List homomorphisms

Troels Henriksen (athas@sigkill.dk)

DIKU
University of Copenhagen

29-11-2021

Preliminaries

Material
- *The Third Homomorphism Theorem* (Gibbons; 1995)
- *Construction of List Homomorphisms by Tupling and Fusion* (Hu, Iwasaki, Takeichi; 1996)

Approach
- **This material is dense.**
- Do not get dissuaded!
  - ‣ Ask about notation that is unclear.
- Proofs in the papers, not the lecture.
- We focus on intuition about working algebraically with (parallel!) programs.
- My notation is slightly different than in the papers.
  - ‣ I write *hom* for homomorphisms (and leave out neutral elements).
  - ‣ I write *map f* instead of $f*$.

Definition: list homomorphism

A function h is called a list homomorphism if there exists a binary operator $\odot$ such that for all finite lists x, y

$$h\,(x \mathbin{+\!\!+} y) = h\,x \odot h\,y$$

where $\mathbin{+\!\!+}$ means list concatenation.

- Intuitively, homomorphisms are functions where we can split the input into chunks arbitrarily, recursively process the cunks, and combine the partial results.
- Each "chunk" can be processed in parallel.

Examples of list homomorphisms

Summation

$$sum\ [] = 0$$
$$sum\ [x] = x$$
$$sum\ (x \mathbin{+\!\!+} y) = sum\ x + sum\ y$$

Length

$$len\ [] = 0$$
$$len\ [x] = 1$$
$$len\ (x \mathbin{+\!\!+} y) = len\ x + len\ y$$

Observations

$$h\,(x + y) = h\,x \odot h\,y$$

- $\odot$ is necessarily associative because $+$ is associative.
- $h\,[]$ is necessarily a neutral element for $\odot$.

For concision, we write

$$hom\,(\odot)\,f$$

for a list homomorphism h, where

$$f\,x = h\,[x]$$
$$0_\odot = h[]$$

Examples

$$
\begin{aligned}
sum &= hom\,(+)\,id \\
length &= hom\,(+)\,(const\,1)
\end{aligned}
$$

Reductions

Reductions are homomorphisms

A function of the form

$$hom \ (\odot) \ id$$

is called a *reduction* with operator $\odot$.

- Equivalent to what we would write as $\boxed{\texttt{reduce} \ \odot \ 0_\odot}$ in Futhark.
- For concision we will abbreviate as follows:

$$red \ \odot = hom \ (\odot) \ id$$

  ▸ Given $\odot$, the neutral element $0_\odot$ is always unique (think about why).

Maps

Maps are homomorphisms

A function of the form

$$hom \; (+\!\!\!+) \; ([\cdot] \circ f)$$

is called a *map* with function f.

- $[\cdot]$ is the function $\lambda x \to [x]$ that turns x into $[x]$.
- We'll write this as *map f*.
- Really is just what you expect, no surprises here.

Does this mean you could define map with `reduce` in Futhark?

Maps

Maps are homomorphisms

A function of the form

$$hom \ (+\!\!+) \ ([\cdot] \circ f)$$

is called a *map* with function f.

- $[\cdot]$ is the function $\lambda x \to [x]$ that turns x into $[x]$.
- We'll write this as *map f*.
- Really is just what you expect, no surprises here.

Does this mean you could define map with `reduce` in Futhark?
- **No**, reduce (++) [] is not well-typed.
- Futhark disallows irregular arrays like [[1,2],[3]].
- We will see many things you cannot express in most parallel languages.

Leftwards and rightwards functions

A function on lists h is $\oplus$-leftwards if and only if for all elements a and lists y,

$$h\ ([a] +\!\!\!+ y) = a \oplus h\ y$$

- $\oplus$ need not be associative.
- **Intuitively, what does this say about h?**

Leftwards and rightwards functions

A function on lists h is $\oplus$-leftwards if and only if for all elements a and lists y,

$$h\,([a] \mathbin{+\!\!+} y) = a \oplus h\,y$$

- $\oplus$ need not be associative.
- **Intuitively, what does this say about h?**
- Suppose

$$h\,[] = e$$

  then

$$h\,[a, b, c] = a \oplus (b \oplus (c \oplus e))$$

- A $\oplus$-leftwards function is a right-to-left *fold*, denoted by *foldr* $(\oplus)$ e.
- Symmetrically, for an $\otimes$-rightwards function h,

$$h\,(y \mathbin{+\!\!+} [a]) = h\,y \otimes a$$

  which we denote by *foldl* $(\otimes)$ e.

