

List Homomorphisms

Troels Henriksen (athas@sigkill.dk)

DIKU
University of Copenhagen

28-11-2021

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 $f*$.

List homomorphisms

- Definitions and properties

- Homomorphism theorems

- Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

- Almost homomorphisms

- Deriving initial almost homomorphisms by tupling

- Fusion theorems for homomorphisms

- Fusion of *mss*

Conclusions

List homomorphisms

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 chunks, and combine the partial results.
- Each “chunk” can be processed in parallel.

Examples of list homomorphisms

Summation

$$\text{sum } [] = 0$$

$$\text{sum } [x] = x$$

$$\text{sum } (x ++ y) = \text{sum } x + \text{sum } y$$

Length

$$\text{len } [] = 0$$

$$\text{len } [x] = 1$$

$$\text{len } (x ++ y) = \text{len } x + \text{len } y$$

Observations

$$h (x \oplus y) = h x \odot h y$$

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

For concision, we write

$$\text{hom } (\odot) f$$

for a list homomorphism h , where

$$f x = h [x]$$

$$0_{\odot} = h []$$

Examples

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

Reductions

Reductions are homomorphisms

A function of the form

$$\text{hom } (\odot) \text{ id}$$

is called a *reduction* with operator \odot .

- Equivalent to what we would write as `reduce \odot 0 \odot` in Futhark.
- For concision we will abbreviate as follows:

$$\text{red } \odot = \text{hom } (\odot) \text{ id}$$

- Given \odot , the neutral element 0_{\odot} is always unique (think about why).

Maps

Maps are homomorphisms

A function of the form

$$\text{hom } (+) ([\cdot] \circ f)$$

is called a *map* with function f .

- $[\cdot]$ is the function $\lambda x \rightarrow [x]$ 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 are homomorphisms

A function of the form

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

is called a *map* with function f .

- $[\cdot]$ is the function $\lambda x \rightarrow [x]$ 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 ,

$$h ([a] \uplus y) = a \oplus h y$$

- \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 ?** It's a fold!
- 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* $(\otimes) e$.

List homomorphisms

Definitions and properties

Homomorphism theorems

Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

Almost homomorphisms

Deriving initial almost homomorphisms by tupling

Fusion theorems for homomorphisms

Fusion of *mss*

Conclusions

First Homomorphism Theorem

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

$$\text{hom } (\odot) f = \text{red } (\odot) \circ \text{map } f$$

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 “homomorphism-style” in terms of recursively splitting and combining subresults.
 - ▶ Can systematically turn this into data-parallel map-reduce code.
 - ▶ Essentially *divide and conquer* in different guise.

Second Homomorphism Theorem

If \odot is associative, then

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

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 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.
- **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

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

► Rough steps:

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

2. Prove that this means h is a homomorphism.

To prove

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 .

“Only if” part (where we assume h is a homomorphism)

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

$$h\ (a \# b) = h\ a \oplus h\ b$$

and then

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

To prove

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 (where we prove h is a homomorphism)

- 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.

List homomorphisms

Definitions and properties

Homomorphism theorems

Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

Almost homomorphisms

Deriving initial almost homomorphisms by tupling

Fusion theorems for homomorphisms

Fusion of *mss*

Conclusions

Consider sorting

We can define

$$\text{sort} = \text{foldr ins } []$$

where

$$\begin{aligned} \text{ins } a [] &= [a] \\ \text{ins } a (b : x) &= \begin{cases} a : b : x & \text{if } a \leq b \\ b : \text{ins } a x & \text{otherwise} \end{cases} \end{aligned}$$

That is, $\text{ins } a l$ inserts a at the appropriate location in the sorted list l .

Consider sorting

We can define

$$\text{sort} = \text{foldr ins } []$$

where

$$\begin{aligned} \text{ins } a [] &= [a] \\ \text{ins } a (b : x) &= \begin{cases} a : b : x & \text{if } a \leq b \\ b : \text{ins } a x & \text{otherwise} \end{cases} \end{aligned}$$

That is, $\text{ins } a l$ inserts a at the appropriate location in the sorted list l .

$$\text{sort} = \text{foldl ins}' []$$

where

$$\text{ins}' x a = \text{ins } a x$$

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

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

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

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

for some function *unsort* such that

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

Can you think of such a function?

$$unsort = id$$

This gives

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

Is this a good sorting algorithm?

