Semantics and Pragmatics of New Shortcut Fusion Rules

Janis Voigtländer

Technische Universität Dresden

FLOPS'08

Classical Shortcut Fusion [Gill et al. 1993]

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. 1993]

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

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

Problem: Expressions like

$$sum\ (upTo\ 10)$$

require explicit construction of intermediate results.

Classical Shortcut Fusion [Gill et al. 1993]

Example:

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

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

Problem: Expressions like

$$\mathit{sum}\ (\mathit{upTo}\ 10)$$

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:

$$\mathit{foldr}\ h_1\ h_2\ (\mathit{build}\ g) \rightsquigarrow g\ h_1\ h_2$$

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

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

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

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. 2007]

Producing intermediate results:

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

Consuming intermediate results:

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

$$
\begin{aligned}
&\mathit{normalise} :: ([\mathit{Int}], \mathit{Int}) \to [\mathit{Float}]\\
&\mathit{normalise} = \mathit{pfold}\ \cdots
\end{aligned}
$$

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

Consuming intermediate results:

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

The fusion rule:

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

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

Consuming intermediate results:

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

The fusion rule:

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

Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

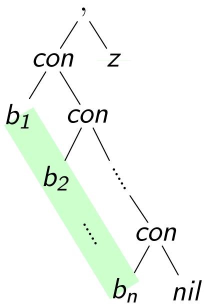
Circular Shortcut Fusion [Fernandes et al. 2007]

Producing intermediate results:

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

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

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

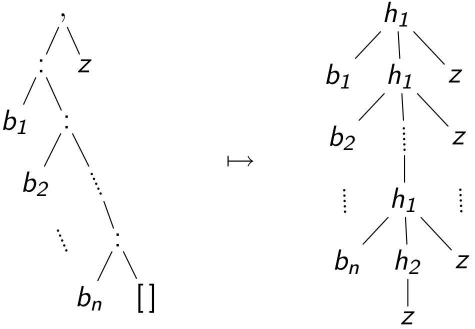


Circular Shortcut Fusion [Fernandes et al. 2007]

Consuming intermediate results:

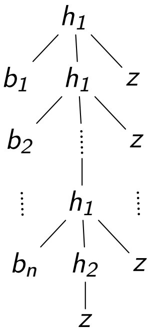
$$pfold :: (b \to a \to z \to a) \to (z \to a) \to ([b], z) \to a$$
$$pfold\ h_1\ h_2\ (bs, z) = foldr\ (\lambda b\ a \to h_1\ b\ a\ z)\ (h_2\ z)\ bs$$

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



Circular Shortcut Fusion [Fernandes et al. 2007]

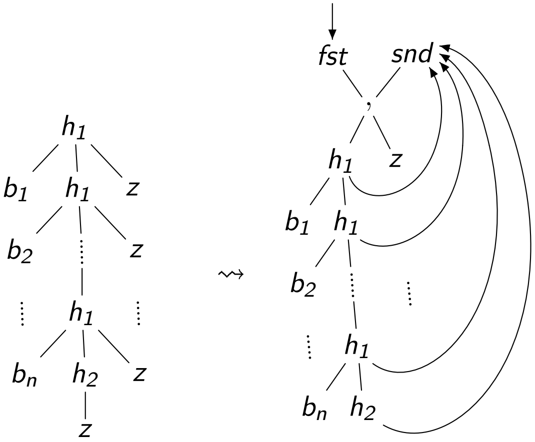
$pfold\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow$



$\rightsquigarrow$

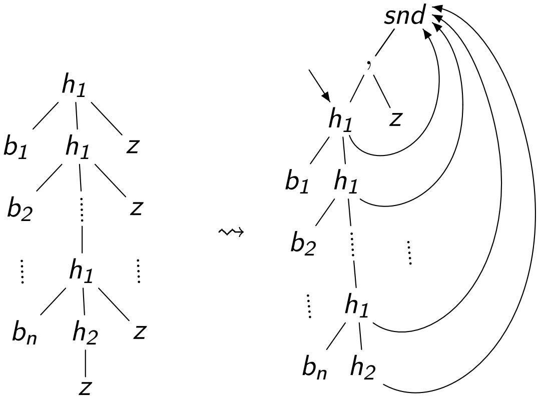
Circular Shortcut Fusion [Fernandes et al. 2007]

