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

J.N. Oliveira

INFOBLENDE SEMINAR SERIES 2016

July 13th, 2016

INESC TEC & University of Minho

Grant FP7-ICT 619606
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’...

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

**Tortuous Convolvulus**
What has this to do with programming?
An example, to begin with

Sorting:

\[ y \text{ Sorts } x = y \text{ Permutes } x \text{ and } y \text{ is ordered} \]

Meaning of clause \( y \text{ is ordered} \) is obvious.

Clause \( y \text{ Permutes } x \) means “\( y \) and \( x \) have the same elements, equaly repeated”.

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

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

So, "bccf" Sorts "cfbc".
An example, to begin with

Sorting:

\[ y \text{ Sorts } x = y \text{ Permutes } x \text{ and } y \text{ is ordered} \]

Meaning of clause \( y \text{ is ordered} \) is obvious.

Clause \( y \text{ Permutes } x \) means “\( y \) and \( x \) have the same elements, equally 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

\[
\textbf{algorithm quicksort} \ (A, lo, hi) \ \textbf{is}
\]
\[
\text{if } lo < hi \ \text{then}
\]
\[
p \coloneqq \text{pivot} \ (A, lo, hi)
\]
\[
\text{left, right} \coloneqq \text{partition} \ (A, p, lo, hi)
\]
\[
\text{quicksort} \ (A, lo, \text{left})
\]
\[
\text{quicksort} \ (A, \text{right}, hi)
\]

\textbf{doubly} recursive?

Where is the hint for \textbf{recursion} in the specification of the previous slide? Nowhere.

Example (continued)

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

```plaintext
algorithm mergesort (A, lo, hi) is
    if lo + 1 < hi then
        mid = ⌊(lo + hi) / 2⌋
        fork mergesort (A, lo, mid)
        mergesort (A, mid, hi)
    join
    merge (A, lo, mid, hi)
```

also doubly recursive? ²

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

![Binary tree diagram](image)

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).
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 $P = \text{politician}$ and $D = \text{diaper}$:

\[
\begin{array}{c}
\overset{\text{dirty} \ (\text{chg} \ x)}{\overset{\text{corrupt} \circ \cdot \text{dirty}}{P}} \leftarrow \overset{\text{dirty}}{\overset{\text{corrupt}}{B}} \rightarrow \overset{\text{corrupt} \circ \cdot \text{dirty}}{D} \rightarrow \overset{\text{chng}}{\text{chngt'}}
\end{array}
\]

$\text{dirty} (\text{chg} \ x) = \text{False}$ induces $\text{chngt'}$ 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”

$$T \xrightarrow{f} A \xleftarrow{g} V$$

(1)

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

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

$$T \leftarrow f \circ g \rightarrow V$$

— the “$T$ is $V$” metaphor — which leaves $A$ implicit.

----------------------------------------

Remarks on notation:

- $x f^o y$ means the same as $y f x$, that is $y = f x$.
- In general, $x R^o y$ asserts the same as $y R x$.
- Relational composition:
  $$y (R \cdot S) x \iff \exists z :: y R z \land z S x$$
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:

\[ \text{Format} \quad \text{String} \]

\[ \text{words} \quad \text{Format} \quad \text{String} \]

\[ \text{String} \]

Only this? No:

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

\[
\text{Format} = ((\gg\text{words})^\circ \cdot \text{words}) \uparrow R
\]  \hspace{1cm} (2)

as specified by some convenient optimization criterion \( R \) (\( \cdot \uparrow \cdot \) 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:

\[ y \left( f \circ g \right) x \]
\[ \Leftrightarrow \{ \text{composition and converse} \} \]
\[ \langle \exists a : a f y : a g x \rangle \]
\[ \Leftrightarrow \{ \text{functions } f \text{ and } g \} \]
\[ \langle \exists a : a = f y : a = g x \rangle \]
\[ \Leftrightarrow \{ \text{one-point quantification} \} \]
\[ f y = g x \]
Metaphorical specifications

