Founded 1989
Editor-in-chief:
Johan T. Jeuring

The Squiggolist

The Squiggolist, Volume 2, Number 2, November 1991

AMELAND — Mysterio
ning. Hollum police is inve
rious case of food-poisonir
keeper of restaurant 'De
ing questioned about t
t ree of the four meals
fact that the thre
come from a
ate pork
as the per
nily around, talkin

AMSTERDAM — Task restriction at
The management of the CWI has sta
ussion on restricting the task of
employes by the CWI. A
agement said: 'Lots of
e staff is waisted
ative work,
nising
co
consult

FLORENCE — New composition discovered. The Museo san Marco in Florence has found a painting ascribed to Beato Angelico o Fra' Angelico that is considered to be the counterpart of the 'Composition of Angels' (1437). The painting is called 'Demonic Composition' and it pictures a repellent rather corpulent male figure with a grey beard and grey hairs worn in a pigtail, that chews quid. The painting proves the associ of Fra' Angelico with demonic compos

Free style spec wrestling II: preorders

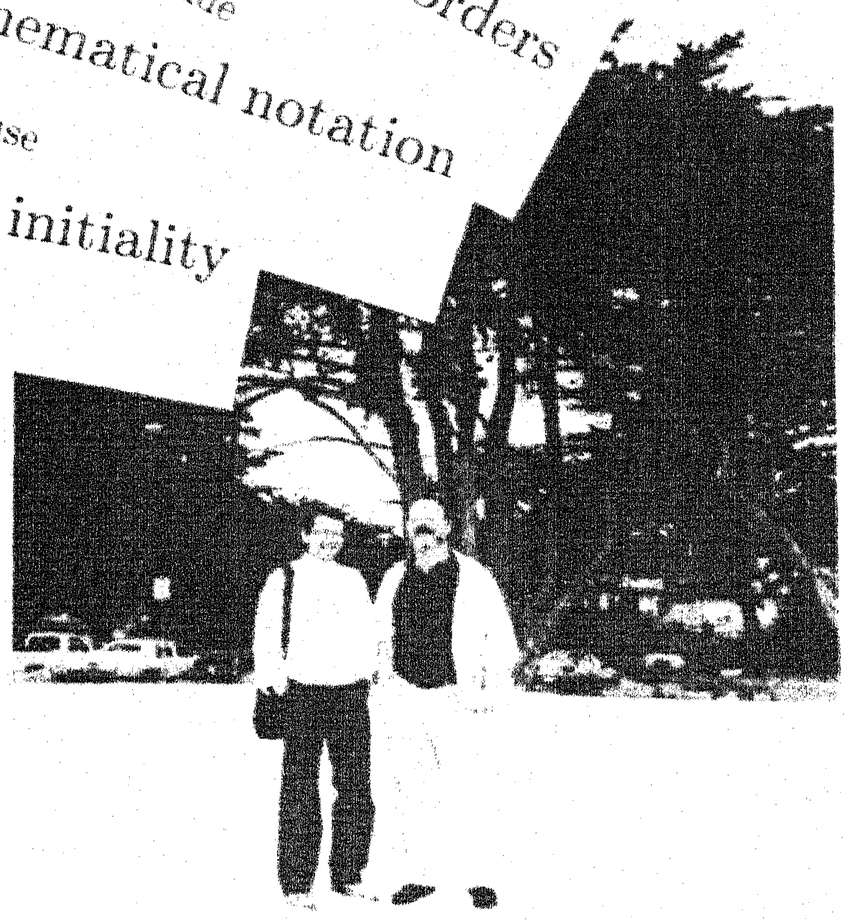
Some ergonomics of mathematical notation

Jaap van der Woude

Roland Backhouse

Many-sorted algebras and initiality

Maarten Fokkinga



The smallest upravel

Richard Bird

be considered a morphism o

On demonic composition

Roland Backhouse

quirements, must be sent to the
for the
A
T

Sums and differentials

Roland Backhouse

BR ELLES — Protest-meeting PH
students. PhD student
Benelux
14 to dis
plorable p
exploited b
to carry ou
ployer bene
too late, sin
was also the
could be sub
many
PhD students bring boxes full of proposals to the EC offices first. A follow-up meeting has been agreed upon.

How to compute $y - x$ or why overloading is a minus

Eerke Boiten
Mathematics and Computer Science
Dept. of Algorithmics & Architecture
P.O. Box 4079
1009 AB Amsterdam
The Netherlands
(jt@cwi.nl)

This is the second issue of the second volume of the Squiggolist. The Squiggolist is a forum for people who work with the Bird–Meertens formalism. It is meant for the quick distribution of short papers, summaries of results, or current points of interest. You cannot subscribe to the Squiggolist: either you receive it, or you don't. I have produced two issues of the Squiggolist each year. This implies that the next Squiggolist will be produced around May next year. Of course, the date of production depends on the number of submitted papers: if all of you submit a paper next week I'll be happy to produce a Squiggolist next month.

Submit your contributions (camera-ready copy or a LaTeX-file) in A4 format. They will be reduced to A5 ($\times$ 0.71), so use pointsize 12. There are no restrictions on the fonts used in the camera-ready copy: it may be LaTeX, handwritten or typewritten, as long as it is black on white, readable and large enough to be turned into A5. I will be the editor, and contributions should be sent to me:

Johan Jeuring
CWI, dept AA
P.O. Box 4079
1009 AB Amsterdam
The Netherlands
email: jt@cwi.nl

Contents

The smallest upravel

Richard S. Bird
Programming Research Group,Oxford University
11 Keble Rd. Oxford, OX1 3QD

June 26, 1991

Introduction

An *unravel* of a sequence x is a bag of nonempty subsequences of x that when shuffled together can give back x. For example, the sequence "accompany" can be unravelled into three lists "acm", "an", and "copy". The order of these lists is not important but duplications do matter; for example, "peptet" can be unravelled into two copies of "pet". Thus, an unravel is essentially a *bag* of sequences and not a list or set.

An unravel is called an *upravel* if all its component sequences are ascending. Since each of "acm", "an", and "copy" are ascending, they give an upravel of "accompany". Each nonempty sequence has at least one upravel, namely the upravel consisting of just singleton sequences. However, of all possible upravels we want to determine one with the least number of elements.

The problem of the smallest upravel was first posed by Kaldewaij [2], who solved it by reducing it to another problem, that of computing a longest decreasing subsequence. Subsequently, Meertens [3] gave a direct solution based on an incremental strategy. Meertens' derivation exploited indeterminacy and involved the invention of a certain partial ordering. Also, the details were rather complicated. Here we follow a direct course, one that eschews indeterminacy and stays within the framework of equational rea-

soning with functions. Some details remain complicated though.

