“Explosive” Programming Controlled by Calculation

AFP’98 — 3rd Summer School on Advanced Functional Programming
12–18 September 1998
Braga
Portugal

J. Nuno Oliveira
Departamento de Informática
Universidade do Minho
Braga
Portugal

email: jno@di.uminho.pt
http://www.di.uminho.pt/~jno

Context

Formal Methods

- Formal specification — yes
- Verification — no!
- **FM**-typical “invent & verify”? — no, transform & **calculate**!

Role of Functional Programming in FMs

- Specification **animation**
- Rapid prototyping and **technology** embedding
- (Categorial) **calculation**

However,

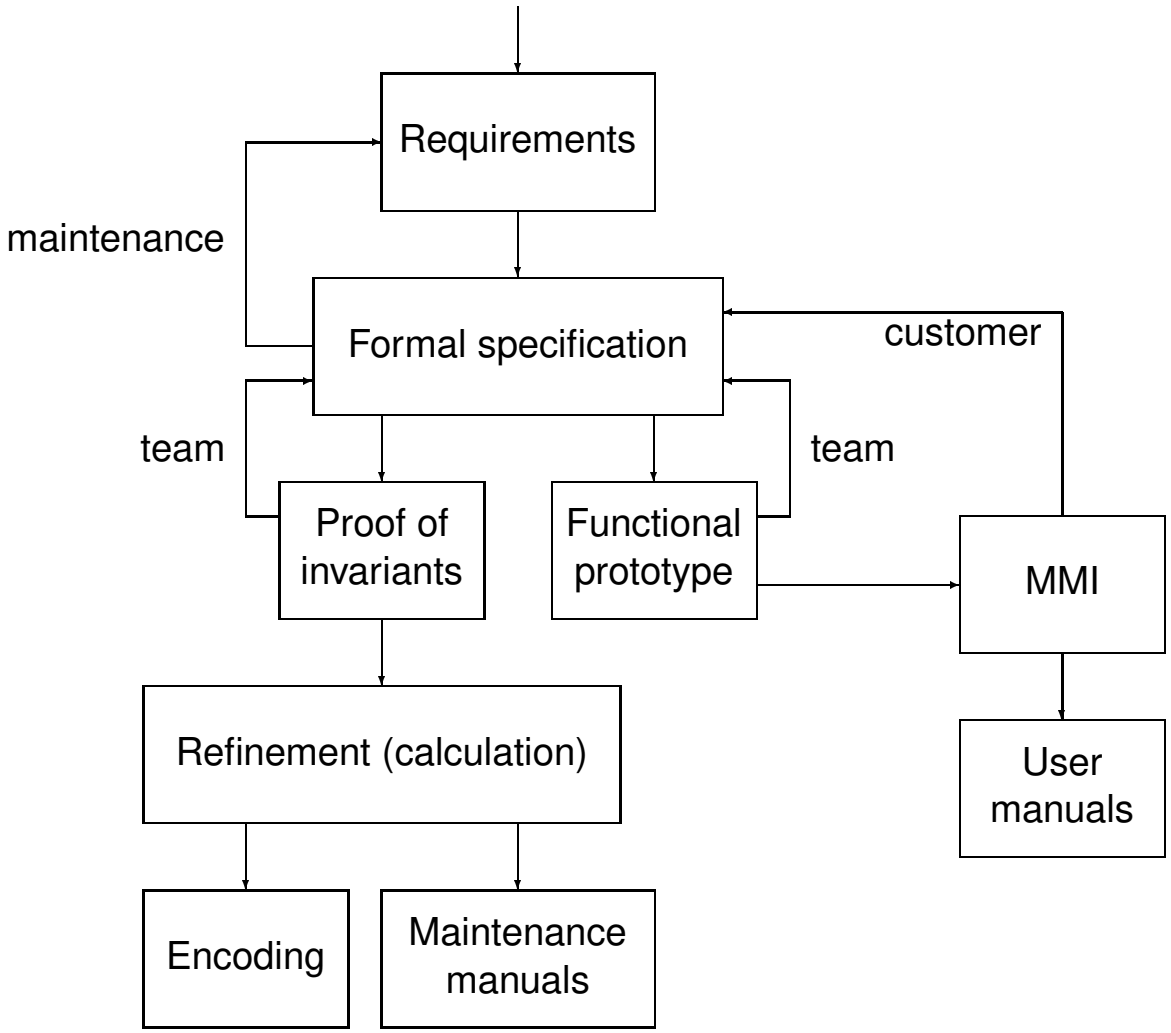
Functional programming as a **means** rather than an **end** in itself.

FMs at UM

Functional animation of formal specs has been **extremely** helpful in the survival of **FM**s at UM:

- Design of the CAMILA language (back to 1980’s: it started as a VDM animator).
- Role in teaching (**FM**s are optional . . .) and in industrial contracts.
- Emphasis on calculations.

Life-cycle



Pragmatics...

