On the 'divide & conquer' metaphor — the '*quinta essentia*' of programming

J.N. Oliveira

25TH INFOBLENDER SEMINAR

July 13th, 2016

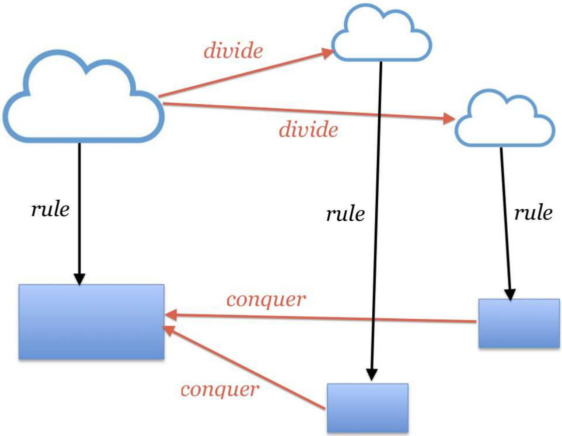
INESC TEC & University of Minho

Grant FP7-ICT 619606

Mac Dictionary ©Apple Inc.

divide and conquer *(or* ***rule****)*

the policy of maintaining control over one's subordinates or subjects by encouraging dissent between them.



Some very good at 'dividing'...

Tortuous Convolvulus
(*Asterix and the Roman Agent*, by Goscinny & Uderzo, Hachette Livre, 1970)

...others (nearly as) good at **conquering**:

What has this to do with programming?

An example, to begin with

Sorting:

$$y \; Sorts \; x \;\; = \;\; y \; Permutes \; x \;\; and \;\; y \; is \; ordered$$

Meaning of clause y *is ordered* is obvious.

Clause y *Permutes* x means "y and x have the same elements, equaly repeated".

EXAMPLE: `"cfbc"` *Permutes* `"fcbc"` *because both have* $\{b \to 1, c \to 2, f \to 1\}$ *elements (a* **bag**, *not a* **set***).*

`"bccf"` *is ordered;* `"cfbc"` *is not (alphabet ordering).*

So, `"bccf"` *Sorts* `"cfbc"`.

An example, to begin with

Sorting:

$$y \ \mathit{Sorts}\ x \quad = \quad y \ \mathit{Permutes}\ x \quad \mathit{and} \quad y \ \mathit{is\ ordered}$$

Meaning of clause *y is ordered* is obvious.

Clause *y Permutes x* means "*y* and *x* have the same elements, equaly repeated".

EXAMPLE: `"cfbc"` *Permutes* `"fcbc"` *because both have* $\{b \rightarrow 1, c \rightarrow 2, f \rightarrow 1\}$ *elements (a* **bag**, *not a* **set***).*

`"bccf"` *is ordered;* `"cfbc"` *is not (alphabet ordering).*

So, `"bccf"` *Sorts* `"cfbc"`.

Example (continued)

Then — why is one of our favourite sorting algorithms [1]

algorithm *quicksort* (A, lo, hi) **is**
  if $lo < hi$ **then**
    $p := pivot\ (A, lo, hi)$
    $left, right := partition\ (A, p, lo, hi)$
    quicksort $(A, lo, left)$
    quicksort $(A, right, hi)$

doubly recursive?

Where is the hint for **recursion** in the specification of the previous slide? Nowhere.

[1] Cf. `https://en.wikipedia.org/wiki/Quicksort#Repeated_elements`.

Example (continued)

And what about the same question, this time for this (parallel!) alternative,

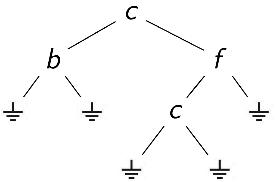
algorithm *mergesort* (A, lo, hi) **is**
  if $lo + 1 < hi$ **then**
    $mid = \lfloor (lo + hi) \; / \; 2 \rfloor$
    fork *mergesort* (A, lo, mid)
    mergesort (A, mid, hi)
    join
    merge (A, lo, mid, hi)

also doubly recursive? [2]

[2] Cf. https://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort.

Example (closing)

Back to *quicksort*, if one inspects the **run-time stack** before the *activation records* of the recursive calls disappear, one will find the pointers there forming a kind of **binary tree**, for instance



when sorting `"cfbc"`.

Textbooks say *quicksort* and *mergesort* are **divide & conquer** algorithms.

How does the **metaphor** with *"divide et impera"* in politics and sociology get into our way?