Specification

We can specify the problem as one of computing **sur**, where

$$\textsf{sur} \quad = \quad \sqcap_{\#}/ \cdot \textsf{all up}\triangleleft \cdot \textsf{unravels}. \tag{1}$$

We omit the standard specification of **up**, the predicate that says a sequence is ascending, and assume familiarity with the notations for reduction (e.g. $\sqcap_{\#}/$), for filter ($\triangleleft$) and for map ($*$). The operator $\sqcap_{\#}$ is here applied to two bags of sequences and selects the smaller of its two arguments under some total ordering $\leq_{\#}$ that respects size, i.e., $bx \leq_{\#} by \;\Rightarrow \#bx \leq \#by$. (By the way, we use x as an identifier for sequences, bx for a bag of sequences, and sbx for a set of bags of sequences.) Use of $\sqcap_{\#}/$ characterises the deterministic approach to algorithm derivation: the ordering $\leq_{\#}$ is fully determinate, it is just that we haven't chosen it yet. By constrast, the specification

$$\textsf{sur} \quad \in \quad \mathrm{Min}(\#) \cdot \textsf{all up}\triangleleft \cdot \textsf{unravels} \tag{2}$$

characterises the indeterminate approach. Equation (2) reads: **sur** x is specified to be a member of the set of smallest upravels of x. There is greater freedom of action with (2), freedom that for some problems is essential, but the price to be paid is a move to non-equational reasoning. It is a bold decision to prefer the more restrictive (1) over (2) since there is no guarantee that an efficient algorithm will emerge.

The interesting task is to specify **unravels**. This function has type

$$\textsf{unravels} \in [A] \rightarrow \left\{ \wr [A]^{+} \int \right\}$$

for some ordered type A such as characters or numbers. Thus, **unravels** takes a possibly empty sequence and returns a set of bags of nonempty sequences. We shall specify **unravels** in two ways, one by an inductive definition, and one using relational inverse.

To motivate the inductive definition, suppose sbx is the set of unravels of x. Each unravel of $[a] +\!\!+ x$ can be obtained by taking some $bx \in sbx$ and

adding $[a]$ as a new component of bx, or by prefixing a to some existing component of bx. If we do this in all possible ways, we obtain all the unravels of $[a] +\!\!+ x$. The empty sequence has one unravel, namely the empty bag. Hence we have

$$\textsf{unravels} \;=\; \oplus^\circ \not\leftarrow \{\!|\,|\!\}. \tag{3}$$

where

$$a \oplus bx \;=\; \{\!|[a] +\!\!+ x|\!\} \uplus (bx - \{\!|x|\!\}) \mid x \in bx\} \cup \{\!|[a]|\!\} \uplus bx\} \tag{4}$$

and the decorated operator $\oplus^\circ$ is defined by

$$a \oplus^\circ sbx \;=\; \cup/\,(a\oplus) * sbx. \tag{5}$$

In general, if $\otimes \in A \times B \to \{B\}$, then $\otimes^\circ \in A \times \{B\} \to \{B\}$. In the present case, we have $B = \{\!|[A]^+|\!\}$.

In the second method for specifying **unravels**, we first specify the inverse function **ravels**. This function takes a bag of sequences and shuffles them together in all possible ways. Thus, **ravels** returns a set of sequences. For possibly empty sequences x and y, the shuffles $x \bowtie y$ are defined by taking $x \bowtie [\,] = [\,] \bowtie x = \{x\}$ and

$$([a] +\!\!+ x) \bowtie ([b] +\!\!+ y) = ([a] +\!\!+) * (x \bowtie ([b] +\!\!+ y)) \cup ([b] +\!\!+) * (([a] +\!\!+ x) \bowtie y).$$

Thus, $\bowtie$ has type $[A] \times [A] \to \{[A]\}$. However, we can restrict $\bowtie$ to have type $[A]^+ \times [A]^+ \to \{[A]^+\}$ by modifying the above definition to exclude the empty sequence.

For the next step we need some new notation. For $\oplus \in B^2 \to \{B\}$, introduce the lifted operator $\oplus^\bullet$ defined by

$$sa \oplus^\bullet sb \;=\; \cup/\,\{a \oplus b \mid a \in sa \wedge b \in sb\}$$

Thus, $\oplus^\bullet$ has type $\{B\}^2 \to \{B\}$. In particular, $\bowtie^\bullet$ takes pairs of sets of (nonempty) sequences to a set of sequences. It is associative and commutative, with identity element $\{[\,]\}$. Commutativity is easy to prove, but associativity is a little trickier and we omit details. Note that $\{a\} \oplus^\bullet sb = a \oplus^\circ sb$.

Now we define $\mathsf{ravels} = \bowtie^\bullet / \cdot \tau*$, where $\tau\, x = \{x\}$. For example,

$$\begin{aligned}
& \mathsf{ravels}\,[\![[1],[2,3],[4]]\!] \\
= \;& \{[1]\} \bowtie^\bullet \{[2,3]\} \bowtie^\bullet \{[4]\} \\
= \;& \{[1]\} \bowtie^\bullet \{[4,2,3],\ [2,4,3],\ [2,3,4]\} \\
= \;& \{[1,4,2,3],[4,1,2,3],[4,2,1,3],[4,2,3,1],\ldots\}
\end{aligned}$$

The result is the set of twelve permutations of $\{1,2,3,4\}$ in which 2 precedes 3.

By definition, the unravels of x are just those bags of sequences that when ravelled give a set that contains x. We therefore have

$$\mathsf{unravels} \;=\; \mu(\bowtie^\bullet / \cdot \tau*), \tag{6}$$

where the operator μ is defined by

$$\mu F\, b = \{a \mid b \in F\, a\}.$$

If $F \in A \to \{B\}$, then $\mu F \in B \to \{A\}$. The function μF corresponds to the relational converse of F when F is interpreted as a relation R such that aRb just when $b \in F\, a$. Many of the properties of μ are closely related to Inv, the inverse image function.

