Automatic Detection of Recursion Patterns

Final thesis presentation

Jasper Van der Jeugt

June 14, 2013

Overview

Introduction

Folds, builds, and fusion

Automatic detection

Evaluation

Future work

Recursion patterns: what and why?

$$\begin{aligned}
&\mathit{upper} :: \mathit{String} \rightarrow \mathit{String} \\
&\mathit{upper}\ [\,] \qquad\quad = [\,] \\
&\mathit{upper}\ (x : \mathit{xs}) = \\
&\quad \mathit{toUpper}\ x : \mathit{upper}\ \mathit{xs}
\end{aligned}$$

Recursion patterns: what and why?

$$
\begin{aligned}
&map :: (a \to b) \to [a] \to [b] \\
&map\ _\ [\,] \qquad\quad = [\,] \\
&map\ f\ (x : xs) = f\ x : map\ f\ xs
\end{aligned}
$$

Recursion patterns: what and why?

$$upper :: String \to String$$
$$upper = map\ toUpper$$

Recursion patterns: what and why?

$$
\begin{aligned}
&\mathit{evens} :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{evens}\ [\,] \qquad\quad\;\; = [\,] \\
&\mathit{evens}\ (x : \mathit{xs}) \\
&\quad \mid \mathit{even}\ x \qquad = x : \mathit{evens}\ \mathit{xs} \\
&\quad \mid \mathit{otherwise} = \mathit{evens}\ \mathit{xs}
\end{aligned}
$$

Recursion patterns: what and why?

$$
\begin{aligned}
& \mathit{filter} \\
& \quad :: (a \rightarrow \mathit{Bool}) \rightarrow [a] \rightarrow [a] \\
& \mathit{filter}\ _\ [\,] \qquad = [\,] \\
& \mathit{filter}\ p\ (x : xs) \\
& \quad \mid p\ x \qquad\quad = \\
& \quad\quad x : \mathit{filter}\ p\ xs \\
& \quad \mid \mathit{otherwise} = \mathit{filter}\ p\ xs
\end{aligned}
$$

Recursion patterns: what and why?

$$
\begin{aligned}
&evens :: [Int] \rightarrow [Int] \\
&evens = filter\ even
\end{aligned}
$$

Recursion patterns: what and why?

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

Recursion patterns: what and why?

$$
\begin{array}{l}
\mathit{foldr} \\
\quad :: \;\; (a \to b \to b) \\
\quad \to b \to [a] \to b \\
\mathit{foldr}\ _\ z\ [\,] \qquad\quad = z \\
\mathit{foldr}\ f\ z\ (x : xs) = \\
\quad f\ x\ (\mathit{foldr}\ f\ z\ xs)
\end{array}
$$

Recursion patterns: what and why?

$$sum :: [\mathit{Int}] \rightarrow \mathit{Int}$$
$$sum = foldr\ (+)\ 0$$

Recursion patterns: what and why?

Advantages

Easier to understand

Conciseness $\Rightarrow$ less bugs

Prove properties only once

Optimizations

"Why Functional Programming Matters", John Hughes, 1984

Recursion patterns: what and why?

Disadvantages

Not everyone writes in this style

Folds for other datatypes?

Optimizations?

Overview

Introduction

Folds, builds, and fusion

Automatic detection

Evaluation

Future work

Folds, builds, and fusion

$$map :: (a \to b) \to [a] \to [b]$$
$$map\ f = foldr\ ((:) \circ f)\ [\,]$$

Folds, builds, and fusion

$$
\begin{aligned}
&\mathit{filter} \\
&\quad :: (a \to \mathit{Bool}) \to [a] \to [a] \\
&\mathit{filter}\ p = \mathit{foldr} \\
&\quad (\lambda x\ l \to \\
&\qquad \textbf{if}\ p\ x\ \textbf{then}\ x : l\ \textbf{else}\ l) \\
&\quad []
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{aligned}
g\ [\,] &= v \\
g\ (x : xs) &= f\ x\ (g\ xs)
\end{aligned}
$$

$$\Leftrightarrow \quad g = \mathit{foldr}\ f\ v$$

"A tutorial on the universality and expressiveness of fold", Graham Hutton, 1999

Folds, builds, and fusion

$$\begin{aligned}
&\mathit{foldr}\ (+)\ 0 \\
&\quad (1\ :\ (2\ :\ (3\ :\ (4\ :\ [\,]))))) \\
&\equiv (1 + (2 + (3 + (4 + 0))))
\end{aligned}$$