Homomorphic sorting

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

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

for some function *unsort* such that

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

Can you think of such a function?

$$unsort = id$$

This gives

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

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.

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

- 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.

List homomorphisms

Definitions and properties

Homomorphism theorems

Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

Almost homomorphisms

Deriving initial almost homomorphisms by tupling

Fusion theorems for homomorphisms

Fusion of *mss*

Conclusions

Almost homomorphisms

Some functions are not homomorphisms.

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

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 $x \uplus y$ 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

$$\begin{aligned} \text{mean } [] &= (0, 0) \\ \text{mean } [x] &= (x, 1) \\ \text{mean } (x \uplus y) &= \text{mean } x \odot \text{mean } y \end{aligned}$$

where

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

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

$$\pi_1 \circ \text{mean}$$

Notation

The function π_i extracts the i th component of a tuple (counting from 1).

List homomorphisms

Definitions and properties

Homomorphism theorems

Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

Almost homomorphisms

Deriving initial almost homomorphisms by tupling

Fusion theorems for homomorphisms

Fusion of *mss*

Conclusions

The maximum segment sum problem

Problem statement

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

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?

Specification of *mss*

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

Note that max^s and map^s are *set* operations (not lists).

- 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”.

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

$$segs [] = \{\}$$

$$segs [x] = \{[x]\}$$

$$segs (xs \# ys) = segs\ xs \cup segs\ ys \cup (tails\ xs\ \mathcal{X}_{\#}\ inits\ ys)$$

$$inits [] = []$$

$$inits [x] = [[x]]$$

$$inits (xs \# ys) = inits\ xs \# map\ (xs \#)\ (inits\ ys)$$

$$tails [] = []$$

$$tails [x] = [[x]]$$

$$tails (xs \# ys) = map\ (\#ys)\ (tails\ xs) \# tails\ ys$$

$$[x_1, \dots, x_n] \mathcal{X}_{\oplus} [y_1, \dots, y_m] = \{x_1 \oplus y_1, \dots, x_1 \oplus y_m, \dots, x_n \oplus y_1, \dots, x_n \oplus y_m\}$$

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

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

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

Example

$$(+1) \triangle (\cdot 2) \triangle (-3) = \lambda x \rightarrow (x + 1, x \cdot 2, x - 3)$$

$$(+) \triangle (\cdot) \triangle (-) = \lambda(x, y) \rightarrow (x + y, x \cdot y, x - y)$$

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

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

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

Example

$$(+1) \triangle (\cdot 2) \triangle (-3) = \lambda x \rightarrow (x + 1, x \cdot 2, x - 3)$$

$$(+) \triangle (\cdot) \triangle (-) = \lambda(x, y) \rightarrow (x + y, x \cdot y, x - y)$$

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

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, \dots, h_n be mutually defined as follows:

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

Then

$$\Delta_1^n h_i = \text{hom } (\Delta_1^n \oplus_i) (\Delta_1^n f_i)$$

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

Notation reminder

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

Writing *segs* in the required form

We have

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

and want n mutually recursive functions

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

- *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 *ys*.

Considering *inits* and *tails*

$$\begin{aligned} \text{inits } [] &= [] \\ \text{inits } [x] &= [[x]] \\ \text{inits } (xs \mathbin{++} ys) &= \underline{\text{inits } xs} \mathbin{++} \text{map } (\underline{xs} \mathbin{++}) (\text{inits } ys) \end{aligned}$$

$$\begin{aligned} \text{tails } [] &= [] \\ \text{tails } [x] &= [[x]] \\ \text{tails } (xs \mathbin{++} ys) &= \text{map } (\mathbin{++} \underline{ys}) (\text{tails } xs) \mathbin{++} \underline{\text{tails } ys} \end{aligned}$$

We need to tuple with *id*:

$$\begin{aligned} \text{id } [] &= [] \\ \text{id } [x] &= [x] \\ \text{id } (xs \mathbin{++} ys) &= \text{id } xs \mathbin{++} \text{id } ys \end{aligned}$$

- To summarise, our tupled function will be

$$\text{segs } \triangle \text{ inits } \triangle \text{ tails } \triangle \text{ id}$$