It seems a long way from (6) to (3); nevertheless, the inductive definition can be synthesised from (6) in a relatively straightforward manner. We will not give details, but there are two key facts in the synthesis. First, we can use the specialisation lemma (for bags) to obtain $\mathsf{ravels} = \bowtie^\circ \not\leftarrow \{[\,]\}$. Second, we use the following theorem, cited without proof, for the relational converse of a right-reduction:

Theorem 1 *Suppose $F \in [\![A]\!] \to \{B\}$ is defined by $F = \oplus^\circ \not\leftarrow e$, where $\oplus \in A \times B \to \{B\}$ and $e \in \{B\}$. Then*

$$\mu F \;=\; \mathsf{con}\,\{[\![\,]\!]\} \diamond \mu(\mathsf{con}\, e) \;\cup\; (\rightarrowtail^\circ \cdot \mathsf{id} \times \mu F) \diamond \mu(\oplus),$$

where $\diamond$ is Kleilsi composition: $F \diamond G = \cup / \cdot F* \cdot G$, and $\rightarrowtail$ is defined by $a \rightarrowtail x = [\![a]\!] \uplus x$. The constant function con *is defined by* con $a\, b = a$, *and $(f \times g)(a,b) = (f\, a, g\, b)$.*

Derivation

The strategy is to start with the characterisation of **unravels** as a right-reduction and apply the promotion theorem for right-reductions. An alternative strategy is to start with Theorem 1 head for a greedy solution. For details of the second approach, consult [1].

Using the inductive characterisation of **unravels**, specification (1) gives $\mathsf{sur} = \sqcap_{\#}/ \cdot \mathsf{upravels}$, where

$$\mathsf{upravels} \;=\; \mathsf{all}\,\mathsf{up}\triangleleft \cdot \oplus^{\circ} \not\leftarrow \{\wr\,\int\}. \tag{7}$$

We split the problem up in this way because we want to treat each part separately. We use the promotion theorem for right-reductions to simplify (7). In one form, this theorem states that $h \cdot \oplus \not\leftarrow e = \otimes \not\leftarrow e'$ provided $h\,e = e'$ and $h \cdot (a\oplus) = (a\otimes) \cdot h$. Another form of the theorem appears below.

For the first proviso, we have $\mathsf{all}\,\mathsf{up} \triangleleft \{\wr\,\int\} = \{\wr\,\int\}$. For the second proviso, we argue

$$\begin{array}{ll}
 & \mathsf{all}\,\mathsf{up}\triangleleft \cdot (a\oplus^{\circ}) \\
= & \quad \{\ \text{definition of } \oplus^{\circ}\ \} \\
 & \mathsf{all}\,\mathsf{up}\triangleleft \cdot \cup/ \cdot (a\oplus)* \\
= & \quad \{\ \triangleleft \text{ promotion and } * \text{ distributivity}\ \} \\
 & \cup/ \cdot (\mathsf{all}\,\mathsf{up}\triangleleft \cdot (a\oplus))* \\
= & \quad \{\ \text{introducing } \otimes \text{ (see below)}\ \} \\
 & \cup/ \cdot (\mathsf{all}\,\mathsf{up} \rightarrow (a\otimes),\ \mathsf{con}\,\{\,\})* \\
= & \quad \{\ \text{law: } \cup/ \cdot (p \rightarrow f, \mathsf{con}\,\{\,\})* = \cup/ \cdot f* \cdot p\triangleleft\ \} \\
 & \cup/ \cdot (a\otimes)* \cdot \mathsf{all}\,\mathsf{up}\triangleleft \\
= & \quad \{\ \text{definition of } \otimes^{\circ}\ \} \\
 & (a\otimes^{\circ}) \cdot \mathsf{all}\,\mathsf{up}\triangleleft
\end{array}$$

The operator $\otimes$ is similar to $\oplus$ but returns bags of upsequences. It is defined by

$$a \otimes bx \;=\; \{\wr[a] +\!\!+ x\int \uplus\, (bx - \wr x\int) \mid x \in bx \wedge a \leq \mathsf{hd}\,x\} \cup \{\wr[a]\int \uplus\, bx\}.$$

We omit the verification of the equation

$$\mathsf{all\,up}\triangleleft \cdot (a\oplus) = (\mathsf{all\,up} \rightarrow (a\otimes), \mathsf{con}\,\{\,\})$$

used in the derivation above. The promotion theorem for right-reductions now gives us that

$$\mathsf{upravels} = \otimes^\circ \not\leftarrow \{\wr\,\int\}.$$

To simplify $\mathsf{sur} = \sqcap_\#/ \cdot \mathsf{upravels}$ we again appeal to the promotion theorem. However, this time we need the more general form of the theorem, namely that $h \cdot \otimes \not\leftarrow e = \odot \not\leftarrow e'$ if and only if $h\,e = e'$ and

$$h \cdot (a\otimes) \cdot \otimes \not\leftarrow e = (a\odot) \cdot h \cdot \otimes \not\leftarrow e$$

Introducing $f = g \pmod h$ as an abbreviation for $f \cdot h = g \cdot h$, the condition above reads: $h \cdot (a\otimes) = (a\odot) \cdot h \pmod{\otimes \not\leftarrow e}$. The following argument is carried out modulo $\otimes^\circ \not\leftarrow \{\wr\,\int\}$, i.e., modulo upravels:

$$\begin{array}{ll} & \sqcap_\#/ \cdot (a\otimes^\circ) \\ = & \quad \{\ \text{definition of } \otimes^\circ\ \} \\ & \sqcap_\#/ \cdot \cup/ \cdot (a\otimes)* \\ = & \quad \{\ /\ \text{promotion and } *\ \text{distributivity}\ \} \\ & \sqcap_\#/ \cdot (\sqcap_\#/ \cdot (a\otimes))* \\ = & \quad \{\ \text{defining } \odot \text{ by } (a\odot) = \sqcap_\#/ \cdot (a\otimes)\ \} \\ & \sqcap_\#/ \cdot (a\odot)* \\ = & \quad \{\ \text{claim; see below}\ \} \\ & (a\odot) \cdot \sqcap_\#/ \end{array}$$

The claim in the last step refers to the property

$$(8) \qquad \sqcap_\#/ \cdot (a\odot)* \cdot \mathsf{upravels} \;=\; (a\odot) \cdot \sqcap_\#/ \cdot \mathsf{upravels}$$

that we have yet to justify for a suitable choice of $\leq_\#$. Assuming (8) can be satisfied, the promotion theorem gives that $\mathsf{sur} = \odot \not\leftarrow \wr\,\int$ since $\sqcap_\#/\,\{\wr\,\int\} = \wr\,\int$.