Need for context-sensitive “textual abstraction” function:

$$f\left(\begin{array}{l}\text{Our technology is based on } \textit{formal} \text{ methods which rely on a } \textit{functional} \text{ programming discipline which is } \textit{categorially} \text{ sound.}\end{array}\right) = \begin{array}{l}\text{Our technology is based on } \ldots \text{ methods which rely on a } \ldots \text{ programming discipline which is } \ldots \text{ sound.}\end{array}$$

Example

In the **SOUR** contract (EU 379, © Olivetti Italy):

- Specs led to about 4 200 lines of CAMILA (*cf.* 31 400 lines of Visual Basic/C++).
- (Not everything was calculated.)
- Braga team: 2 lecturers + 2 final year students

Motivation

- Sometime ago: our students of the **FM** course were asked to design a **data mining** library, to become available in ORACLE technology.

Good to get them a job ($) and hide the **FM** disease . . .

- A group assigned to the *bill of materials* functionality produced a fairly compact ORACLE implementation of *part explosion* . . .
- . . . but their derivation included false steps (!).
- Sample of how cumbersome **pointwise** reasoning may happen to become in software design.
- Good programming intuition, anyway.

By reworking their calculation — this time in the **pointfree** style — we became aware of how close they were to other groups calculating seemingly disparate functionalities.

Need to redesign the library in a **polytypic** style.

Main points

- In many situations, **data mining** amounts to extracting knowledge from a complex data structure, to be **accumulated** via a binary **operator** which provides for the intended abstraction (academic example: summing up the elements of a list).
- The algebraic structure which underlies this kind of operation is the ubiquitous **monoid**.
- We concentrate on monoidal reductions within a particular class of *general **hierarchical** systems*.
- This arises as a **base-functor** construction.
- We describe how (polytypic) "**flatten**" evolves towards "**explosion**", once one more parameter is added to the hierarchical construction.
- We will witness a smooth move from **monoids** to (almost) **vector spaces** as target reduction algebras.

A hierarchical datatype construction

Why do we say that the UNIX or WINDOWS file system is **hierarchical**?

Recall old file systems such as CPM or many others — these were flat structures mapping file **names** to file **attributes**:

$$I \rightharpoonup File \tag{1}$$

NB: Read "$A \rightharpoonup B$" as the datatype of *finite partial functions*.

In the Haskell illustrations we will write `Ffun a b` for $a \rightharpoonup b$, assuming

```
> type Ffun a b = [(a,b)]
```

subject to the obvious datatype **invariant**. □

In modelling hierarchical file systems, we have to upgrade (1) to something like

$$I \rightharpoonup (File + Directory)$$

where $Directory$ is again a $I \rightharpoonup (File + Directory)$ and so on.

So we obtain a recursive datatype of shape

$$X \;=\; I \rightharpoonup (File + X) \tag{2}$$

Another example: generalized lists,

```
> data GList a  = In [ Either a (GList a) ]
```

— *cf.* equation

$$X \;=\; (a + X)^\star \tag{3}$$

regarded another hierarchical datatype, this time extending $a^\star$.

Generic construction:

$$\underbrace{\mathsf{F}A}_{\text{type functor}} \quad \text{used in} \quad \overbrace{X = \underbrace{\mathsf{F}(A + X)}_{\text{base functor}}}^{\mathsf{H}A}$$

$\mathsf{H}A$ = hierarchical extension of $\mathsf{F}A$.

Taking $\mathsf{F}A$'s base functor into account, we have

$$\mathsf{F}A \;=\; \underbrace{\mathsf{B}(A, \mathsf{F}A)}_{\text{base functor of } \mathsf{F}}$$

and then define

$$\begin{aligned} \mathsf{H}A &= \mathsf{F}(A + \mathsf{H}A) \\ &= \mathsf{B}(A + \mathsf{H}A, \mathsf{F}(A + \mathsf{H}A)) \\ &= \underbrace{\mathsf{B}(A + \mathsf{H}A, \mathsf{H}A)}_{\mathsf{B}'(A, \mathsf{H}A)} \end{aligned}$$

In the $GList$ example,

$$\begin{aligned} \mathsf{F}A &= A^\star \\ \mathsf{B}(A, X) &= 1 + A \times X \\ \mathsf{B}'(A, X) &= 1 + (A + X) \times X \end{aligned}$$

Adding one more parameter:

This is the case of interest.

An example:

$$\begin{aligned} \mathsf{F}(A,B) &= (A \times B)^\star \\ \mathsf{B}(A,B,X) &= 1 + (A \times B) \times X \\ \mathsf{B}'(A,B,X) &= 1 + (A \times B + X) \times X \end{aligned}$$

that is, equation

$$X = ((A + X) \times B)^\star \tag{4}$$

cf.

```
>  data H a b  = In [ (Either a (H a b), b) ]
```

Is this datatype useful? Think of the following interpretation:

- the Bs as natural numbers.
- $(A \times \mathbb{N})^\star$ as a model of lists of votes (elections within some organization)
- Extension (4) capturing the fact that organization is hierarchically structured.
- $(i_1\, a, n)$ — *i.e.* `(Left a, n)` — meaning *candidate* a *has obtained* n *votes*.
- $(i_2\, x, n)$ — *i.e.* `(Right x, n)` — meaning n *is the* ***weight*** *of the votes to be found in* x, *e.g.* lecturer votes are twice as strong as those of students, *etc.*