First Homomorphism Theorem

Every monomorphism can be written as the composition of a reduction and a map.

$$hom \ (\odot) \ f = red \ (\odot) \circ map \ f$$

Conversely, every such composition is a homomorphism.

- **This is why $map - reduce$ compositions are so crucial.**
- Sometimes it is easiest to invent parallel algorithms by thinking "homormorphism-style" in terms of recursively splitting and combining subresults.
    - Can systematically turn this into data-parallel map-reduce code.

If $\odot$ is associative, then

$$\begin{aligned}
hom\ (\odot)\ f &= foldr\ (\oplus)\ 0_\odot \quad \text{where } x \oplus y = f\ x \odot y \\
&= foldl\ (\otimes)\ 0_\otimes \quad \text{where } x \otimes y = x \odot f\ y
\end{aligned}$$

That is, *every homomorphism is both a leftwards and a rightwards function.*

- **Intuition:** as a homomorphism allows us to split "however we want", we can also split consistently with a left (or right) bias.
- **What about the other direction?**

Third Homomorphism Theorem

If h is leftwards and rightwards, then h is a homomorphism.

- **Why this is useful:** If we can come up with left-folds and right-folds that both solve the problem, then a homomorphism must also exist.
- **Put another way:** If a problem can be solved with both a *foldr* and a *foldl*, then a "parallel" algorithm for the problem also exists.

Third Homomorphism Theorem

If h is leftwards and rightwards, then h is a homomorphism.

- **Why this is useful:** If we can come up with left-folds and right-folds that both solve the problem, then a homormorphism must also exist.
- **Put another way:** If a problem can be solved with both a *foldr* and a *foldl*, then a "parallel" algorithm for the problem also exists.
- **Unfortunately:**
    - Not all homomorphisms are usefully parallel.
    - The proof does not tell us how to *construct* a practical homomorphism from the left and right folds.

Proof sketch (full details in Gibbons' paper)

The statement to prove

If h is leftwards and rightwards, then h is a homomorphism.

Steps:

1. Prove that
$$h\,v = h\,x \land h\,w = h\,y \Rightarrow h\,(v + w) = h\,(x + y)$$

   ▸ Rough steps:

   $$\begin{aligned} h\,(v + w) &= \ldots \\ &= h\,(v + y) \quad \text{(By treating } h \text{ as leftwards function.)} \\ &= \ldots \\ &= h\,(x + y) \quad \text{(By treating } h \text{ as rightwards function.)} \end{aligned}$$

2. Prove that this means h is a homomorphism.

The function h is a homomorphism if and only if the implication

$$h\,v = h\,x \land h\,w = h\,y \Rightarrow h\,(v \mathbin{+\!\!+} w) = h\,(x \mathbin{+\!\!+} y)$$

holds for all lists v, w, x, y.

"Only if" part

If h is a homomorphism then for all a, b there is a $\oplus$ such that,

$$h\,(a \mathbin{+\!\!+} b) = h\,a \oplus h\,b$$

and then

$$
\begin{aligned}
h\,(v \mathbin{+\!\!+} w) &= h\,v \oplus h\,w && \text{(Def. of homomorphism)} \\
&= h\,x \oplus h\,y && \text{(LHS of implication)} \\
&= h\,(x \mathbin{+\!\!+} y) && \text{(Def. of homomorphism)}
\end{aligned}
$$

The function h is a homomorphism if and only if the implication

$$h\,v = h\,x \wedge h\,w = h\,y \Rightarrow h\,(v + w) = h\,(x + y)$$

holds for all lists v, w, x, y.

"If" part

- Choose g such that $h \circ g \circ h = h$ and define

$$a \odot b = h\,(g\,a + g\,b)$$

- (We'll show that g exists in a moment.)
- We have $h\,x = h\,(g\,(h\,x))$ and $h\,y = h\,(g\,(h\,y))$.
- With $v = g\,(h\,x)$ and $w = g\,(h\,y)$ then

$$
\begin{aligned}
h\,(x + y) &= h\,(v + w) \\
&= h\,(g\,(h\,x) + g\,(h\,y)) \\
&= h\,x \odot h\,y
\end{aligned}
$$

Existence of g

Given h, we must construct g such that

$$h \circ g \circ h = h$$

- Can we be sure such a g exists for any h?
  - ▸ **Yes:** To compute $g \, x$, enumerate domain of h and produce first y such that $h \, y = x$.
  - ▸ **Theoretically sound, but not really practical.**
  - ▸ When we apply the Third Homomorphism Theorem we better come up with a more reasonable g than this.

