Circular vs. Higher-Order Shortcut Fusion

Janis Voigtländer

Technische Universität Dresden

March 30th, 2009

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example: $upTo\ n = go\ 1$

$$\begin{aligned} &\textbf{where}\ go\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,] \\ &\qquad\qquad\qquad\quad \textbf{else}\ \ i : go\ (i+1) \end{aligned}$$

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example:
$$\begin{aligned} &\mathit{upTo}\ n = \mathit{go}\ 1\\ &\qquad \textbf{where}\ \mathit{go}\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,]\\ &\qquad\qquad\qquad\qquad\quad\ \textbf{else}\ \ i : \mathit{go}\ (i+1) \end{aligned}$$

$$\begin{aligned} \mathit{sum}\ [\,] &= 0\\ \mathit{sum}\ (x : \mathit{xs}) &= x + \mathit{sum}\ \mathit{xs} \end{aligned}$$

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example: $upTo\ n = go\ 1$

$$\begin{aligned} &\textbf{where}\ go\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,] \\ &\qquad\qquad\qquad\qquad \textbf{else}\ \ i : go\ (i+1) \end{aligned}$$

$$\begin{aligned} sum\ [\,] &= 0 \\ sum\ (x : xs) &= x + sum\ xs \end{aligned}$$

Problem: Expressions like

$$sum\ (upTo\ 10)$$

require explicit construction of intermediate results.

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example: $upTo\ n = go\ 1$
$\quad\quad \textbf{where}\ go\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,]$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{else}\ \ i : go\ (i+1)$

$$sum\ [\,] \quad\quad = 0$$
$$sum\ (x : xs) = x + sum\ xs$$

Problem: Expressions like

$$sum\ (upTo\ 10)$$

require explicit construction of intermediate results.

Solution: 1. Write *upTo* in terms of *build*.

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example:
$$\begin{aligned} &upTo\ n = go\ 1 \\ &\quad\quad \textbf{where}\ go\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,] \\ &\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{else}\ \ i : go\ (i+1) \end{aligned}$$

$$\begin{aligned} sum\ [\,] &= 0 \\ sum\ (x : xs) &= x + sum\ xs \end{aligned}$$

Problem: Expressions like

$$sum\ (upTo\ 10)$$

require explicit construction of intermediate results.

Solution:

1. Write *upTo* in terms of *build*.
2. Write *sum* in terms of *foldr*.

Classical Shortcut Fusion [Gill et al., FPCA'93]

Example:
$$\begin{aligned} & upTo\ n = go\ 1 \\ & \quad \textbf{where}\ go\ i = \textbf{if}\ i > n\ \textbf{then}\ [\,] \\ & \qquad\qquad\qquad\qquad \textbf{else}\ \ i : go\ (i+1) \end{aligned}$$

$$\begin{aligned} sum\ [\,] &= 0 \\ sum\ (x : xs) &= x + sum\ xs \end{aligned}$$

Problem: Expressions like

$$sum\ (upTo\ 10)$$

require explicit construction of intermediate results.

Solution:

1. Write *upTo* in terms of *build*.
2. Write *sum* in terms of *foldr*.
3. Use the following fusion rule:

$$foldr\ h_1\ h_2\ (build\ g) \rightsquigarrow g\ h_1\ h_2$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow c \rightarrow (a, z)) \rightarrow c \rightarrow ([b], z)$$
$$\mathit{buildp}\ g\ \ c = g\ \ (:)\ [\,]\ c$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$\mathit{buildp}\ g\ c = g\ (:)\ [\,]\ c$$

$$\mathit{filterAndCount} :: (b \to \mathit{Bool}) \to [b] \to ([b], \mathit{Int})$$
$$\mathit{filterAndCount}\ f = \mathit{buildp}\ \cdots$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$\mathit{buildp}\ g\ c = g\ (:)\ [\,]\ c$$

Consuming intermediate results:

$$\mathit{pfold} :: (b \to a \to z \to a) \to (z \to a) \to ([b], z) \to a$$
$$\mathit{pfold}\ h_1\ h_2\ (bs, z) = \mathit{foldr}\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ bs$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$buildp :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$buildp\ g\ \ c = g\ \ (:)\ [\,]\ c$$

Consuming intermediate results:

$$pfold :: (b \to a \to z \to a) \to (z \to a) \to ([b], z) \to a$$
$$pfold\ \ h_1\ \ h_2\ (bs, z) = foldr\ (\lambda b\ \ a \to h_1\ \ b\ \ a\ \ z)\ (h_2\ \ z)\ bs$$

$$normalise :: ([Int], Int) \to [Float]$$
$$normalise = pfold\ \ \cdots$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow c \rightarrow (a, z)) \rightarrow c \rightarrow ([b], z)$$
$$\mathit{buildp}\ g\ \ c = g\ (:)\ [\,]\ c$$

