Programming from metaphorisms

Institute of Cybernetics

Tallinn — April 28th, 2016

J.N. Oliveira

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

INESC TEC & University of Minho

Metaphorism < metaphor

Cognitive linguistics versus Chomskian **generative** linguistics

- Information science is based on Chomskian **generative grammars**
- Semantics is a "quotient" of syntax
- **Cognitive linguistics** has emerged meanwhile
- Emphasis on conceptual **metaphors** — the basic building block of semantics
- *Metaphors we live by* (Lakoff and Johnson, 1980).

Metaphors we live by

A **cognitive metaphor** is a device whereby the meaning of an idea (concept) is carried by another, e.g.

> *She* **counterattacked** *with a* **winning** *argument*

— the underlying metaphor is ARGUMENT IS WAR.

Metaphor TIME IS MONEYunderlies everyday phrases such as e.g.:

> *You are* **wasting** *my time*

> **Invest** *your time in something else.*

Metaphoric language

Attributed to Mark Twain:

> *"Politicians and diapers should be changed often and for the same reason".*

(*'No jobs for the boys'* in metaphorical form.)

Metaphor structure, where P = politician and D = diaper:

$$
\begin{tikzcd}
P \arrow[loop above, "\color{red}{chngt'}"] & & D \arrow[ll, "corrupt^{\circ} \cdot dirty"'] \arrow[loop above, "chng"] \arrow[dl, "dirty"] \\
& \mathbb{B} \arrow[from=1-1, "corrupt"'] &
\end{tikzcd}
$$

$dirty\ (chng\ x) = False$ induces $chngt'$ over P, and so on.

Formal metaphors

In his *Philosophy of Rhetoric*, Richards (1936) finds three kernel ingredients in a metaphor, namely

- a **tenor** (e.g. *politicians*)
- a **vehicle** (e.g. *diapers*)
- an implicit, shared **attribute**.

Formally, we have a "cospan"

$$\begin{tikzcd} \mathbf{T} \arrow[dr, "f"'] & & \mathbf{V} \arrow[dl, "g"] \\ & A & \end{tikzcd} \tag{1}$$

where functions $f : \mathbf{T} \to A$ and $g : \mathbf{V} \to A$ extract the common **attribute** (A) from **tenor** ($\mathbf{T}$) and **vehicle** ($\mathbf{V}$).

Formal metaphors

The cognitive, æsthetic, or witty power of a **metaphor** is obtained by *hiding* A, thereby establishing a *composite*, **binary relationship**

$$\mathbf{T} \xleftarrow{f^\circ \cdot g} \mathbf{V}$$

— the "**T** is **V**" metaphor — which leaves A implicit.

Remarks *on notation:*

- $x\ f^\circ\ y$ *means the same as* $y\ f\ x$, *that is* $y = f\ x$.
- *In general,* $x\ R^\circ\ y$ *asserts the same as* $y\ R\ x$.
- *Relational composition:*

$$y\ (R \cdot S)\ x \quad \mathit{iff} \quad \langle \exists\ z\ ::\ y\ R\ z \wedge z\ S\ x \rangle$$

Metaphors in science

Scientific expression is inherently metaphoric.

Such metaphors convey the meaning of a **complex**, new concept in terms of a **simpler**, familiar one:

> *The cell* **envelope** ... *proteins* **behave** ... **colonies** *of bacteria* ... *electron* **cloud** ...

Mathematics terminology inherently metaphoric too, cf. e.g.

- **polynomial** functor ...
- vector **addition** ...

(algebraic structure sharing) and so is **computing** terminology in general:

- ... stack, queue, pipe, memory, driver, ...

"Metaphoric" software design?

Text formatting example:

$$
\begin{tikzcd}
{[\mathit{String}]} & & \mathit{String} \arrow[ll, "\mathit{Format}"'] \arrow[dl, "\mathit{words}"] \\
& {[\mathit{String}]} \arrow[from=1-1, "(\gg\!\!= \mathit{words})"'] &
\end{tikzcd}
$$

Only this? No:

Formatting consists in (re)introducing white space evenly throughout the output text lines,

$$\mathit{Format} = ((\gg\!\!= \mathit{words})^{\circ} \cdot \mathit{words}) \restriction R \qquad (2)$$

as specified by some convenient **optimization** *criterion* R

($\cdot \restriction \cdot$ operator to be explained soon.)

Metaphorical specifications

Problem statements are often **metaphorical** in a formal sense — **input-output** relations in which

- some hidden information is **preserved** (the **invariant** part)
- some form of **optimization** takes place (the **variant** part).