Further examples

Hierarchical file systems fall into this class, for $\mathsf{F}(A,B) = B \rightharpoonup A$, *cf.* equation

$$X = B \rightharpoonup (A + X)$$

and

```
> data H a b  = In (Ffun b (Either a (H a b)))
```

and so do **production trees** in the *bill of materials* problem, *cf.*

$$X \;=\; (A + X) \rightharpoonup B$$

that is

```
> data H a b  = In (Ffun (Either a (H a b)) b)
```

More examples could be provided — **budgeting**, **sub-contracting**, *etc.*

From $flatten$ to $explode$

What is the $A^\star$ instantiation of the polytypic function $flatten$? It should be the **identity** function, because lists are "already flat".

What about its hierarchical derivation, H $= GList$?

$$\begin{array}{rcl} flatten & : & \mathsf{H}\, A \longrightarrow A^\star \\ flatten\, l & \stackrel{\text{def}}{=} & \mathbin{+\!\!+}_{x \leftarrow l} \left\{ \begin{array}{lcl} (x = i_1\, a) & \Rightarrow & [a] \\ (x = i_2\, l') & \Rightarrow & flatten\, l' \end{array} \right. \end{array} \tag{5}$$

where $\mathbin{+\!\!+}_{x \leftarrow l} \ldots$ indicates the iteration of binary list-concatenation ($x +\!\!+ y$) to a sequence of lists, that is,

$$\begin{array}{rcl} \mathbin{+\!\!+} [\,] & = & [\,] \\ \mathbin{+\!\!+}\, cons(x, l) & = & x +\!\!+ (\mathbin{+\!\!+}\, l) \end{array} \tag{6}$$

Or, if you like:

```
> flatten :: H a -> [a]
> flatten (In []) = []
> flatten (In (Left a:l)) = wrap a ++ flatten (In l)
> flatten (In (Right l':l)) =
>                       flatten l' ++ flatten (In l)
```

where $wrap\, a = [a]$.

Pointfree notation is by far the most economic:

$$flatten \circ [\ nil, cons\ ] \ = \ [\ nil, +\!\!+ \circ ([\ wrap, flatten\ ] \times flatten)\ ]$$

where nil is constant function $\underline{[\ ]}$:

$$\begin{array}{ccccccc}
\mathsf{H}\,A & \xleftarrow{\delta} & (A + \mathsf{H}\,A) & \xleftarrow{[nil,cons]} & 1 + (A + \mathsf{H}\,A) \times (A + \mathsf{H}\,A) & \xleftarrow{1+id\times\delta^{-1}} & 1 + (A + \mathsf{H}\,A) \times \mathsf{H}\,A \\
\downarrow{\scriptstyle flatten} & & & & & & \downarrow{\scriptstyle 1+(A+flatten)\times flatten} \\
A^\star & \xleftarrow[[\ nil,+\!\!+\ ]]{} & 1 + A^\star \times A^\star & & \xleftarrow[1+[\ wrap,id\ ]\times id]{} & & 1 + (A + A^\star) \times A^\star
\end{array}$$

Or, if you like,

- $flatten$ is catamorphism $(\!|[\ nil, +\!\!+ \circ ([\ wrap, id\ ] \times id)\ ]|\!)$
- $flatten$ is hylomorphism $[\!|[\ nil, +\!\!+\ ] \circ \nu, \omega|\!]$,

$$\begin{array}{ccc}
\mathsf{H}\,A & \xrightarrow{\omega} & 1 + (A + \mathsf{H}\,A) \times \mathsf{H}\,A \\
f\downarrow & & \downarrow{\scriptstyle 1+(A+f)\times f} \\
A^\star & \xleftarrow[[\ nil,+\!\!+\ ]]{} 1 + A^\star \times A^\star & \xleftarrow{\nu} 1 + (A + A^\star) \times A^\star
\end{array}$$

where $\omega = [\ nil, cons\ ]^{-1}$ and $\nu = 1 + [\ wrap, id\ ] \times id$.

So we have

$$\mathsf{H}\,A \underset{lift = (i_1)^\star \text{ (\textit{i.e.}, extend, expand)}}{\overset{flatten \text{ (\textit{i.e.}, compress, reduce)}}{\rightleftarrows}} A^\star$$

$flatten$ is an instance of the generic functional pattern which captures **monoidal reductions**, which are very typical "data-mining" operations.

A monoidal reduction is a hylomorphism whose target algebra is a **monoid** M,

$$\begin{array}{ccc}
T & \xrightarrow{\beta} & \mathsf{F}\,T \\
f\downarrow & & \downarrow{\scriptstyle \mathsf{F}\,f} \\
M & \xleftarrow[[\ \underline{\epsilon},\theta\ ]]{} 1 + M \times M & \xleftarrow{\nu} \mathsf{F}\,M
\end{array}$$

(ϵ is the unit; θ is a binary associative operator.)

