A Hierarchical Polytypic Construction and Associated Reduction Morphisms

DRAFT!

ACL+LOGCOMP'99 — Workshop
10–11 February 1999
Coimbra
Portugal

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

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

Context

DRAFT!

Formal Methods

- Formal specification — yes
- Verification — no!
- "Invent & verify"? — no, transform & **calculate**!

Mathematics of program construction

- 1970's — program transformation
- 1990's — program calculation
- 2000's — process calculation

Vital role of categorial foundations

- Made it possible to **integrate** seemingly disparate schools of thought
- Category theoretic notions more and more familiar to (functional?) programmers.

Categorial toolboxes

Toolbox = a universal construction + functor packaged with a little bit of theory:

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

Example

The **Coproduct ($A+B$) toolbox**:

Coproduct fusion : $f \cdot [\, g, h \,] = [\, f \cdot g, f \cdot h \,]$ (1)

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "g"'] \arrow[ddr, "f \cdot g"'] & A+B \arrow[d, "{[\, g,h \,]}"] & B \arrow[l, "i_2"'] \arrow[dl, "h"] \arrow[ddl, "f \cdot h"] \\
& C \arrow[d, "f"] & \\
& D &
\end{tikzcd}$$

Coproduct reflection : $[\, i_1, i_2 \,] = id_{A+B}$ (2)

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "i_1"'] & A+B \arrow[d, "id_{A+B}"] & B \arrow[l, "i_2"'] \arrow[dl, "i_2"] \\
& A+B &
\end{tikzcd}$$

Coproduct absorption : $[\, g, h \,] \cdot (i+j) = [\, g \cdot i, h \cdot j \,]$ (3)

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[d, "i"'] & A+B \arrow[d, "i+j"'] & B \arrow[l, "i_2"'] \arrow[d, "j"] \\
D \arrow[r, "i_1"] \arrow[dr, "g"'] & D+E \arrow[d, "{[\, g,h \,]}"] & E \arrow[l, "i_2"] \arrow[dl, "h"] \\
& C &
\end{tikzcd}$$

Coproduct cancellation : $[\, g, h \,] \cdot i_1 = g \;,\; [\, g, h \,] \cdot i_2 = h$ (4)

... plus the "functorial goodies":

$+$-functor :

$$(g \cdot h) + (i \cdot j) = (g + i) \cdot (h + j) \quad (5)$$

$+$-functor-id :

$$id_A + id_B = id_{A+B} \quad (6)$$

DRAFT!

i_1 is natural :

$$(i + j) \cdot i_1 = i_1 \cdot i \quad (7)$$

i_2 is natural :

$$(i + j) \cdot i_2 = i_2 \cdot i \quad (8)$$

Compare $A + B$ with `union A B` in C/C++, or with `Either A B` in HASKELL.

Ahh... HASKELL — go and get it !

Other basic toolboxes

- Product $A \times B$
- Exponentials A^B
- Initial (0) and terminal (1) datatypes
- Pointers $(1 + A)$
- Polynomials (*e.g.* $1 + A \times X$)
- *etc.*

The role of functors

The "Manes & Arbib" school:

E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1986. D. Gries, series editor.

Functors "in the large"

- Datatypes "are" (endo) functors,

$$\begin{tikzcd}
A \arrow[d, "f"] & A^\star \arrow[d, "f^\star"] \\
B & B^\star
\end{tikzcd}$$

DRAFT!

and (of course!)

$$
\begin{aligned}
id^\star &= id \\
(f \cdot g)^\star &= f^\star \cdot g^\star
\end{aligned}
$$

(how special this was to write about LISP `mapcar` 20 years ago ...)

- **Polymorphism** = natural transformation,

$$
\begin{tikzcd}
A \arrow[d, "f"'] & A^\star \arrow[r, "elems_A"] \arrow[d, "f^\star"'] & \mathcal{P}\,A \arrow[d, "\mathcal{P}\,f"] \\
B & B^\star \arrow[r, "elems_B"'] & \mathcal{P}\,B
\end{tikzcd}
$$

cf. the *theorem for free* of the *elems* function:

$$\mathcal{P}\,f \cdot elems \;=\; elems \cdot f^\star \tag{9}$$

First steps 10 years ago:

Quotation (from *Theorems for free!*, by Philip Wadler, ACM FPL&CA 1989):

> *From the type of a polymorphic function we can derive a theorem that is satisfies.* (...) *How useful are the theorems so generated? Only time and experience will tell* (...)