INVARIANT PART:

$$
\begin{array}{ll}
 & y\ (f^{\circ} \cdot g)\ x \\
\Leftrightarrow & \quad \{\ \text{composition and converse}\ \} \\
 & \langle \exists\ a\ :\ a\ f\ y :\ a\ g\ x \rangle \\
\Leftrightarrow & \quad \{\ \text{functions } f \text{ and } g\ \} \\
 & \langle \exists\ a\ :\ a = f\ y :\ a = g\ x \rangle \\
\Leftrightarrow & \quad \{\ \text{one-point quantification}\ \} \\
 & (f\ y) = (g\ x)
\end{array}
$$

Metaphorical specifications

VARIANT PART:

$$
\begin{array}{ll}
& y\ (S \upharpoonright R)\ x \\
\Leftrightarrow & \qquad \{ \text{ check definition (18) below } \} \\
& y\ (S \cap R\ /\ S^{\circ})\ x \\
\Leftrightarrow & \qquad \{ \text{ meet } \} \\
& y\ S\ x \wedge y\ (R\ /\ S^{\circ})\ x \\
\Leftrightarrow & \qquad \{ \text{ division (more about this below) } \} \\
& y\ S\ x \wedge \langle \forall\ y'\ :\ y'\ S\ x :\ y\ R\ y' \rangle\ x
\end{array}
$$

Altogether:

> *According to criteria R, y is (among) the* **best** *outputs of S for input x.*

Metaphorical specifications

INVARIANT + VARIANT parts:

$$M = (f^\circ \cdot g) \upharpoonright R \qquad \begin{tikzcd} & \mathbf{T} & \\ \mathbf{T} \arrow[ur, "R"] \arrow[dr, "f"'] & & \mathbf{V} \arrow[ul, "M"'] \arrow[ll, "f^\circ \cdot g"'] \arrow[dl, "g"] \\ & A & \end{tikzcd} \tag{3}$$

Meaning of $y\ M\ x$:

- $f\ y = g\ x$ (the information preserved);
- output y is "best" among all other y' such that $f\ y' = f\ x$ (this is the **optimization**).

Metaphorisms

Term "metaphorism" refers to metaphors involving tree-like, inductive types, e.g.

- **Source code refactoring** — the meaning of the source **program** is preserved, the target code being better styled wrt. coding conventions and best practices.
- **Change of base** (numeric representation) — the numbers represented by the source and the result are the same, cf. the *representation changers* of Hutton and Meijer (1996).
- **Sorting** — the bag (multiset) of elements of the source list is preserved, the optimization consisting in obtaining an *ordered* output.

etc

More about (relation) notation

Relation division is for relational composition what whole division is for multiplication of natural numbers, compare property

$$z \times y \leqslant x \Leftrightarrow z \leqslant x \div y$$

— $x \div y$ is the **largest** number that multiplied by y approximates x — with property

$$Q \cdot S \leqslant R \Leftrightarrow Q \leqslant R / S \tag{4}$$

— R / S is the **largest** relation that chained with S approximates R.

(Both are Galois connections.)

More about (relation) notation

Moreover, we can define a kind of **symmetric division** by

$$\frac{S}{R} = (S^\circ / R^\circ)^\circ \cap R^\circ / S^\circ \qquad \begin{tikzcd} B \arrow[dr, "R"'] & & C \arrow[ll, "\frac{S}{R}"'] \arrow[dl, "S"] \\ & A & \end{tikzcd} \tag{5}$$

Pointwise:

$$b \, \frac{S}{R} \, c \ \Leftrightarrow \ \langle \forall\, a \ :: \ a\, R\, b \Leftrightarrow a\, S\, c \rangle \tag{6}$$

In the case of functions:

$$\frac{f}{g} = g^\circ \cdot f \tag{7}$$

"Rational" relations

So **metaphors** are nicely described by "fractions" $\frac{f}{g}$ which, incidentally, share several properties (when paralleled with) **rational** numbers, e.g.

$$\left(\frac{f}{g}\right)^{\circ} = \frac{g}{f} \tag{8}$$

$$\frac{id}{g} \cdot \frac{h}{k} \cdot \frac{f}{id} = \frac{h \cdot f}{k \cdot g} \tag{9}$$

Moreover, metaphors are closed by intersection:

$$\frac{f}{g} \cap \frac{h}{k} = \frac{f \vartriangledown h}{g \vartriangledown k} \tag{10}$$

where $(f \vartriangledown h)\ x = (f\ x, h\ x)$ is the **pairing** operator.

Predicates and diagonals

As in the POLITICS IS DIRT metaphor, metaphors can involve predicates p, q, ... for instance

$$y \frac{true}{q} x = q\ y$$

where $true$ is the everywhere-true predicate.