Folds, builds, and fusion

$$
\begin{aligned}
&\textbf{data}\ \mathit{Tree}\ a \\
&\quad = \mathit{Leaf}\ a \\
&\quad \mid\ \mathit{Branch}\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Leaf} \\
&\quad :: a \to \mathit{Tree}\ a \\
&\mathit{Branch} \\
&\quad :: \mathit{Tree}\ a \to \mathit{Tree}\ a \to \mathit{Tree}\ a
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{aligned}
&\mathit{Leaf}\\
&\quad :: a \to \mathit{Tree}\ a\\
&\mathit{Branch}\\
&\quad :: \mathit{Tree}\ a \to \mathit{Tree}\ a \to \mathit{Tree}\ a
\end{aligned}
$$

$$
\begin{aligned}
\mathit{leaf} &:: a \to r\\
\mathit{branch} &:: r \to r \to r
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{aligned}
&\mathit{foldTree} \\
&\quad :: \ (a \to r) \qquad\qquad \text{-- leaf} \\
&\quad \to (r \to r \to r) \quad \text{-- branch} \\
&\quad \to \ \mathit{Tree}\ a \to r \\
&\mathit{foldTree}\ l\ b\ (\mathit{Leaf}\ \ x) \qquad\quad = l\ x \\
&\mathit{foldTree}\ l\ b\ (\mathit{Branch}\ x\ y) = \\
&\quad b\ (\mathit{foldTree}\ l\ b\ x) \\
&\quad\quad (\mathit{foldTree}\ l\ b\ y)
\end{aligned}
$$

Folds, builds, and fusion

Template Haskell: compile-time type-safe metaprogramming

```
$(deriveFold
    '' Tree "foldTree")
```

Folds, builds, and fusion

Folds capture a *consumption* pattern of a list

Builds capture a *production* pattern of a list

Folds, builds, and fusion

$$\begin{aligned} &\mathit{build} \\ &\quad :: \ (\forall b. \\ &\qquad\quad (a \to b \to b) \to b \to b) \\ &\quad \to [a] \\ &\mathit{build}\ g = g\ (:)\ [\,] \end{aligned}$$

Folds, builds, and fusion

$$
\begin{array}{l}
\mathit{replicate} :: \mathit{Int} \to a \to [a] \\
\mathit{replicate}\ n\ x = \\
\quad \mathit{build}\ \$\ \lambda \mathit{cons}\ \mathit{nil} \to \\
\quad\quad \mathbf{let}\ g\ n \mid n \leqslant 0 \quad\quad = \mathit{nil} \\
\quad\quad\quad\quad\quad\ \mid \mathit{otherwise} = \\
\quad\quad\quad\quad\quad\quad \mathit{cons}\ x\ (g\ (n - 1)) \\
\quad\quad \mathbf{in}\ g\ n
\end{array}
$$

Folds, builds, and fusion

Foldr/build-fusion:

$$
\begin{aligned}
&\mathit{foldr}\ \mathit{cons}\ \mathit{nil}\ (\mathit{build}\ g) \\
&\equiv g\ \mathit{cons}\ \mathit{nil}
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{aligned}
& \mathit{sum}\ (\mathit{replicate}\ n\ x) \\
\equiv\quad & \{\ \text{Inline } \mathit{sum}\ \} \\
& \mathit{foldr}\ (+)\ 0\ (\mathit{replicate}\ n\ x)
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{array}{l}
\mathit{foldr}\ (+)\ 0\ (\mathit{replicate}\ n\ x) \\
\equiv \quad \{\ \text{Inline } \mathit{replicate}\ \} \\
\mathit{foldr}\ (+)\ 0\ (\\
\quad \mathit{build}\ (\lambda \mathit{cons}\ \mathit{nil} \rightarrow \\
\qquad \textbf{let}\ g\ n \mid n \leqslant 0 \qquad = \mathit{nil} \\
\qquad\qquad\quad \mid \mathit{otherwise} = \\
\qquad\qquad\qquad \mathit{cons}\ x\ (g\ (n-1)) \\
\qquad \textbf{in}\ g\ n))
\end{array}
$$

Folds, builds, and fusion

$$\equiv \quad \{ \text{ Foldr/build-fusion } \}$$

$$
\begin{array}{l}
(\lambda \mathit{cons}\ \mathit{nil} \rightarrow \\
\quad \textbf{let}\ g\ n \mid n \leqslant 0 \qquad = \mathit{nil} \\
\qquad\qquad\quad \mid \mathit{otherwise} = \\
\qquad\qquad\qquad \mathit{cons}\ x\ (g\ (n - 1)) \\
\quad \textbf{in}\ g\ n)\ (+)\ 0
\end{array}
$$