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



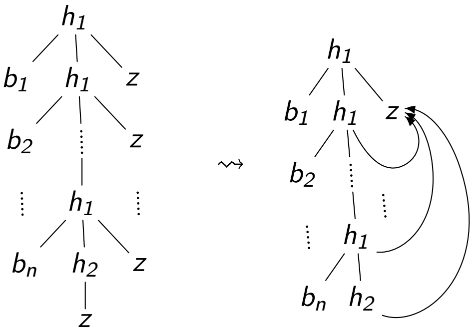
Circular Shortcut Fusion [Fernandes et al. 2007]

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



Circular Shortcut Fusion [Fernandes et al. 2007]

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

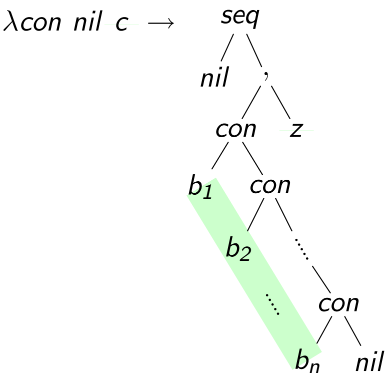


A Problem with Selective Strictness

Producing intermediate results:

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

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

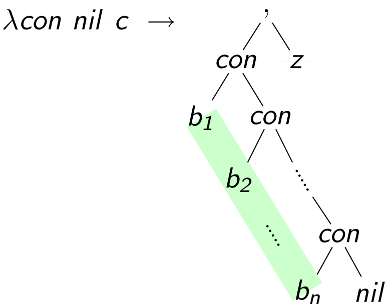


A Problem with Selective Strictness

Producing intermediate results:

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

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



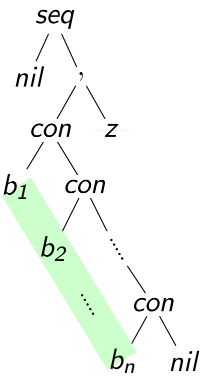
A Problem with Selective Strictness

Producing intermediate results:

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

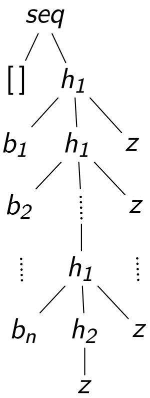
In Haskell, g could also be, for example, of the following form:

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



A Problem with Selective Strictness

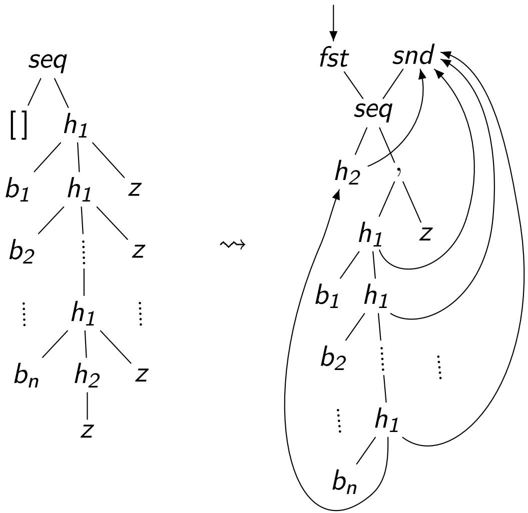
This would lead to the following replacement:



$\rightsquigarrow$

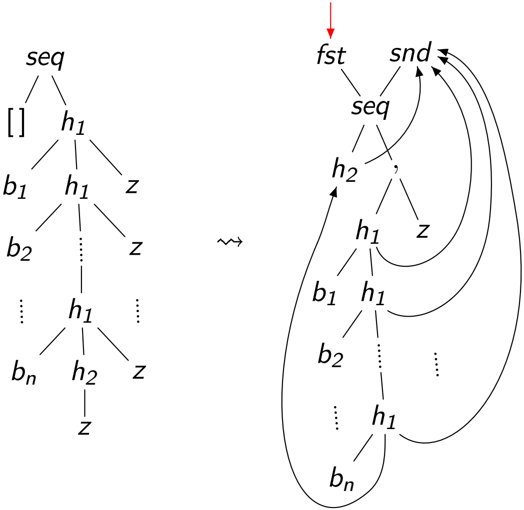
A Problem with Selective Strictness

This would lead to the following replacement:



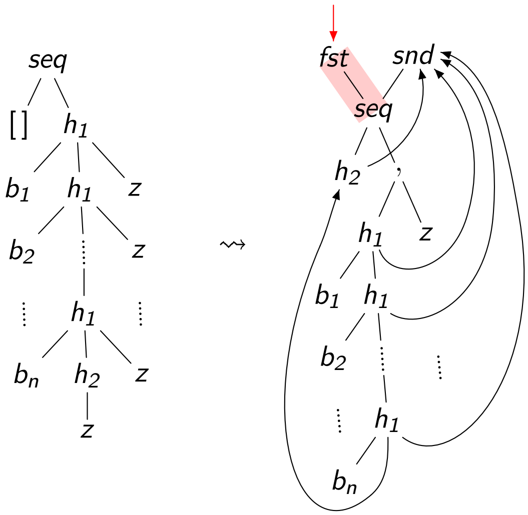
A Problem with Selective Strictness

This would lead to the following replacement:



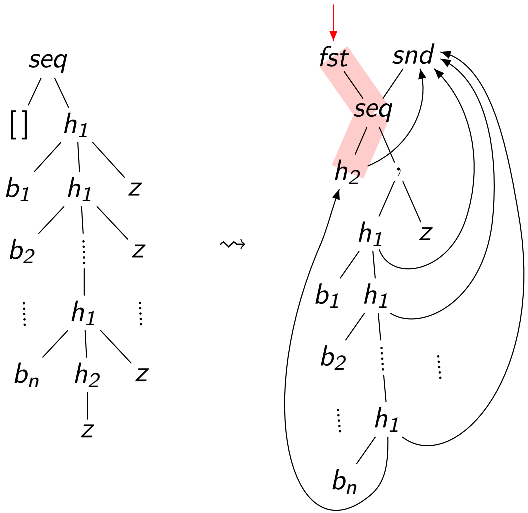
A Problem with Selective Strictness

This would lead to the following replacement:



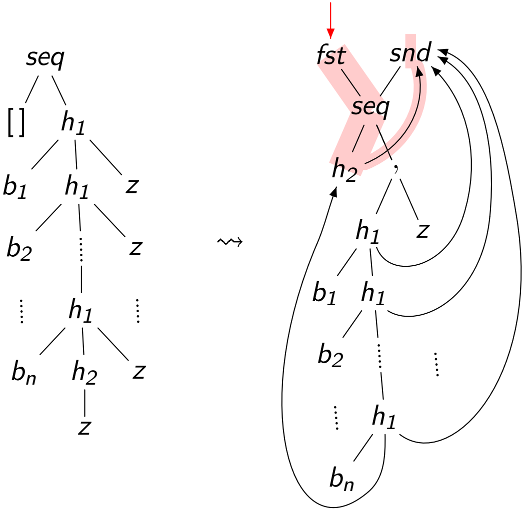
A Problem with Selective Strictness

This would lead to the following replacement:



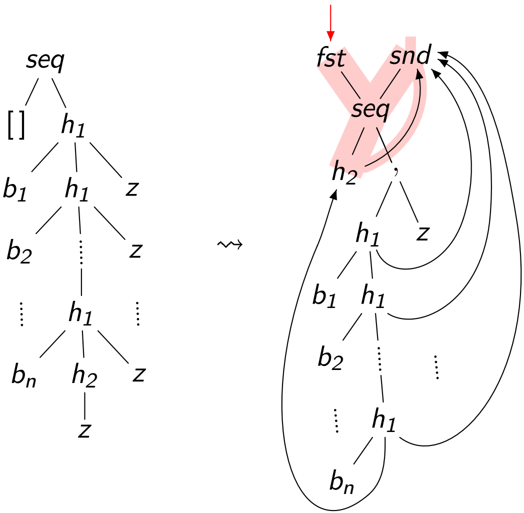
A Problem with Selective Strictness

This would lead to the following replacement:



A Problem with Selective Strictness

This would lead to the following replacement:



Total and Partial Correctness

Theorem 1

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

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

Total and Partial Correctness

Theorem 1

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

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

Theorem 2

Without preconditions,

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

Replacing Circularity by Higher-Orderedness

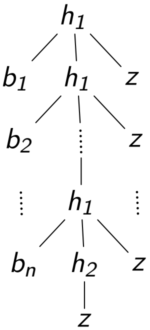
$$\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ []\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

Replacing Circularity by Higher-Orderedness

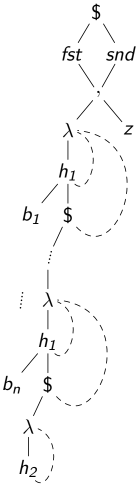
$$
\begin{aligned}
\mathit{pfold}\ h_1\ h_2\ (g\ (:)\ [\,]\ c) \rightsquigarrow\ & \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a \\
& \textbf{case}\ g\ (\lambda b\ k\ z \rightarrow h_1\ b\ (k\ z)\ z)\ (\lambda z \rightarrow h_2\ z)\ c \\
& \textbf{of}\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