Consuming intermediate results:

$$\mathit{pfold} :: (b \rightarrow a \rightarrow z \rightarrow a) \rightarrow (z \rightarrow a) \rightarrow ([b], z) \rightarrow a$$
$$\mathit{pfold}\ \ h_1\ \ h_2\ (bs, z) = \mathit{foldr}\ (\lambda b\ \ a \rightarrow h_1\ \ b\ \ a\ \ z)\ (h_2\ \ z)\ bs$$

The fusion rule:

$$\mathit{pfold}\ \ h_1\ \ h_2\ (\mathit{buildp}\ g\ \ c)$$
$$\rightsquigarrow$$
$$\mathbf{let}\ (a, z) = g\ (\lambda b\ \ a \rightarrow h_1\ \ b\ \ a\ \ z)\ (h_2\ \ z)\ c\ \mathbf{in}\ a$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$\mathit{buildp}\ g\ \ c = g\ (:)\ [\,]\ c$$

Consuming intermediate results:

$$\mathit{pfold} :: (b \to a \to z \to a) \to (z \to a) \to ([b], z) \to a$$
$$\mathit{pfold}\ \ h_1\ \ h_2\ (bs, z) = \mathit{foldr}\ (\lambda b\ a \to h_1\ \ b\ a\ z)\ (h_2\ \ z)\ bs$$

The fusion rule:

$$\mathit{pfold}\ \ h_1\ \ h_2\ (\mathit{buildp}\ g\ \ c)$$
$$\leadsto$$
$$\mathbf{let}\ (a, z) = g\ (\lambda b\ a \to h_1\ \ b\ a\ z)\ (h_2\ \ z)\ c\ \mathbf{in}\ a$$

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$buildp :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$buildp\ g\ c = g\ (:)\ [\,]\ c$$

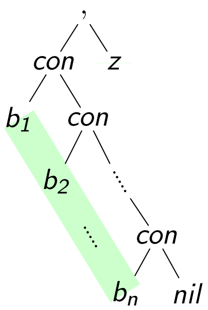
Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow c \rightarrow (a, z)) \rightarrow c \rightarrow ([b], z)$$
$$\mathit{buildp}\ g\ c = g\ (:)\ [\,]\ c$$

The type of g forces it to be essentially of the following form:

$\lambda con\ nil\ c \rightarrow$

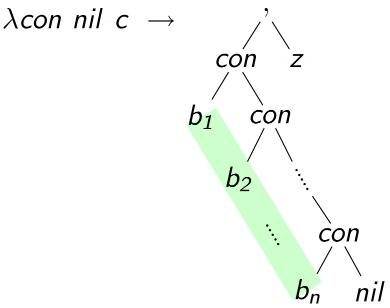


Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Producing intermediate results:

$$buildp :: (\forall a.\ (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow c \rightarrow (a, z)) \rightarrow c \rightarrow ([b], z)$$
$$buildp\ g\ c = g\ (:)\ []\ c$$

The type of g forces it to be essentially of the following form:



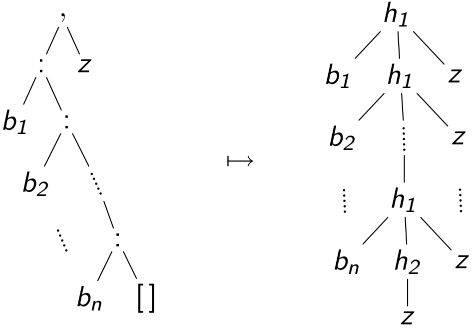
Formal justification: free theorems [Wadler, FPCA'89]

Circular Shortcut Fusion [Fernandes et al., Haskell'07]

Consuming intermediate results:

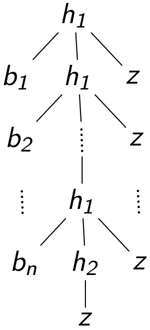
$$pfold :: (b \to a \to z \to a) \to (z \to a) \to ([b], z) \to a$$
$$pfold\ h_1\ h_2\ (bs, z) = foldr\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ bs$$

A concrete output (*buildp g c*) will be consumed as follows:



Circular Shortcut Fusion [Fernandes et al., Haskell'07]