Put in another way, we can encode predicates in the form of **diagonal** metaphors:

$$p? = id \cap \frac{true}{p} \tag{11}$$

that is,

$$y\ (p?)\ x \Leftrightarrow (y = x) \wedge (p\ y)$$

holds.

Weakest preconditions

More generally,

$$f \cap \frac{true}{q} = q? \cdot f \qquad\qquad f \cap \frac{p}{true} = f \cdot p?$$

hold. Moreover, equality

$$f \cap \frac{p}{true} = \frac{true}{q} \cap f$$

expresses a **weakest** precondition (p) / **strongest** postcondition (q) relationship.

Another way to write this:

$$f \cdot p? = q? \cdot f \quad \Leftrightarrow \quad p = q \cdot f \tag{12}$$

Divide & conquer metaphors

Can we derive **programs** from a given **metaphor**

$$M = \frac{f}{g} \upharpoonright R \tag{13}$$

by calculation?

By this law of shrinking

$$(S \cdot f) \upharpoonright R = (S \upharpoonright R) \cdot f \tag{14}$$

we can shift f out of the metaphor:

$$\frac{f}{g} \upharpoonright R = (\frac{id}{g} \upharpoonright R) \cdot f$$

This is known as the *inverse of a function* refinement strategy.

Divide & conquer metaphors

Divide & conquer programming calls for an auxiliary structure $\mathbf{W}$ between vehicle and tenor,

$$\mathbf{T} \longleftarrow \mathbf{W} \longleftarrow \mathbf{V}$$

intended to gain **control** of the "pipeline".

This can be done in two ways.

Assume a **surjection** $h : \mathbf{W} \to \mathbf{T}$ on the **tenor** side, that is, $\rho\, h = h \cdot h^{\circ} = id$.

Range of a function:
$y' \ (h \cdot h^{\circ}) \ y \Leftrightarrow \langle \exists\, x \ :: \ y' = y \wedge y = h\ x \rangle.$

Divide & conquer metaphors

Then $h : \mathbf{W} \to \mathbf{T}$ provides an intermediate **representation** of the tenor.

As we shall see shortly, the splitting works as follows

$$
\begin{tikzcd}
\mathbf{T} & \mathbf{W} \arrow[l, "h"'] \arrow[dr, "h"'] & & \mathbf{V} \arrow[ll, "X"'] \arrow[lll, bend right, dotted, "\frac{f}{g} \upharpoonright R"'] \arrow[ddl, "f"] \\
& & \mathbf{T} \arrow[d, "g"'] & \\
& & A &
\end{tikzcd}
\qquad (15)
$$

provided one can find a relation X such that $h \cdot X = \frac{f}{g} \upharpoonright R$.

Note how the **outer** metaphor gives way to an **inner** metaphor between the vehicle ($\mathbf{V}$) and the intermediate type ($\mathbf{W}$).

Divide & conquer metaphors

Alternatively, we can imagine *surjection* h working on the vehicle side, say $h : \mathbf{W} \to \mathbf{V}$ in

$$
\begin{tikzcd}
\mathbf{T} \arrow[dd, "g"'] & & \mathbf{W} \arrow[ll, "Y"'] \arrow[dl, "h"] & \mathbf{V} \arrow[l, "h^\circ"'] \arrow[lll, bend right, dotted, "\frac{f}{g} \upharpoonright R"'] \\
& \mathbf{V} \arrow[dl, "f"] & & \\
A & & &
\end{tikzcd}
\tag{16}
$$

and try and find relation Y such that $Y \cdot h^\circ = \frac{f}{g} \upharpoonright R$.

Note how intermediate type $\mathbf{W}$ acts is **representation** of $\mathbf{T}$ or $\mathbf{V}$ in, respectively, (15) and (16) — h acts as a typical data refinement **abstraction** function.

Divide & conquer metaphors

Let us calculate "**conquer**" step Y in the first place:

$$
\begin{array}{cl}
& \frac{f}{g} \upharpoonright R \\
= & \qquad \{ \text{ identity of composition } \} \\
& (\frac{f}{g} \upharpoonright R) \cdot id \\
= & \qquad \{ \; h \text{ assumed to be a surjection, } h \cdot h^{\circ} = id \; \} \\
& (\frac{f}{g} \upharpoonright R) \cdot h \cdot h^{\circ} \\
= & \qquad \{ \text{ law (14) } \} \\
& \underbrace{(\frac{f \cdot h}{g} \upharpoonright R)}_{Y} \cdot h^{\circ}
\end{array}
$$