**Variant part:**

$$y \ (S \upharpoonright R) \ x$$

$$\iff \ \{ \text{anticipating definition (21) below} \}$$

$$y \ (S \cap R / S^o) \ x$$

$$\iff \ \{ \ y \ (S \cap R) \ x = y \ S \ x \land y \ R \ x \ \}$$

$$y \ S \ x \land y \ (R / S^o) \ x$$

$$\iff \ \{ \text{division (more about this below)} \}$$

$$y \ S \ x \land \langle \forall \ y' : y' \ S \ x : y \ R \ y' \rangle$$

Altogether:

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

**Invariant + Variant parts:**

\[
M = (f^\circ \cdot g) \upharpoonright R
\]

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**).
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

\[ z \times y \leq x \iff z \leq x \div y \]

meaning

\[ x \div y \text{ is the largest number that multiplied by } y \text{ approximates } x \]

with property

\[ Q \cdot S \subseteq R \iff Q \subseteq R / S \quad (4) \]

— \( R / S \) 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

\[
\frac{S}{R} = (S^\circ / R^\circ)^\circ \cap R^\circ / S^\circ
\]  \hspace{1cm} (5)

Pointwise:

\[
b \frac{S}{R} c \iff \langle \forall a :: a R b \iff a S c \rangle
\]  \hspace{1cm} (6)

In the case of functions:

\[
y \frac{f}{g} x \iff g y = f x
\]  \hspace{1cm} (7)
Metaphors = "rational" relations

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

\[
\left( \frac{f}{g} \right)^\circ = \frac{g}{f}, \quad \frac{f}{id} = f
\]

\[
\frac{id}{g} \cdot \frac{h}{k} \cdot \frac{f}{id} = \frac{h \cdot f}{k \cdot g}
\]

Moreover, metaphors are closed by intersection:

\[
\frac{f}{g} \cap \frac{h}{k} = \frac{f \upharpoonright h}{g \upharpoonright k}
\]

where $(f \upharpoonright h) x = (f x, h x)$ is the pairing operator.
Predicates and diagonals

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

\[
y \frac{true}{q} \quad x = q \; y
\]

where \textit{true} is the everywhere-true predicate.

Put in another way, we can encode predicates in the form of \textbf{diagonal} metaphors:

\[
p? = \text{id} \cap \frac{true}{p}
\]

that is,

\[
y (p?) \; x \iff (y = x) \land (p \; y)
\]

holds.
Weakest preconditions

More generally,

\[ f \cap \frac{\text{true}}{q} = q? \cdot f \quad \quad f \cap \frac{p}{\text{true}} = f \cdot p? \]

hold. Moreover, equality

\[ f \cap \frac{p}{\text{true}} = \frac{\text{true}}{q} \cap f \]

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

Another way to write this:

\[ f \cdot p? = q? \cdot f \quad \Leftrightarrow \quad p = q \cdot f \quad (12) \]
Post-conditioned metaphors

Special case of metaphor shrinking relevant in the sequel:

\[
\frac{f}{g} \mid true \quad q
\]  

(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):

\[
Sort = \frac{bag}{bag} \uparrow \frac{true}{ordered}
\]  

(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 \textit{post-conditioned}:

\[
\frac{f \upharpoonright \text{true}}{g \upharpoonright \text{true}} = q? \cdot \frac{f}{g}
\]

Thus the \textbf{sorting} metaphor (14)

\[
\text{Sort} = \frac{\text{bag} \upharpoonright \text{true}}{\text{bag} \upharpoonright \text{ordered}}
\]

re-writes to:

\[
\text{Sort} = \text{ordered}? \cdot \text{Perm} \quad \text{where} \quad \text{Perm} = \frac{\text{bag}}{\text{bag}}
\] (15)

So \( y \text{ Perm } x \) means that \( y \) is a \textit{permutation} of \( x \).
Can we derive **programs** from a given **metaphor**

\[
M = \frac{f}{g} \upharpoonright R
\]  

by calculation?