Metaphors

Metaphors are everywhere

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 MONEY underlies 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:

$$\begin{tikzcd} P \arrow[loop left, "chngt'"] \arrow[dr, "corrupt"'] & & D \arrow[ll, "corrupt^{\circ} \cdot dirty"'] \arrow[loop right, "chng"] \arrow[dl, "dirty"] \\ & \mathbb{B} & \end{tikzcd}$$

$dirty\ (chng\ x) = False$ 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:*

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

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:

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

Only this? No:

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

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

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

($\cdot \upharpoonright \cdot$ 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:

$$
\begin{array}{ll}
& y\ (S \upharpoonright R)\ x \\
\Leftrightarrow & \quad \{\ \text{anticipating definition (21) below}\ \} \\
& y\ (S \cap R\ /\ S^\circ)\ x \\
\Leftrightarrow & \quad \{\ y\ (S \cap R)\ x = y\ S\ x \wedge y\ R\ x\ \} \\
& y\ S\ x \wedge y\ (R\ /\ S^\circ)\ x \\
\Leftrightarrow & \quad \{\ \text{division (more about this below)}\ \} \\
& y\ S\ x \wedge \langle \forall\ y'\ :\ y'\ S\ x :\ y\ R\ y' \rangle
\end{array}
$$

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 $f\ y' = g\ x$ (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

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

meaning

> $x \div y$ *is the* **largest** *number that multiplied by* y *approximates* x

with property

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

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

(Both are so-called 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:

$$y \, \frac{f}{g} \, x \;\Leftrightarrow\; g\, y = f\, x \tag{7}$$

Metaphors = "rational" relations

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

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

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

Moreover, metaphors are closed by intersection:

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

where $(f \triangledown h)\ x = (f\ x, h\ x)$ 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:

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

Post-conditioned metaphors

Special case of metaphor shrinking relevant in the sequel:

$$\frac{f}{g} \upharpoonright \frac{true}{q} \tag{13}$$

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

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

Example of (13):

$$Sort = \frac{bag}{bag} \upharpoonright \frac{true}{ordered} \tag{14}$$

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*:

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

Thus the **sorting** metaphor (14)

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

re-writes to:

$$Sort = ordered? \cdot Perm \quad \textbf{where} \quad Perm = \frac{bag}{bag} \tag{15}$$

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

Divide & conquer metaphors

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

$$M = \frac{f}{g} \upharpoonright R \tag{16}$$

by calculation?

By this law of shrinking

$$(S \cdot f) \upharpoonright R = (S \upharpoonright R) \cdot f \tag{17}$$

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

D&C programming consists in adding an intermediate, 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:
$y'\ (h \cdot h^{\circ})\ y \Leftrightarrow y' = y \wedge \langle \exists\ x\ ::\ y = h\ x \rangle$.

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

$$
\begin{tikzcd}
\mathbf{T} & & \mathbf{W} \arrow[ll, "h"'] \arrow[dr, "h"'] & & \mathbf{V} \arrow[ll, "X"'] \arrow[llll, dotted, bend right, "\frac{f}{g}\upharpoonright R"'] \arrow[ddl, "f"] \\
& & & \mathbf{T} \arrow[dr, "g"'] & \\
& & & & A
\end{tikzcd}
\tag{18}
$$

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 $h : \mathbf{W} \rightarrow \mathbf{V}$ in

$$
\begin{tikzcd}
\mathbf{T} \arrow[ddr, "g"'] & & & \mathbf{W} \arrow[lll, "Y"'] \arrow[dl, "h"] & \mathbf{V} \arrow[l, "h^\circ"'] \arrow[llll, dotted, bend right, "\frac{f}{g} \restriction R"'] \\
& & \mathbf{V} \arrow[dl, "f"] & & \\
& A & & &
\end{tikzcd}
\tag{19}
$$

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

Note how intermediate type $\mathbf{W}$ acts as **representation** of $\mathbf{T}$ or $\mathbf{V}$ in, respectively, (18) and (19) — h acts as a typical data refinement **abstraction** function.