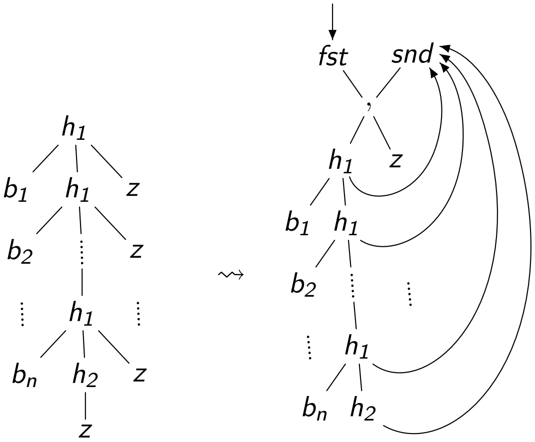
$pfold\ h_1\ h_2\ (g\ (:)\ []\ c) \rightsquigarrow$



$\rightsquigarrow$

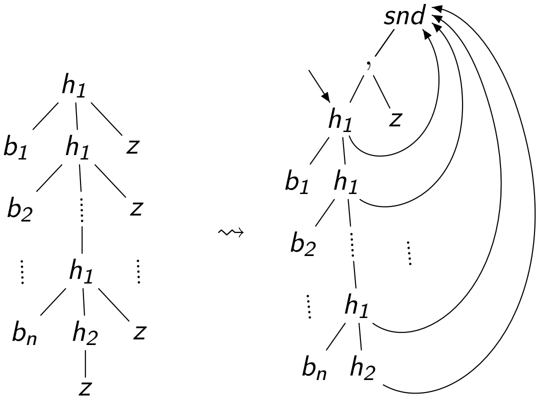
Circular Shortcut Fusion [Fernandes et al., Haskell'07]

$$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$



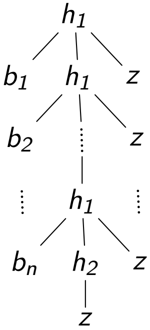
Circular Shortcut Fusion [Fernandes et al., Haskell'07]

$$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

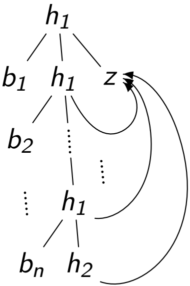


Circular Shortcut Fusion [Fernandes et al., Haskell'07]

$$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$



$\rightsquigarrow$



This is Where I got Interested

- Free-theorems-based transformations had been studied before.

This is Where I got Interested

- Free-theorems-based transformations had been studied before.
- . . . but been found to not be totally correct when considering certain language features [Johann and V., POPL'04].

This is Where I got Interested

- Free-theorems-based transformations had been studied before.
- ...but been found to not be totally correct when considering certain language features [Johann and V., POPL'04].
- Circular shortcut fusion depends on evaluation order, which is precisely a "dangerous" corner for free theorems.

This is Where I got Interested

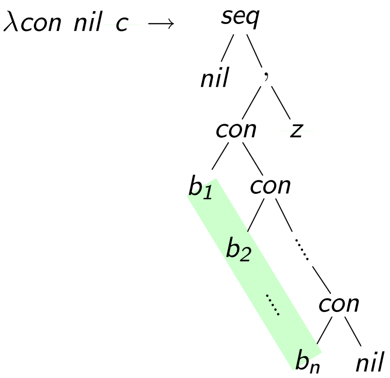
- Free-theorems-based transformations had been studied before.
- ...but been found to not be totally correct when considering certain language features [Johann and V., POPL'04].
- Circular shortcut fusion depends on evaluation order, which is precisely a "dangerous" corner for free theorems.
- So would it be possible to manufacture counterexamples?

A Problem with Selective Strictness

Producing intermediate results:

$$buildp :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$buildp\ g\ c = g\ (:)\ [\,]\ c$$

In Haskell, g could also be, for example, of the following form:



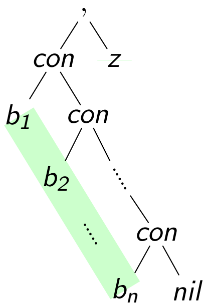
A Problem with Selective Strictness

Producing intermediate results:

$$\mathit{buildp} :: (\forall a.\ (b \to a \to a) \to a \to c \to (a, z)) \to c \to ([b], z)$$
$$\mathit{buildp}\ g\ c = g\ (:)\ []\ c$$

The type of g forces it to be essentially of the following form:

$\lambda \mathit{con}\ \mathit{nil}\ c \to$



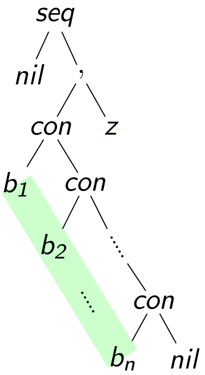
A Problem with Selective Strictness

Producing intermediate results:

$$buildp :: (\forall a.\ (b \rightarrow a \rightarrow a) \rightarrow a \rightarrow c \rightarrow (a, z)) \rightarrow c \rightarrow ([b], z)$$
$$buildp\ g\ c = g\ (:)\ [\,]\ c$$