Consider sorting

We can define

$$sort = foldr\ ins\ []$$

where

$$ins\ a\ [] = [a]$$

$$ins\ a\ (b : x) = \begin{cases} a : b : x & \text{if } a \leqslant b \\ b : ins\ a\ x & \text{otherwise} \end{cases}$$

That is, *ins a l* inserts *a* at the appropriate location in the sorted list *l*.

Consider sorting

We can define

$$sort = foldr\ ins\ []$$

where

$$ins\ a\ [] = [a]$$

$$ins\ a\ (b : x) = \begin{cases} a : b : x & \text{if } a \leqslant b \\ b : ins\ a\ x & \text{otherwise} \end{cases}$$

That is, *ins a l* inserts *a* at the appropriate location in the sorted list *l*.

$$sort = foldl\ ins'\ []$$

where

$$ins'\ x\ a = ins\ a\ x$$

As *sort* is both leftwards and rightwards, there is a homomorphic sorting algorithm.

Homomorphic sorting

By the Third Homomorphism Theorem, we get a homomorphism $hom \odot [\cdot]$ where

$$u \odot v = sort \ (unsort \ u \ +\!\!+ \ unsort \ v)$$

for some function *unsort* such that

$$sort \circ unsort \circ sort = sort$$

Can you think of such a function?

Homomorphic sorting

By the Third Homomorphism Theorem, we get a homomorphism $hom \odot [\cdot]$ where

$$u \odot v = sort \ (unsort \ u + unsort \ v)$$

for some function *unsort* such that

$$sort \circ unsort \circ sort = sort$$

Can you think of such a function?

$$unsort = id$$

This gives

$$u \odot v = sort \ (u + v)$$

Is this a good sorting algorithm?

Homomorphic sorting

By the Third Homomorphism Theorem, we get a homomorphism $hom \odot [\cdot]$ where

$$u \odot v = sort\ (unsort\ u \mathbin{+\!\!+} unsort\ v)$$

for some function *unsort* such that

$$sort \circ unsort \circ sort = sort$$

Can you think of such a function?

$$unsort = id$$

This gives

$$u \odot v = sort\ (u \mathbin{+\!\!+} v)$$

Is this a good sorting algorithm?

Homomorphic sorting in words

"To sort the concatenation of two sorted lists u and v, concatenate u and v and then sort the result using some other sorting algorithm."

Improving the homomorphic sort

- In the paper, Gibbons goes on to mechanically derive a more efficient $\odot$, based on the property that u will be sorted.

$$
\begin{aligned}
u \odot v &= sort \ (u + v) \\
&= foldl \ ins' \ [] \ (u + v) \\
&= foldl \ ins' \ (foldl \ ins' \ [] \ u) \ v \\
&= foldl \ ins' \ (sort \ u) \ v \\
&= foldl \ ins' \ u \ v
\end{aligned}
$$

- Now $u \odot v$ merges two sorted lists in $O(n^2)$ time.
- By also assuming v is sorted, Gibbons eventually derives mergesort.
  - Read the paper!
- **Not parallel**, but we could in principle have derived a parallel $\odot$ instead.

Almost homomorphisms

Some functions are not homomorphisms.

- Knowing the means of arrays x and y does not let us compute the mean of $x + y$.

But they can be turned into homomorphisms if we compute a little extra information.

- If we know the means x_{mean}, y_{mean} and sizes x_n, y_n of x, y, we can compute the mean and size of $x + y$ as

$$\frac{x_{mean} \cdot x_n + y_{mean} \cdot y_n}{x_n + y_n}$$

and

$$x_n + y_n$$

This is called an *almost homomorphism*.

mean **as an almost homomorphism**

$$mean\ [] = (0, 0)$$
$$mean\ [x] = (x, 1)$$
$$mean\ (x + y) = mean\ x \odot mean\ y$$

where

$$(x_{mean}, x_n) \odot (y_{mean}, y_n) = (\frac{x_{mean} \cdot x_n + y_{mean} \cdot y_n}{x_n + y_n}, x_n + y_n)$$

To compute the actual mean we then project out the first component:

$$\pi_1 \circ mean$$

The maximum segment sum problem

Problem statement

Given a list A, find the largest sum of a subsequence $A[i : j]$.

Examples

$$mss\ [0, 3, -2] = 3$$
$$mss\ [4, -1, 0] = 4$$
$$mss\ [0, 3, -2, 4, -1, 0] = 5$$

Not a homomorphism

- Knowing $mss\ x$ and $mss\ y$ is not enough to give us $mss(x + y)$.
- E.g. $x = [0, 3, -2], y = [4, -1, 0]$ as above.