Tupling $segs \triangle inits \triangle tails \triangle id$

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

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

Goal

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

Derivation

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

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

All derivations

Skeleton as a reminder

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

Derivation

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

The almost homomorphism

$$\text{segs} = \pi_1 \circ \text{hom } (\Delta_1^n \oplus_i) (\Delta_1^n f_i)$$

List homomorphisms

Definitions and properties

Homomorphism theorems

Applying the third homomorphism theorem

Deriving efficient almost homomorphisms

Almost homomorphisms

Deriving initial almost homomorphisms by tupling

Fusion theorems for homomorphisms

Fusion of *mss*

Conclusions

Fusion

Reminder of where we are

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

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 $\text{hom}(\oplus) f$ be given. If there exists \otimes such that

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

then

$$h \circ \text{hom}(\oplus) f = \text{hom}(\otimes) (h \circ f)$$

Example of applying fusion theorem

- Suppose $h = (2 \cdot), (\oplus) = (\uparrow), f = \text{abs}$.

(\uparrow is the “max” operator)

- ▶ What does $(2 \cdot) \circ \text{hom}(\uparrow) \text{abs}$ 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 $h = (2 \cdot)$, $(\oplus) = (\uparrow)$, $f = \text{abs}$. (\uparrow is the “max” operator)
 - What does $(2 \cdot) \circ \text{hom } (\uparrow) \text{ abs}$ 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

$$\begin{aligned}(2 \cdot) \circ \text{hom } (\uparrow) \text{ abs} &= \text{hom } (\uparrow) ((2 \cdot) \circ \text{abs}) \\ &= \text{red } (\uparrow) \circ \text{map } ((2 \cdot) \circ \text{abs})\end{aligned}$$

Sadly this is not yet enough

Reminder:

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

where

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

- We want to fuse $segs$ into $(map^s \text{ sum})$
- 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 ($i = 1, \dots, n$) and a function $H = h_1 \times \dots \times h_n$ 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: crossing functions, sometimes called *maps*

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

Example of applying almost fusion theorem

Suppose we want to fuse

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

where

$$\begin{aligned} f_1 x &= x & (x_{\text{mean}}, x_n) \oplus_1 (y_{\text{mean}}, y_n) &= \frac{x_{\text{mean}} \times x_n + y_{\text{mean}} \times y_n}{x_n + y_n} \\ f_2 x &= 1 & (x_{\text{mean}}, x_n) \oplus_2 (y_{\text{mean}}, y_n) &= x_n + y_n \end{aligned}$$

We must define

$$\begin{aligned} h_1 &= \log \\ h_2 &= ? \\ \otimes_1 &= ? \\ \otimes_2 &= ? \end{aligned}$$

such that

$$\begin{aligned} h_1 (x \oplus_1 y) &= (h_1 \times h_2) x \otimes_1 (h_1 \times h_2) y \\ h_2 (x \oplus_2 y) &= (h_1 \times h_2) x \otimes_2 (h_1 \times h_2) y \end{aligned}$$

Finding h_2, \otimes_2

With

$$\begin{aligned}(x_{mean}, x_n) \otimes_2 (y_{mean}, y_n) &= x_n + y_n \\ h_2 x &= x\end{aligned}$$

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

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

Applying almost fusion

So we can fuse

$$\log \circ \pi_1 \circ \text{hom} (\Delta_1^2 \oplus i) (\Delta_1^2 f_i)$$

to

$$\pi_1 \circ \text{hom} (\Delta_1^2 \otimes i) (\Delta_1^2 (h_i \circ f_i))$$

where

$$h_1 x = \log x$$

$$h_2 x = x$$

$$(x_{\text{mean}}, x_n) \otimes_1 (y_{\text{mean}}, y_n) = \log \left(\frac{\exp(x_{\text{mean}}) \times x_n + \exp(y_{\text{mean}}) \times y_n}{x_n + y_n} \right)$$

$$(x_{\text{mean}}, x_n) \otimes_2 (y_{\text{mean}}, y_n) = x_n + y_n$$

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

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

$$h_j ((s_1, i_1, t_1, d_1) \oplus_j (s_2, i_2, t_2, d_2)) = \\ (h_1 s_1, h_2 i_1, h_3 t_1, h_4 d_1) \otimes_j (h_1 s_2, h_2 i_2, h_3 t_2, h_4 d_2)$$

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

- **Remember!** We start out knowing $h_1 = map^s \text{ sum}$.
- As we determine more h s we can solve more equations.