Functors "in the small"

(or bringing Tarski+Kleene into the club)

$A^\star$ "internals" = it is the *least fixpoint* — *initial* solution — to equation

$$X \;\cong\; 1 + A \times X$$

In general

$$X \;\cong\; \mathsf{F}\,X$$

is a domain equation, and its fixpoint solution $\mu\mathsf{F}$ is the carrier of the initial F-algebra $(\mu\mathsf{F},\ \mu\mathsf{F} \xleftarrow{in} \mathsf{F}\,\mu\mathsf{F})$:

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[r, bend left, "out"] \arrow[r, phantom, "\cong"] & \mathsf{F}\,\mu\mathsf{F} \arrow[l, bend left, "in"]
\end{tikzcd}
$$

DRAFT!

You said "initial"?

You did. So we get another "toolbox", **the initial F-algebra toolbox**, this time involving F-algebras.

The arrow which mediates in and another F-algebra $(A,\ A \xleftarrow{\alpha} \mathsf{F}\,A)$ is unique and is called *catamorphism* (from $\kappa\alpha\tau\alpha$ = "downwards"):

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[d, "f = (\!|\alpha|\!)_{\mathsf{F}}"'] & \mathsf{F}\,\mu\mathsf{F} \arrow[l, "in"'] \arrow[d, "\mathsf{F}(\!|\alpha|\!)_{\mathsf{F}}"] \\
A & \mathsf{F}\,A \arrow[l, "\alpha"]
\end{tikzcd}
$$

The arrows which are relevant in this context are the F-algebra *homomorphisms*:

$$
\begin{tikzcd}
A \arrow[d, "f"'] & \mathsf{F}\,A \arrow[l, "\alpha"] \arrow[d, "\mathsf{F}\,f"] \\
B & \mathsf{F}\,B \arrow[l, "\beta"]
\end{tikzcd}
$$

Therefore:

$$\textbf{Cata-fusion :}\quad f \cdot (\!|\alpha|\!) = (\!|\beta|\!) \quad \text{if} \quad f \cdot \alpha = \beta \cdot \mathsf{F}\,f \tag{10}$$

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[d, "(\!|\alpha|\!)"'] & \mathsf{F}\,\mu\mathsf{F} \arrow[l, "in"'] \arrow[d, "\mathsf{F}(\!|\alpha|\!)"] \\
A \arrow[d, "f"'] & \mathsf{F}\,A \arrow[l, "\alpha"] \arrow[d, "\mathsf{F}\,f"] \\
B & \mathsf{F}\,B \arrow[l, "\beta"]
\end{tikzcd}
$$

$$\textbf{Cata-cancellation :}\quad (\!|\alpha|\!) \cdot in = \alpha \cdot \mathsf{F}\,(\!|\alpha|\!) \tag{11}$$

DRAFT!

Cata-reflection : $\;(\![in]\!) = id$ (12)

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[d, "(\![in]\!)"'] & \mathsf{F}\,\mu\mathsf{F} \arrow[l, "in"'] \arrow[d, "\mathsf{F}\,(\![in]\!)"] \\
\mu\mathsf{F} & \mathsf{F}\,\mu\mathsf{F} \arrow[l, "in"]
\end{tikzcd}
$$

What about **Cata-absorption**? Where is the associated functor?

Type functors

In writing

$$A^\star \;\cong\; \mathsf{F}\,A^\star$$

for $\mathsf{F}\,X = 1 + A \times X$ we don't want A to be regarded as a constant datatype. So we should give room for A as a parameter and write

$$A^\star \;\cong\; \mathsf{B}(A, A^\star)$$

for $\mathsf{B}(X,Y) = 1 + X \times Y$.