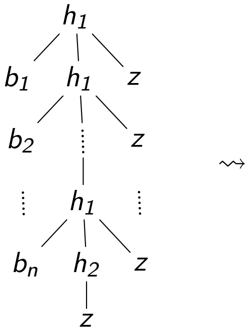
In Haskell, g could also be, for example, of the following form:

$\lambda con\ nil\ c \rightarrow$



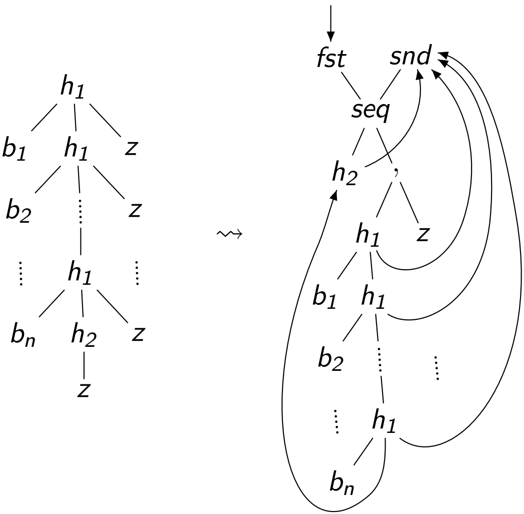
A Problem with Selective Strictness

This would lead to the following replacement:



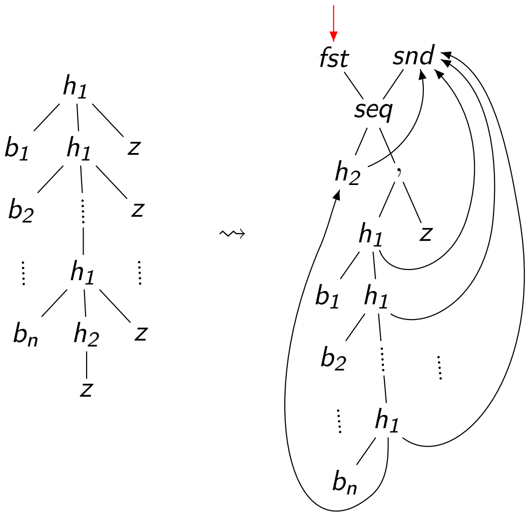
A Problem with Selective Strictness

This would lead to the following replacement:



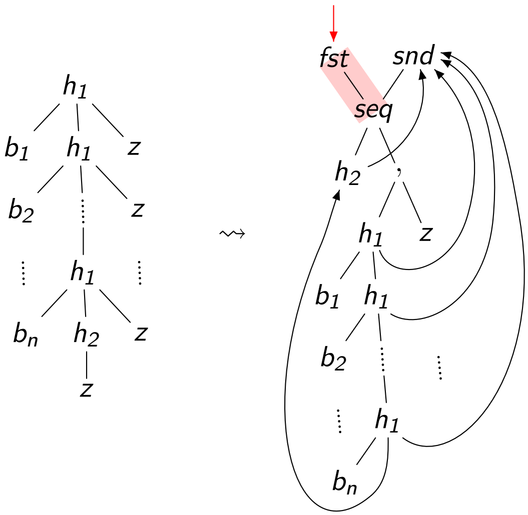
A Problem with Selective Strictness

This would lead to the following replacement:



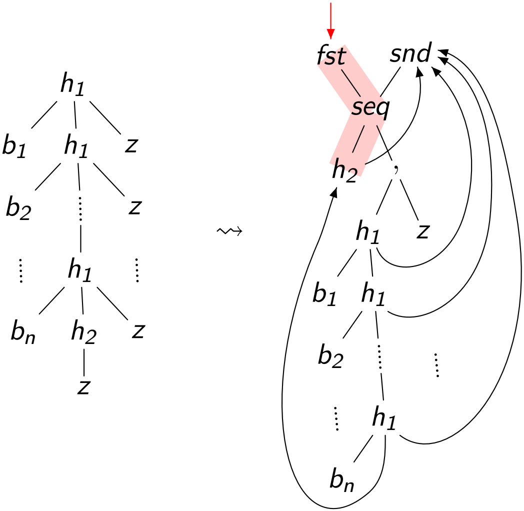
A Problem with Selective Strictness

This would lead to the following replacement:



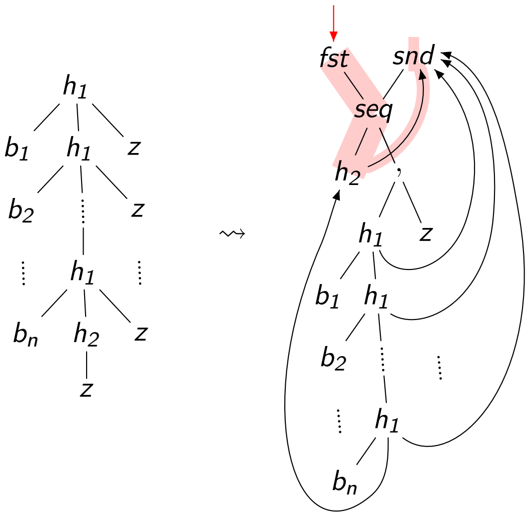
A Problem with Selective Strictness

This would lead to the following replacement:



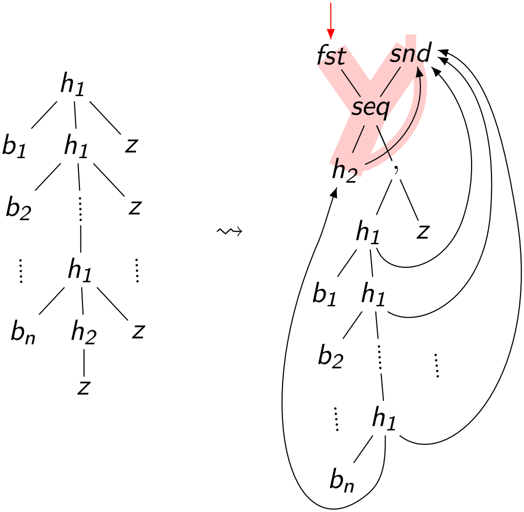
A Problem with Selective Strictness

This would lead to the following replacement:



A Problem with Selective Strictness

This would lead to the following replacement:



Total and Partial Correctness [V., FLOPS'08]

Theorem 1

If $h_2 \perp \neq \perp$ and $h_1 \perp \perp \perp \neq \perp$, then

$$pfold\ h_1\ h_2\ (buildp\ g\ c)$$
$$=$$
$$\mathbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \mathbf{in}\ a$$

Total and Partial Correctness [V., FLOPS'08]

Theorem 1

If $h_2 \perp \neq \perp$ and $h_1 \perp \perp \perp \neq \perp$, then

$$\begin{array}{c} \mathit{pfold}\ h_1\ h_2\ (\mathit{buildp}\ g\ c) \\ = \\ \mathbf{let}\ (a, z) = g\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ c\ \mathbf{in}\ a \end{array}$$

Theorem 2

Without preconditions,

$$\begin{array}{c} \mathit{pfold}\ h_1\ h_2\ (\mathit{buildp}\ g\ c) \\ \sqsupseteq \\ \mathbf{let}\ (a, z) = g\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ c\ \mathbf{in}\ a \end{array}$$

Replacing Circularity by Higher-Orderedness

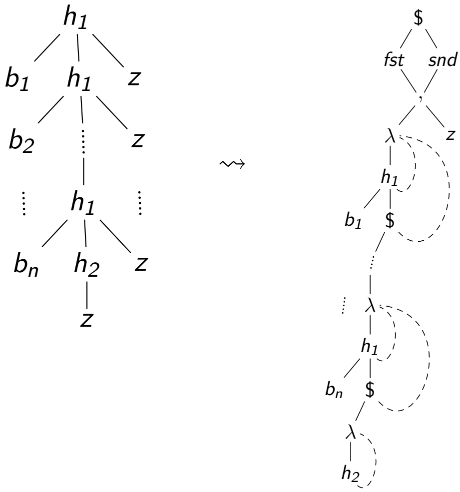
$$\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

Replacing Circularity by Higher-Orderedness

$$
\begin{aligned}
pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \leadsto\ & \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a \\
& \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c \\
& \textbf{of}\ (k, z) \rightarrow k\ z
\end{aligned}
$$

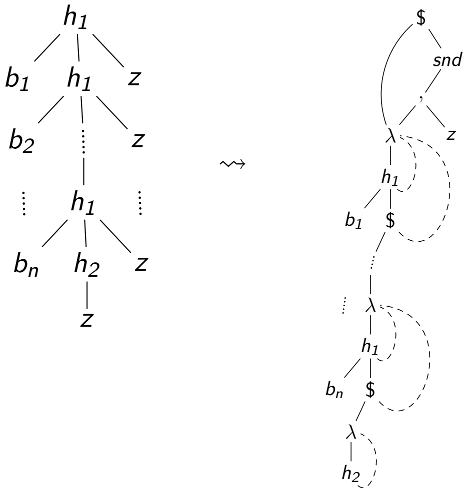
Replacing Circularity by Higher-Orderedness

$$\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \leadsto \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c\ \textbf{of}\ (k, z) \rightarrow k\ z$$