By this law of shrinking

\[
(S \cdot f) \upharpoonright R = (S \upharpoonright R) \cdot f
\]  

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 $W$ between vehicle and tenor,

\[ T \leftarrow W \leftarrow V \]

intended to gain control of the “pipeline”.

This can be done in two ways. Assume a surjection $h : W \rightarrow T$ on the tenor side, that is, $\rho h = h \cdot h^0 = id$.

---

**Range of a function:**

\[ y' (h \cdot h^0) y \iff y' = y \land (\exists x :: y = h x) \]
Divide & conquer metaphors

Then $h : W \rightarrow T$ provides an intermediate representation of the tenor.

As we shall see shortly, the splitting works as follows

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 ($V$) and the intermediate type ($W$).
Alternatively, we can imagine *surjection* \( h \) working on the *vehicle* side, say \( h : W \rightarrow V \) in

\[
\begin{array}{c}
T \\
\downarrow g \\
A \\
\downarrow f \\
V \\
\end{array}
\quad \begin{array}{c}
\quad Y \\
\uparrow f^{\uparrow R} \\
W \\
\uparrow h \\
V \\
\end{array}
\]  

(19)

and try and find relation \( Y \) such that \( Y \cdot h^\circ = f^{\uparrow R} \).

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

Quicksort — example of (18):

Mergesort — example of (19):
Another (a bit degenerate) example

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

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

(Recall Google Map-Reduce.)
Divide & conquer metaphors

Let us calculate “conquer” step $Y$ (19) in the first place:

\[
\begin{align*}
  f \quad \uparrow R \\
  g
  &= \{ \text{identity of composition} \} \\
  (f \uparrow R) \cdot id
  &= \{ h \text{ assumed to be a surjection, } h \cdot h^o = id \} \\
  (f \uparrow R) \cdot h \cdot h^o
  &= \{ \text{law (17)} \} \\
  (f \cdot h \uparrow R) \cdot h^o
\end{align*}
\]
Divide & conquer metaphors

Altogether:

$$\frac{f}{g} \upharpoonright R = (\frac{f \cdot h}{g} \upharpoonright R) \cdot h^\circ$$

for $h$ surjective \hspace{1cm} (20)

In a diagram, completing (19):

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

\[ S \upharpoonright R = S \cap R/S^\circ. \]  \hspace{1cm} (21)

to solve equation

for \( X \) (next slide).
“Hard Split, Easy Join”

\[
\begin{align*}
\begin{array}{c}
f \\
g \\
\end{array}
& \\
\uparrow R \\
\hline
\begin{array}{c}
f \\
g \\
\end{array} \\
\cap R / \begin{array}{c} f \\
g \end{array} \\
\hline
= & \quad \{ (21) \ ; \ \text{converse of a metaphor (8)} \} \\
\begin{array}{c}
f \\
g \end{array}
\cap R / \begin{array}{c} f \\
g \end{array} \\
\hline
h \cdot h^\circ \cdot \left( \begin{array}{c} f \\
g \end{array} \cap R / \begin{array}{c} f \\
g \end{array} \right) \\
= & \quad \{ \text{injective } h^\circ \text{ distributes by } \cap \} \\
= & \quad \{ \text{injective } h^\circ \text{ distributes by } \cap \} \\
\end{align*}
\]

(Thumb rule: the converse of a function is always \textbf{injective}.)

\[
\begin{align*}
\begin{array}{c}
f \\
g \end{array}
& \\
\uparrow R \\
\hline
\begin{array}{c}
f \\
g \end{array} \\
\cap R / \begin{array}{c} f \\
g \end{array} \\
\hline
h \cdot h^\circ \cdot \left( \begin{array}{c} f \\
g \end{array} \cap R / \begin{array}{c} f \\
g \end{array} \right) \\
= & \quad \{ \text{injective } h^\circ \text{ distributes by } \cap \} \\
= & \quad \{ \text{injective } h^\circ \text{ distributes by } \cap \} \\
\end{align*}
\]
“Hard Split, Easy Join”