In general, datatypes "are" functors built on top of other (base) functors:

$$\underbrace{\mathsf{T}A}_{\text{type functor}} \quad\cong\quad \underbrace{\mathsf{B}(A,\mathsf{T}A)}_{\text{base functor}}$$

Example

$$\mathsf{BTree}\,A \;\cong\; \underbrace{1 + A \times (\mathsf{BTree}\,A)^2}_{\substack{\text{polynomial} \\ \text{base functor} \\ \mathsf{B}(X,Y) = 1 + X \times Y^2}}$$

TYPE FUNCTOR DEFINITION:

$$
\begin{tikzcd}
A \arrow[d, "f"'] & \mathsf{T}\,A \arrow[d, "\mathsf{T}\,f = (\![in_B \cdot \mathsf{B}(f,id)]\!)"'] & & \mathsf{B}(A,\mathsf{T}\,A) \arrow[ll, "in_A"'] \arrow[d, "\mathsf{B}(id,\mathsf{T}\,f)"] \\
B & \mathsf{T}\,B & \mathsf{B}(B,\mathsf{T}\,B) \arrow[l, "in_B"] & \mathsf{B}(A,\mathsf{T}\,B) \arrow[l, "\mathsf{B}(f,id)"]
\end{tikzcd}
$$

Now, **Cata-absorption**

$$(\![\alpha]\!) \cdot \mathsf{T}\,f \;=\; (\![\alpha \cdot \mathsf{B}(f,id)]\!) \tag{13}$$

is but a consequence of the fact that, for a polynomial B, $\mathsf{B}(f,id)$ is a natural transformation:

$$
\begin{tikzcd}
A \arrow[d, "f"'] & \mathsf{T}\,A \arrow[d, "\mathsf{T}\,f"'] & & \mathsf{B}(A,\mathsf{T}\,A) \arrow[ll, "in_A"'] \arrow[d, "\mathsf{B}(id,\mathsf{T}\,f)"] \\
B & \mathsf{T}\,B \arrow[d, "(\![\alpha]\!)"'] & \mathsf{B}(B,\mathsf{T}\,B) \arrow[l, "in_B"] \arrow[d, "\mathsf{B}(B,(\![\alpha]\!))"] & \mathsf{B}(A,\mathsf{T}\,B) \arrow[l, "\mathsf{B}(f,id)"] \arrow[d, "\mathsf{B}(A,(\![\alpha]\!))"] \\
 & C & \mathsf{B}(B,C) \arrow[l, "\alpha"] & \mathsf{B}(A,C) \arrow[l, "\mathsf{B}(f,id)"]
\end{tikzcd}
$$

But there is more, *e.g.* the *banana-split* law

$$\langle (\![g]\!), (\![k]\!) \rangle = (\![\langle g \cdot \mathsf{F}\,\pi_1, k \cdot \mathsf{F}\,\pi_2 \rangle]\!) \tag{14}$$

scaling-up the *exchange-law*

$$\langle [\,f,h\,], [\,g,k\,] \rangle \;=\; [\,\langle f,g\rangle, \langle h,k\rangle\,] \tag{15}$$

to catas, *etc.*

Dualizing . . . :

$A \times B$ dualizes $A + B$, 1 dualizes 0?

So terminal solutions dualize the initial ones and we get *anamorphisms* ($\alpha\nu\alpha$= "upwards") as catamorphism duals:

$$
\begin{tikzcd}
\mathsf{T}\,A & \mathsf{B}(A,\mathsf{T}\,A) \arrow[l, "in"'] \\
B \arrow[u, "[\![h]\!]"] \arrow[r, "h"'] & \mathsf{B}(A,B) \arrow[u, "\mathsf{B}(id,[\![h]\!])"']
\end{tikzcd}
$$

generated by *co-algebra* h.

Coalgebras, anamorphisms and terminal solutions move us from **programs** to **processes** — *cf.* work by Luís Barbosa in the LOG-COMP project.

DRAFT!
DRAFT!