Before tackling (8), let us rewrite the equation for $\odot$ using the definition of $\otimes$:

$$(9)\qquad a \odot bx \;=\; \sqcap_{\#}/\,\{\wr[a] +\!\!+ x\int \uplus\,(bx - \wr x\int) \mid x \in bx \wedge a \leq \mathsf{hd}\,x\}$$
$$\sqcap_{\#}(\wr[a]\int \uplus\, bx).$$

We would like to simplify (9) by choosing a suitable definition of $\leq_{\#}$; at the same time we will choose $\leq_{\#}$ to satisfy (8).

The ordering $\leq_{\#}$ has to respect $\#$, so its definition has to take the form

$$bx \leq_{\#} by \;\equiv\; \#bx < \#by \vee (\#bx = \#by \wedge bx \leq_B by),$$

where $\leq_B$ is some ordering on $B = \wr[A]^+\int$. From (9) and the definition of $\leq_{\#}$ we obtain

$$a \odot bx \;=\; \begin{cases} \wr[a]\int \uplus\, bx & \text{if } by = \{\,\} \\ \sqcap_B/\,\{\wr[a] +\!\!+ x\int \uplus\,(bx - \wr x\int) \mid x \in by\} & \text{otherwise} \\ \text{where } by = \{x \mid x \in bx \wedge a \leq \mathsf{hd}\,x\} & \end{cases}$$

The ordering $\leq_B$ is an ordering on bags, and the only reasonable candidate is a lexicographic ordering:

$$bx \leq_B by \;\equiv\; bx = \wr\int \vee \sqcap_L/\,bx < \sqcap_L/\,by \vee$$
$$(\sqcap_L/\,bx = \sqcap_L/\,by) \wedge (bx - \wr\sqcap_L/\,bx\int \leq_B by - \wr\sqcap_L/\,by\int),$$

where $\leq_L$ is some ordering on $L = [A]^+$. Thus, $\leq_B$ is the lexicographic ordering on bags generated by the ordering $\leq_L$ on its elements.

In order to use $\leq_B$ to simplify the definition of $\odot$ still further, we would like to define $\leq_L$ so that

$$(10)\quad x <_L y \;\Rightarrow\; \wr[a] +\!\!+ x\int \uplus\,(bx - \wr x\int) <_B \wr[a] +\!\!+ y\int \uplus\,(bx - \wr y\int)$$

for all $x, y \in bx$ with $a \leq \mathsf{hd}\,x$ and $a \leq \mathsf{hd}\,y$. Then we would have

$$a \odot bx \;=\; \begin{cases} \wr[a]\int \uplus\, bx & \text{if } by = \{\,\} \\ \wr[a] +\!\!+ \sqcap_L/\,by\int \uplus\,(bx - \wr\sqcap_L/\,by\int) & \text{otherwise} \\ \text{where } by = \{x \mid x \in bx \wedge a \leq \mathsf{hd}\,x\}. & \end{cases}$$

Now (10) is satisfied if the following two conditions hold:

$$\begin{aligned} a \leq \mathsf{hd}\, x &\Rightarrow [a] +\!\!+ x <_L x \\ x <_L y &\Rightarrow [a] +\!\!+ x <_L [a] +\!\!+ y. \end{aligned}$$

The second condition is satisfied if we choose $\leq_L$ to be a lexicographic ordering on $[A]$, and the first is satisfied if we reverse the standard convention and assign $x \leq y$ when y is an initial segment of x. (In the reversed ordering $[\,]$ is the *last* entry in the dictionary.) Thus, we take

$$x \leq_L y \;\equiv\; y = [\,] \;\vee\; \mathsf{hd}\, x < \mathsf{hd}\, y \;\vee\; (\mathsf{hd}\, x = \mathsf{hd}\, y \wedge \mathsf{tail}\, x \leq_L \mathsf{tail}\, y).$$

As a final optimisation to the program for $\odot$, we can take the step of representing bags by lists of sequences in ascending order under $\leq_L$. We then get

$$a \odot xs \;=\; \begin{cases} xs +\!\!+ [[a]] & \text{if } zs = [\,] \\ ys +\!\!+ [[a] +\!\!+ \mathsf{hd}\, zs] +\!\!+ \mathsf{tail}\, zs & \text{otherwise} \\ \text{where } (ys, zs) = (((< a) \cdot \mathsf{hd} \triangleleft xs, ((a \leq) \cdot \mathsf{hd}) \triangleleft xs). \end{cases}$$

We can also express this program recursively: we have $a \odot [\,] = [[a]]$ and

$$a \odot ([x] +\!\!+ xs) \;=\; \begin{cases} [[a] +\!\!+ x] +\!\!+ xs & \text{if } a \leq \mathsf{hd}\, x \\ [x] +\!\!+ (a \odot xs) & \text{otherwise} \end{cases}$$

Evaluation of $a \odot xs$ can also be implemented by binary search since the first elements of sequences in xs are in increasing order. That means sur can be implemented in $O(n \log n)$ steps, where n is the length of the input.

The claim

But we still have to prove claim (8), namely

$$\sqcap_{\#}/ \cdot (a\odot)* \cdot \mathsf{upravels} = (a\odot) \cdot \sqcap_{\#}/ \cdot \mathsf{upravels}.$$

Certainly, (8) follows if $(a\odot)$ is *monotonic* under $\leq_{\#}$ (modulo **upravels**), i.e.,

$$xs \leq_{\#} ys \;\Rightarrow\; a \odot xs \leq_{\#} a \odot ys$$

for all $xs, ys \in \mathsf{upravels}\, x$. However, $(a\odot)$ is *not* monotonic under $\leq_\#$. A simple counterexample is given by the fact that we have

$$[[1,2],\ [3,4,6]] <_\# [[1,2,3],\ [4],\ [6]]$$

but applying $(5\odot)$ to these sequences gives

$$[[1,2],\ [3,4,6],\ [5]] >_\# [[1,2,3],\ [4],\ [5,6]]$$

Compare (8) with the following valid identity:

$$(11) \qquad \sqcap_B/ \cdot (a\odot)* \cdot \mathsf{upravels} \;=\; (a\odot) \cdot \sqcap_B/ \cdot \mathsf{upravels}$$

Equation (11) holds because $(a\odot)$ is monotonic under $\leq_B$ modulo $\mathsf{upravels}$. The proof involves a fairly straightforward case analysis, but we omit it for reasons of space.

Well, I have been unable to find a proof of (8). Fortunately, this does not mean we are stuck for we can prove another fact:

$$(12) \qquad \sqcap_\#/ \cdot \mathsf{upravels} \;=\; \sqcap_B/ \cdot \mathsf{upravels}$$

In words, although $\leq_\#$ is a different ordering than $\leq_B$, the smallest upravel under one is the smallest upravel under the other. This claim is not obvious (witness the complexity of the proof given below).

