Higher-order matching for program transformation refactoring

Ganesh Sittampalam
MAG

- Annotate source code with hints for complex optimisations
- Maintain unoptimised, easy-to-read code
- Compiler automatically applies optimisation
  - Displays calculation – or details of failure
Refactoring

• Apply the same transformations
• Now at *edit* time not *compile* time
• Can work with optimised code
• Want the inverse transformation too
Cat-elimination

\[
\begin{align*}
\text{reverse } [] &= [] \\
\text{reverse } (x:xs) &= \text{reverse } xs ++ [x]
\end{align*}
\]

\[\Rightarrow\]

\[
\begin{align*}
\text{reverse } xs &= \text{reverse'} \ x s \ [x] \\
\text{reverse'} \ [] \ ys &= ys \\
\text{reverse'} \ (x:xs) \ ys &= \text{reverse'} \ x s \ (x:ys)
\end{align*}
\]
Cat-elimination

Specification:

\[
\text{reverse } xs = \text{reverse' } xs \; []
\]

\[
\text{reverse' } xs \; ys = \text{reverse } xs \; ++ \; ys
\]

Laws:

\[
(xs \; ++ \; ys) \; ++ \; zs = xs \; ++ \; (ys \; ++ \; zs)
\]

+ some definitions
Canned recursion on lists

`foldr` is the natural fold on lists

\[
\text{foldr } f \ e \ [] = e \\
\text{foldr } f \ e \ (x:xs) = f \ x \ (\text{foldr } f \ e \ xs)
\]

\[
\text{reverse } xs = \text{foldr } (\lambda t \ ts \rightarrow ts ++ [t]) \ [] \ xs
\]
List fusion

Suppose $f (a \oplus b) = a \otimes f b \ \forall \ a, b$

Then:

$$f (a_1 \oplus (a_2 \oplus (a_3 \oplus ... (a_n \oplus e))))$$

$$= a_1 \otimes f (a_2 \oplus (a_3 \oplus ... (a_n \oplus e)))$$

$$= a_1 \otimes (a_2 \otimes f (a_3 \oplus ... (a_n \oplus e)))$$

$$= ...$$

$$= a_1 \otimes (a_2 \otimes (a_3 \otimes ... (a_n \otimes f e)))$$
Fusion rule

\[ f (\text{foldr} (\oplus) \ e \ xs) = \text{foldr} (\otimes) \ e' \ xs \]

if

\[ f \ \text{strict} \]

\[ f \ e = e' \]

\[ \lambda \ a \ b \rightarrow f (a \oplus b) = \lambda \ a \ b \rightarrow a \otimes f b \]
Applying fusion

\[ f (\text{foldr} (\oplus) e xs) = \text{foldr} (\otimes) e' xs \]

If \( f \) strict, \( f e = e' \)

\[ \lambda a b \to f (a \oplus b) = \lambda a b \to a \otimes f b \]

\[ \text{reverse'} xs ys = \text{reverse} xs ++ ys \]

\[ = \text{foldr} (\lambda t ts \to ts ++ [t]) [ ] ++ ys \]

- Pick subexpression
- Try to apply fusion
Applying fusion

\[
f \left( \text{foldr} \left( \oplus \right) e \; xs \right) = \text{foldr} \left( \otimes \right) e' \; xs
\]

If \( f \) strict, \( f \; e = e' \)

\[\lambda \; a \; b \rightarrow f \left( a \oplus b \right) = \lambda \; a \; b \rightarrow a \otimes f \; b\]

\[
\text{reverse'} \; xs \; ys = \text{reverse} \; xs \; ++ \; ys
\]

\[= \text{foldr} \left( \lambda \; ts \rightarrow ts \; ++ \; \left[ t \right] \right) \left[ \right] \; ++ \; ys\]

• Pick subexpression

• Try to apply fusion
Applying fusion

\[ f (\text{foldr} (\oplus) e \, xs) = \text{foldr} (\otimes) e' \, xs \]

**If** \( f \) strict, \( f \, e = e' \)

\[ \lambda \, a \, b \rightarrow f (a \oplus b) = \lambda \, a \, b \rightarrow a \otimes f \, b \]

\[ \text{foldr} (\lambda \, t \, ts \rightarrow ts \, ++ \, [t]) \, [\, ] \, ++ \]

\[
  
  f \quad := \quad (++)
  
  (\oplus) \quad := \quad \lambda \, t \, ts \rightarrow ts \, ++ \, [t]
  
  e \quad := \quad []
  