But is an almost homomorphism

- With human creativity we can come up with an associative operator on 4-tuples that lets us solve MSS as an almost homomorphism.
- But can we also derive it systematically?

$$mss = max^s \circ (map^s \ sum) \circ segs$$

Note that max^s and map^s are set operations.

- We will try to *systematically* derive an efficient almost homomorphism from the specification.
- **Well-known technique in (academic) functional programming**, often called things like "constructive algorithmics" or "program calculation".

$$
\begin{array}{ll}
mss & = max^s \circ (map^s\ sum) \circ segs \\[2ex]
segs\ [] & = \{\} \\
segs\ [x] & = \{[x]\} \\
segs\ (xs + ys) & = segs\ xs \cup segs\ ys \cup (tails\ xs\ \mathcal{X}_{+}\ inits\ ys) \\[2ex]
inits\ [] & = [] \\
inits\ [x] & = [[x]] \\
inits\ (xs + ys) & = inits\ xs + map\ (xs +)\ (inits\ ys) \\[2ex]
tails\ [] & = [] \\
tails\ [x] & = [[x]] \\
tails\ (xs + ys) & = map\ (+ys)\ (tails\ xs) + tails\ ys \\[2ex]
[x_1, \cdots, x_n]\ \mathcal{X}_{\oplus}\ [y_1, \cdots, y_m] & = \{x_1 \oplus y_1, \cdots, x_1 \oplus y_m, \cdots, x_n \oplus y_1, \cdots, x_n \oplus y_m\}
\end{array}
$$

Tupling

$$(f \vartriangle g)\, x = (f\, x, g\, x)$$

So $f \vartriangle g$ is a function that applies both f and g to its argument.

$$a(\oplus \vartriangle \otimes)b = (a \oplus b, a \otimes b)$$

Example

$$(+1) \vartriangle (\cdot 2) \vartriangle (-3) = \lambda x \to (x + 1, x \cdot 2, x - 3)$$

$$(+) \vartriangle (\cdot) \vartriangle (-) = \lambda(x, y) \to (x + y, x \cdot y, x - y)$$

Tupling

$$(f \vartriangle g) \; x = (f \; x, g \; x)$$

So $f \vartriangle g$ is a function that applies both f and g to its argument.

$$a(\oplus \vartriangle \otimes)b = (a \oplus b, a \otimes b)$$

Example

$$(+1) \vartriangle (\cdot 2) \vartriangle (-3) = \lambda x \to (x + 1, x \cdot 2, x - 3)$$
$$(+) \vartriangle (\cdot) \vartriangle (-) = \lambda(x, y) \to (x + y, x \cdot y, x - y)$$

$$\Delta_1^n f_i = f_1 \vartriangle \cdots \vartriangle f_n$$

So Δ_1^n tuples the n functions f_i:

$$(\Delta_1^n f_i) \; x = (f_1 \; x, \cdots, f_n \; x)$$

Tupling of homomorphisms

Let $h_1, \cdots, h_n$ be mutually defined as follows:

$$
\begin{aligned}
h_j \; [] &= 0_{\oplus_j} \\
h_j \; [x] &= f_j \; x \\
h_j \; (xs + ys) &= ((\Delta_1^n h_i) \; xs) \oplus_j ((\Delta_i^n h_i) \; ys)
\end{aligned}
$$

Then

$$
\Delta_1^n h_i = hom \; (\Delta_1^n \oplus_i) \; (\Delta_1^n f_i)
$$

Implication: tupling mutually recursive functions that *traverse the same list in a specific way* will produce a list homomorphism.

Notation reminder

$$
(\Delta_1^n f_i) \; x = (f_1 \; x, \cdots, f_n \; x)
$$

Writing *segs* in the required form

We have

$$segs\ [] = \{\}$$
$$segs\ [x] = \{[x]\}$$
$$segs\ (xs + ys) = \underline{segs\ xs} \cup \underline{segs\ ys} \cup (\underline{tails\ xs}\ \mathcal{X}_{+}\ \underline{inits\ ys})$$

and want n mutually recursive functions

$$h_j\ [] = 0_{\oplus_j}$$
$$h_j\ [x] = f_j\ x$$
$$h_j\ (xs + ys) = ((\Delta_1^n h_i)\ xs) \oplus_j ((\Delta_i^n h_i)\ ys)$$

- *segs* must be tupled with *tails* because *segs* and *tails* traverse the same list *xs*.
- *segs* must be tupled with *inits* because *segs* and *inits* traverse the same list *xs*.

