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

$reverse$ [] = []

$reverse$ (x:xs) = $reverse\ xs$ ++ [x]

➔

$reverse\ xs$ = $reverse'\ xs$ []

$reverse'$ [] ys= ys

$reverse'$ (x:xs) ys= $reverse'\ xs$ (x:ys)

Cat-elimination

Specification:

reverse xs = *reverse' xs* []

reverse' xs ys = *reverse xs* ++ *ys*

Laws:

(*xs* ++ *ys*) ++ *zs* = *xs* ++ (*ys* ++ *zs*)

+ some definitions

Canned recursion on lists

foldr is the natural fold on lists

$foldr\ f\ e\ [\,] = e$

$foldr\ f\ e\ (x{:}xs) = f\ x\ (foldr\ f\ e\ xs)$

$reverse\ xs = foldr\ (\lambda t\ ts \rightarrow ts \mathbin{+\!+} [t])\ [\,]\ xs$

List fusion

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

Then:

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

Fusion rule

$f\ (foldr\ (\oplus)\ e\ xs) = foldr\ (\otimes)\ e'\ xs$

if

f strict

$f\ e = e'$

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

Applying fusion

$f\ (foldr\ (\oplus)\ e\ xs) = foldr\ (\otimes)\ e'\ xs$

If f strict, $f\ e = e'$

$\lambda\ a\ b \longrightarrow f\ (a \oplus b) = \lambda\ a\ b \longrightarrow a \otimes f\ b$

$reverse'\ xs\ ys = reverse\ xs\ {+}{+}\ ys$

$= foldr\ (\lambda t\ ts \longrightarrow ts\ {+}{+}\ [t])\ [\ ]\ {+}{+}\ ys$

- Pick subexpression
- Try to apply fusion

Applying fusion

$f\ (foldr\ (\oplus)\ e\ xs) = foldr\ (\otimes)\ e'\ xs$

If f strict, $f\ e = e'$

$\lambda\ a\ b \rightarrow f\ (a \oplus b) = \lambda\ a\ b \rightarrow a \otimes f\ b$

$reverse'\ xs\ ys = reverse\ xs \mathbin{++} ys$

$= foldr\ (\lambda t\ ts \rightarrow ts \mathbin{++} [t])\ [\ ] \mathbin{++} ys$

- Pick subexpression
- Try to apply fusion

Applying fusion

$f\ (foldr\ (\oplus)\ e\ xs) = foldr\ (\otimes)\ e'\ xs$

If f strict, $f\ e = e'$

$\lambda\ a\ b \longrightarrow f\ (a \oplus b) = \lambda\ a\ b \longrightarrow a \otimes f\ b$

$foldr\ (\lambda t\ ts \longrightarrow ts \mathbin{+\!+} [t])\ [\,]\ \mathbin{+\!+}$

$f \quad := \ (\mathbin{+\!+})$

$(\oplus) := \ \lambda t\ ts \rightarrow ts \mathbin{+\!+} [t]$

$e \quad := \ []$

Applying fusion

$f\ (foldr\ (\oplus)\ e\ xs) = foldr\ (\otimes)\ e'\ xs$

If f strict, $f\ e = e'$

$\lambda\ a\ b \rightarrow f\ (a \oplus b) = \lambda\ a\ b \rightarrow a \otimes f\ b$

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

$f \quad := \ ({+}{+})$

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

$e \quad := \ []$

- Substitute into side conditions

Applying fusion

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

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

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

- Rewrite exhaustively
- η-expand where needed

Applying fusion

$\lambda\ ts \rightarrow ts = e'$

$\lambda\ a\ b \rightarrow (\texttt{++})\ (b \texttt{ ++ } [a])$

$\quad = \lambda\ a\ b \rightarrow a \otimes ((\texttt{++})\ b)$

- Rewrite exhaustively
- η-expand where needed

Applying fusion

$$\lambda\, ts \rightarrow ts = e'$$

$$\lambda\, a\; b\; ts \rightarrow (b \mathbin{+\!+} [a]) \mathbin{+\!+} ts$$
$$= \lambda\, a\; b \rightarrow a \otimes ((\mathbin{+\!+})\; b)$$

- Rewrite exhaustively
- η-expand where needed

Applying fusion

$\lambda\ ts \rightarrow ts = e'$

$\lambda\ a\ b\ ts \rightarrow b \mathbin{++} (a{:}ts)$

$= \lambda\ a\ b \rightarrow a \otimes ((\mathbin{++})\ b)$

Applying fusion

$\lambda\, ts \rightarrow ts = e'$

$\lambda\, a\; b\; ts \rightarrow b \mathbin{++} (a{:}ts)$

$\quad = \lambda\, a\; b \rightarrow a \otimes ((\mathbin{++})\; b)$

$e' \quad := \lambda\, ts \rightarrow ts$

$(\otimes) \quad := \lambda\, t\; f\; ts \rightarrow f\; (t{:}ts)$

Higher-order matching

- Various algorithms
- All solve for ϕ in the equation

$$\phi P = T$$

ϕ a substitution, P and T λ-terms

P contains free variables, T closed

- Vary in
  - Restrictions on P
  - Which solutions are returned
  - More solutions ➔ More restrictions

Fast reverse

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

➔

$$reverse\ xs = reverse'\ xs\ []$$

$$reverse'\ \mathrm{xs\ ys} =$$

$$foldr\ (\lambda\ t\ f\ ts \rightarrow f\ (t{:}ts))\ (\lambda\ ts \rightarrow \mathrm{ts})\ xs\ ys$$

Fast reverse

$$reverse\ xs = foldr\ (\lambda\ t\ ts \rightarrow ts \mathbin{++} [t])\ []\ xs$$

$$\Leftrightarrow$$

$$reverse\ xs = reverse'\ xs\ []$$

$$reverse'\ \mathrm{xs}\ \mathrm{ys} =$$

$$foldr\ (\lambda\ t\ f\ ts \rightarrow f\ (t{:}ts))\ (\lambda\ ts \rightarrow \mathrm{ts})\ xs\ ys$$

Warm fusion

xs = *foldr* (:) [] *xs*

reverse xs = *reverse* (*foldr* (:) [] *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?