Folds, builds, and fusion

$$\equiv \quad \{\ \beta\text{-reduction}\ \}$$

$$\begin{array}{l} \textbf{let}\ g\ n \mid n \leqslant 0 \quad\quad = 0 \\ \qquad\quad\ \mid \mathit{otherwise} = \\ \qquad\qquad x + (g\ (n - 1)) \\ \textbf{in}\ g\ n \end{array}$$

Folds, builds, and fusion

$$
\begin{array}{l}
\mathit{buildTree} \\
\quad :: \ (\forall b. \\
\qquad\quad (a \to b) \to \\
\qquad\quad (b \to b \to b) \to \\
\qquad\quad b) \\
\quad \to \mathit{Tree}\ a \\
\mathit{buildTree}\ g = g\ \mathit{Leaf}\ \mathit{Branch}
\end{array}
$$

Folds, builds, and fusion

$ $(\mathit{deriveBuild}$
$\quad$ ‘‘ Tree `"buildTree"`)

Folds, builds, and fusion

Foldr/build-fusion for *Tree*:

$$
\begin{aligned}
&\mathit{foldTree}\ \mathit{leaf}\ \ \mathit{branch}\ (\mathit{buildTree}\ g) \\
&\equiv g\ \mathit{leaf}\ \ \mathit{branch}
\end{aligned}
$$

Folds, builds, and fusion

$$
\begin{aligned}
&map :: (a \to b) \to [a] \to [b] \\
&map\ f\ ls = build\ \$\ \lambda cons\ nil \to \\
&\quad foldr \\
&\qquad (\lambda x\ xs \to cons\ (f\ x)\ xs) \\
&\qquad nil\ ls
\end{aligned}
$$

Pipelines:

$$
\ldots \circ map\ f \circ map\ g \circ filter\ h \circ \ldots
$$

Overview

Introduction

Folds, builds, and fusion

Automatic detection

Evaluation

Future work

Automatic detection

Nondeterministic rules for fold:

F-Bind, F-Rec, F-Abs, F-App, F-Case

Automatic detection

F-Bind':

$$\frac{e_1' = [y \mapsto [\,]]e_1 \qquad f \notin fv(e_1) \qquad ws\ fresh \qquad e_2 \ {}_{vs}\overset{f\ \triangle}{\leadsto}{}_{ws}\ e_2' \qquad \{f, y, vs\} \cap fv(e_2') = \emptyset}{f = \lambda y \rightarrow \textbf{case}\ y\ \textbf{of}\ \{[\,] \rightarrow e_1; (v : vs) \rightarrow e_2\} \leadsto f = \lambda y \rightarrow foldr\ (\lambda v\ ws \rightarrow e_2')\ e_1'\ y}$$

Automatic detection

F-REC':

$$\frac{}{f\ x\ {}_{x}\overset{f\ \triangle}{\rightsquigarrow}{}_{y}\quad y}$$

Automatic detection

F-Abs:

$$\frac{e \;{}_{x}\overset{E}{\leadsto}_{y}\; e'}{\lambda z \rightarrow e \;{}_{x}\overset{E}{\leadsto}_{y}\; \lambda z \rightarrow e'}$$

Automatic detection

F-App:

$$\frac{e_1 \; {}_x\overset{E}{\rightsquigarrow}_y \; e_1' \qquad e_2 \; {}_x\overset{E}{\rightsquigarrow}_y \; e_2'}{e_1 \; e_2 \; {}_x\overset{E}{\rightsquigarrow}_y \; e_1' \; e_2'}$$

Automatic detection

F-CASE:

$$\frac{e \;{}_{x}\overset{E}{\leadsto}_{y}\; e' \qquad e_i \;{}_{x}\overset{E}{\leadsto}_{y}\; e_i' \quad (\forall i)}{\textbf{case}\ e\ \textbf{of}\ \overline{p \rightarrow e} \;{}_{x}\overset{E}{\leadsto}_{y}\; \textbf{case}\ e'\ \textbf{of}\ \overline{p \rightarrow e'}}$$

Automatic detection

$$
\begin{aligned}
&\mathit{sum} = \lambda y \to \textbf{case } y \textbf{ of} \\
&\quad [\,] \qquad\quad \to 0 \\
&\quad (v : \mathit{vs}) \to (+)\ v\ (\mathit{sum}\ \mathit{vs})
\end{aligned}
$$