$$\begin{array}{ll}
\textit{inits } [] & = [] \\
\textit{inits } [x] & = [[x]] \\
\textit{inits } (xs \mathbin{+\!\!+} ys) & = \underline{\textit{inits } xs} \mathbin{+\!\!+} \textit{map } (\underline{xs}\mathbin{+\!\!+}) \; (\textit{inits } ys)
\end{array}$$

$$\begin{array}{ll}
\textit{tails } [] & = [] \\
\textit{tails } [x] & = [[x]] \\
\textit{tails } (xs \mathbin{+\!\!+} ys) & = \textit{map } (\mathbin{+\!\!+}\underline{ys}) \; (\textit{tails } xs) \mathbin{+\!\!+} \underline{\textit{tails } ys}
\end{array}$$

We need to tuple with *id*:

$$\begin{array}{ll}
\textit{id } [] & = [] \\
\textit{id } [x] & = [x] \\
\textit{id } (xs \mathbin{+\!\!+} ys) & = \textit{id } xs \mathbin{+\!\!+} \textit{id } ys
\end{array}$$

- To summarise, our tupled function will be

$$\textit{segs } \vartriangle \textit{inits } \vartriangle \textit{tails } \vartriangle \textit{id}$$

Still need to phrase this in the form

$$
\begin{aligned}
h_j \, [] &= 0_{\oplus_j} \\
h_j \, [x] &= f_j \, x \\
h_j \, (xs + ys) &= ((\Delta_1^n h_i) \, xs) \oplus_j ((\Delta_i^n h_i) \, ys)
\end{aligned}
$$

I.e. derive $f_1, \oplus_1$ for *segs*, $f_2, \oplus_2$ for *inits*, $f_3, \oplus_3$ for *tails*, $f_4, \oplus_4$ for *id*.

- Actually straightforward: pick corresponding recursive calls from the tuples.

Deriving f_1 and $\oplus_1$

Original

$$
\begin{aligned}
segs\ [] &= \{\} \\
segs\ [x] &= \{[x]\} \\
segs\ (xs + ys) &= \underline{segs\ xs} \cup \underline{segs\ ys} \cup (\underline{tails\ xs}\ \mathcal{X}_+\ \underline{inits\ ys})
\end{aligned}
$$

Goal

$$
\begin{aligned}
h_1\ [] &= 0_{\oplus_1} \\
h_1\ [x] &= f_1\ x \\
h_1\ (xs + ys) &= ((\Delta_1^n h_i)\ xs) \oplus_1 ((\Delta_i^n h_i)\ ys)
\end{aligned}
$$

Derivation

$$
\begin{aligned}
f_1\ x &= \{[x]\} \\
(s_1, i_1, t_1, d_1) \oplus_1 (s_2, i_2, t_2, d_2) &= s_1 \cup s_2 \cup (t_1\ \mathcal{X}_+\ i_2)
\end{aligned}
$$

Intuition s_1 to $segs\ xs$, i_1 to $inits\ xs$, t_1 to $tails\ xs$, d_1 to $id\ xs$, s_2 to $segs\ ys$, i_2 to $inits\ ys$, t_2 to $tails\ ys$, d_2 to $id\ ys$.

All derivations

Skeleton as a reminder

$$
\begin{aligned}
h_j \; [] &= 0_{\oplus_j} \\
h_j \; [x] &= f_j \; x \\
h_j \; (xs + ys) &= ((\Delta_j^n h_i) \; xs) \oplus_j ((\Delta_i^n h_i) \; ys)
\end{aligned}
$$

Derivation

$$
\begin{aligned}
f_1 \; x &= \{[x]\} \\
(s_1, i_1, t_1, d_1) \oplus_1 (s_2, i_2, t_2, d_2) &= s_1 \cup s_2 \cup (t_1 \; \mathcal{X}_{+} \; i_2) \\
f_2 \; x &= [[x]] \\
(s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2) &= i_1 + map \; (d_1 +) \; i_2 \\
f_3 \; x &= [[x]] \\
(s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2) &= map \; (d_2 +) \; t_1 + t_2 \\
f_4 \; x &= [x] \\
(s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2) &= d_1 + d_2
\end{aligned}
$$

The almost homomorphism

$$
segs = \pi_1 \circ hom \; (\Delta_1^n \oplus_i) \; (\Delta_1^n f_i)
$$

Fusion

$$mss = max^s \circ (map^s \ sum) \circ segs$$

where

$$segs = \pi_1 \circ hom \ (\Delta_1^4 \oplus_i) \ (\Delta_1^4 f_i)$$

Fusion

Merging adjacent operations to avoid intermediate structures or to allow further simplification.