Question:

Are monoids enough for this purpose? No, as we shall see.

Vote lists

Let us speculate about what should happen to $flatten$ in case we use bifunctor $\mathsf{F}(A,B) = (A \times B)^\star$ — recall "vote lists":

$$X \;=\; ((A + X) \times B)^\star$$

Proposal:

$$\begin{array}{rcl} flatten\,[\,] & = & [\,] \\ flatten\,(cons(x,l)) & = & \left\{ \begin{array}{lcl} x = (i_1\,a, b) & \Rightarrow & [(a,b)] \\ x = (i_2\,l', b) & \Rightarrow & b \otimes flatten\,l' \end{array} \right. +\!\!+\; flatten\,l \end{array}$$

where

$$b \otimes l \stackrel{\text{def}}{=} [(a, b \times n) \mid (a,b) \leftarrow l]$$

Prototype:

```
> flatten :: Num b => H a b -> [(a,b)]
> flatten (In []) = []
> flatten (In ((Left a, b):l)) =
>              wrap a b ++ flatten (In l)
> flatten (In ((Right l',b):l)) =
>              otimes b (flatten l') ++ flatten (In l)
```

where

```
> wrap a b = [(a,b)]
```

and

```
> otimes b l = [ (a,b*n) | (a,n) <- l ]
```

Animation:

```
> GBList> flatten x
> [("John",5), ("Mary",6), ("James",4)]
> GBList>
```

for

```
> x = In [ (Left "John", 5),
>          (Right (In [
>                              (Left "Mary", 3)
```

```
>                         ]),2),
>              (Left "James", 4)
>          ]
```

This code is justified by hylomorphism diagram

$$\begin{array}{ccc} \mathsf{H}(A,B) & \xrightarrow{\omega=[\ nil,cons\ ]^{-1}} & \mathsf{J}\,\mathsf{H}(A,B) \\ \downarrow{\scriptstyle flatten} & & \downarrow{\scriptstyle \mathsf{J}\,flatten} \\ (A\times B)^\star & \xleftarrow[[\ nil,+\!\!+\ ]]{} 1+(A\times B)^\star\times(A\times B)^\star & \xleftarrow{\nu} \mathsf{J}\,(A\times B)^\star \end{array} \tag{7}$$

where $\mathsf{J}\,X = 1+((A+X)\times B)\times X$.

About arrow $\nu = 1+[\ wrap, \otimes\circ swap\ ]\circ distl\times id$:

$$\begin{array}{c} 1+((A+(A\times B)^\star)\times B)\times(A\times B)^\star \\ \downarrow{\scriptstyle 1+distl\times id} \\ 1+(A\times B+(A\times B)^\star\times B)\times(A\times B)^\star \\ \downarrow{\scriptstyle 1+(id+swap)\times id} \\ 1+(A\times B+B\times(A\times B)^\star)\times(A\times B)^\star \\ \downarrow{\scriptstyle 1+[\ wrap,\otimes\ ]\times id} \\ 1+(A\times B)^\star\times(A\times B)^\star \end{array}$$

($distl$ means “distribute left”).

Question:

What’s new in the 2-parameter case? — the addition of operator $\otimes$ to the target monoidal structure $M=(A\times B)^\star, \epsilon=[\ ]$ and $\theta=+\!\!+$.

Note that operator $\otimes$ is **external** to the monoid — it bears functionality $\otimes : B\times M \longrightarrow M$.

A better (generic) definition of $\otimes$ based on the *strength* ϕ of the list functor,

$$B\times(A\times B)^\star \xrightarrow{swap} (A\times B)^\star\times B \xrightarrow{\phi} ((A\times B)\times B)^\star \tag{8}$$

followed by

$$((A\times B)\times B)^\star \xrightarrow{assocr^\star} (A\times(B\times B))^\star \xrightarrow{(id\times(\times))^\star} (A\times B)^\star \tag{9}$$

where

$$(\times)\ :\ B\times B\longrightarrow B \tag{10}$$

and $swap$, $assocr$ *etc.* denote the same functions as in Bird-Moor's **Algebra of Programming** textbook.

Bill of materials

Recall

$$X \;=\; (A + X) \rightharpoonup B$$

for $B = \mathbb{N}$.

The specification of $flatten$ becomes

$$\begin{array}{rcl} flatten & : & \mathsf{H}(A,B) \longrightarrow (A \rightharpoonup B) \\ flatten\,\sigma & \stackrel{\text{def}}{=} & \bigoplus_{x \in dom\,\sigma} \left\{ \begin{array}{lcl} (x = i_1\,a) & \Rightarrow & \left(\begin{array}{c} a \\ \sigma\,x \end{array} \right) \\ (x = i_2\,\sigma') & \Rightarrow & (\sigma\,x) \otimes flatten\,\sigma' \end{array} \right. \end{array} \tag{11}$$

where

$$v \oplus r \;=\; v \dagger r \dagger \left(\begin{array}{c} a \\ v\,a + r\,a \end{array} \right)_{a \in (dom\,v) \cap (dom\,r)}$$

is **multiset** union, and

$$n \otimes v \;=\; \underbrace{v \oplus \cdots \ldots \oplus v}_{n \text{ times}}$$

