation of different parameter cycles
  - correctness proof
- full paper

Thanks

for your attention!

and for inspiration, and many discussions, to:

- Doaitse Swierstra
- Vincent van Oostrom