How does (12) help? The answer seems a devious one. Having identified $\leq_B$ and proved (12), we can replay the derivation, but with the new starting point $\mathsf{sur} = \sqcap_B/ \cdot \mathsf{upravels}$. One has to check that the new definition of $\odot$, namely $(a\odot) = \sqcap_B/ \cdot (a\otimes)$, defines the same operation as before (it does, but we omit details). Then we can use (11) to complete the derivation. Admittedly, this method is inferior to one based on (8) but it is perfectly valid.

The proof of (12) is quite complicated. We are going to use a cut and paste argument to replace one upravel of x by another, so we need to know that what we cut and paste remain subsequences of x. For this reason we need to keep track of the *positions* of elements in x. Replace each element a in x with a pair (a, j), where j is the position of a in x and so $0 \leq j < \#x$. Each upravel is now a bag (or sequence) of sequences of

pairs. Each sequence in each upravel is ordered on first components (the sequence is an upsequence), but also on second components (the upsequence is a subsequence of x). Note, however, that the definitions of $\leq_\#$ and $\leq_B$ depend only on the first components of pairs. In effect, we are proving a one-dimensional result (about the elements of x) by using a two-dimensional argument (about the elements and their positions in x). The proof hinges on the following lemma:

Lemma 1 *Let xs be the smallest upravel of x under $\leq_\#$ and suppose each element of xs is a sequence of pairs in the manner described above. Then for all $y, z \in xs$ with $y <_L z$,*

$$(a, p) \in y \wedge (b, q) \in z \quad \Rightarrow \quad a \leq b \vee p > q.$$

Proof. The argument is by contradiction. Suppose the conclusion is false and there is some $y, z \in xs$, with $y <_L z$, and some $(a, p) \in y$ and $(b, q) \in z$ with $a > b \wedge p < q$. Without loss of generality, suppose (p, q) is as small as possible under the lexical ordering on pairs. Let

$$y = u +\!\!+ [(a, p)] +\!\!+ v \quad \text{and} \quad z = w +\!\!+ [(b, q)] +\!\!+ t$$

Suppose u has last element (c, n) and w has last element (d, r) where, to avoid a case analysis, we take $(-\infty, -\infty)$ as last element if u or v is empty. Since y and z are upsequences we have

$$c \leq a \wedge n < p \quad \text{and} \quad d \leq b \wedge r < q$$

Combined with the assumption $a > b \wedge p < q$, we get

$$c \leq a \wedge n < q \quad \text{and} \quad d < a \wedge r < q \tag{13}$$

By the assumption that (a, p) and (b, q) are the earliest critical pair, we have that neither $(c, n), (b, q)$ or $(a, p), (d, r)$ are critical pairs, and so

$$c \leq b \vee n > q \quad \text{and} \quad a \leq d \vee p > r \tag{14}$$

Combining (13) and (14) we get

$$c \leq b \wedge n < q \quad \text{and} \quad d < a \wedge r < p$$

and so

$$y' = u +\!\!+ [(b,q)] +\!\!+ t \qquad \text{and} \qquad z' = w +\!\!+ [(a,p)] +\!\!+ v$$

are upsequences of x. Define a new upravel xs' by replacing y, z with y', z'. We have $\#xs' = \#xs$ and $xs' <_B xs$ since $y' <_L y$, whence $xs' <_\# xs$. A contradiction. □

Armed with this result, we can prove (12). Suppose $\sqcap_\#/\,\mathsf{upravels}\, x = xs$ and $xs = [x_1, x_2, \ldots, x_n]$. Let $ys = [y_1, y_2, \ldots, y_m]$ be another upravel of x, so $m \geq n$. We aim to show that $xs <_B ys$. Since xs is not a proper initial segment of ys (otherwise xs and ys cannot ravel to the same sequence x), there is a j with $x_i = y_i$ for $1 \leq i < j$, and $x_j \neq y_j$. We have to show that $x_j <_L y_j$.

If x_j is an initial segment of y_j, then we can replace x_j by y_j in xs, deleting all elements in y_j but not in x_j from other components in xs. The result is another upravel xs' with $\#xs' \leq \#xs$ and $xs' <_B xs$. Hence $xs' <_\# xs$, a contradiction.

So, x_j is not an initial segment of y_j and we have $x_j = u +\!\!+ [(a,p)] +\!\!+ v$ and $y_j = u +\!\!+ [(b,q)] +\!\!+ w$ for some $a \neq b$ and $p \neq q$. We show that the assumption $a > b$ leads to a contradiction. If $a > b$, then (b,q) cannot appear in x_j or in any x_i with $i < j$, so it appears in some x_k of xs, where $k > j$. Let $x_k = w +\!\!+ [(b,q)] +\!\!+ s$. If $q < p$, then we can replace x_j and x_k in xs with new upsequences $u +\!\!+ [(b,q),(a,p)] +\!\!+ v$ and $w +\!\!+ s$. This gives a new upravel xs' with $\#xs' = \#xs$ and $xs' <_B xs$. Hence $xs' <_\# xs$, a contradiction.

In the final case, we have $a > b$ and $p < q$. But appeal to the lemma shows that we get a contradiction in this case too. □

Comments

There are good points and bad points about the above derivation of an efficient program for the smallest upravel. One good point is that we have a simply structured derivation based on equational reasoning. Moreover, the invention of the ordering $\leq_\#$ was systematic and rabbit-free. The only

problem is that the proof of the equation (8) is missing. The bad point is in the last part with the truly awful proof of (12).

Here are three questions well worth answering:

- Starting with (2), is there a simpler derivation of the final algorithm?
- What is the proof of (8)?
- What is a better proof of (12)?

References

[1] R. S. Bird (1991), Unravelling greedy algorithms. *Journal of Functional Programming* vol X, no X (1991)

[2] A. Kaldewaij (1985), On the decomposition of sequences into ascending subsequences. *Information Processing Letters*, vol 21, page 69.

[3] L. Meertens, (1985), Some more examples of algorithmic developments. *IFIP Wg2.1 Working Paper*, Pont a Mousson, France.

Many-sorted algebras and initiality

Maarten M Fokkinga, University of Twente

The categorical notion of just 'algebra' is general enough to cover also 'many-sorted' algebras. This holds also with respect to initiality. A real generalisation of 'algebra' yields 'dialgebra'.