Examples again, please

Quicksort — example of (18):

$$
\begin{tikzcd}
{[a]} & \mathbf{BTree}\ a \arrow[l, "flatten"'] \arrow[dr, "flatten"'] & & {[a]} \arrow[ll, "X"'] \arrow[dd, "bag"] \arrow[lll, dotted, bend right, "\frac{bag}{bag} \upharpoonright \frac{true}{ordered}"'] \\
 & & {[a]} \arrow[dr, "bag"'] & \\
 & & & Bag\ a
\end{tikzcd}
$$

Mergesort — example of (19):

$$
\begin{tikzcd}
{[a]} \arrow[ddr, "bag"'] & & & \mathbf{LTree}\ a \arrow[lll, "Y"'] \arrow[dl, "tips"] & {[a]} \arrow[l, "tips^{\circ}"'] \arrow[llll, dotted, bend right, "\frac{bag}{bag} \upharpoonright \frac{true}{ordered}"'] \\
 & & {[a]} \arrow[dl, "bag"] & & \\
 & Bag\ a & & &
\end{tikzcd}
$$

Another (a bit degenerate) example

Matrix-matrix multiplication (*mmm*) — example of (19):

$$\begin{tikzcd}
Mat\ a \arrow[dr, "id"'] & & \mathsf{LTree}\,(Vec\ a, Vec\ a) \arrow[ll, "Y"'] \arrow[d, "pack"] & (Mat\ a, Mat\ a) \arrow[l, "unpack = pack^{\circ}"'] \arrow[lll, dotted, bend right, "\frac{mmm}{id} \upharpoonright \frac{true}{true}"'] \\
& Mat\ a & (Mat\ a, Mat\ a) \arrow[l, "mmm"] &
\end{tikzcd}$$

Equation is $Y \cdot unpack = mmm$, since $\frac{mmm}{id} \upharpoonright \frac{true}{true} = mmm$.

(Recall Google Map-Reduce.)

Divide & conquer metaphors

Let us calculate "**conquer**" step Y (19) in the first place:

$$
\begin{array}{cl}
& \frac{f}{g} \upharpoonright R \\
= & \{ \text{ identity of composition } \} \\
& (\frac{f}{g} \upharpoonright R) \cdot id \\
= & \{ \ h \text{ assumed to be a surjection, } h \cdot h^{\circ} = id \ \} \\
& (\frac{f}{g} \upharpoonright R) \cdot h \cdot h^{\circ} \\
= & \{ \text{ law (17) } \} \\
& \underbrace{(\frac{f \cdot h}{g} \upharpoonright R)}_{Y} \cdot h^{\circ}
\end{array}
$$

Divide & conquer metaphors

Altogether:

$$\frac{f}{g} \upharpoonright R \;=\; \left(\frac{f \cdot h}{g} \upharpoonright R\right) \cdot h^\circ \qquad \text{for } h \text{ surjective} \tag{20}$$

In a diagram, completing (19):

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

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

$$S \upharpoonright R \;=\; S \cap R/S^{\circ}. \qquad (21)$$

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"

$$
\begin{array}{cl}
& \dfrac{f}{g} \upharpoonright R \\
= & \qquad \{ \ (21)\ ;\ \text{converse of a metaphor (8)} \ \} \\
& \dfrac{f}{g} \cap R \mathbin{/} \dfrac{g}{f} \\
= & \qquad \{ \ h \text{ assumed to be a surjection, } \rho\, h = h \cdot h^{\circ} = id \ \} \\
& h \cdot h^{\circ} \cdot (\dfrac{f}{g} \cap R \mathbin{/} \dfrac{g}{f}) \\
= & \qquad \{ \ \text{injective } h^{\circ} \text{ distributes by } \cap \ \} \\
& h \cdot (\dfrac{f}{g \cdot h} \cap h^{\circ} \cdot R \mathbin{/} \dfrac{g}{f})
\end{array}
$$

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

"Hard Split, Easy Join"

We recall property

$$R \,/\, \frac{g}{f} = (R \,/\, g) \cdot f \tag{22}$$

— which follows from (4) — and carry on:

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

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:

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

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:

$$
\frac{f}{g} \upharpoonright R \quad = \quad h \cdot (\frac{f}{g \cdot h} \cap h^{\circ} \cdot (R \,/\, g) \cdot f) \qquad \text{for } h \text{ surjective} \quad (23)
$$

Back to post-conditioned metaphors

Recall (15)

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

from slide 28.

For this special case, "Hard Split, Easy Join" (23) boils down to

$$q? \cdot \frac{f}{g} = h \cdot p? \cdot \frac{f}{g \cdot h} \qquad \text{for } h \text{ surjective and } p = q \cdot h \tag{24}$$

see next slide.

Back to post-conditioned metaphors

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

The counterpart of (20) is even more immediate:

$$
q? \cdot \frac{f}{g} = \underbrace{q? \cdot \frac{f \cdot h}{g}}_{Y} \cdot h^\circ \qquad \text{for } h \text{ surjective} \tag{25}
$$

What happens next?