Divide & conquer metaphors

Altogether:

$$\frac{f}{g} \upharpoonright R \;=\; \left(\frac{f \cdot h}{g} \upharpoonright R\right) \cdot h^{\circ} \qquad \text{for } h \text{ surjective} \tag{17}$$

In a diagram, completing (16):

$$\begin{tikzcd}
& \mathbf{T} & & & \\
\mathbf{T} \arrow[ur, bend left, "R"] \arrow[ddr, "g"'] & & & \mathbf{W} \arrow[ull, "\frac{f\cdot h}{g}\upharpoonright R"'] \arrow[lll, "\frac{f\cdot h}{g}"'] \arrow[dl, "h"] & \mathbf{V} \arrow[l, "h^{\circ}"'] \arrow[ulll, dotted, bend right, "\frac{f}{g}\upharpoonright R"'] \\
& & \mathbf{V} \arrow[dl, "f"] & & \\
& A & & &
\end{tikzcd}$$

Strategy is known by "**Easy Split**, **Hard Join**" (Howard, 1994), where "Split" (resp. "Join") stands for "divide" (resp. "conquer")

Thus the hard work is deferred to the **conquer** stage.

Divide & conquer metaphors

Next we calculate the alternative "**Hard Split**, **Easy Join**" strategy. We will need

$$S \upharpoonright R \;=\; S \cap R/S^{\circ}. \tag{18}$$

to solve equation

$$\begin{tikzcd}
\mathbf{T} & \mathbf{W} \arrow[l, "h"'] \arrow[dr, "h"'] & & \mathbf{V} \arrow[lll, bend right, dotted, "\frac{f}{g}\upharpoonright R"'] \arrow[ll, "X"'] \arrow[dd, "f"] \\
& & \mathbf{T} \arrow[dr, "g"'] & \\
& & & A
\end{tikzcd}$$

for X (next slide).

"Hard Split, Easy Join"

$$
\begin{array}{ll}
& \frac{f}{g} \upharpoonright R \\
= & \qquad \{ \ (18) \ ; \text{converse of a metaphor } (8) \ \} \\
& \frac{f}{g} \cap R \,/\, \frac{g}{f} \\
= & \qquad \{ \ h \text{ assumed to be a surjection, } \rho\, h = h \cdot h^{\circ} = id \ \} \\
& h \cdot h^{\circ} \cdot (\frac{f}{g} \cap R \,/\, \frac{g}{f}) \\
= & \qquad \{ \ \text{injective } h^{\circ} \text{ distributes by } \cap \ \} \\
& h \cdot (\frac{f}{g \cdot h} \cap h^{\circ} \cdot R \,/\, \frac{g}{f})
\end{array}
$$

(Thumb rule: the converse of a function is always injective.)

"Hard Split, Easy Join"

We record property

$$R \,/\, \frac{g}{f} = (R \,/\, g) \cdot f \tag{19}$$

which follows from (4) and carry on:

$$\begin{aligned}
& h \cdot (\frac{f}{g \cdot h} \cap h^\circ \cdot R \,/\, \frac{g}{f}) \\
= \quad & \qquad \{ \text{ above ; shunting } \} \\
& h \cdot \underbrace{(\frac{f}{g \cdot h} \cap h^\circ \cdot (R \,/\, g) \cdot f)}_{X}
\end{aligned}$$

Clearly, the **divide** step X is now where most of the work is done.

"Hard Split, Easy Join"

The choice of intermediate w by X mirrors where the **optimization** has moved to, check this in the pointwise version:

$$
\begin{array}{l}
w\ X\ v \Leftrightarrow \\
\quad \mathbf{let}\ a = f\ v\ \in \\
\qquad (g\ (h\ w) = a) \wedge \langle \forall\ t\ :\ a = g\ t :\ (h\ w)\ R\ t \rangle
\end{array}
$$

In words:

> *Given vehicle v, X will select those w that represent tenors $(h\ w)$ with the same attribute (a) as vehicle v, and that are* **best** *among all other tenors t exhibiting the same attribute a.*

Altogether:

$$
\frac{f}{g} \upharpoonright R \quad = \quad h \cdot (\frac{f}{g \cdot h} \cap h^{\circ} \cdot (R \,/\, g) \cdot f) \qquad \text{for } h \text{ surjective} \qquad (20)
$$

Post-conditioned metaphors

Finally consider the following pattern of metaphor shrinking

$$\frac{f}{g} \upharpoonright \frac{true}{q} \tag{21}$$

indicating that only outputs satisfying q are regarded as **good enough**.

Thus q acts as a **post-condition** on $\frac{f}{g}$.

Example of (21):

$$Sort = \frac{bag}{bag} \upharpoonright \frac{true}{ordered}$$

In this metaphor, function *bag* extracts the bag (**multiset**) of elements of a finite list and predicate *ordered* checks whether it is ordered.