NB: $v \dagger r$ means the *overwriting* of map v by map r; $dom\,v$ means the *domain* of v). □

Prototype:

```
> flatten :: (Num b, Eq a)  => H a b -> Ffun a b
> flatten (In sigma) =
>         oPlus([ case x of
>                       Left  a -> wrap a (ap sigma x)
>                       Right b -> otimes (ap sigma x)
>                                         (flatten b)
>               | x <- dom sigma ])
```

Animation: for

```
> x = In [
>          Left "a" |-> 1,
>          Left "b" |-> 2,
>          Right (In [
>                       Left "a" |-> 3,
>                       Left "c" |-> 1
>                     ]) |-> 3
>        ]
```

we expect

```
> BillOfMaterials> flatten x
> [("b",2), ("c",3), ("a",10)]
> BillOfMaterials>
```

to happen.

Clearly, $flatten$ instantiates to **part explosion**.

NB:

- $A \rightharpoonup \mathbb{N}$ is a model of multisets.
- If we identify A with the dimensions of a **vector space**, then each $v \in A \rightharpoonup \mathbb{N}$ may be understood as a vector.
- For instance, 3-dimensional vector $v = 2x+3y+z$ described by multiset $m = \begin{pmatrix} x & y & z \\ 2 & 3 & 1 \end{pmatrix}$.
- Under the analogy, $\oplus$ becomes **vectorial sum** and $\otimes$ becomes **scalar** (external) multiplication.
- In fact, later calculations will require $\otimes$ to distribute over $\oplus$, *etc.*

□

What we have assumed:

```
> x |-> y = (x,y)                -- syntactic sugar
>
> wrap a b = [ a |-> b ]
>
```

```
> oplus m n = monff m n (uncurry (+))
>
> otimes b l = [ (a,b*n) | (a,n) <- l ]
```

plus a little finite-function **library** behind this.

File system

Recall

$$X \;=\; B \rightharpoonup (A + X)$$

Proposal:

$$\begin{array}{rcl} flatten & : & \mathsf{H}(A,B) \longrightarrow (B \rightharpoonup A) \\ flatten\,\sigma & \stackrel{\text{def}}{=} & \bigcup_{b \in dom\,\sigma} \; let \quad x = \sigma\, b \\ & & \qquad\quad in \quad \left\{ \begin{array}{lcl} (x = i_1\, a) & \Rightarrow & \left(\begin{array}{c} b \\ a \end{array} \right) \\ (x = i_2\, \sigma') & \Rightarrow & b \otimes flatten\,\sigma' \end{array} \right. \end{array}$$

Thinking of $B = String$, why not

$$b \otimes \sigma \;=\; ((\lambda s.b +\!\!+ \texttt{"/"} +\!\!+ s) \rightharpoonup A)\sigma \;?$$

Clearly, $flatten$ instantiates to UNIX `tar` command.

Prototype:

```
> wrap a b = [ a |-> b ]
>
> oplus x = plus x
>
> otimes b l = [ (b++"/"++s,a) | (s,a) <- l ]
>
> oPlus x = foldl (oplus) bottom x
>
> flatten :: Eq a => H a [Char] -> [([Char],a)]
> flatten (In sigma) =
```

```
>          oPlus([ case ap sigma b of
>                       Left  a -> wrap b a
>                       Right x -> otimes b (flatten x)
>                | b <- dom sigma ])
```

Animation: for

```
> x = In
>     [
>      "usr" |-> Right (In
>         [
>                 "local" |-> Right (In
>                      [
>                              "lib" |-> Right (In
>                                       [
>                                        "hugs" |-> Right (In
>                                                   [
>                                                    "lib" |-> Right (In
>                                                              [
>                                                               "Prelude.hs" |-> Left "-rw-r--r--
>
>                                                              ])
>                                                   ])
>                                       ])
>             ])
>  ]),
>      "home" |-> Right (In
>                 [
>                 "jno" |-> Right (In
>                           [
>                            "work" |-> Right (In
>                                      [
>                                       "explode.hs" |-> Left "-rw-r--r--   1 jno      users
>                                      ])
>                           ])
>
>                 ])
>     ]
```

we get the “ `tar` of x”:

```
> FileSystem> flatten x
> [("usr/local/lib/hugs/lib/Prelude.hs","-rw-r--r--   1
> root     root        52137 Oct 17  1997"),
> ("home/jno/work/explode.hs","-rw-r--r--   1 jno
>  users        1292 Sep 01 10:26")]
> FileSystem>
```

Putting things together

We are ready to present the **generic** function of which three instantiations have been considered.