Example of fusion

$$map \ f \circ map \ g = map \ (f \circ g)$$

Can we fuse $segs$ **into** $(map^s \ sum)$**?**

Fusion theorem for homomorphisms

Let h and $hom\ (\oplus)\ f$ be given. If there exists $\otimes$ such that

$$\forall x, y.\ h\ (x \oplus y) = h\ x \otimes h\ y$$

then

$$h \circ hom\ \oplus\ f = hom\ (\otimes)\ (h \circ f)$$

Example of applying fusion theorem

- Suppose $h = (2 \cdot), (\oplus) = (\uparrow), f = abs.$ ($\uparrow$ is the "max" operator)
  - What does $(2\cdot) \circ hom \ (\uparrow) \ abs$ compute?
- Must find $\otimes$ such that

$$\forall x, y. \ 2 \cdot (x \uparrow y) = (2 \cdot x) \otimes (2 \cdot y)$$

Example of applying fusion theorem

- Suppose $h = (2 \cdot), (\oplus) = (\uparrow), f = abs.$ ($\uparrow$ is the "max" operator)
  - What does $(2\cdot) \circ hom\ (\uparrow)\ abs$ compute?
- Must find $\otimes$ such that

$$\forall x, y.\ 2 \cdot (x \uparrow y) = (2 \cdot x) \otimes (2 \cdot y)$$

We fiddle a bit and find

$$x \otimes y = x \uparrow y$$

- Which means we can fuse as

$$(2\cdot) \circ hom\ (\uparrow)\ abs = hom\ (\uparrow)\ ((2\cdot) \circ abs)$$
$$= red\ (\uparrow) \circ map\ ((2\cdot) \circ abs)$$

Sadly this is not yet enough

Reminder:

$$mss = max^s \circ (map^s \; sum) \circ segs$$

where

$$segs = \pi_1 \circ hom \; (\Delta_1^4 \oplus_i) \; (\Delta_1^4 f_i)$$

- We want to fuse *segs* into $(map^s \; sum)$
- Since *segs* is not a homomorphism but an *almost homomorphism*, we cannot apply the fusion theorem.

Fortunately:
- There is also a fusion theorem for almost homomorphisms.
- Idea is to create a function H that "contains" h but also transforms the other tuple elements as necessary.

Fusion theorem for almost homomorphisms

Let h and $hom\ (\Delta_1^n\oplus_i)\ (\Delta_1^n f_i)$ be given. If there exists $\otimes_i\ (i = 1, \cdots, n)$ and a function $H = h_1 \times \cdots \times h_n$ where $h_1 = h$ such that for all i,

$$\forall x, y.\ h_i(x \oplus_i y) = H\,x \otimes_i H\,y$$

then

$$h \circ \pi_1 \circ hom\ (\Delta_1^n\oplus_i)\ (\Delta_1^n f_i) = \pi_1 \circ hom\ (\Delta_1^n\otimes_i)\ (\Delta_1^n(h_i \circ f_i))$$

Notation note: crossing functions, sometimes called *maps*

$$(h_1 \times \cdots \times h_n)\ (x_1, \cdots, x_n) = (h_1\,x_1, \cdots, h_n\,x_n)$$

Example of applying almost fusion theorem

Suppose we want to fuse

$$log \circ \pi_1 \circ hom \ (\Delta_1^2 \oplus_i) \ (\Delta_1^2 f_i)$$

where

$$f_1 \ x = x \quad (x_{mean}, x_n) \oplus_1 (y_{mean}, y_n) = \frac{x_{mean} \times x_n + y_{mean} \times y_n}{x_n + y_n}$$
$$f_2 \ x = 1 \quad (x_{mean}, x_n) \oplus_2 (y_{mean}, y_n) = x_n + y_n$$

We must define

$$h_1 = log$$
$$h_2 = ?$$
$$\otimes_1 = ?$$
$$\otimes_2 = ?$$

such that

$$h_1 \ (x \oplus_1 y) = (h_1 \times h_2) \ x \otimes_1 (h_1 \times h_2) \ y$$
$$h_2 \ (x \oplus_2 y) = (h_1 \times h_2) \ x \otimes_2 (h_1 \times h_2) \ y$$