We recall property

\[ R / \frac{g}{f} = (R / g) \cdot f \]  \hspace{1cm} (22)

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

\[ h \cdot \left( \frac{f}{g \cdot h} \cap h^\circ \cdot R / \frac{g}{f} \right) \]

\[ = \{ \text{above ; shunting} \} \]

\[ h \cdot \left( \frac{f}{g \cdot h} \cap h^\circ \cdot (R / g) \cdot f \right) \]

\[ \underbrace{X}_{\text{divide step}} \]

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:

\[
\begin{align*}
\text{let } a &= f \, v \in \\
& (g \, (h \, w) = a) \land \langle \forall \ t : a = g \, t : (h \, w) \, R \, t \rangle
\end{align*}
\]

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:

\[
\frac{f \upharpoonright R}{g} = h \cdot (\frac{f}{g \cdot h} \cap h^\circ \cdot (R \, / \, g) \cdot f) \quad \text{for } h \text{ surjective} \quad (23)
\]
Recall (15)

\[
\text{Sort} = \text{ordered } \cdot \text{Perm} \quad \text{where} \quad \text{Perm} = \frac{\text{bag}}{\text{bag}}
\]

from slide 28.

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

\[
q? \cdot \frac{f}{g} = h \cdot p? \cdot \frac{f}{g \cdot h} \quad \text{for } h \text{ surjective and } p = q \cdot h \quad (24)
\]

see next slide.
Back to post-conditioned metaphors

\[
q? \cdot id \cdot \frac{f}{g} = \begin{cases} \text{ } h \text{ assumed surjective} \end{cases}
q? \cdot h \cdot h^\circ \cdot \frac{f}{g}
= \begin{cases} \text{ switch to } \wp p (12), \text{ cf. } q? \cdot h = h \cdot p? \end{cases}
\]

\[
h \cdot p? \cdot \frac{f}{g \cdot h}
\quad \quad \text{X}
\]

The counterpart of (20) is even more immediate:

\[
q? \cdot \frac{f}{g} = q? \cdot \frac{f}{g} \cdot h^\circ \quad \text{for } h \text{ surjective} \quad (25)
\]
What happens next?
In a diagram

Case (18), for instance:

Legend:

\[ h = (\lfloor k \rfloor) \rightarrow k \]
will be the final \textbf{conquer} step

\[ X = \lfloor D \rfloor \rightarrow D \]
will be the final \textbf{divide} step

Final D&C program will be as simple as

\[ P = k \cdot (G \cdot P) \cdot D \]

This is known as a (relational) \textbf{hylomorphism}.

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

\[ \text{tree } A ::= \text{null} \mid \text{fork} \left( \text{tree } A, A, \text{tree } A \right) \]

The function flatten : list A → tree A is defined by

\[ \text{flatten} = \left[ \text{nil}, \text{join} \right], \]

where 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

\[
\begin{align*}
\text{ordered} & \cdot \text{perm} \\
\quad & \text{[since flatten is a function]} \\
\quad & \text{ordered} \cdot \text{flatten} \cdot \text{flat} \cdot \text{check} \\
\quad & \text{[claim: ordered} \cdot \text{flatten} = \text{flatten} \cdot \text{inordered} \text{[see below]} \text{]} \\
\quad & \text{flat} \cdot \text{inordered} \cdot \text{flatten} \cdot \text{check} \\
\quad & \text{[converses]} \\
\quad & \text{flat} \cdot \text{perm} \cdot \text{flatten} \cdot \text{inordered} \\
\quad & \quad \text{[fusion, for an appropriate definition of split]} \\
\quad & \text{flatten} \cdot \left[ \text{nil}, \text{split}^* \right] \\
\end{align*}
\]

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

\[ \text{inordered} = \left[ \text{null}, \text{fork} \cdot \text{check} \right] \]

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

\[ \forall b \in \text{intree} \ x \Rightarrow bRa \land \forall b \in \text{intree} \ y \Rightarrow aRb. \]

The relation intree is the membership test for trees. Introducing \( Ff = f \times \text{id} \times f \) 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

\[ \forall b : b \text{ intree } x \Rightarrow bRb \land \forall b : b \text{ intree } y \Rightarrow aRb. \]

Thus check' is similar to check except for the switch to lists.