Pre-requisites:

1. A "flat" two-parameter datatype $\mathsf{F}(A, B)$ defined by linear domain equation $X = 1 + (A \times B) \times X$.
2. Its hierarchical extension, $\mathsf{H}(A, B)$, defined by domain equation

$$X = \mathsf{F}((A + X), B)$$

3. A "wrap up" arrow $\mathsf{F}(A, B) \xleftarrow{wrap} A \times B$.
4. A target monoid algebra whose carrier is the flat type,

$$\mathsf{F}(A, B) \xleftarrow{[\ \underline{\epsilon}, \theta\ ]} 1 + \mathsf{F}(A, B) \times \mathsf{F}(A, B)$$

enriched with a "scalar multiplication" operator,

$$\mathsf{F}(A, B) \xleftarrow{\otimes} B \times \mathsf{F}(A, B)$$

which extends naturally to $\mathsf{H}(A, B)$.

We have been interested in **hylomorphism** $[\![[\ \underline{\epsilon}, \theta\ ] \circ \nu, \beta]\!]$ depicted by

$$\begin{array}{ccccc} \mathsf{H}(A,B) & & \xrightarrow{\quad\quad\beta\quad\quad} & & \mathsf{J}\,\mathsf{H}(A,B) \\ \Big\downarrow f & & & & \Big\downarrow \mathsf{J}\,f \\ \mathsf{F}(A,B) & \xleftarrow[[\ \underline{\epsilon}, \theta\ ]]{} & 1 + \mathsf{F}(A,B) \times \mathsf{F}(A,B) & \xleftarrow[\nu]{} & \mathsf{J}\,\mathsf{F}(A,B) \end{array} \tag{12}$$

where

$$\mathsf{J}\,X = 1 + ((A + X) \times B) \times X$$

β is the appropriate input "parser" and

$$\nu = 1 + [\ wrap, \otimes \circ swap\ ] \circ distl \times id$$

Deriving a linear implementation

Recall that we want $flatten$ to run on top of ORACLE.

Our purpose is to try and find an arrow ν' able to provide a linear recursive path alternative to $\nu \circ \mathsf{J}\, f$ in diagram (12):

$$\begin{array}{ccccc}
\mathsf{H}(A,B) & \xrightarrow{\quad\beta\quad} & 1+\mathsf{F}(A,B)\times\mathsf{H}(A,B) & \xleftarrow[\nu']{} & \mathsf{J}\,\mathsf{H}(A,B) \\
\downarrow{\scriptstyle f} & & \downarrow{\scriptstyle 1+\mathsf{F}(A,B)\times f} & & \downarrow{\scriptstyle \mathsf{J}\,f} \\
\mathsf{F}(A,B) & \xleftarrow[[\,\underline{\epsilon},\theta\,]]{} & 1+\mathsf{F}(A,B)\times\mathsf{F}(A,B) & \xleftarrow{\nu} & \mathsf{J}\,\mathsf{F}(A,B)
\end{array}$$

This is a state-of-the-art calculation, assuming natural isomorphisms

$$A \times B + C \times B \xleftarrow{distl} (A + C) \times B$$

and

$$B \times A \xleftarrow{swap} (A \times B$$

and fact

$$[\,g, h\,] \times f \;=\; [\,g \times f, h \times f\,] \circ distl \tag{13}$$

Important

Vital role of categorial foundations, namely the universal constructions packaged with their **little** but **powerful** theory, consisting of

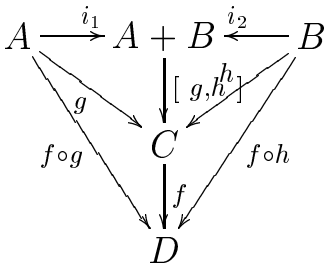
- a **fusion** law
- a **reflection** law
- an **absorption** law
- a **cancellation** law

+ natural transformations (Wadler's *theorems for free!*), *etc.*

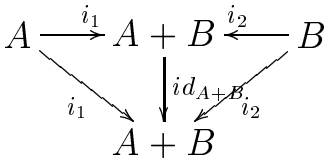
Example

Coproduct — the `Either a b , either f g` datatype:

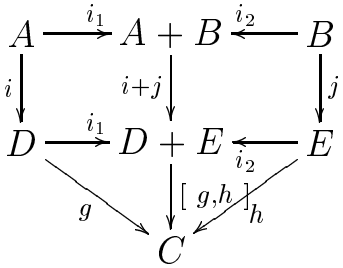
Coproduct fusion : $$f \circ [\ g, h\ ] = [\ f \circ g, f \circ h\ ] \quad (14)$$



Coproduct reflection : $$[\ i_1, i_2\ ] = id_{A+B} \quad (15)$$



Coproduct absorption : $$[\ g, h\ ] \circ (i + j) = [\ g \circ i, h \circ j\ ] \quad (16)$$



Coproduct cancellation : $$[\ g, h\ ] \circ i_1 = g\ ,\ [\ g, h\ ] \circ i_2 = h \quad (17)$$