$\rightsquigarrow$

$$
\begin{aligned}
&\mathit{sum} = \lambda y \to \\
&\quad \mathit{foldr}\ (\lambda v\ \mathit{ws} \to (+)\ v\ \mathit{ws})\ 0\ y
\end{aligned}
$$

Automatic detection

$$
\begin{aligned}
&\mathit{foldl} = \lambda f\ z\ y \rightarrow \textbf{case}\ y\ \textbf{of} \\
&\quad [\,] \qquad\quad\ \rightarrow z \\
&\quad (v : vs) \rightarrow \mathit{foldl}\ f\ (f\ z\ v)\ vs
\end{aligned}
$$

$\rightsquigarrow$

$$
\begin{aligned}
&\mathit{foldl} = \lambda f\ z\ y \rightarrow \mathit{foldr} \\
&\quad (\lambda v\ vs\ z \rightarrow vs\ (f\ z\ v)) \\
&\quad (\lambda z \rightarrow z)\ y\ z
\end{aligned}
$$

Automatic detection

Nondeterministic rules for build:

B-Bind, B-Rec, B-Nil,
B-Cons, B-Build, B-Case

Automatic detection

Implementation: GHC core

$$
\begin{array}{ll}
\textbf{data } \mathit{Expr}\ b & \\
\quad = & \mathit{Var}\ \mathit{Id} \\
\quad \mid & \mathit{Lit}\ \mathit{Literal} \\
\quad \mid & \mathit{App}\ (\mathit{Expr}\ b)\ (\mathit{Arg}\ b) \\
\quad \mid & \mathit{Lam}\ b\ (\mathit{Expr}\ b) \\
\quad \ldots &
\end{array}
$$

Automatic detection

Advantages of GHC core

Simpler expression type
Multiple passes
Type information

Automatic detection

GHC plugin framework

$$\textbf{module}\ \mathit{WhatMorphism}\ \textbf{where}$$

$$\begin{aligned}
&\mathit{whatMorphism} \\
&\quad ::\ \mathit{CoreProgram} \\
&\quad \to \mathit{CoreM}\ \mathit{CoreProgram} \\
&\mathit{whatMorphism} = \ldots
\end{aligned}$$

Automatic detection

GHC plugin framework

```
ghc
  -package what-morphism
  -fplugin WhatMorphism
   Main.hs
```

Automatic detection

WhatMorphism:

WhatMorphism.Fold
WhatMorphism.Build
WhatMorphism.Inliner
WhatMorphism.Fusion

Overview

Introduction
Folds, builds, and fusion
Automatic detection
Evaluation
Future work

Evaluation

Folds:

Package	**Total**	List	ADT	V. arg.	N. rec.	**HLint**
Cabal-1.16.0.3	20	11	9	6	0	9
containers-0.5.2.1	100	11	89	41	11	1
cpphs-1.16	5	2	3	3	0	1
darcs-2.8.4	66	65	8	1	0	6
ghc-7.6.3	327	216	111	127	9	26
hakyll-4.2.2.0	5	1	4	3	0	0
haskell-src-exts-1.13.5	37	11	26	15	0	2
hlint-1.8.44	6	3	3	1	0	0
hscolour-1.20.3	4	4	0	0	0	2
HTTP-4000.2.8	6	6	0	2	0	3
pandoc-1.11.1	15	15	0	1	0	2
parsec-3.1.3	3	3	0	1	0	0
snap-core-0.9.3.1	4	3	1	1	0	0

Evaluation

Folds:

Better results than HLint
Both ADTs and lists
Superset

Evaluation

Builds:

Package	**Total**	List	ADT	Rec.
Cabal-1.16.0.3	101	81	20	5
containers-0.5.2.1	25	2	23	12
cpphs-1.16	6	5	1	3
darcs-2.8.4	354	354	0	26
ghc-7.6.3	480	178	302	53
hakyll-4.2.2.0	22	18	4	2
haskell-src-exts-1.13.5	140	74	66	16
hlint-1.8.44	69	62	7	1
hscolour-1.20.3	33	33	0	2
HTTP-4000.2.8	11	11	0	5
pandoc-1.11.1	97	97	0	16
parsec-3.1.3	10	10	0	0
snap-core-0.9.3.1	4	4	0	0

Evaluation

Foldr/build-fusion:

Manual intervention needed
Manually optimised code
Integration with existing libraries