Example for $j = 1$

Equation

$$\begin{aligned} \text{map}^s \text{sum} ((s_1, i_1, t_1, d_1) \oplus_1 (s_2, i_2, t_2, d_2)) = \\ (\text{map}^s \text{sum } s_1, h_2 \ i_1, h_3 \ t_1, h_4 \ d_1) \otimes_1 (\text{map}^s \text{sum } s_2, h_2 \ i_2, h_3 \ t_2, h_4 \ d_2) \end{aligned}$$

Calculation

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

Matching last expression with RHS of the equation gives

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

Example for $j = 2$

Equation

$$\begin{aligned} \text{map sum } ((s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2)) = \\ (\text{map}^s \text{ sum } s_1, h_2 \ i_1, h_3 \ t_1, h_4 \ d_1) \otimes_2 (\text{map}^s \text{ sum } s_2, h_2 \ i_2, h_3 \ t_2, h_4 \ d_2) \end{aligned}$$

Calculation

$$\begin{aligned} & \text{map sum } ((s_1, i_1, t_1, d_1) \oplus_2 (s_2, i_2, t_2, d_2)) \\ &= \text{map sum } (i_1 \# \text{map } (d_1 \#) \ i_2) \\ &= \text{map sum } i_1 \# \text{map sum } (\text{map } (d_1 \#) \ i_2) \\ &= \text{map sum } i_1 \# \text{map } (\text{sum} \circ (d_1 \#)) \ i_2 \\ &= \text{map sum } i_1 \# \text{map } (\text{sum } d_1 \#) \ (\text{map sum } i_2) \end{aligned}$$

Matching last expression with RHS of the equation gives

$$\begin{aligned} h_4 &= \text{sum} \\ (s_1, i_1, t_1, d_1) \otimes_2 (s_2, i_2, t_2, d_2) &= i_1 \# \text{map } (d_1 \#) \ i_2 \end{aligned}$$

If we continue

$$h_1 = \text{map}^s \text{ sum}$$

$$h_2 = \text{map sum}$$

$$h_3 = \text{map sum}$$

$$h_4 = \text{sum}$$

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

$$(s_1, i_1, t_1, d_1) \otimes_2 (s_2, i_2, t_2, d_2) = i_1 \uplus \text{map } (d_1 +) i_2$$

$$(s_1, i_1, t_1, d_1) \otimes_3 (s_2, i_2, t_2, d_2) = \text{map } (+d_2) t_1 \uplus t_2$$

$$(s_1, i_1, t_1, d_1) \otimes_4 (s_2, i_2, t_2, d_2) = d_1 + d_2$$

The f parts

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

$$f_1 x = \{[x]\}$$

$$f_2 x = [[x]]$$

$$f_3 x = [[x]]$$

$$f_4 x = [x]$$

$$f'_1 x = (h_1 \circ f_1) x = ((map^s sum) \circ f_1) x = \{x\}$$

$$f'_2 x = (h_2 \circ f_2) x = ((map sum) \circ f_2) x = [x]$$

$$f'_3 x = (h_3 \circ f_3) x = ((map sum) \circ f_3) x = [x]$$

$$f'_4 x = (h_4 \circ f_4) x = (sum \circ f_4) x = x$$

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

$$(s_1, i_1, t_1, d_1) \otimes'_1 (s_2, i_2, t_2, d_2) = s_1 \uparrow s_2 \uparrow (t_1 + i_2)$$

$$(s_1, i_1, t_1, d_1) \otimes'_2 (s_2, i_2, t_2, d_2) = i_1 \uparrow (d_1 + i_2)$$

$$(s_1, i_1, t_1, d_1) \otimes'_3 (s_2, i_2, t_2, d_2) = (t_1 + d_2) \uparrow t_2$$

$$(s_1, i_1, t_1, d_1) \otimes'_4 (s_2, i_2, t_2, d_2) = d_1 + d_2$$

And you get to experience this in the assignment!

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.

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!)