Detailed reasoning is as follows:

$$
\begin{array}{lcl}
f & = & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap, \otimes \circ swap\ ] \circ distl \times id) \circ (1 + ((id + f) \times id) \times f) \circ \beta \\
& = & \quad \{\ \text{bifunctors } + \text{ and } \times\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap, \otimes \circ swap\ ] \circ distl \circ ((id + f) \times id) \times f) \circ \beta \\
& = & \quad \{\ distl \text{ is natural}\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap, \otimes \circ swap\ ] \circ (id + f \times id) \circ distl \times f) \circ \beta \\
& = & \quad \{\ +\text{-absorption and } swap \text{ is natural}\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap, \otimes \circ (id \times f) \circ swap\ ] \circ distl \times f) \circ \beta \\
& = & \quad \{\ \text{fact (19) below and identity of composition}\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap, f \circ \otimes \circ swap\ ] \circ distl \times f \circ id) \circ \beta \\
& = & \quad \{\ \text{bifunctor } \times\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + ([\ wrap, f \circ \otimes \circ swap\ ] \times f) \circ (distl \times id)) \circ \beta \\
& = & \quad \{\ \text{fact (13)}\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap \times f, f \circ \otimes \circ swap \times f\ ] \circ distl \circ (distl \times id)) \circ \beta \\
& = & \quad \{\ \text{identity of composition and bifunctor } \times\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap \times f, (f \times f) \circ (\otimes \circ swap \times id)\ ] \circ distl \circ (distl \times id)) \circ \beta \\
& = & \quad \{\ \text{bifunctor } +\ \} \\
& & [\ \underline{\epsilon}, \theta\ ] \circ (1 + [\ wrap \times f, (f \times f) \circ (\otimes \circ swap \times id)\ ]) \circ (1 + distl \circ (distl \times id)) \circ \\
& = & \quad \{\ +\text{-absorption, } +\text{-fusion and fact (20) below}\ \} \\
& & [\ \underline{\epsilon}, [\ \theta \circ (wrap \times f), \underbrace{f \circ \theta \circ (\otimes\ \circ swap \times id)}_{\rho}\ ]\ ] \circ \underbrace{(1 + distl \circ (distl \times id)) \circ \beta}_{\beta'} \\
& = & \quad \{\ \text{introducing abbreviations } \rho \text{ and } \beta'\ \} \\
& & [\ \underline{\epsilon}, [\ \theta \circ (wrap \times f), \rho\ ]\ ] \circ \beta' \\
& = & \quad \{\ \text{forcing } \epsilon \text{ in, which is the unit of } \theta\ \} \\
& & [\ \underline{\epsilon}, [\ \theta \circ (wrap \times f), \theta \circ \langle \underline{\epsilon}, \rho \rangle\ ]\ ] \circ \beta' \\
& = & \quad \{\ \text{reverse } +\text{-fusion followed by exchange law}\ \} \\
& & [\ \underline{\epsilon}, \theta \circ \langle [\ wrap \circ \pi_1, \underline{\epsilon}\ ], [\ f \circ \pi_2, \rho\ ] \rangle\ ] \circ \beta'
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \text{ expansion of } \rho \text{ followed by reverse } + \text{-fusion } \} \\
& [\ \underline{\epsilon}, \theta\ ] \circ (1 + \langle [\ wrap \circ \pi_1, \underline{\epsilon}\ ], f \circ [\ \pi_2, \theta \circ (\otimes \circ swap \times id)\ ] \rangle) \circ \beta' \\
= & \{ \text{ reverse } \times \text{-absorption and bifunctor } + \} \\
& [\ \underline{\epsilon}, \theta\ ] \circ (1 + id \times f) \circ (1 + \langle [\ wrap \circ \pi_1, \underline{\epsilon}\ ], [\ \pi_2, \theta \circ (\otimes \circ swap \times id)\ ] \rangle) \circ \beta' \\
= & \{ \text{ expanding } \beta' \text{ and introducing } \nu' \} \\
& [\ \underline{\epsilon}, \theta\ ] \circ (1 + id \times f) \circ \nu' \circ \beta
\end{array}
$$

From this reasoning we extract

$$\nu' \ = \ (1 + \langle [\ wrap \circ \pi_1, \underline{\epsilon}\ ], [\ \pi_2, \theta \circ (\otimes \circ swap \times id)\ ] \rangle) \circ (1 + distl \circ (distl \times id))$$

which is equivalent to

$$\nu' \ = \ 1 + \langle [\ wrap \circ \pi_1 \circ distl, \underline{\epsilon}\ ], \theta \circ [\ \underline{\epsilon}, \otimes \circ swap\ ] \times id \rangle \circ (distl \times id) \quad (18)$$

Important

Two facts required above,

$$f \circ \otimes \ = \ \otimes \circ (id \times f) \quad (19)$$

$$f \circ \theta \ = \ \theta \circ (f \times f) \quad (20)$$

can be derived provided properties

$$b \otimes (x\ \theta\ y) \ = \ (b \otimes x)\ \theta\ (b \otimes y) \quad (21)$$

and

$$(n \star b) \otimes x \ = \ n \otimes (b \otimes x) \quad (22)$$

hold, where

$$B \xleftarrow{\star} B \times B$$

is the *internal multiplication* operator which underlies the generic definition of scalar multiplication:

$$B \times \mathsf{F}(A, B) \xrightarrow{swap} \mathsf{F}(A, B) \times B \xrightarrow{\varphi} \mathsf{F}(A, B \times B) \xrightarrow{\mathsf{F}(id, \star)} \mathsf{F}(A, B) \quad (23)$$