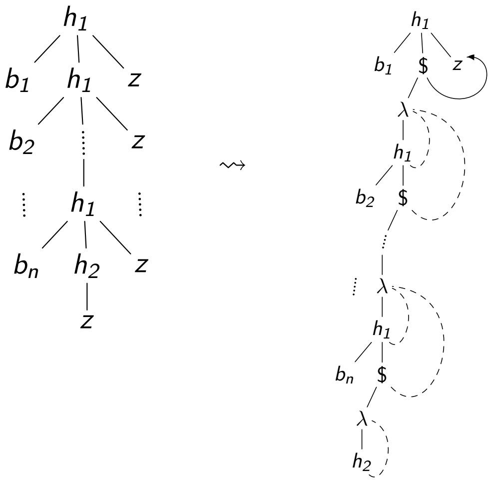
Replacing Circularity by Higher-Orderedness

$$
\begin{aligned}
pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow\ & \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c \\
& \textbf{of}\ (k, z) \rightarrow k\ z
\end{aligned}
$$



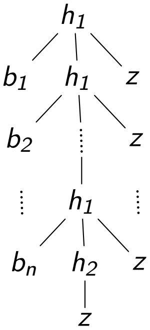
Replacing Circularity by Higher-Orderedness

$$\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \mathbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c$$
$$\mathbf{of}\ (k, z) \rightarrow k\ z$$

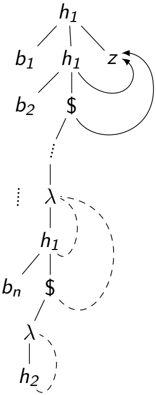


Replacing Circularity by Higher-Orderedness

$$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{case}\ g\ (\lambda b\ k\ z \to h_1\ b\ (k\ z)\ z)\ (\lambda z \to h_2\ z)\ c\ \textbf{of}\ (k, z) \to k\ z$$

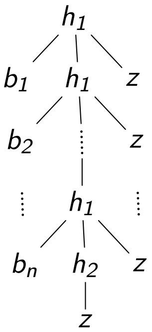


$\rightsquigarrow$

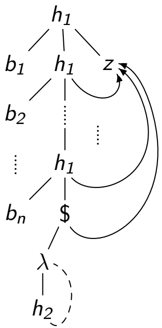


Replacing Circularity by Higher-Orderedness

$$\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c\ \textbf{of}\ (k, z) \rightarrow k\ z$$

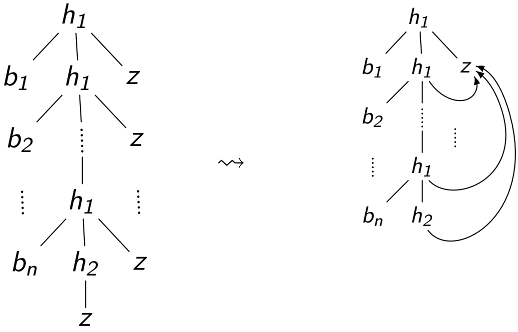


$\rightsquigarrow$



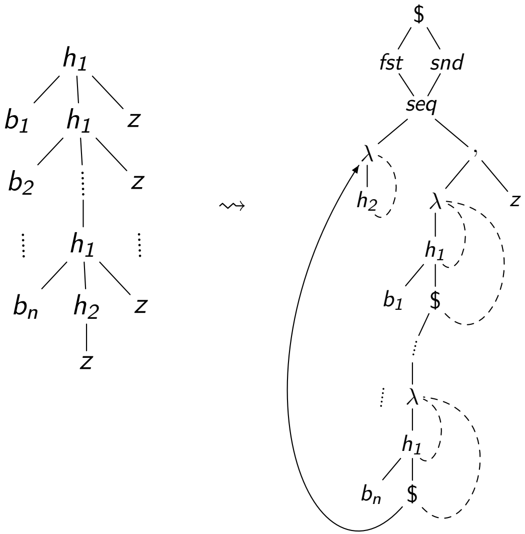
Replacing Circularity by Higher-Orderedness

$$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c$$
$$\textbf{of}\ (k, z) \rightarrow k\ z$$



No Problem with Selective Strictness

For a g of the problematic form considered earlier:



Total Correctness [V., FLOPS'08]

Theorem 3

Without preconditions,

$$pfold\ h_1\ h_2\ (buildp\ g\ c)$$
$$=$$
$$\textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c\ \textbf{of}\ (k, z) \rightarrow k\ z$$

Total Correctness [V., FLOPS'08]

Theorem 3

Without preconditions,

$$pfold\ h_1\ h_2\ (buildp\ g\ c)$$
$$=$$
$$\textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c\ \textbf{of}\ (k, z) \rightarrow k\ z$$

Circular vs. Higher-Order Shortcut Fusion

Which flavour is better?

