Chapter 25. Fusion and Tupling

Zhenjiang Hu, Wei Zhang

School of Computer Science
Peking University

December 27, 2023

Outline

max

Consider the function to compute the maximum of a list (by reusing *sort*):

$$\begin{aligned} max &: [Int] \to Int \\ max &= head \cdot sort \end{aligned}$$

max

Consider the function to compute the maximum of a list (by reusing *sort*):

$$\begin{array}{lcl} max & : & [Int] \to Int \\ max & = & head \cdot sort \end{array}$$

where *sort* is defined by

$$\begin{array}{lcl} sort & = & foldr\ insert\ [\,] \\ insert\ a\ [] & = & [a] \\ insert\ a\ (b : x) & = & \textbf{if}\ a \geq b\ \textbf{then}\ a : (b : x) \\ & & \textbf{else}\ b : insert\ a\ x. \end{array}$$

How to eliminate all intermediate results in computing *max*?

reverse

Consider the following function to reverse a list:

$$
\begin{array}{lcl}
\mathit{rev}\ x & = & \mathit{fastrev}'\ x\ [\,] \\
\mathit{fastrev}'\ x\ y & = & \mathit{reverse}\ x +\!\!+ y \\
 & & \textbf{where} \\
 & & \quad \mathit{reverse} = \mathit{foldr}\ (\lambda a\ r.\ r +\!\!+ [a])\ [\,]
\end{array}
$$

How to eliminate the intermediate list in computing $\mathit{fastrev}'$?

reverse

Consider the following function to reverse a list:

$$
\begin{array}{lcl}
\mathit{rev}\ x & = & \mathit{fastrev}'\ x\ [\,] \\
\mathit{fastrev}'\ x\ y & = & \mathit{reverse}\ x +\!\!+ y \\
 & & \textbf{where} \\
 & & \quad \mathit{reverse} = \mathit{foldr}\ (\lambda a\ r.\ r +\!\!+ [a])\ [\,]
\end{array}
$$

How to eliminate the intermediate list in computing $\mathit{fastrev}'$?

Exercise

Show evaluation steps of $\mathit{rev}\ [1, 2, 3, 4]$, and explain that $(\mathit{rev}\ \mathit{xs})$ is a quadratic program.

Fusion Law for Foldr

Lemma (Foldr Fusion)

$$\frac{f\,(a \oplus r) = a \otimes f\ r}{f \cdot \mathit{foldr}\ (\oplus)\ e = \mathit{foldr}\ (\otimes)\ (f\ e)}$$

Fusion Law for Foldr

Lemma (Foldr Fusion)

$$\frac{f(a \oplus r) = a \otimes f\ r}{f \cdot foldr\ (\oplus)\ e = foldr\ (\otimes)\ (f\ e)}$$

Or written as

$$\frac{f(a \oplus r) = a \otimes f\ r}{f \cdot \oplus \not\leftarrow_e = \otimes \not\leftarrow_{f\ e}}$$

Fusion: max

Consider the fusion for *max*:

$$max = head \cdot foldr\ insert\ [\,]$$

where we assume that $max\ [\,] = -\infty$.

Fusion: max

Consider the fusion for *max*:

$$max = head \cdot foldr\ insert\ [\,]$$

where we assume that $max\ [\,] = -\infty$.

To apply the foldr fusion lemma, we consider calculation of *head* (*insert a r*).

We calculate as follows.