With

$$(x_{mean}, x_n) \otimes_2 (y_{mean}, y_n) = \quad x_n + y_n$$
$$h_2\ x = \quad x$$

then

$$
\begin{aligned}
h_2\ (x \oplus_2 y) &= x_n + y_n \\
&= (log\ x_{mean}, h_2\ x_n) \otimes_2 (log\ y_{mean}, h_2\ y_n) \\
&= (log \times id)\ x \otimes_2 (log \times id)\ y \\
&= H\ x \otimes_2 H\ y
\end{aligned}
$$

With

$$\begin{aligned}
(x_{mean}, x_n) \otimes_1 (y_{mean}, y_n) &= log \left(\frac{exp \; x_{mean} \times x_n + exp \; y_{mean} \times y_n}{x_n + y_n} \right) \\
h_1 \; x &= log \; x \\
h_2 \; x &= x
\end{aligned}$$

then

$$\begin{aligned}
h_1 \; (x \oplus_1 y) &= h_1 \left((x_{mean}, x_n) \oplus_1 (y_{mean}, y_n) \right) \\
&= log \left(\frac{x_{mean} \times x_n + y_{mean} \times y_n}{x_n + y_n} \right) \\
&= (log \; x_{mean}, x_n) \otimes_1 (log \; y_{mean}, y_n) \\
&= (log \times id) \; x \otimes_1 (log \times id) \; y \\
&= H \; x \otimes_1 H \; y
\end{aligned}$$

Applying almost fusion

So we can fuse

$$log \circ \pi_1 \circ hom \ (\Delta_1^2 \oplus_i) \ (\Delta_1^2 f_i)$$

to

$$\pi_1 \circ hom \ (\Delta_1^2 \otimes_i) \ (\Delta_1^2 (h_i \circ f_i))$$

where

$$h_1 \ x = log \ x$$
$$h_2 \ x = x$$
$$(x_{mean}, x_n) \otimes_1 (y_{mean}, y_n) = log \ (\frac{exp \ x_{mean} \times x_n + exp \ y_{mean} \times y_n}{x_n + y_n})$$
$$(x_{mean}, x_n) \otimes_2 (y_{mean}, y_n) = x_n + y_n$$

$$mss = max^s \circ map^s \; sum \circ \pi_1 \circ hom \; (\Delta_1^4 \oplus_i) \; (\Delta_1^4 f_i)$$

We need to find $h_1, h_2, h_3, h_4, \otimes_1, \otimes_2, \otimes_3, \otimes_4$ that solve these equations ($j = 1, \cdots, 4$):

$$h_j \; ((s_1, i_1, t_1, d_1) \oplus_j (s_2, i_2, t_2, d_2)) =$$
$$(h_j \; s_1, h_2 \; i_1, h_3 \; t_1, h_4 \; d_1) \otimes_j (h_j \; s_s, h_2 \; i_2, h_3 \; t_3, h_4 \; d_2)$$

Procedure: calculate LHS of equation j by promoting h_j into result and determine $h_j, \otimes_j$ by matching with its RHS.

Example for $j = 1$

Equation

$$map^s\ sum\ ((s_1, i_1, t_1, d_1) \oplus_1 (s_2, i_2, t_2, d_2)) =$$
$$(map^s\ sum\ s_1, h_2\ i_1, h_3\ t_1, h_4\ d_1) \otimes_1 (map^s\ sum\ s_s, h_2\ i_2, h_3\ t_3, h_4\ d_2)$$

Calculation

$$map^s\ sum\ ((s_1, i_1, t_1, d_1) \oplus_1 (s_2, i_2, t_2, d_2))$$
$$= (map^s\ sum)\ (s_1 \cup s_2 \cup (t_1\ \mathcal{X}_+\ i_2))$$
$$= map^s\ sum\ s_1 \cup map^s\ sum\ s_2 \cup map^s\ sum(t_1\ \mathcal{X}_+\ i_2))$$
$$= map^s\ sum\ s_1 \cup map^s\ sum\ s_2 \cup (t_1\ \mathcal{X}_{map^s\ sum\circ +}\ i_2)$$
$$= map^s\ sum\ s_1 \cup map^s\ sum\ s_2 \cup (map\ sum\ t_1\ \mathcal{X}_+\ map\ sum\ i_2)$$

Matching last expression with RHS of the equation gives

$$h_2 = h_3 = map\ sum$$
$$(s_1, i_1, t_1, d_1) \otimes_1 (s_2, i_2, t_2, d_2) = s_1 \cup s_2 \cup (t_1\ \mathcal{X}_+\ i_2)$$

Example for $j = 2$

Equation

$$map\ sum\ ((s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2)) =$$
$$(map^s\ sum\ s_1, h_2\ i_1, h_3\ t_1, h_4\ d_1) \otimes_2 (map^s\ sum\ s_s, h_2\ i_2, h_3\ t_3, h_4\ d_2)$$