Replacing Circularity by Higher-Orderedness

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

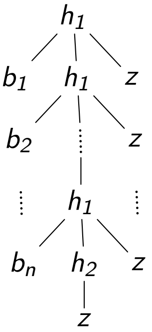


$\rightsquigarrow$

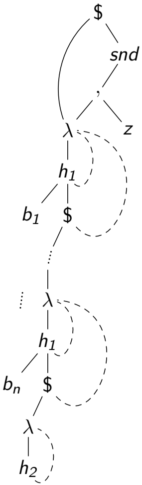


Replacing Circularity by Higher-Orderedness

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

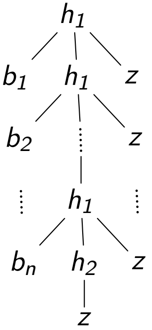


$\rightsquigarrow$

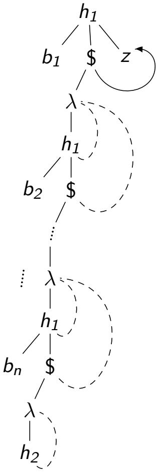


Replacing Circularity by Higher-Orderedness

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

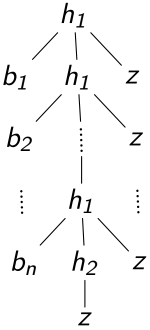


$\rightsquigarrow$

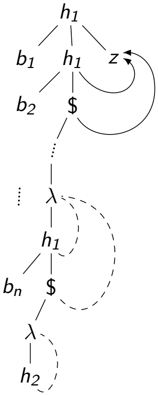


Replacing Circularity by Higher-Orderedness

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

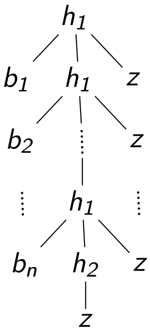


$\rightsquigarrow$

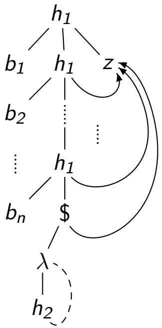


Replacing Circularity by Higher-Orderedness

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

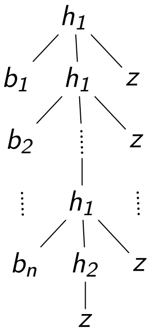


$\rightsquigarrow$

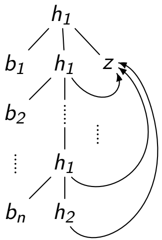


Replacing Circularity by Higher-Orderedness

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

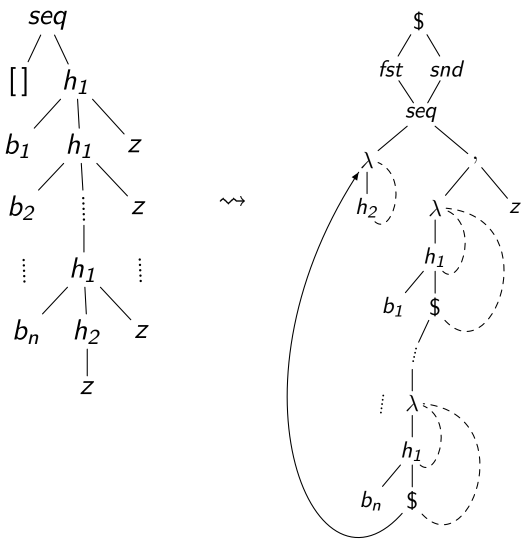


$\rightsquigarrow$



No Problem with Selective Strictness

For a g of the problematic form considered earlier:



Total Correctness

Theorem 3

Without preconditions,

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

Total Correctness

Theorem 3

Without preconditions,

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

Tricky Sharing Issues — Circular Shortcut Fusion

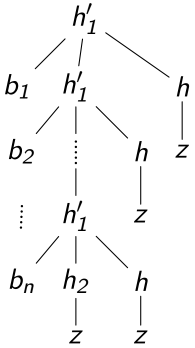
$$pfold\ h_1\ h_2\ (buildp\ g\ c) \rightsquigarrow \textbf{let}\ (a, z) = g\ (\lambda b\ a \rightarrow h_1\ b\ a\ z)\ (h_2\ z)\ c\ \textbf{in}\ a$$

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

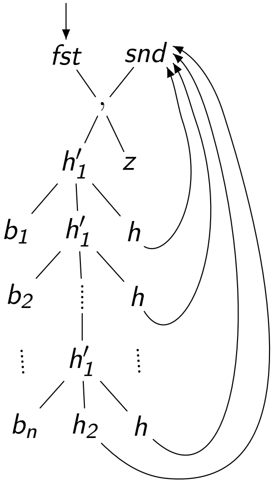
Tricky Sharing Issues — Circular Shortcut Fusion