- For the case of $r = []$, we have:

$$
\begin{array}{ll}
 & \mathit{head}\ (\mathit{insert}\ a\ [\,]) \\
= & \quad \{\ \text{def. of } \mathit{insert}\ \} \\
 & \mathit{head}\ [a] \\
= & \quad \{\ \text{def. of } \mathit{head}\ \} \\
 & a
\end{array}
$$

Fusion Example: max

- For the case of $r = b : x$, we have:

$$
\begin{aligned}
& \mathit{head}\ (\mathit{insert}\ a\ (b : x)) \\
= & \quad \{\ \text{def. of } \mathit{insert}\ \} \\
& \mathit{head}\ (\textbf{if}\ a \geq b\ \textbf{then}\ a : (b : x)\ \textbf{else}\ b : \mathit{insert}\ a\ x) \\
= & \quad \{\ \text{distribute } \mathit{head} \text{ over } \mathit{if}\ \} \\
& \textbf{if}\ a \geq b\ \textbf{then}\ \mathit{head}\ (a : (b : x))\ \textbf{else}\ \mathit{head}\ (b : \mathit{insert}\ a\ x) \\
= & \quad \{\ \text{def. of } \mathit{head}\ \} \\
& \textbf{if}\ a \geq b\ \textbf{then}\ a\ \textbf{else}\ b \\
= & \quad \{\ \text{assumption: } r = b : x\ \} \\
& \textbf{if}\ a \geq \mathit{head}\ r\ \textbf{then}\ a\ \textbf{else}\ \mathit{head}\ r
\end{aligned}
$$

In summary, we have

$$head\ (insert\ a\ r) = a \otimes head\ r$$
$$\textbf{where}\ a \otimes r = \textbf{if}\ a \geq r\ \textbf{then}\ a\ \textbf{else}\ r$$

It follows from the foldr fusion lemma that we get the following new definition for *max*.

$$max = foldr\ (\otimes)\ (-\infty)$$

A linear time program!

Fusion Example: Fast Reverse

Consider fusion of the following program:

$$fastrev' \ x \ y = reverse \ x +\!\!+ y$$

Fusion Example: Fast Reverse

Consider fusion of the following program:

$$fastrev' \ x \ y = reverse \ x +\!\!+ \ y$$

Exercise

What is the intermediate list in the above computation?

Fusion Example: Fast Reverse

Consider fusion of the following program:

$$fastrev' \ x \ y = reverse \ x +\!\!+ \ y$$

Exercise

What is the intermediate list in the above computation?

We can see where fusion calculation is application if we rewrite the definition.

$$\begin{aligned} fastrev' \ x &= (+\!\!+) \ (reverse \ x) \\ &= ((+\!\!+) \cdot foldr \ (\lambda a \ r. \ r +\!\!+ [a]) \ [\,]) \ x \end{aligned}$$

Let us calculate the fusion condition:

$$
\begin{aligned}
& (+\!\!+)\ (r +\!\!+ [a]) \\
= & \quad \{\ \eta \text{ expansion } \} \\
& \lambda y.\ (+\!\!+)\ (r +\!\!+ [a])\ y \\
= & \quad \{\ \text{section notation } \} \\
& \lambda y.\ (r +\!\!+ [a]) +\!\!+ y) \\
= & \quad \{\ \text{associativity of } +\!\!+ \ \} \\
& \lambda y.\ r +\!\!+ ([a] +\!\!+ y)
\end{aligned}
$$

Marching it with $a \otimes ((+\!\!+)\ r)$ gives

$$a \otimes r' = \lambda y.\ r'\ ([a] +\!\!+ y)$$

So we get

$$\begin{aligned} \mathit{fastrev}'\ x &= \mathit{foldr}\ (\otimes)\ ((+\!\!+)\ [])\ x \\ \textbf{where}\ a \otimes r' &= \lambda y.\ r'\ ([a] +\!\!+ y) \end{aligned}$$

So we get

$$\begin{aligned} \mathit{fastrev}'\ x &= \mathit{foldr}\ (\otimes)\ ((+\!\!+)\ [])\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!\!+ y) \end{aligned}$$

which is the same as

$$\begin{aligned} \mathit{fastrev}'\ x &= \mathit{foldr}\ (\otimes)\ \mathit{id}\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!\!+ y) \end{aligned}$$

So we get

$$\begin{aligned} &fastrev'\ x = foldr\ (\otimes)\ ((+\!\!+)\ [])\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!\!+ y) \end{aligned}$$

which is the same as

$$\begin{aligned} &fastrev'\ x = foldr\ (\otimes)\ id\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!\!+ y) \end{aligned}$$