1 Introduction You might wonder whether the notion of algebra is rich enough to model also so-called many-sorted algebras. For example, in a traditional formulation, the collection

$$(bool, nat;\ \ true, false, bool\text{-}to\text{-}nat, zero, succ, equal)$$

is or suggests a two-sorted algebra, the two sorts (types) being *bool* and *nat*. In view of the typing $bool\text{-}to\text{-}nat : bool \to nat$ and $equal : nat \times nat \to bool$ both sorts are needed simultaneously to specify the operations.

We shall show that by instantiating the base category to a product category, the conventional category $Alg(F)$ consists of many-sorted algebras indeed. Besides that, a single initial many-sorted algebra can be expressed as many initial single-sorted algebras. Thus the existence conditions and the construction for initial algebras over a product category are reduced to initial algebras over the component categories. These two results say that the theory for just 'normal' algebras also applies to many-sorted algebras.

We formalise only the case "many = two". It has the advantage that the formulas are simpler than in the general case, whereas all essential aspects are covered. You can easily generalise the discussion to "many = n" for arbitrary natural n.

2 Algebras Recall the categorical formalisation of algebras. Let $\mathbf{C}$ be a category, and F be an endofunctor on $\mathbf{C}$. An **F-algebra** in $\mathbf{C}$ is: a morphism $\phi : Fa \to_{\mathbf{C}} a$ in $\mathbf{C}$, for some object $a = U\phi$ called the carrier of ϕ. Let ϕ, ψ be F-algebras; then an **F-homomorphism** from ϕ to ψ is: a morphism f in $\mathbf{C}$ satisfying $\phi ; f = Ff ; \psi$, denoted $f : \phi \to_F \psi$. These algebras, as objects, and homomorphisms, as morphisms, constitute a category called $Alg(F)$; the composition (hence also the identities) are inherited from the underlying category $\mathbf{C}$. Initiality in $Alg(F)$ of an F-algebra α means that for each F-algebra ϕ there exists precisely one homomorphism from α to ϕ, which we denote by $(\!|\alpha \to \phi|\!)_F$. If α and F are clear from the context we write $(\!|\phi|\!)_F$ or just $(\!|\phi|\!)$. Thus,

$$\alpha ; x = Fx ; \phi \quad \equiv \quad x = (\!|\phi|\!) \qquad \text{cata-CHARN}$$

If $Alg(F)$ has an initial object, we let μF denote one; it is called an (or the) initial F-algebra.

Whenever we say that $\mathbf{C}$ is 'the base category' we omit explicit mentioning of $\mathbf{C}$; in particular, we then write $\to_{\mathbf{C}}$ as just $\to$. Finally recall that $\vartriangle$ is the tupling or pairing operation.

3 Many-sortedness The example in the introduction motivates the following definition.

- A **two-sorted $\dagger,\ddagger$-algebra** is: a pair (ϕ,ψ) with $\phi : a \dagger b \to_{\mathbf{A}} a$ and $\psi : a \ddagger b \to_{\mathbf{B}} b$, for some a, b called the sorts of the two-sorted algebra.
- A **two-sorted $\dagger,\ddagger$-homomorphism** from (ϕ,ψ) to (χ,ω) is: a pair (f,g) with

$$\phi ; f = f \dagger g ; \chi \qquad \text{and} \qquad \psi ; g = f \ddagger g ; \omega .$$

For this to make sense $\dagger$ and $\ddagger$ should be bifunctors with common source $\mathbf{A} \times \mathbf{B}$ and targets $\mathbf{A}$ and $\mathbf{B}$ respectively, for some categories $\mathbf{A}$ and $\mathbf{B}$. Clearly, the two-sorted algebras with their homomorphisms constitute a category, 2-$Alg(\dagger,\ddagger)$ say.

There is, however, a simpler definition for two-sorted algebras. Recall the notion of product category: its objects and homomorphisms are pairs, and composition etc, are defined coordinatewise. Taking $\dagger$ and $\ddagger$ as above, the composite $\dagger \vartriangle \ddagger$ is an endofunctor on $\mathbf{A} \times \mathbf{B}$. Spelling out what it means to be an object or morphism in $Alg(\dagger \vartriangle \ddagger)$ (having underlying category $\mathbf{A} \times \mathbf{B}$), you'll see that these are exactly the two-sorted $\dagger,\ddagger$-algebras and homomorphisms defined above.

$$\begin{array}{ll} & (\phi,\psi) \text{ in } Alg(\dagger \vartriangle \ddagger) \\ \equiv & \\ & (\phi,\psi) : (\dagger \vartriangle \ddagger)(a,b) \to_{\mathbf{A}\times\mathbf{B}} (a,b) \qquad \text{for some } a,b \\ \equiv & \\ & \phi : a \dagger b \to_{\mathbf{A}} a \quad \text{and} \quad \psi : a \ddagger b \to_{\mathbf{B}} b \qquad \text{for some } a,b \end{array}$$

and further

$$\begin{array}{ll} & (f,g) : (\phi,\psi) \to_{Alg(\dagger\vartriangle\ddagger)} (\chi,\omega) \\ \equiv & \\ & (\phi,\psi) ; (f,g) = (\dagger \vartriangle \ddagger)(f,g) ; (\chi,\omega) \\ \equiv & \\ & \phi ; f = f \dagger g ; \chi \quad \text{and} \quad \psi ; g = f \ddagger g ; \omega . \end{array}$$

Thus, 2-$Alg(\dagger,\ddagger) = Alg(\dagger \vartriangle \ddagger)$, and the prefix "two-sorted $\dagger,\ddagger$-" equivales just "$\dagger \vartriangle \ddagger$-".

4 Theorem *Let $\dagger,\ddagger$ be bifunctors. Suppose that $\mu(\dagger b)$ and $\mu(a\ddagger)$ exist for all a, b. Then there exist endofunctors F, G such that*

$$(\mu F, \mu G) \text{ is initial in } Alg(\dagger \vartriangle \ddagger) .$$

Specifically, $F = I \dagger L$ and $G = (U\mu F)\ddagger$, where L is the map functor induced by $\ddagger$.

For this to make sense it is required that there are categories $\mathbf{A}, \mathbf{B}$, and that functors $\dagger,\ddagger$ have source $\mathbf{A}\times\mathbf{B}$ and target $\mathbf{A}$ and $\mathbf{B}$ respectively, and that F, G are endofunctors on $\mathbf{A} \times \mathbf{B}$; it follows that $L : \mathbf{A} \to \mathbf{B}$.