Post-conditioned metaphors

The following equality shows why these metaphors are referred to as *post-conditioned*:

$$\frac{f}{g} \upharpoonright \frac{true}{q} = q? \cdot \frac{f}{g}$$

Example (sorting):

$$Sort = ordered? \cdot Perm \quad \textbf{where} \quad Perm = \frac{bag}{bag} \tag{22}$$

where $y\ Perm\ x$ means that y is a *permutation* of x.

For this special case, "Hard Split, Easy Join" (20) boils down to

$$q? \cdot \frac{f}{g} = h \cdot p? \cdot \frac{f}{g \cdot h} \qquad \text{for } h \text{ surjective and } p = q \cdot h \tag{23}$$

see next slide.

Post-conditioned metaphors

$$
\begin{array}{ll}
& q? \cdot id \cdot \dfrac{f}{g} \\
= & \qquad \{ \ h \text{ assumed surjective } \} \\
& q? \cdot h \cdot h^\circ \cdot \dfrac{f}{g} \\
= & \qquad \{ \ \text{switch to } \mathrm{WP}\ p \ (12), \text{ cf. } q? \cdot h = h \cdot p? \ \} \\
& h \cdot \underbrace{p? \cdot \dfrac{f}{g \cdot h}}_{X}
\end{array}
$$

The counterpart of (17) is even more immediate:

$$
q? \cdot \frac{f}{g} = \underbrace{q? \cdot \frac{f \cdot h}{g}}_{Y} \cdot h^\circ \qquad \text{for } h \text{ surjective} \tag{24}
$$

Metaphorisms

Metaphorisms are metaphors over **inductive** types.

The tree-like structure of the intermediate type $\mathbf{W}$ will be central to the derivation of **programs** from **divide & conquer** metaphors.

Eventually, $\mathbf{W}$ will disappear, leaving its mark in the algorithmic process only.

This is why this refinement strategy is often known as "changing the **virtual** data structure" (Swierstra and de Moor, 1993).

Now we know more about the types involved — assuming such **initial**, term-algebras exist for functors $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$, respectively.

$$\mathbf{T} \xleftarrow{\mathsf{in}_{\mathbf{T}}} \mathbf{F}\ \mathbf{T}$$

$$\mathbf{W} \xleftarrow{\mathsf{in}_{\mathbf{W}}} \mathbf{G}\ \mathbf{W}$$

$$\mathbf{V} \xleftarrow{inV} \mathbf{H}\ \mathbf{V}$$

Initial algebras

Take $\mathsf{T} \xleftarrow{\mathsf{in}_{\mathsf{T}}} \mathsf{F}\ \mathsf{T}$, for instance. The unique F-homomorphism from the initial $\mathsf{T} \xleftarrow{\mathsf{in}_{\mathsf{T}}} \mathsf{F}\ \mathsf{T}$ to any other (relational) algebra $A \xleftarrow{R} \mathsf{F}\ A$ is written $(\!|R|\!)$

$$
\begin{tikzcd}
\mathsf{T} \arrow[d, "X = (\!|R|\!)"'] \\
A
\end{tikzcd}
\quad \Leftrightarrow \quad
\begin{tikzcd}
\mathsf{T} \arrow[d, "X"'] & \mathsf{F}\ \mathsf{T} \arrow[l, "\mathsf{in}_{\mathsf{T}}"'] \arrow[d, "\mathsf{F}\ X"] \\
A & \mathsf{F}\ A \arrow[l, "R"]
\end{tikzcd}
$$

and is termed **catamorphism** (or **fold**) over R:

$$X = (\!|R|\!) \quad \Leftrightarrow \quad X \cdot \mathsf{in}_{\mathsf{T}} = R \cdot (\mathsf{F}\ X) \tag{25}$$

$$S \cdot (\!|R|\!) = (\!|Q|\!) \quad \Leftarrow \quad S \cdot R = Q \cdot \mathsf{F}\ S \tag{26}$$

$$(\!|R|\!) \cdot \mathsf{in}_{\mathsf{T}} \quad = \quad R \cdot \mathsf{F}\ (\!|R|\!) \tag{27}$$

Sorting example (details)