Calculation

$$map\ sum\ ((s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2))$$
$$= map\ sum\ (i_1 + map\ (d_1+)\ i_2)$$
$$= map\ sum\ i_1 + map\ sum\ (map\ (d_1+)\ i_2)$$
$$= map\ sum\ i_1 + map\ (sum \circ (d_1+))\ i_2$$
$$= map\ sum\ i_1 + map\ (sum\ d_1+)\ (map\ sum\ i_2)$$

Matching last expression with RHS of the equation gives

$$h_4 = sum$$
$$(s_1, i_1, t_1, d_1) \otimes_2 (s_2, i_2, t_2, d_2) = i_1 + map\ (d_1+)\ i_2$$

$$h_1 = map^s \; sum$$
$$h_2 = map \; sum$$
$$h_3 = map \; sum$$
$$h_4 = sum$$

$$(s_1, i_1, t_1, d_1) \otimes_1 (s_2, i_2, t_2, d_2) = s_1 \cup s_2 \cup (t_1 \; \mathcal{X}_+ \; i_2)$$
$$(s_1, i_1, t_1, d_1) \otimes_2 (s_2, i_2, t_2, d_2) = i_1 + map \; (d_1+) \; i_2$$
$$(s_1, i_1, t_1, d_1) \otimes_3 (s_2, i_2, t_2, d_2) = map \; (+d_2) \; t_1 + t_2$$
$$(s_1, i_1, t_1, d_1) \otimes_4 (s_2, i_2, t_2, d_2) = d_1 + d_2$$

The f parts

To apply the Almost Homomorphism Fusion Theorem we also need fused f functions:

$$f_1\ x = \{[x]\}$$
$$f_2\ x = [[x]]$$
$$f_3\ x = [[x]]$$
$$f_4\ x = [x]$$

$$f_1'\ x = ((map^s\ sum) \circ f_1)\ x = \{x\}$$
$$f_2'\ x = ((map\ sum) \circ f_2)\ x = [x]$$
$$f_3'\ x = ((map\ sum) \circ f_3)\ x = [x]$$
$$f_4'\ x = (sum \circ f_4)\ x = [x]$$

And finally we obtain:

$$map^s\ sum \circ \pi_1 \circ hom\ (\Delta_1^4 \oplus_i)\ (\Delta_1^4 f_i) = \pi_1 \circ hom\ (\Delta_1^4 \otimes_i)\ (\Delta_1^4 f_i')$$

Finishing up

$$mss = max^s \circ \pi_1 \circ hom \, (\Delta_1^4 \otimes_i) \, (\Delta_1^4 f_i')$$

We can fuse max^s with the remaining almost homomorphism with

$$H = max^s \times max \times max \times id$$

and get the final almost homomorphism

$$mss = \pi_1 \circ hom \, (\Delta_1^4 \otimes_i') \, id$$

where

$$
\begin{aligned}
(s_1, i_1, t_1, d_1) \otimes_1' (s_2, i_2, t_2, d_2) &= s_1 \uparrow s_2 \uparrow (t_1 + i_2) \\
(s_1, i_1, t_1, d_1) \otimes_2' (s_2, i_2, t_2, d_2) &= i_1 \uparrow (d_1 + i_2) \\
(s_1, i_1, t_1, d_1) \otimes_3' (s_2, i_2, t_2, d_2) &= (t_1 + d_2) \uparrow t_2 \\
(s_1, i_1, t_1, d_1) \otimes_4' (s_2, i_2, t_2, d_2) &= d_1 + d_2
\end{aligned}
$$

What has been accomplished?

We went from

$$mss = max^s \circ (map^s \ sum) \circ segs$$

to

$$mss = \pi_1 \circ hom \ (\Delta_1^4 \otimes_i') \ id$$

- Specification was an inefficient $O(n^2)$ algorithm involving computationally difficult objects (sets).
- Final derivation has only (tuples of) scalars as intermediate values.
  - You could easily translate the final bit into Futhark or any other parallel language.

Takeaways

What do we make of all this?

- Easy to get lost in the details of formalisms.
- Only a subset of the programs described can be implemented properly in real languages.
- These theorems and techniques **do not give us algorithms for free**.
- They **help us structure our search for efficient implementations**.
- Usually easier to find small functions that satisfy specific properties than to come up with algorithms from scratch using sheer creativity.
- (But maybe some of this could be mechanised? Maybe a fun (research) project!)