Hylomorphisms:

A hylomorphism $[\![g, h]\!]$ is the combination of a cata and a ana of the same type:

$$[\![g, h]\!] = (\!|g|\!) \cdot [\!(h)\!]$$

The hylomorphism is a very general programming concept:

$$\begin{tikzcd}
C & \mathsf{B}(A, C) \arrow[l, "g"'] \\
\mathsf{T}\,A \arrow[u, "(\!|g|\!)"] & \mathsf{B}(A, \mathsf{T}\,a) \arrow[l, "in"'] \arrow[u, "\mathsf{B}(id,(\!|g|\!))"'] \\
B \arrow[u, "{[\!(h)\!]}"] \arrow[r, "h"'] \arrow[uu, bend left=60, "{[\![g,h]\!]}"] & \mathsf{B}(A, B) \arrow[u, "\mathsf{B}(id,[\!(h)\!])"'] \arrow[uu, bend right=60, "{\mathsf{B}(id,[\![g,h]\!])}"']
\end{tikzcd}$$

By the way, *factorial* is a (list) hylomorphism, *quick sort* is a binary tree hylomorphism, *merge sort*

$$\begin{tikzcd}
A^+ \arrow[rr, bend left, "out"] \arrow[d, "mSort"'] & A + A^+ \times A^+ \arrow[d, "id+mSort\times mSort"] & A + A \times A^+ \arrow[l, "id+sep"] \arrow[dl, dotted] \\
A^+ & A + A^+ \times A^+ \arrow[l, "{[\,singl, merge\,]}"] &
\end{tikzcd}$$

is a leaf tree hylomorphism, *etc. etc.*

Category theory has made a "revolution" in standard algorithm theory...

Comments:

- In the 16th century: *12.cu.p̃.18.ce.p̃.27.co.p̃.17*
- In the 20th century: $12x^3 + 18x^2 + 27x + 17$
- In the 21st century (?): **pointfree** programming — recall John Backus *FP* — versus pointwise (conventional) programming.
- The pointfree style can be easily adopted in *e.g.* HASKELL.

- In fact, catamorfisms are *generic folds*, *cf.* `foldr` in HASKELL, *etc.*

You said "generic"?

You did. So we get another step in the *program abstraction* ladder: note that the **Cata-toolbox** was described parametrically on functors F, B *etc.*.

Another most important consequence of going categorial: *polymorphism* gives place to *polytypism*, that is to say:

Polytypism = higher-order polymorphism

Polytypism is <u>the</u> exciting R&D area on algorithmics these days....

Eventually: The Polytypic Hirerachical Construction

Motivation

- Sometime ago: students of the Minho **FM** course asked to design a **data mining** library.
- 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.

DRAFT!

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

DRAFT!

Main points

- **Data mining** is but knowledge extraction and **accumulation** 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 are interested on monoidal reductions within a particular class of *general **hierarchical** systems*.
- This arises as a **base-functor** construction.
- (Polytypic) "**flatten**" evolves towards "**explosion**", once one more parameter is added to the hierarchical construction.
- We 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**:

$$I \rightharpoonup File \tag{16}$$

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 (16) 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

$$X \;\cong\; I \rightharpoonup (File + X) \tag{17}$$

Another example: generalized lists,

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

— *cf.* equation

$$X \;\cong\; (a + X)^{\star} \tag{18}$$

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

Generic construction:

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

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

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

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

and then define

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

In the $GList$ example,

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

DRAFT!

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

$$X \;\cong\; ((A + X) \times B)^{\star} \tag{19}$$

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.
- $(A \times I\!N)^{\star}$ as a model of lists of votes (elections within some organization)
- Extension (19) 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

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

and

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

DRAFT!

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

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

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, $\mathsf{H} = GList$?

$$\begin{array}{rcl} flatten & : & \mathsf{H}\,A \longrightarrow A^\star \\ flatten\, l & \stackrel{\mathrm{def}}{=} & \mathop{+\!\!\!+}_{x \leftarrow l} \left\{ \begin{array}{lcl} (x = i_1\, a) & \Rightarrow & [a] \\ (x = i_2\, l') & \Rightarrow & flatten\, l' \end{array} \right. \end{array} \qquad (20)$$

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

$$\begin{array}{rcl} \mathop{+\!\!\!+}[\,] & = & [\,] \\ \mathop{+\!\!\!+} cons(x, l) & = & x + (\mathop{+\!\!\!+}\, l) \end{array} \qquad (21)$$