- $\mathbf{T}$ = finite cons-**lists**, $\mathsf{in}_{\mathbf{T}} = [nil, cons]$.
- $\mathbf{W}$ = binary leaf **trees**, $\mathbf{W} \xleftarrow{\mathsf{in}_{\mathbf{W}}=[leaf,fork]} \mathbf{F}\,\mathbf{W}$ where $\mathbf{F}\,f = id + (f \times f)$.
- $bag = (\!|k|\!)$ — converts finite lists to **bags** (multisets of elements).
- $h = tips = (\!|[singl, conc]|\!)$ where $singl\ x = [x]$ and $conc\ (x, y) = x + \!\!+\ y$. (Surjection h lists the leafs of a tree.)
- $ordered = (\!|[nil, cons] \cdot (id + mn?)|\!)$ where $mn\ (x, xs) = \langle \forall\ x'\ :\ x' \in_{\mathbf{T}} xs :\ x' \leqslant x \rangle$, $\in_{\mathbf{T}}$ denoting list membership.[1]

[1] Predicate $mn\ (x, xs)$ ensures that list $x : xs$ is such that x is at most the minimum of xs, if it exists.

Result needed (**F**-congruences)

Say that equivalence relation R is a **congruence** for algebra $h : \mathbf{F}\ A \rightarrow A$ of functor **F** wherever

$$h \cdot (\mathbf{F}\ R) \ \subseteq \ R \cdot h \qquad i.e. \qquad y\ (\mathbf{F}\ R)\ x \Rightarrow (h\ y)\ R\ (h\ x) \tag{28}$$

hold. Then this is the same as stating:

$$R \cdot h = R \cdot h \cdot (\mathbf{F}\ R) \tag{29}$$

For $h = \mathsf{in}$ initial, (29) is equivalent to:

$$R = (\!| R \cdot \mathsf{in} |\!) \tag{30}$$

(29,30) useful: inductive **equivalence relation** generated by a fold is such that the recursive branch **F** can be added or removed where convenient.

Permutations (example)

For $R = Perm$ (22), for instance, (30) unfolds into

$$Perm \cdot \mathrm{in} = Perm \cdot \mathrm{in} \cdot (F\ Perm)$$

whose useful part is

$$Perm \cdot cons = Perm \cdot cons \cdot (id \times Perm)$$

i.e.

$$y\ Perm\ (a : x) = \langle \exists\ z\ \ :\ \ z\ Perm\ x :\ \ y\ Perm\ (a : z) \rangle$$

written pointwise. In words:

> *Permuting a sequence with at least one element is the same as adding it to the front of a permutation of the tail and permuting again.*

"Easy Split, Hard Join"

Let us use **mergesort** as example, which relies on *leaf trees* based on functor $\mathsf{K}\ f = id + f^2$, as W is of shape $\mathsf{W} = L + \mathsf{W}^2$.

We go back to (24), the instance of (16) which fits the sorting metaphorism:

$$q? \cdot \frac{bag}{bag} = \underbrace{q? \cdot \frac{bag \cdot tips}{bag}}_{Y = (\!|Z|\!)} \cdot tips^{\circ}$$

Recall $tips = (\!| t |\!)$ where [2]

$$\begin{array}{l} t = [singl, conc] \\ singl\ a = [a] \\ conc\ (x, y) = x + y \end{array}$$

[2] Also note that the empty list is treated separately from this scheme.

"Easy Split, Hard Join"

Our aim is to calculate Z, the $\mathbf{K}$-algebra which shall control the *conquer* step:

$$
\begin{array}{ll}
& (\!| Z |\!) = q? \cdot \dfrac{bag}{bag} \cdot (\!| t |\!) \\
\Leftarrow & \quad \{ \ \text{fusion (26) ; functor } \mathbf{K} \ \} \\
& q? \cdot \dfrac{bag}{bag} \cdot t = Z \cdot (\mathbf{K}\ q?) \cdot \mathbf{K}\ \dfrac{bag}{bag} \\
\Leftarrow & \quad \{ \ \text{(29) ; Leibniz} \ \} \\
& q? \cdot \dfrac{bag}{bag} \cdot t = Z \cdot \mathbf{K}\ q?
\end{array}
$$

(Left pending: $\frac{bag}{bag}$ is a $\mathbf{K}$-congruence for algebra t.)

"Easy Split, Hard Join"

Next, we head for a functional implementation $z \subseteq Z$:

$$z \cdot \mathbf{K}\ q? \subseteq q? \cdot \frac{bag}{bag} \cdot t$$

$$\Leftarrow \qquad \{ \text{ cancel } q? \text{ assuming } z \cdot \mathbf{K}\ q? = q? \cdot z \ (12) \ \}$$

$$z \ \subseteq \ \frac{bag \cdot t}{bag}$$

Algebra $z : \mathbf{K}\ \mathbf{T} \rightarrow \mathbf{T}$ should implement (inner) metaphor $\frac{bag \cdot t}{bag}$, essentially requiring that z preserves the bag of elements of the lists involved.