We now reason:

\[
\begin{align*}
\text{perm} \cdot \text{flat} \cdot \text{fork} \cdot \text{check} \\
= \left\{ \text{catamorphisms, since flatten} = \left[ \text{nil}, \text{join} \right] \right\} \\
\text{perm} \cdot \text{join} \cdot \text{f} \cdot \text{flatten} \cdot \text{check} \\
= \left\{ \text{claim: flatten} \cdot \text{check} = \text{check} \cdot \text{f} \cdot \text{flatten} \right\} \\
\text{perm} \cdot \text{join} \cdot \text{check'} \cdot \text{f} \cdot \text{flatten} \\
= \left\{ \text{claim: perm} \cdot \text{join} = \text{perm} \cdot \text{join} \cdot \text{f} \cdot \text{perm} \right\} \\
\text{perm} \cdot \text{join} \cdot \text{f} \cdot \text{perm} \cdot \text{check'} \cdot \text{f} \cdot \text{flatten} \\
= \left\{ \text{claim: perm} \cdot \text{check'} = \text{check} \cdot \text{f} \cdot \text{perm} \cdot \text{func.; functions} \right\} \\
\text{perm} \cdot \text{join} \cdot \text{check'} \cdot \text{f} \cdot \text{perm} \cdot \text{flat} \\
\quad \quad \left\{ \text{taking split} \subseteq \text{check'} \cdot \text{join}^* \cdot \text{perm} \right\} \\
\text{split} = \left[ \text{base}, \text{step} \right] \cdot \text{embed}. \\
\end{align*}
\]

The fusion conditions are:

\[ \text{split} \cdot \text{id} \cdot \text{check'} \cdot \text{join} \subseteq \text{check'} \cdot \text{join}^* \cdot \text{perm} \cdot \text{cons}. \]

These conditions are satisfied by taking

\[
\begin{align*}
\text{split} \left( a, \left( x, y \right) \right) & = \left\{ \left( x \right), \left( a \right) \right\}, \quad \text{if aRb} \\
& = \left\{ \left( x, b \right), \left( a \right) + y \right\}, \quad \text{otherwise}. \\
\end{align*}
\]

Finally, appeal to the hylomorphism theorem gives that \( X = \text{flatten} \cdot \left[ \text{nil}, \text{split}^* \right] \) is the least solution of the equation
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 $W$ will be central to the derivation of programs from divide & conquer metaphors.

Eventually, $W$ 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 $F$, $G$ and $H$, respectively.

\[
\begin{align*}
T & \xrightarrow{\text{in}_T} F T \\
W & \xrightarrow{\text{in}_W} G W \\
V & \xrightarrow{\text{in}_V} H V
\end{align*}
\]
Initial algebras

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

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

\[
X = (|R|) \Leftrightarrow X \cdot \text{in}_T = R \cdot (F X) \quad (26)
\]
\[
S \cdot (|R|) = (|Q|) \Leftrightarrow S \cdot R = Q \cdot F S \quad (27)
\]
\[
(|R|) \cdot \text{in}_T = R \cdot F (|R|) \quad (28)
\]
Sorting example (details)

- **T** = finite cons-lists, \( \text{in}_T = [\text{nil}, \text{cons}] \).

- **W** = binary leaf trees, \( \text{in}_W = [\text{leaf}, \text{fork}] \) where \( F f = \text{id} + (f \times f) \).