Circular vs. Higher-Order Shortcut Fusion

Which flavour is better?

- Intellectually, I find the circular approach more fascinating.

Circular vs. Higher-Order Shortcut Fusion

Which flavour is better?

- Intellectually, I find the circular approach more fascinating.
- But semantically, the high-order approach is more robust.

Circular vs. Higher-Order Shortcut Fusion

Which flavour is better?

- Intellectually, I find the circular approach more fascinating.
- But semantically, the high-order approach is more robust.
- Performance measurements do not give a very clear picture.

Circular vs. Higher-Order Shortcut Fusion

Which flavour is better?

- Intellectually, I find the circular approach more fascinating.
- But semantically, the high-order approach is more robust.
- Performance measurements do not give a very clear picture.
- There are interesting interactions with rather low-level details of the language implementation!

Tricky Sharing Issues — Circular Shortcut Fusion

$$\textit{pfold}\ h_1\ h_2\ (\textit{buildp}\ g\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

Tricky Sharing Issues — Circular Shortcut Fusion

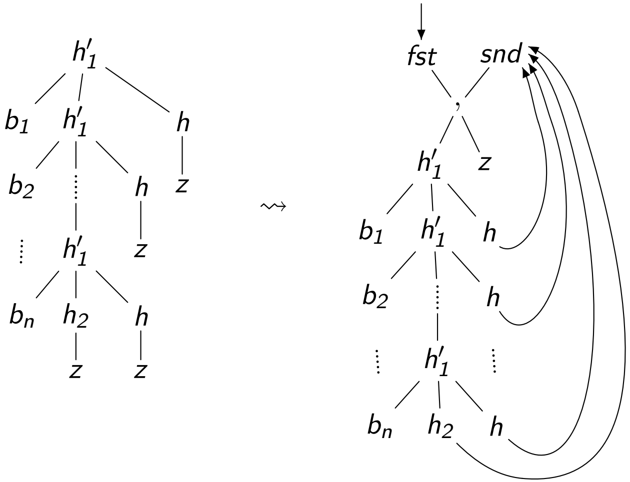
$$pfold\ h_1\ h_2\ (buildp\ g\ c) \leadsto \mathbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \mathbf{in}\ a$$

If $h_1 = \lambda b\ a\ z \rightarrow h_1'\ b\ a\ (h\ z)$,

Tricky Sharing Issues — Circular Shortcut Fusion

$$pfold\ h_1\ h_2\ (buildp\ g\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

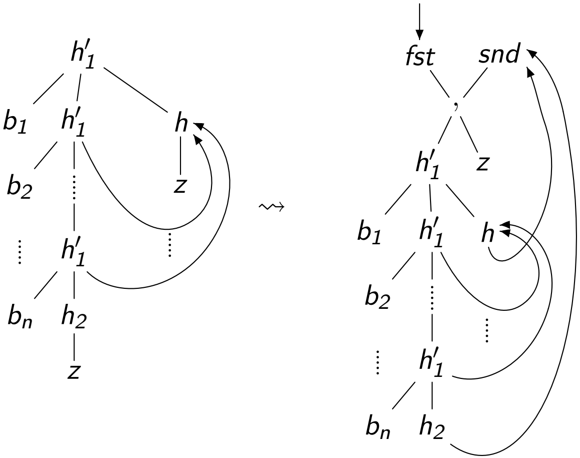
If $h_1 = \lambda b\ a\ z \to h_1'\ b\ a\ (h\ z)$, then:



Tricky Sharing Issues — Circular Shortcut Fusion

$$pfold\ h_1\ h_2\ (buildp\ g\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

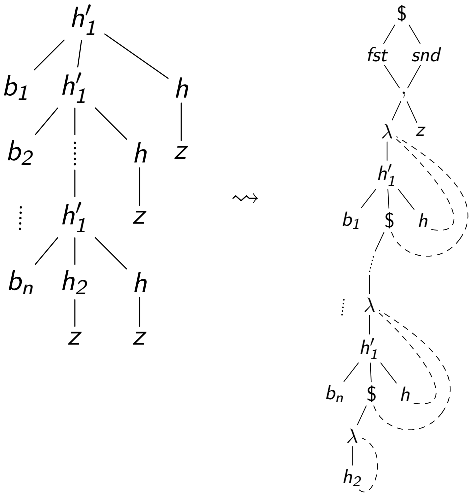
If $h_1 = \lambda b\ a\ z \rightarrow h_1'\ b\ a\ (h\ z)$, then using full laziness:



Tricky Sharing Issues — Higher-Order Shortcut Fusion

$$\begin{aligned} pfold\ h_1\ h_2\ (buildp\ g\ c) \rightsquigarrow\ & \textbf{case}\ g\ (\lambda b\ k\ z \to h_1\ b\ (k\ z)\ z)\ (\lambda z \to h_2\ z)\ c \\ & \textbf{of}\ (k, z) \to k\ z \end{aligned}$$

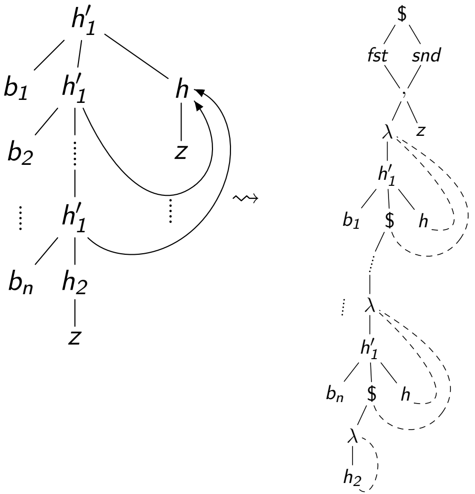
If $h_1 = \lambda b\ a\ z \to h_1'\ b\ a\ (h\ z)$, then:



Tricky Sharing Issues — Higher-Order Shortcut Fusion

$$\mathit{pfold}\ h_1\ h_2\ (\mathit{buildp}\ g\ c) \rightsquigarrow \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c\ \textbf{of}\ (k, z) \rightarrow k\ z$$

If $h_1 = \lambda b\ a\ z \rightarrow h_1'\ b\ a\ (h\ z)$, then using full laziness:



What can be Learnt

- Both semantic and pragmatic considerations can motivate studying new rules as well as new combinators.

What can be Learnt

- Both semantic and pragmatic considerations can motivate studying new rules as well as new combinators.
- These lessons also inform new developments for more classical shortcut fusion techniques.

What can be Learnt

- Both semantic and pragmatic considerations can motivate studying new rules as well as new combinators.
- These lessons also inform new developments for more classical shortcut fusion techniques.
- There is still an interesting design space to explore!

Recent (and Future?) Developments

- [Pardo et al., PEPM'09] study circular and higher-order shortcut fusion in the presence of monads.

Recent (and Future?) Developments

- [Pardo et al., PEPM'09] study circular and higher-order shortcut fusion in the presence of monads.
- From a semantics perspective, the circular flavour is again more intriguing.

Recent (and Future?) Developments

- [Pardo et al., PEPM'09] study circular and higher-order shortcut fusion in the presence of monads.
- From a semantics perspective, the circular flavour is again more intriguing.
- The higher-order flavour is (again) more generally applicable.

Recent (and Future?) Developments

- [Pardo et al., PEPM'09] study circular and higher-order shortcut fusion in the presence of monads.
- From a semantics perspective, the circular flavour is again more intriguing.
- The higher-order flavour is (again) more generally applicable.
- It should be interesting to investigate the interplay with other fusion work involving monads [V., MPC'08].

References I

J.P. Fernandes, A. Pardo, and J. Saraiva.
A shortcut fusion rule for circular program calculation.
In *Haskell Workshop, Proceedings*, pages 95–106. ACM Press, 2007.

A. Gill, J. Launchbury, and S.L. Peyton Jones.
A short cut to deforestation.
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 223–232. ACM Press, 1993.

P. Johann and J. Voigtländer.
Free theorems in the presence of seq.
In *Principles of Programming Languages, Proceedings*, pages 99–110. ACM Press, 2004.

References II

A. Pardo, J.P. Fernandes, and J. Saraiva.
Shortcut fusion rules for the derivation of circular and higher-order monadic programs.
In *Partial Evaluation and Program Manipulation, Proceedings*, pages 81–90. ACM Press, 2009.

S.L. Peyton Jones and D. Lester.
A modular fully-lazy lambda lifter in Haskell.
Software Practice and Experience, 21(5):479–506, 1991.

J. Svenningsson.
Shortcut fusion for accumulating parameters & zip-like functions.
In *International Conference on Functional Programming, Proceedings*, pages 124–132. ACM Press, 2002.

References III

- J. Voigtländer.
  Asymptotic improvement of computations over free monads.
  In *Mathematics of Program Construction, Proceedings*, volume 5133 of *LNCS*, pages 388–403. Springer-Verlag, 2008.
- J. Voigtländer.
  Semantics and pragmatics of new shortcut fusion rules.
  In *Functional and Logic Programming, Proceedings*, volume 4989 of *LNCS*, pages 163–179. Springer-Verlag, 2008.
- P. Wadler.
  Theorems for free!
  In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.