That is,

$$\begin{aligned} fastrev'\ []\ y &= y \\ fastrev'\ (a : r)\ y &= fastrev'\ r\ (a : y) \end{aligned}$$

So we get

$$\begin{aligned} &fastrev'\ x = foldr\ (\otimes)\ ((+\!+)\ [])\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!+ y) \end{aligned}$$

which is the same as

$$\begin{aligned} &fastrev'\ x = foldr\ (\otimes)\ id\ x \\ &\quad \textbf{where}\ a \otimes r' = \lambda y.\ r'\ ([a] +\!+ y) \end{aligned}$$

That is,

$$\begin{aligned} fastrev'\ []\ y &= y \\ fastrev'\ (a : r)\ y &= fastrev'\ r\ (a : y) \end{aligned}$$

A linear time algorithm!

Homework BMF 3-1

Using the foldr fusion lemma to prove the following two equations.

1. $foldr\ (\oplus)\ e \cdot map\ f = foldr\ (\lambda a\ r.f\ a \oplus r)\ e$
2. $map\ f \cdot map\ g = map\ (f \cdot g)$

Outline

Exercise

What is the time complexity for the following function that computes the maximum element from a list.

$$
\begin{array}{lll}
maximum\ [a] & = & a \\
maximum\ (a : x) \mid a > maximum\ x & = & a \\
\quad \mid otherwise & = & maximum\ x
\end{array}
$$

Enumerating Bigger Elements

Enumerate all bigger elements in a list. An element is bigger if it is greater than the sum of the elements that follow it till the end of the list.

$$\textit{biggers}\ [3, 10, 4, -2, 1, 3] = [10, 4, 3]$$

Enumerating Bigger Elements

Enumerate all bigger elements in a list. An element is bigger if it is greater than the sum of the elements that follow it till the end of the list.

$$biggers\ [3, 10, 4, -2, 1, 3] = [10, 4, 3]$$

$$biggers\ [] = []$$
$$biggers\ (a : x) = \textbf{if}\ a > sum\ x\ \textbf{then}\ a : biggers\ x\ \textbf{else}\ biggers\ x$$

$$sum\ [] = 0$$
$$sum\ (a : x) = a + sum\ x$$

Enumerating Bigger Elements

Enumerate all bigger elements in a list. An element is bigger if it is greater than the sum of the elements that follow it till the end of the list.

$$biggers\ [3, 10, 4, -2, 1, 3] = [10, 4, 3]$$

$$biggers\ [] = []$$
$$biggers\ (a : x) = \textbf{if}\ a > sum\ x\ \textbf{then}\ a : biggers\ x\ \textbf{else}\ biggers\ x$$

$$sum\ [] = 0$$
$$sum\ (a : x) = a + sum\ x$$

How can we optimize this program?

Enumerating Bigger Elements

Enumerate all bigger elements in a list. An element is bigger if it is greater than the sum of the elements that follow it till the end of the list.

$$biggers\ [3, 10, 4, -2, 1, 3] = [10, 4, 3]$$

$$biggers\ [] = []$$
$$biggers\ (a : x) = \textbf{if}\ a > sum\ x\ \textbf{then}\ a : biggers\ x\ \textbf{else}\ biggers\ x$$

$$sum\ [] = 0$$
$$sum\ (a : x) = a + sum\ x$$

How can we optimize this program?

Exercise: Is *biggers* a foldr?