In a diagram

Case (18), for instance:

$$
\begin{tikzcd}
\mathbf{T} \arrow[loop left, "q?"] & \mathbf{G}\,\mathbf{T} \arrow[l, "k"'] \\
\mathbf{W} \arrow[loop left, "p?"] \arrow[u, "(\!|k|\!)"] & \mathbf{G}\,\mathbf{W} \arrow[l, "\mathsf{in}_{\mathbf{W}}"'] \arrow[u, "\mathbf{G}\,(\!|k|\!)"'] \\
\mathbf{T} \arrow[u, "[\!(D)\!]"] \arrow[r, "D"'] & \mathbf{G}\,\mathbf{T} \arrow[u, "\mathbf{G}\,[\!(D)\!]"']
\end{tikzcd}
$$

Legend:

$h = (\!|k|\!)$ — k *will be the final* **conquer** *step*

$X = [\!(D)\!]$ — D *will be the final* **divide** *step*

Final D&C program will be as simple as

$$P = k \cdot (\mathbf{G}\ P) \cdot D$$

This is known as a (relational) **hylomorphism**.

Technical details in the appendix and in (Oliveira, 2015).

Background — AoP, pp.154–155

154 6 / *Recursive Programs*

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 $\mathit{flatten} : \mathit{list}\ A \leftarrow \mathit{tree}\ A$ 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

6.6 / *Sorting by selection* 155

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 broad class of relational specifications.

Merit of **relation algebra** — typed, calculational and productive.

Overall aim: **scientific** software engineering (as SE "founding fathers" planned in 1969...)

Annex

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

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

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

$$X = (\!|R|\!) \quad \Leftrightarrow \quad X \cdot \mathsf{in}_\mathbf{T} = R \cdot (\mathbf{F}\ X) \tag{26}$$

$$S \cdot (\!|R|\!) = (\!|Q|\!) \quad \Leftarrow \quad S \cdot R = Q \cdot \mathbf{F}\ S \tag{27}$$

$$(\!|R|\!) \cdot \mathsf{in}_\mathbf{T} \quad = \quad R \cdot \mathbf{F}\ (\!|R|\!) \tag{28}$$

Sorting example (details)

- $\mathbf{T}$ = finite cons-**lists**, $\mathsf{in}_{\mathbf{T}} = [nil, cons]$.
- $\mathbf{W}$ = binary leaf **trees**, $\mathbf{W} \xleftarrow{\mathsf{in}_{\mathbf{W}} = [leaf, \mathbf{fork}]} \mathbf{F}\ \mathbf{W}$ 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.[3]

[3] 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 $h : \mathbf{F}\ A \to A$ of functor **F** wherever

$$h \cdot (\mathbf{F}\ R) \ \subseteq\ R \cdot h \qquad i.e. \qquad y\ (\mathbf{F}\ R)\ x \Rightarrow (h\ y)\ R\ (h\ x) \tag{29}$$

hold. Then this is the same as stating:

$$R \cdot h = R \cdot h \cdot (\mathbf{F}\ R) \tag{30}$$

For $h = \mathsf{in}$ initial, (30) is equivalent to:

$$R = (\!|R \cdot \mathsf{in}|\!) \tag{31}$$

(30,31) 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 $R = Perm$ (15), for instance, (31) unfolds into

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

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 (25), the instance of (19) 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 [4]

$$t = [singl, conc]$$
$$singl\ a = [a]$$
$$conc\ (x, y) = x +\!\!+\ y$$

[4] 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:

$$
\begin{array}{ll}
& (\!|Z|\!) = q? \cdot \frac{bag}{bag} \cdot (\!|t|\!) \\
\Leftarrow & \quad \{ \text{ fusion (27) ; functor } \mathbf{K} \ \} \\
& q? \cdot \frac{bag}{bag} \cdot t = Z \cdot (\mathbf{K}\ q?) \cdot \mathbf{K}\ \frac{bag}{bag} \\
\Leftarrow & \quad \{ \text{ (30) ; Leibniz } \} \\
& q? \cdot \frac{bag}{bag} \cdot t = Z \cdot \mathbf{K}\ q?
\end{array}
$$

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

"Easy Split, Hard Join"

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

$$
\begin{array}{ll}
& z \cdot \mathbf{K}\ q? \subseteq q? \cdot \dfrac{bag}{bag} \cdot t \\
\Leftarrow & \quad \{ \text{ cancel } q? \text{ assuming } z \cdot \mathbf{K}\ q? = q? \cdot z \ (12) \ \} \\
& z \subseteq \dfrac{bag \cdot t}{bag}
\end{array}
$$

Algebra $z : \mathbf{K}\ \mathbf{T} \to \mathbf{T}$ 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:

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

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

Details in (Oliveira, 2015).

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.