Evaluation

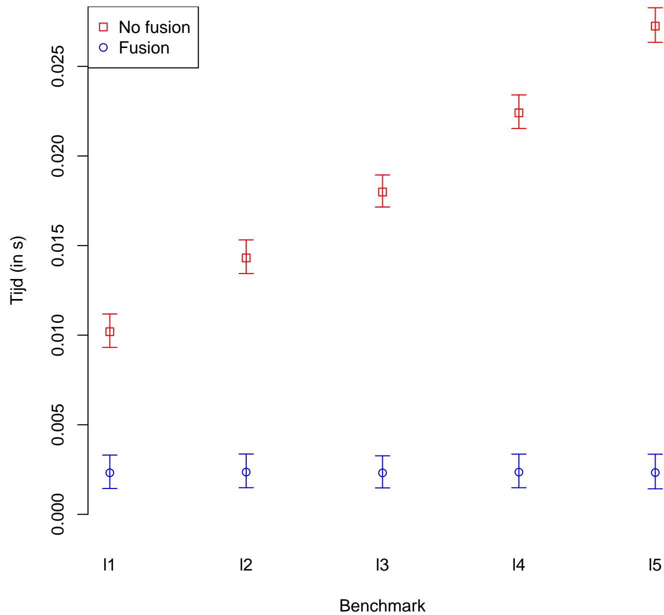
$$\mathit{l1}, \mathit{l2}, \mathit{l3}, \mathit{l4}, \mathit{l5} :: \mathit{Int} \to \mathit{Int}$$
$$\mathit{l1}\ n = \mathit{suml}\ (1\ `\mathit{uptol}`\ n)$$
$$\mathit{l2}\ n = \mathit{suml}\ (\mathit{mapl}\ (+1)$$
$$\quad (1\ `\mathit{uptol}`\ n))$$
$$\mathit{l3}\ n = \mathit{suml}\ (\mathit{mapl}\ (+1)$$
$$\quad (\mathit{mapl}\ (+1)$$
$$\qquad (1\ `\mathit{uptol}`\ n)))$$

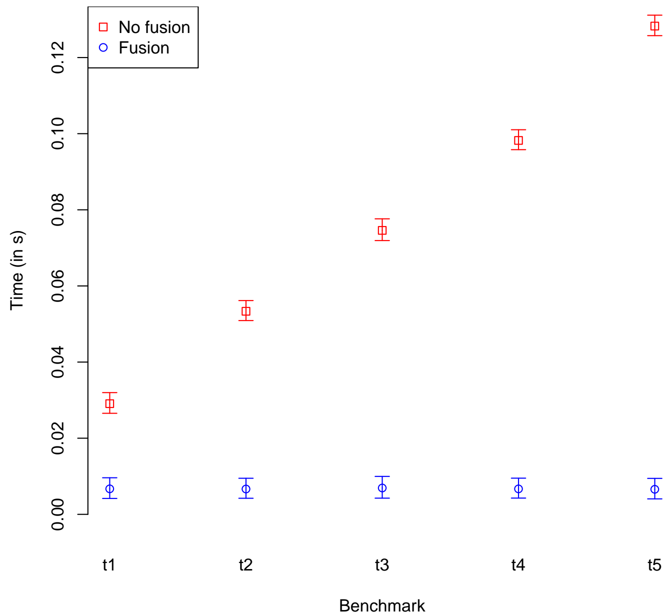
...

Evaluation

$$\mathit{t1}, \mathit{t2}, \mathit{t3}, \mathit{t4}, \mathit{t5} :: \mathit{Int} \to \mathit{Int}$$
$$\mathit{t1}\ n = \mathit{sumt}\ (1\ \text{`}\mathit{uptot}\text{`}\ n)$$
$$\mathit{t2}\ n = \mathit{sumt}\ (\mathit{mapt}\ (+1)$$
$$\quad (1\ \text{`}\mathit{uptot}\text{`}\ n))$$
$$\mathit{t3}\ n = \mathit{sumt}\ (\mathit{mapt}\ (+1)$$
$$\quad (\mathit{mapt}\ (+1)$$
$$\quad\quad (1\ \text{`}\mathit{uptot}\text{`}\ n)))$$

...


No fusion
Fusion
Tijd (in s)
0.000
0.005
0.010
0.015
0.020
0.025
I1
I2
I3
I4
I5
Benchmark



No fusion
Fusion
Time (in s)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
t1
t2
t3
t4
t5
Benchmark

Overview

Introduction
Folds, builds, and fusion
Automatic detection
Evaluation
Future work

Future work

GADTs
Mutually recursive datatypes & folds
foldr/foldr-fusion
Stream fusion

Questions?