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:

$$flatten \cdot [\, nil, cons\,] \;=\; [\, nil, + \cdot ([\, wrap, flatten\,] \times flatten)\,]$$

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

$$\begin{tikzcd} \mathsf{H}\,A \arrow[d, "flatten"] & (A + \mathsf{H}\,A)^\star \arrow[l, "\delta"'] & 1 + (A + \mathsf{H}\,A) \times (A + \mathsf{H}\,A)^\star \arrow[l, "{[\, nil, cons\,]}"'] & 1 + (A + \mathsf{H}\,A) \times \mathsf{H}\,A \arrow[l, "1 + id \times \delta^{-1}"'] \arrow[d, "1 + (A + flatten) \times flatten"] \\ A^\star & 1 + A^\star \times A^\star \arrow[l, "{[\, nil, +\,]}"] & & 1 + (A + A^\star) \times A^\star \arrow[ll, "{1 + [\, wrap, id\,] \times id}"] \end{tikzcd}$$

Or, if you like,

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

$$\begin{tikzcd} \mathsf{H}\,A \arrow[d, "f"'] \arrow[rr, "\omega"] & & 1 + (A + \mathsf{H}\,A) \times \mathsf{H}\,A \arrow[d, "1 + (A + f) \times f"] \\ A^\star & 1 + A^\star \times A^\star \arrow[l, "{[\, nil, +\,]}"] & 1 + (A + A^\star) \times A^\star \arrow[l, "\nu"] \end{tikzcd}$$

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

So we have

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

i $flatten$ 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,

$$\begin{tikzcd} T \arrow[d, "f"'] \arrow[rr, "\beta"] & & \mathsf{F}\,T \arrow[d, "\mathsf{F}\,f"] \\ M & 1 + M \times M \arrow[l, "{[\, \underline{\epsilon}, \theta\,]}"] & \mathsf{F}\,M \arrow[l, "\nu"] \end{tikzcd}$$

DRAFT!

(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":

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

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

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

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

$$\begin{tikzcd} \mathsf{H}(A,B) \arrow[rr, "\omega = [\,nil, cons\,]^{-1}"] \arrow[d, "flatten"'] & & \mathsf{J}\,\mathsf{H}(A,B) \arrow[d, "\mathsf{J}\,flatten"] \\ (A\times B)^\star & 1 + (A\times B)^\star \times (A\times B)^\star \arrow[l, "{[\,nil, +\!\!+\,]}"] & \mathsf{J}\,(A\times B)^\star \arrow[l, "\nu"] \end{tikzcd} \quad (22)$$

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

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

$$\begin{tikzcd} 1 + ((A + (A\times B)^\star)\times B)\times (A\times B)^\star \arrow[d, "1 + distl \times id"] \\ 1 + (A\times B + (A\times B)^\star \times B)\times (A\times B)^\star \arrow[d, "1 + (id + swap)\times id"] \\ 1 + (A\times B + B\times (A\times B)^\star)\times (A\times B)^\star \arrow[d, "{1 + [\,wrap, \otimes\,]\times id}"] \\ 1 + (A\times B)^\star \times (A\times B)^\star \end{tikzcd}$$

($distl$ 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,

$$B \times (A\times B)^\star \xrightarrow{swap} (A\times B)^\star \times B \xrightarrow{\phi} ((A\times B)\times B)^\star \qquad (23)$$

DRAFT!
DRAFT!

followed by

$$((A \times B) \times B)^\star \xrightarrow{assocr^\star} (A \times (B \times B))^\star \xrightarrow{(id \times (\times))^\star} (A \times B)^\star \qquad (24)$$

where

$$(\times) \;:\; B \times B \longrightarrow B \qquad (25)$$

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

Bill of materials

Recall

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

for $B = I\!N$.