Standard z is the well-known **list merge** function that merges two ordered lists into an ordered list. Check that this behaviour is required by the last assumption above: $z \cdot \mathbf{K}\ q? = q? \cdot z$.

"Hard Split, Easy Join"

Calculations in this case (cf. **quicksort**) are more elaborate.

Recall the overall scheme, tuned for this case:

$$
\begin{tikzcd}
\mathbf{T} & \mathbf{W} \arrow[l, "\llparenthesis \mathit{flatten} \rrparenthesis"'] \arrow[dr, "\llparenthesis \mathit{flatten} \rrparenthesis"'] & & & \mathbf{V} \arrow[lll, "X"'] \arrow[llll, dotted, bend right, "q? \cdot \frac{bag}{bag}"'] \arrow[ddl, "\llparenthesis \mathit{bag} \rrparenthesis"] \\
& & \mathbf{T} \arrow[dr, "\llparenthesis \mathit{bag} \rrparenthesis"'] & & \\
& & & A &
\end{tikzcd}
$$

$\mathbf{W} = 1 + A \times \mathbf{W}^2$ in this case, in which h instantiates to *flatten*, the fold which does **inorder traversal** of $\mathbf{W}$.

Details in (Oliveira, 2015).

AoP, pp.154–155

Quicksort

The so-called 'advanced' sorting algorithms (quicksort, mergesort, heapsort, and so on) all use some form of tree as an intermediate datatype. Here we sketch the development of Hoare's quicksort (Hoare 1962), which follows the path of selection sort quite closely.

Consider the type *tree A* defined by

$$\mathit{tree}\ A \ ::=\ \mathit{null} \mid \mathit{fork}\,(\mathit{tree}\ A, A, \mathit{tree}\ A).$$

The function *flatten* : *list A* ← *tree A* is defined by

$$\mathit{flatten} \ = \ (\![\mathit{nil}, \mathit{join}]\!),$$

where $\mathit{join}\,(x, a, y) = x +\!\!+ [a] +\!\!+ y$. Thus *flatten* produces a list of the elements in a tree in left to right order.

In outline, the derivation of quicksort is

$$\begin{array}{cl}
 & \mathit{ordered} \cdot \mathit{perm} \\
\supseteq & \{\text{since } \mathit{flatten} \text{ is a function}\} \\
 & \mathit{ordered} \cdot \mathit{flatten} \cdot \mathit{flatten}^\circ \cdot \mathit{perm} \\
= & \{\text{claim: } \mathit{ordered} \cdot \mathit{flatten} = \mathit{flatten} \cdot \mathit{inordered} \text{ (see below)}\} \\
 & \mathit{flatten} \cdot \mathit{inordered} \cdot \mathit{flatten}^\circ \cdot \mathit{perm} \\
= & \{\text{converses}\} \\
 & \mathit{flatten} \cdot (\mathit{perm} \cdot \mathit{flatten} \cdot \mathit{inordered})^\circ \\
\supseteq & \{\text{fusion, for an appropriate definition of } \mathit{split}\} \\
 & \mathit{flatten} \cdot (\![\mathit{nil}, \mathit{split}^\circ]\!)^\circ .
\end{array}$$

In quicksort we head for an algorithm expressed as a hylomorphism using trees as an intermediate datatype.

The coreflexive *inordered* on trees is defined by

$$\mathit{inordered} \ = \ (\![\mathit{null}, \mathit{fork} \cdot \mathit{check}]\!)$$

where the coreflexive *check* holds for (x, a, y) if

$$(\forall b : b \ \mathit{intree}\ x \Rightarrow bRa) \ \ \wedge \ \ (\forall b : b \ \mathit{intree}\ y \Rightarrow aRb).$$

The relation *intree* is the membership test for trees. Introducing $\mathsf{F}f = f \times \mathit{id} \times f$ for brevity, the proviso for the fusion step in the above calculation is

To establish this condition we need the coreflexive *check′* that holds for (x, a, y) if

$$(\forall b : b \ \mathit{inlist}\ x \Rightarrow bRa) \ \ \wedge \ \ (\forall b : b \ \mathit{inlist}\ y \Rightarrow aRb).$$

Thus *check′* is similar to *check* except for the switch to lists.