\]
Applying fusion

\[ f (\text{foldr} \ (\oplus) \ e \ xs) = \text{foldr} \ (\otimes) \ e' \ xs \]

If \( f \) strict, \( f e = e' \)

\[ \lambda \ a \ b \rightarrow f (a \oplus b) = \lambda \ a \ b \rightarrow a \otimes f \ b \]

\[ \text{foldr} \ (\lambda t \ ts \rightarrow ts ++ [t]) \ [ ] ++ \]

\[ f := (++) \]

\( (\oplus) := \lambda t \ ts \rightarrow ts ++ [t] \)

\[ e := [] \]

- Substitute into side conditions
Applying fusion

\((++)([]) = e'\)

\(\lambda a b \rightarrow (++) (b ++ [a])\)

\(= \lambda a b \rightarrow a \otimes ((++) b)\)

- Rewrite exhaustively
- \(\eta\)-expand where needed
Applying fusion

\[ \lambda \; ts \rightarrow ts = e' \]

\[ \lambda \; a \; b \rightarrow (++) \; (b \; ++ \; [a]) \]

\[ = \lambda \; a \; b \rightarrow a \otimes (++) \; b \]

- Rewrite exhaustively
- \(\eta\)-expand where needed
Applying fusion

\[ \lambda \; ts \rightarrow ts = e' \]

\[ \lambda \; a \; b \; ts \rightarrow (b ++ [a]) \; ++ \; ts \]
\[ = \lambda \; a \; b \rightarrow a \otimes ((++) \; b) \]

- Rewrite exhaustively
- \( \eta \)-expand where needed
Applying fusion

\[ \lambda \; ts \rightarrow ts = e' \]

\[ \lambda \; a \; b \; ts \rightarrow b \; ++ \; (a:ts) \]

\[ = \lambda \; a \; b \rightarrow a \; \otimes \; ((++) \; b) \]
Applying fusion

\[
\lambda \, ts \rightarrow ts = e'
\]

\[
\lambda \, a \, b \, ts \rightarrow b \, ++ \, (a:ts)
= \lambda \, a \, b \rightarrow a \, \otimes \, ((++ \, b)
\]

\[
e' \, := \, \lambda \, ts \rightarrow ts
\]
\[
(\otimes) \, := \, \lambda \, t \, f \, ts \rightarrow f \, (t:ts)
\]
Higher-order matching

• Various algorithms
• All solve for \( \phi \) in the equation
  \[ \phi P = T \]
  \( \phi \) a substitution, \( P \) and \( T \) \( \lambda \)-terms
  \( P \) contains free variables, \( T \) closed

• Vary in
  – Restrictions on \( P \)
  – Which solutions are returned
  – More solutions \( \Rightarrow \) More restrictions
Fast reverse

\[
\begin{align*}
\text{reverse } xs &= \text{foldr } (\lambda \ t \ ts \rightarrow ts ++ [t]) \ [] \ xs \\
\rightarrow \\
\text{reverse } xs &= \text{reverse}' \ xs \ [] \\
\text{reverse}' \ xs \ ys &= \\
&= \text{foldr } (\lambda \ t \ f \ ts \rightarrow f \ (t:ts)) \ (\lambda \ ts \rightarrow ts) \ xs \ ys
\end{align*}
\]
Fast reverse

reverse \( xs = foldr (\lambda \ t \ ts \rightarrow ts ++ [t]) [] xs \)

\(\Leftrightarrow\)

reverse \( xs = reverse' \ xs [] \)

reverse' \( xs \ ys =
foldr (\lambda \ t \ f \ ts \rightarrow f (t:ts)) (\lambda \ ts \rightarrow ts) \ xs \ ys \)
Warm fusion

\[ xs = \text{foldr} \ (\cdot) \ [] \ xs \]

\[ \text{reverse} \ xs = \text{reverse} \ (\text{foldr} \ (\cdot) \ [] \ xs) \]

- Can introduce folds by fusion
- Fusion transformations merge into one
Other examples

• Tree traversals
  – Flattening a tree
  – Alpha-beta pruning
• Tupling
  – Fibonacci etc etc
• Some kinds of deforestation
• Fix-point fusion
Conclusions etc

• Complex rewrite rules
  ➞ good specifications for refactoring
  – Good for recursive programs
  – Need HOM to solve
• More integration between browser + compiler?
• More ideas of applications?
• Can we always invert things?