Definition (Mutumorphism)

Functions $f_1, \ldots, f_n$ are said to form a mutumorphism if each f_i $(i = 1, 2, \ldots, n)$ is defined in the following form:

$$\begin{array}{lcl} f_i\ [] & = & e_i \\ f_i\ (a : x) & = & a \oplus_i (f_1\ x, f_2\ x, \ldots, f_n\ x) \end{array}$$

where e_i $(i = 1, 2, \ldots, n)$ are given constants and $\oplus_i$ $(i = 1, 2, \ldots, n)$ are given binary functions. We represent the function $f\ x = (f_1\ x, \ldots, f_n\ x)$ as follows.

$$f = [\![(e_1, \ldots, e_n), (\oplus_1, \ldots, \oplus_n)]\!].$$

Expressive Power of Mutumorphism

- *foldr* is a special case:

$$foldr\ (\oplus)\ e = [\![(e), (oplus)]\!]$$

- It covers all primitive recursive functions on lists.

$$\begin{aligned} prim\ [\,] &= e \\ prim\ (a : x) &= F(a, x, prim\ x) \end{aligned}$$

This is because we can *prim* is mutually defined with the identity function on lists.

biggers as a Mutumorphism

$$
\begin{aligned}
\mathit{biggers} &= \mathit{fst} \circ [\![([], 0), (\oplus_1, \oplus_2)]\!] \\
&\textbf{where } a \oplus_1 (r, s) = \textbf{if } a > s \textbf{ then } a : r \textbf{ else } r \\
&\qquad\quad a \oplus_2 (r, s) = a + s
\end{aligned}
$$

Lemma (Mutu-Tupling)

$$
\begin{aligned}
&[\![(e_1, e_2, \ldots, e_n), (\oplus_1, \oplus_2, \ldots, \oplus_n)]\!] \\
&\quad = \mathit{foldr}\ (\oplus)\ (e_1, e_2, \ldots, e_n) \\
&\qquad \textbf{where}\ a \oplus r = (a \oplus_1 r, a \oplus_2 r, \ldots, a \oplus_n r)
\end{aligned}
$$

Consider, as an example, to apply the mutu-tupling lemma to *biggers*.

$$
\begin{aligned}
& \mathit{biggers} \\
= & \quad \{ \text{ mutumorphism for } \mathit{biggers} \} \\
& \mathit{fst} \circ [\![([], 0), (\oplus_1, \oplus_2)]\!] \\
= & \quad \{ \text{ mutu-tupling lemma } \} \\
& \mathit{fst} \circ \mathit{foldr}\ (\oplus)\ ([], 0) \\
& \quad \textbf{where}\ a \oplus (r, s) = (\textbf{if}\ a > s\ \textbf{then}\ a : r\ \textbf{else}\ r, a + s)
\end{aligned}
$$

Inlining *foldr* in the derived program gives the following readable recursive program:

$$
\begin{aligned}
&\mathit{biggers}\ x = \textbf{let}\ (r, s) = \mathit{tup}\ x\ \textbf{in}\ r \\
&\quad \textbf{where}\ \mathit{tup}\ [] = ([], 0) \\
&\qquad\quad \mathit{tup}\ (a : x) = \textbf{let}\ (r, s) = \mathit{tup}\ x \\
&\qquad\qquad\qquad\qquad \textbf{in}\ (\textbf{if}\ a > s\ \textbf{then}\ a : r\ \textbf{else}\ r, a + s)
\end{aligned}
$$

Lemma (Foldr-Tupling)

$$(\mathit{foldr}\ (\oplus_1)\ e_1\ x,\ \mathit{foldr}\ (\oplus_2)\ e_2\ x) = \mathit{foldr}\ (\oplus)\ (e_1, e_2)\ x$$
$$\textbf{where}\ a \oplus (r_1, r_2) = (a \oplus_1 r_1, a \oplus_2 r_2)$$

For example, the following program for computing the average of a list:

$$average\ x = sum\ x / length\ x$$

can be transformed into the following with the foldr-tupling lemma.

$$\begin{aligned} average\ x = {} & \textbf{let}\ (s, l) = tup\ x\ \textbf{in}\ s/l \\ & \textbf{where}\ tup = foldr\ (\lambda a\ (s, l).\ (a + s, 1 + l))\ (0, 0) \end{aligned}$$

Homework BMF 3-2

(1) Using tupling transformation to derive an efficient program for computing *tailsums*.

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

(2) Code the efficient program in Haskell.