We now reason:

$$\begin{array}{cl}
 & \mathit{perm} \cdot \mathit{flatten} \cdot \mathit{fork} \cdot \mathit{check} \\
= & \{\text{catamorphisms, since } \mathit{flatten} = (\![\mathit{nil}, \mathit{join}]\!)\} \\
 & \mathit{perm} \cdot \mathit{join} \cdot \mathsf{F}\,\mathit{flatten} \cdot \mathit{check} \\
= & \{\text{claim: } \mathsf{F}\,\mathit{flatten} \cdot \mathit{check} = \mathit{check}' \cdot \mathsf{F}\,\mathit{flatten}\} \\
 & \mathit{perm} \cdot \mathit{join} \cdot \mathit{check}' \cdot \mathsf{F}\,\mathit{flatten} \\
= & \{\text{claim: } \mathit{perm} \cdot \mathit{join} = \mathit{perm} \cdot \mathit{join} \cdot \mathsf{F}\,\mathit{perm}\} \\
 & \mathit{perm} \cdot \mathit{join} \cdot \mathsf{F}\,\mathit{perm} \cdot \mathit{check}' \cdot \mathsf{F}\,\mathit{flatten} \\
= & \{\text{claim: } \mathsf{F}\,\mathit{perm} \cdot \mathit{check}' = \mathit{check}' \cdot \mathsf{F}\,\mathit{perm}\text{; functors}\} \\
 & \mathit{perm} \cdot \mathit{join} \cdot \mathit{check}' \cdot \mathsf{F}(\mathit{perm} \cdot \mathit{flatten}) \\
\supseteq & \{\text{taking } \mathit{split} \subseteq \mathit{check}' \cdot \mathit{join}^\circ \cdot \mathit{perm}\} \\
 & \mathit{split}^\circ \cdot \mathsf{F}(\mathit{perm} \cdot \mathit{flatten}).
\end{array}$$

Formal proofs of the three claims are left as exercises. In words, *split* is defined by the rule that if $(y, a, z) = \mathit{split}\ x$, then $y +\!\!+ [a] +\!\!+ z$ is a permutation of x with bRa for all b in y and aRb for all b in z. As in the case of selection sort, we can implement *split* with a catamorphism on non-empty lists:

$$\mathit{split} \ = \ (\![\mathit{base}, \mathit{step}]\!) \cdot \mathit{embed}.$$

The fusion conditions are:

$$\begin{array}{rcl}
\mathit{base} & \subseteq & \mathit{check}' \cdot \mathit{join}^\circ \cdot \mathit{perm} \cdot \mathit{wrap} \\
\mathit{split} \cdot (\mathit{id} \times \mathit{check}' \cdot \mathit{join}) & \subseteq & \mathit{check}' \cdot \mathit{join}^\circ \cdot \mathit{perm} \cdot \mathit{cons}.
\end{array}$$

These conditions are satisfied by taking

$$\begin{array}{rcl}
\mathit{base}\ a & = & ([\,], a, [\,]) \\
\mathit{step}\,(a, (x, b, y)) & = & \begin{cases} ([a] +\!\!+ x, b, y), & \text{if } aRb \\ (x, b, [a] +\!\!+ y), & \text{otherwise.} \end{cases}
\end{array}$$

Finally, appeal to the hylomorphism theorem gives that $X = \mathit{flatten} \cdot (\![\mathit{nil}, \mathit{split}^\circ]\!)^\circ$ is the least solution of the equation

Wrapping up

We have generalized the calculation of **quicksort** given in the AoP textbook (Bird and de Moor, 1997).

Generic calculation of the refinement of **metaphorisms** into **hylomorphisms** by *changing the virtual data structure*.

Metaphorism identified as a class of relational specifications.

Currently working out the **text formatting** metaphorism, where R in $M = (f^{\circ} \cdot g) \upharpoonright R$ includes more than one optimization criterion, e.g.

- fixed maximum number of characters per output line
- maximize number of words per line (minimize white space)

Metaphorism — exploratory concept (recent research topic).

References

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall, 1997.

B.T. Howard. Another iteration on Darlington's 'A Synthesis of Several Sorting Algorithms'. Technical Report KSU CIS 94-8, Department of Computing and Information Sciences, Kansas State University, 1994.

G. Hutton and E. Meijer. Back to basics: Deriving representation changers functionally. *JFP*, 6(1):181–188, 1996.

G. Lakoff and M. Johnson. *Metaphors we live by*. University of Chicago Press, Chicago, 1980. ISBN 978-0-226-46800-6.

J.N. Oliveira. Metaphorisms in programming. In *RAMiCS 2015*, volume 9348 of *LNCS*, pages 171–190. Springer-Verlag, 2015. doi: 10.1007/978-3-319-24704-5_11.

I.A. Richards. *The Philosophy of Rhetoric*. Oxford University Press, 1936.

D. Swierstra and O. de Moor. Virtual data structures. In B. Möller, H. Partsch, and S. Schuman, editors, *Formal Program Development*, volume 755 of *LNCS*, pages 355–371. Springer-Verlag, 1993. ISBN 978-3-540-57499-6.