The specification of $flatten$ becomes

$$flatten \;:\; \mathsf{H}(A,B) \longrightarrow (A \rightharpoonup B)$$

$$flatten\,\sigma \;\stackrel{\rm def}{=}\; \bigoplus_{x \in \textit{dom}\,\sigma} \begin{cases} (x = i_1\,a) & \Rightarrow \begin{pmatrix} a \\ \sigma\,x \end{pmatrix} \\ (x = i_2\,\sigma') & \Rightarrow (\sigma\,x) \otimes flatten\,\sigma' \end{cases} \qquad (26)$$

where

$$v \oplus r \;=\; v \dagger r \dagger \begin{pmatrix} a \\ v\,a + r\,a \end{pmatrix}_{a \in (\textit{dom}\,v) \cap (\textit{dom}\,r)}$$

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) =
```


DRAFT!

```
>         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:

- $A \rightharpoonup I\!N$ is a model of multisets.
- If we identify A with the dimensions of a **vector space**, then each $v \in A \rightharpoonup I\!N$ 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.*

DRAFT!

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

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

Proposal:

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

Thinking of $B = String$, why not

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

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

Prototype:

```
> wrap a b = [ a |-> b ]
>
> oplus x = plus x
```


DRAFT!

```
>
> 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
```


DRAFT!

```
>  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 $X \cong 1 + (A \times B) \times X$.
2. Its hierarchical extension, $\mathsf{H}(A,B)$, defined by domain equation

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

3. A "wrap up" arrow $\mathsf{F}(A,B) \xrightarrow{wrap} A \times B$.
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** $[\![\, [\,\underline{\epsilon},\theta\,] \cdot \nu, \beta]\!]$ depicted by

$$\begin{tikzcd} \mathsf{H}(A,B) \arrow[rr, "\beta"] \arrow[d, "f"] & & \mathsf{J}\,\mathsf{H}(A,B) \arrow[d, "\mathsf{J}\,f"'] \\ \mathsf{F}(A,B) & 1 + \mathsf{F}(A,B) \times \mathsf{F}(A,B) \arrow[l, "{[\,\underline{\epsilon},\theta\,]}"] & \mathsf{J}\,\mathsf{F}(A,B) \arrow[l, "\nu"] \end{tikzcd} \tag{27}$$

where

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

β is the appropriate input "parser" and

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

DRAFT!

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 $\nu \cdot \mathsf{J}\,f$ in diagram (27):

$$\begin{tikzcd} \mathsf{H}(A,B) \arrow[rr, bend left, "\beta"] \arrow[d, "f"] & 1 + \mathsf{F}(A,B) \times \mathsf{H}(A,B) \arrow[d, "1+\mathsf{F}(A,B)\times f"] & \mathsf{J}\,\mathsf{H}(A,B) \arrow[l, "\nu'"] \arrow[d, "\mathsf{J}\,f"'] \\ \mathsf{F}(A,B) & 1 + \mathsf{F}(A,B) \times \mathsf{F}(A,B) \arrow[l, "{[\,\underline{\epsilon},\theta\,]}"] & \mathsf{J}\,\mathsf{F}(A,B) \arrow[l, "\nu"] \end{tikzcd}$$

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

$$[\, g, h \,] \times f \;=\; [\, g \times f, h \times f \,] \cdot distl \tag{28}$$

Detailed reasoning is as follows:

DRAFT!

$$
\begin{array}{cl}
f & = \ [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap, \otimes \cdot swap \,] \cdot distl \times id) \cdot (1 + ((id + f) \times id) \times f) \cdot \beta \\
= & \qquad \{ \text{ bifunctors } + \text{ and } \times \ \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap, \otimes \cdot swap \,] \cdot distl \cdot ((id + f) \times id) \times f) \cdot \beta \\
= & \qquad \{ \ distl \text{ is natural } \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap, \otimes \cdot swap \,] \cdot (id + f \times id) \cdot distl \times f) \cdot \beta \\
= & \qquad \{ \ +\text{-absorption and } swap \text{ is natural } \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap, \otimes \cdot (id \times f) \cdot swap \,] \cdot distl \times f) \cdot \beta \\
= & \qquad \{ \text{ fact (30) below and identity of composition } \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap, f \cdot \otimes \cdot swap \,] \cdot distl \times f \cdot id) \cdot \beta \\