Proof We shall synthesise a pair F, G and an expression for $(\!|(\mu F, \mu G) - \ \ldots)\!|_{\dagger\vartriangle\ddagger}$ such that assertion

$$(x,y) \ : \ (\mu F, \mu G) \ \rightarrow_{\dagger\vartriangle\ddagger} \ (\phi,\psi) \qquad \equiv \qquad (x,y) = (\!|(\mu F, \mu G) - (\phi,\psi)|\!)_{\dagger\vartriangle\ddagger}$$

is valid, thus establishing initiality of $(\mu F, \mu G)$ in $Alg(\dagger \vartriangle \ddagger)$.

For whatever F and G are going to be, put $a, b = U\mu F, U\mu G$. Let (ϕ,ψ) be an arbitrary $\dagger \vartriangle \ddagger$-algebra, say

$$\begin{array}{lcl} \phi & : & c \dagger d \rightarrow c \\ \psi & : & c \ddagger d \rightarrow d \end{array}$$

for some c, d. It follows that $x : a \rightarrow c$ and $y : b \rightarrow d$ (for the x, y in the desired equivalence). Now we argue

$$\begin{array}{ll} & (x,y) : (\mu F, \mu G) \rightarrow_{\dagger\vartriangle\ddagger} (\phi,\psi) \\ \equiv & \quad \text{product category} \\ & \mu F \,;\, x = x \dagger y \,;\, \phi \ \ \wedge \ \ \mu G \,;\, y = x \ddagger y \,;\, \psi \\ \equiv & \quad \text{bifunctor (aim: express } y \text{ as a homomorphism from } \mu G) \\ & \mu F \,;\, x = x \dagger y \,;\, \phi \ \ \wedge \ \ \mu G \,;\, y = id_a \ddagger y \,;\, x \ddagger id_d \,;\, \psi \\ \equiv & \quad \textbf{define } G := a\ddagger = (U\mu F)\ddagger, \text{ cata-CHARN} \\ & \mu F \,;\, x = x \dagger y \,;\, \phi \ \ \wedge \ \ y = (\!|x \ddagger id_d \,;\, \psi|\!)_G \\ \equiv & \quad \text{cata-FACTORN1 (applied to the rhs of } y), \\ & \quad \textbf{abbreviate } f = (\!|\psi|\!)_{c\ddagger} \text{ and } \textbf{define } L = \text{map functor induced by } \ddagger \\ & \mu F \,;\, x = x \dagger y \,;\, \phi \ \ \wedge \ \ y = Lx \,;\, f \\ \equiv & \quad \text{substitute } y = Lx \,;\, f \text{ in left conjunct, functor} \\ & \mu F \,;\, x = (I \dagger L)x \,;\, id_c \dagger f \,;\, \phi \ \ \wedge \ \ y = Lx \,;\, f \\ \equiv & \quad \textbf{define } F = I \dagger L, \text{ cata-CHARN} \\ & x = (\!|id_c \dagger f \,;\, \phi|\!)_F \ \ \wedge \ \ y = Lx \,;\, f \\ \equiv & \quad \text{product category} \\ & (x,y) = ((\!|id_c \dagger f \,;\, \phi|\!)_F, \ \ Lx \,;\, f). \end{array}$$

Thus we have found the required definitions of F, G and $(\!|(\mu F, \mu G) - \ \ldots)\!|_{\dagger\vartriangle\ddagger}$. □

5 Dialgebras The notion of algebra is not general enough to cover distribution properties and algebra-like structures that you encounter in practice. A slight generalisation of algebra yields the notion of dialgebra. Briefly, for dialgebras the target type may be Ga rather than just a. The formal definition reads as follows. An F,G-**dialgebra** is: a morphism ϕ such that $\phi : Fa \rightarrow Ga$ for some $a = U\phi$ called the **carrier** of ϕ. For this to make sense it is required that F and G are functors, $F, G : \mathbf{A} \rightarrow \mathbf{C}$ say, and $\mathbf{A}, \mathbf{C}$ are categories (and $\mathbf{C}$ is taken as the base one).

Let ϕ, ψ be F, G-dialgebras. An F, G-**homomorphism** from ϕ to ψ is: a morphism f for which $\phi ; Gf = Ff ; \psi$ denoted $f : \phi \rightarrow_{F,G} \psi$. It follows that $f : U\phi \rightarrow U\psi$. These dialgebras and homomorphisms constitute a category, called $DiAlg(F, G)$. You may now prove yourself that $Alg(F) = DiAlg(F, I)$; here it follows that $\mathbf{A} = \mathbf{C}$ since the source and target of I are equal. Co-algebras are obtained by taking $F, G = I, G$.

6 Bialgebras A datatype like *stack* with operations *empty*, *push*, *isempty*, *top* and *pop* has not the form of an algebra $\phi : Fa \rightarrow a$, but is rather a pair (ϕ, ψ) with $\phi : Fa \rightarrow a$ and $\psi : a \rightarrow Ga$, for some endofunctors F, G. To be specific, for *stack* we have

$$\begin{array}{lcllcll} \phi & = & empty \triangledown push & : & 1 + b \times a \rightarrow a & = & Fa \rightarrow a \\ \psi & = & isempty \vartriangle top \vartriangle pop & : & a \rightarrow bool \times b \times a & = & a \rightarrow Ga \end{array}$$

where b is the type of the stacked values (and a is the type of the stacks themselves). We call such a pair (ϕ, ψ) a (single-sorted) F, G-**bialgebra**. An F, G-**bialgebra homomorphism** from (ϕ, ψ) to (χ, ω) is a morphism f satisfying

$$\phi ; f = Ff ; \chi \qquad \text{and} \qquad \psi ; Gf = f ; \omega .$$

Clearly, these bialgebras and homomorphisms form a category, called $BiAlg(F, G)$, that is built upon the base category.

As for many-sorted algebras, a bialgebra is a particular dialgebra. Let F, G be endofunctors (on the base category), and consider $DiAlg(F \vartriangle I, I \vartriangle G)$. Then we find

$$\begin{array}{ll} & (\phi, \psi) \text{ in } DiAlg(F \vartriangle I, I \vartriangle G) \\ \equiv & \\ & (\phi, \psi) : (F \vartriangle I)a \rightarrow (I \vartriangle G)a \quad \text{for some } a \\ \equiv & \\ & \phi : Fa \rightarrow a \quad \text{and} \quad \psi : a \rightarrow Ga \quad \text{for some } a \end{array}$$

and moreover

$$\begin{array}{ll} & f : (\phi, \psi) \rightarrow_{DiAlg(F \vartriangle I, I \vartriangle G)} (\chi, \omega) \\ \equiv & \\ & (\phi, \psi) ; (I \vartriangle G)f = (F \vartriangle I)f ; (\chi, \omega) \\ \equiv & \\ & \phi ; f = Ff ; \chi \quad \text{and} \quad \psi ; Gf = f ; \omega . \end{array}$$

So, this proves $BiAlg(F, G) = DiAlg(F \vartriangle I, I \vartriangle G)$. The generalisation to two-sorted bialgebras is straightforward. It is an exercise for you to define the notion formally.