- **bag** = \( \langle k \rangle \) — converts finite lists to bags (multisets of elements).

- **h** = **tips** = \( \langle [\text{singl}, \text{conc}] \rangle \) where \( \text{singl} x = \{x\} \) and \( \text{conc} (x, y) = x + y \). (Surjection \( h \) lists the leaves of a tree.)

- **ordered** = \( \langle [\text{nil}, \text{cons}] \cdot (\text{id} + \text{mn}?) \rangle \) where \( \text{mn} (x, xs) = \langle \forall x' : x' \epsilon_T xs : x' \leq x \rangle \), \( \epsilon_T \) denoting list membership.\(^3\)

---

\(^3\)Predicate \( \text{mn} (x, xs) \) 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 $h : F A \to A$ of functor $F$ wherever

$$h \cdot (F R) \subseteq R \cdot h \quad \text{i.e.} \quad y (F R) x \Rightarrow (h y) R (h x) \quad (29)$$

hold. Then this is the same as stating:

$$R \cdot h = R \cdot h \cdot (F R) \quad (30)$$

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

$$R = (R \cdot \text{in}) \quad (31)$$

(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

\[
Perm \cdot \text{in} = Perm \cdot \text{in} \cdot (F \ Perm)
\]

whose useful part is

\[
Perm \cdot \text{cons} = Perm \cdot \text{cons} \cdot (id \times Perm)
\]

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 $\mathbf{K} f = id + f^2$, as $\mathbf{W}$ is of shape $\mathbf{W} = L + \mathbf{W}^2$.

We go back to (25), the instance of (19) which fits the sorting metaphorism:

\[
q? \cdot \frac{\text{bag}}{\text{bag}} = q? \cdot \frac{\text{bag} \cdot \text{tips}}{\text{bag}} \cdot \text{tips}^\circ \\
\underbrace{Y = (|Z|)}
\]

Recall $\text{tips} = (|t|)$ where \(^4\)

\[
t = [\text{singl}, \text{conc}] \\
\text{singl} \ a = [a] \\
\text{conc} \ (x, y) = x \oplus y
\]

\(^4\)Also note that the empty list is treated separately from this scheme.
Our aim is to calculate $Z$, the $K$-algebra which shall control the conquer step:

$$\langle Z \rangle = q? \cdot \frac{\text{bag}}{\text{bag}} \cdot \langle t \rangle$$

$$\Leftarrow \{ \text{fusion (27); functor } K \}$$

$$q? \cdot \frac{\text{bag}}{\text{bag}} \cdot t = Z \cdot (K q?) \cdot K \frac{\text{bag}}{\text{bag}}$$

$$\Leftarrow \{ (30); \text{Leibniz} \}$$

$$q? \cdot \frac{\text{bag}}{\text{bag}} \cdot t = Z \cdot K q?$$

(Left pending: $\frac{\text{bag}}{\text{bag}}$ is a $K$-congruence for algebra $t$.)
“Easy Split, Hard Join”

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

\[
z \cdot K \ q? \subseteq q? \cdot \frac{\text{bag} \cdot t}{\text{bag}} \cdot t
\]

\[\iff \{ \text{cancel } q? \text{ assuming } z \cdot K \ q? = q? \cdot z \ (12) \ \}
\]

\[
z \subseteq \frac{\text{bag} \cdot t}{\text{bag}}
\]

Algebra \( z : K \ T \rightarrow T \) should implement (inner) metaphor \( \frac{\text{bag} \cdot t}{\text{bag}} \), 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: \( z \cdot K \ q? = q? \cdot z \)
“Hard Split, Easy Join”

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

Recall the overall scheme, tuned for this case:

\[ W = 1 + A \times W^2 \]

in this case, in which \( h \) instantiates to \textit{flatten}, the fold which does \textbf{inorder traversal} of \( W \).

Details in (Oliveira, 2015).
References