= & \qquad \{ \text{ bifunctor } \times \ \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + ([\, wrap, f \cdot \otimes \cdot swap \,] \times f) \cdot (distl \times id)) \cdot \beta \\
= & \qquad \{ \text{ fact (28) } \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap \times f, f \cdot \otimes \cdot swap \times f \,] \cdot distl \cdot (distl \times id)) \cdot \beta \\
= & \qquad \{ \text{ identity of composition and bifunctor } \times \ \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap \times f, (f \times f) \cdot (\otimes \cdot swap \times id) \,] \cdot distl \cdot (distl \times id)) \cdot \beta \\
= & \qquad \{ \text{ bifunctor } + \ \} \\
 & [\, \underline{\epsilon}, \theta \,] \cdot (1 + [\, wrap \times f, (f \times f) \cdot (\otimes \cdot swap \times id) \,]) \cdot (1 + distl \cdot (distl \times id)) \cdot \beta \\
= & \qquad \{ \ +\text{-absorption, } +\text{-fusion and fact (31) below } \} \\
 & [\, \underline{\epsilon}, [\, \theta \cdot (wrap \times f), \underbrace{f \cdot \theta \cdot (\otimes \ \cdot swap \times id)}_{\rho} \,] \,] \cdot \underbrace{(1 + distl \cdot (distl \times id)) \cdot \beta}_{\beta'} \\
= & \qquad \{ \text{ introducing abbreviations } \rho \text{ and } \beta' \ \} \\
 & [\, \underline{\epsilon}, [\, \theta \cdot (wrap \times f), \rho \,] \,] \cdot \beta' \\
= & \qquad \{ \text{ forcing } \epsilon \text{ in, which is the unit of } \theta \ \} \\
 & [\, \underline{\epsilon}, [\, \theta \cdot (wrap \times f), \theta \cdot \langle \underline{\epsilon}, \rho \rangle \,] \,] \cdot \beta' \\
= & \qquad \{ \text{ reverse } +\text{-fusion followed by exchange law } \} \\
 & [\, \underline{\epsilon}, \theta \cdot \langle [\, wrap \cdot \pi_1, \underline{\epsilon} \,], [\, f \cdot \pi_2, \rho \,] \rangle \,] \cdot \beta'
\end{array}
$$

DRAFT!

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

From this reasoning we extract

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

which is equivalent to

$$\nu' \ = \ 1 + \langle [\, wrap \cdot \pi_1 \cdot distl, \underline{\epsilon} \,], \theta \cdot [\, \underline{\epsilon}, \otimes \cdot swap \,] \times id \rangle \cdot (distl \times id) \quad (29)$$

Important

Two facts required above,

$$f \cdot \otimes \ = \ \otimes \cdot (id \times f) \quad (30)$$

$$f \cdot \theta \ = \ \theta \cdot (f \times f) \quad (31)$$

can be derived provided properties

$$b \otimes (x \, \theta \, y) \ = \ (b \otimes x) \, \theta \, (b \otimes y) \quad (32)$$

and

$$(n \star b) \otimes x \ = \ n \otimes (b \otimes x) \quad (33)$$

hold, where

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

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

$$B \times \mathsf{F}(A, B) \xrightarrow{swap} \mathsf{F}(A, B) \times B \xrightarrow{\varphi} \mathsf{F}(A, B \times B) \xrightarrow{\mathsf{F}(id, \star)} \mathsf{F}(A, B) \quad (34)$$

DRAFT!

“Vector spaces” are required by $flatten$

- Facts (32) and (33) 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

$$(n + m) \otimes x \;=\; (n \otimes x)\,\theta\,(m \otimes x) \qquad (35)$$

- 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:

```
{ Parts      r = ⊥;
  Structure  ff = y;
  while      (ff != ⊥)
             { Unit x = get(dom ff);
               Quantity n = ff x;
               r = r ⊕ { (x==i1 k)   ⇒ (k, n) ;
                       { (x==i2 y')  ⇒ ⊥
               ff = ff \{x} ⊕ { (x == i1 k)   ⇒ ⊥      ;
                              { (x == i2 y')  ⇒ n ⊗ y'
             };
}
```

where

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

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

DRAFT!

```
{ 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**.

DRAFT!

- 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**.

DRAFT!