"Vector spaces" are required by $flatten$

- Facts (21) and (22) can be found in the axiomatization of *vector spaces* in Universal Algebra compendia.
- Adopting the full axiomatization would turn B into a **field** (of scalars) and $\mathsf{F}(A, B)$ into a commutative **group** (of vectors).
- Another operation would become available, that of **scalar addition** (say $+$), which, despite being absent from our reasoning, would make sense in our context, including its axiom

$$(n + m) \otimes x \;=\; (n \otimes x)\, \theta\, (m \otimes x) \tag{24}$$

- In the *bill of materials* context, $+$ would be natural number addition, while in the *file system* context it would bring us into **regular expression** algebra.
- However, hard to imagine scalar **inverses** and cancellations in the *file system* context.

Going imperative

Knowing that ϵ, θ form a monoid makes it possible to convert the **linear** version of f we have derived into a `while`-loop, via the technique of ***accumulation* parameter** introduction:

We omit the details, but present the outcome of the reasoning leading to an iterative version of *explode* in the *bill of materials* context.

Prototype:

```
> flatten' :: (Num b, Eq a) =>
>              H a b -> Ffun a b -> Ffun a b
> flatten' (In sigma) r =
>    if sigma == bottom
>    then r
>    else let ((x,n),ff) = get sigma
>         in flatten' (In (oplus ff
>                                (case x of
```

```
>                                    Left  a -> bottom
>                                    Right b ->
>                                        otimes n (out b))))
>                           (oplus r
>                                  (case x of
>                                    Left  a -> wrap a n
>                                    Right b -> bottom))
```

Animation:

```
> BillOfMaterials> flatten' x bottom
> [("b",2), ("a",10), ("c",3)]
> BillOfMaterials>
```

Writing this in "pseudo-C"-like notation:

$$
\begin{array}{l}
\{\ \texttt{Parts} \quad \texttt{r} = \bot\texttt{;} \\
\quad \texttt{Structure} \quad \texttt{ff} = \texttt{y;} \\
\quad \texttt{while} \quad (\texttt{ff} \mathrel{!}= \bot) \\
\qquad \{\ \texttt{Unit x} = \texttt{get(dom ff);} \\
\qquad\ \texttt{Quantity n} = \texttt{ff x;} \\
\qquad\ \texttt{r} = \texttt{r} \oplus \left\{ \begin{array}{ll} (\texttt{x}==i_1\,\texttt{k}) & \Rightarrow \begin{pmatrix} \texttt{k} \\ n \end{pmatrix} \\ (\texttt{x}==i_2\,\texttt{y'}) & \Rightarrow \bot \end{array} \right. \texttt{;} \\
\qquad\ \texttt{ff} = \texttt{ff} \setminus\{x\} \oplus \left\{ \begin{array}{ll} (x == i_1\,\texttt{k}) & \Rightarrow \bot \\ (x == i_2\,\texttt{y'}) & \Rightarrow n \otimes \texttt{y'} \end{array} \right. \texttt{;} \\
\qquad \};\\
\}
\end{array}
$$

where

- `Parts` abstracts $A \rightharpoonup \mathbb{N}$
- `Unit` abstracts $A + \mathsf{H}(A, \mathbb{N})$
- `Structure` abstracts $\mathsf{H}(A, \mathbb{N})$

A bit of `if-then-else` logic finally turns this into something even simpler:

```
{ Parts r = ⊥;
  Structure ff = y ;
  Quantity n ;
  Unit x;
  while (ff != ⊥)
    { x = get(dom ff) ;
      n = ff x;
      ff = ff \{x};
      if (x == i1 k) r = r ⊕ (k n) ;
      if (x == i1 y') ff = ff ⊕ (n ⊗ y');

    }
}
```

At ORACLE-level — this is straightforwardly encoded as a little piece of PRO-C code + SQL, after a SETS-derivation of the underlying **relational database**, via one of SETS's **recursion removal** laws.

Conclusion

- **Functional programming** is essential to **FM**s
- ... but it is a **part** of the whole story.

Future work

This is work in progress.

Results are incomplete and unsatisfactory in several respects:

- We are still far from a **truly polytypic** characterization of our hierarchical type construction and of its functionality.
- Some attention has been paid to the move towards arbitrary "flat" types $\mathsf{F}(A, B)$ but more work is needed concerning **exponentials** and **contravariance**.

- We have also ignored the key issue of guaranteeing that our hierarchical arbitrary types do in fact **exist** as least fixpoints of their equations.
- Other hierarchical "extensors" might be considered (*e.g.* based on $A + X^n$ rather than on $A + X$) and indeed many hierarchical types exist which are not covered by our constructions.
- The nondeterminism inherent in some of the types we need requires a move from functions to **relations**.