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

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



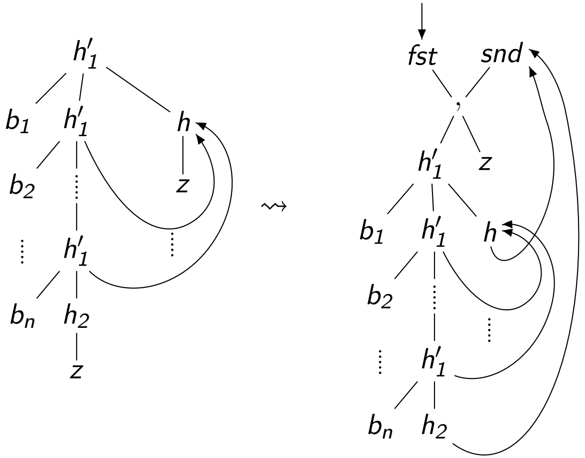
$\rightsquigarrow$



Tricky Sharing Issues — Circular Shortcut Fusion

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

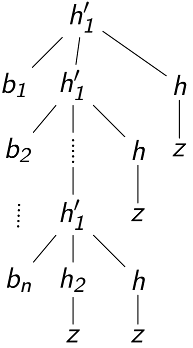
If $h_1 = \lambda b\ a\ z \rightarrow h_1'\ b\ a\ (h\ z)$, then using full laziness:



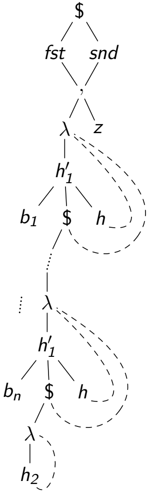
Tricky Sharing Issues — Higher-Order Shortcut Fusion

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

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



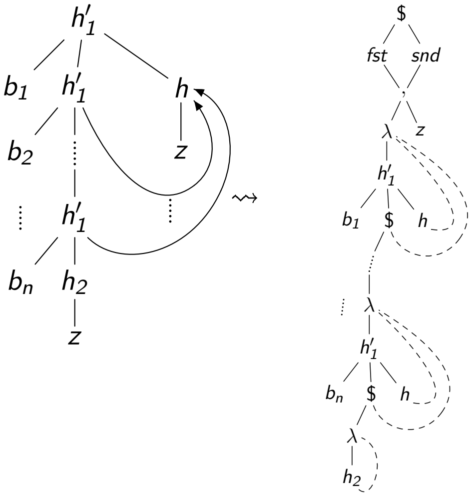
$\rightsquigarrow$



Tricky Sharing Issues — Higher-Order Shortcut Fusion

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

If $h_1 = \lambda b\ a\ z \rightarrow h'_1\ b\ a\ (h\ z)$, then using full laziness:



Also in the Paper

- *pfold*′- and *buildp*′-combinators for better sharing behaviour, with totally correct fusion rules (Theorem 4)

Also in the Paper

- $pfold'$- and $buildp'$-combinators for better sharing behaviour, with totally correct fusion rules (Theorem 4)
- totally correct $foldr/build$-fusion rules (Theorems 5 and 6)

Also in the Paper

- pfold'- and buildp'-combinators for better sharing behaviour, with totally correct fusion rules (Theorem 4)
- totally correct *foldr*/*build*-fusion rules (Theorems 5 and 6)
- a safely approximating (and post-processing-friendly) *destroy*/*unfoldr*-fusion rule (Theorem 7)

Also in the Paper

- *pfold*′- and *buildp*′-combinators for better sharing behaviour, with totally correct fusion rules (Theorem 4)
- totally correct *foldr*/*build*-fusion rules (Theorems 5 and 6)
- a safely approximating (and post-processing-friendly) *destroy*/*unfoldr*-fusion rule (Theorem 7)

...
what about, for example, stream fusion [Coutts et al. 2007] ?

References

D. Coutts, R. Leshchinskiy, and D. Stewart.
Stream fusion: From lists to streams to nothing at all.
International Conference on Functional Programming, 2007.

J.P. Fernandes, A. Pardo, and J. Saraiva.
A shortcut fusion rule for circular program calculation.
Haskell Workshop, 2007.

A. Gill, J. Launchbury, and S.L. Peyton Jones.
A short cut to deforestation.
International Conference on Functional Programming, 1993.

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.
International Conference on Functional Programming, 2002.