7 And so on For more such interesting observations, read my forthcoming thesis.

Free style spec wrestling II: preorders

Jaap van der Woude

18-10-1991

1 Introduction

Some games seem truely masochistic, for instance: give a pointfree proof of something that is easily proved with pointwise reasoning. With a pointfree proof I mean a proof in a pointless world, not just a proof without points in the formulae to be manipulated. Such games are not played just for fun or to keep us from other popular pastimes like ramriding. They kind of test the calculational power of the spec algebra and they may induce further streamlining of the spec calculus. I do admit that I am not spec wizard enough to conclude that the outrageous prooftime I needed until now is sufficient reason to abandon the spec calculus, but I do have my doubts on the current state of affairs.

An almost ridiculous sample-game was presented in the Liber Amicorum for Lambert Meertens (pp 110–115), constructing those proofs took me about three days (not counting the time to shape things up). An easier one (took me six hours, so at least five too many) is the following challenge from Richard Bird, transferred to me by Wim Feijen:

Define for a preorder X, i.e. X is reflexive and transitive, the *leasting* λX by

$$\lambda X.S \;=\; \{m \in S \mid (\mathbf{A}\, s \;:\; s \in S \;:\; m\, X\, s)\}$$

So, λX fed with a set S returns the X-least points of S (by lack of antisymmetry this may be a substantial subset of S).

(1) Challenge *Prove for preorders X and Y the existence of a preorder Z with*

$$\lambda Z \;=\; \lambda Y \circ \lambda X$$

By pointwise reasoning one finds such a Z quite easily: $X \sqcap (\neg X^{\cup} \sqcup Y)$, and it turns out to be unique (thanks Wim).

One way to meet the challenge is to prove that the proposed candidate is a good one (that is what I did first), but that smells like cheating. Here I want to present a proof that at least makes it sweetly reasonable that one could have arrived at the candidate by careful calculation.

2 Translation and technique

Since set theory can be played in an extensional (monotypical) spec algebra, where points are represented as impish leftconditions, we represent sets as leftconditions; so, without name changes, the set S is represented as a spec S such that $S = S \circ \top\!\!\top$. A subset A of S is a leftcondition $A \sqsubseteq S$ and a full relation $A \times B$ on S is given by $A \circ B^{\cup}$. A preorder X is of course a spec such that $I \sqcup X \circ X \sqsubseteq X$.

The definition of λX amounts to: $\lambda X.S$ is the greatest solution of

$$A :: \quad A \sqsubseteq S \quad \wedge \quad A \circ S^{\cup} \sqsubseteq X$$

Thus, by definition of the division operator,

$$\lambda X.S \;=\; S \sqcap X/S^{\cup} \tag{2}$$

The challenge is to construct for preorders X and Y a preorder Z such that *for every* S

$$S \sqcap Z/S^{\cup} \;=\; S \sqcap X/S^{\cup} \sqcap Y/(S \sqcap X/S^{\cup})^{\cup} \tag{3}$$

Since $/S^{\cup}$ is conjunctive we are led to believe that $Z = X \sqcap Z'$ for some Z' probably depending on Y .
Oh ... you are right, it is very well possible you don't know all calculation rules for the division operator by heart. Let me give you a few:

$$P/Q \sqsupseteq R \;\equiv\; P \sqsupseteq R \circ Q \tag{4}$$

$$P \;\sqsupseteq\; (P/Q) \circ Q \tag{5}$$

The first is the defining characterisation of $/Q$ as an "adjoint" of $\circ Q$, the second is a direct consequence: the *cancellation* rule. Two junctivity rules are:

$$(P \sqcap Q)/R \;=\; P/R \sqcap Q/R \tag{6}$$

$$P/(Q \sqcup R) \;=\; P/Q \sqcap P/R \tag{7}$$

The observation that the numerator of $\lambda X.S$ is a preorder invites the following rule to join in:

$$X/P \;=\; X \circ (X/P) \;\Leftarrow\; X \text{ is a preorder} \tag{8}$$

Since in the denominator of $\lambda X.S$ a wokked leftcondition (so a rightcondition) occurs, it might be useful to note for leftcondition L , rightcondition R and arbitrary P and Q :

$$P/R \text{ is a leftcondition} \tag{9}$$

$$P \circ (R \sqcap Q) \;=\; P \circ R \sqcap P \circ Q \tag{10}$$

$$L \circ R \;=\; L \sqcap R \tag{11}$$

$$L_{<} \circ Q \;=\; L \sqcap Q \tag{12}$$

$$L_{<} \;\sqsubseteq\; L \sqcap L^{\cup} \tag{13}$$

I do assume familiarity with the (left) domain operator.

3 Let's do it

First consider the RHS of (3); direct manipulation fails because of lack of suitable rules with conjunctions in the denominator of a division. So I try

$$(\text{LHS} \sqsupseteq P \equiv \text{RHS} \sqsupseteq P) \equiv (\text{LHS} = \text{RHS})$$

a good possibility since "/" occurs in both sides.

$$\begin{aligned}
& S \sqcap X/S^\cup \sqcap Y/(S \sqcap X/S^\cup)^\cup \sqsupseteq P \\
= & \ \{ \text{ assume } S \sqcap X/S^\cup \sqsupseteq P \ ; (4) \ \} \\
& Y \sqsupseteq P \circ (S \sqcap X/S^\cup)^\cup \\
= & \ \{ \ S^\cup \text{ is rightcondition } ; (10) \ \} \\
& Y \sqsupseteq P \circ S^\cup \sqcap P \circ (X/S^\cup)^\cup \\
\Leftarrow & \ \{ \text{ get rid of } /S^\cup \text{ by cancellation } ; \ P \sqsubseteq S \ \} \\
& Y \sqsupseteq P \circ S^\cup \sqcap X^\cup \\
= & \ \{ \text{ shunting } ; (4) \ ; \text{ assumption on } P \ \} \\
& S \sqcap X/S^\cup \sqcap (\neg X^\cup \sqcup Y)/S^\cup \sqsupseteq P \\
= & \ \{ (6) \ \} \\
& S \sqcap (X \sqcap (\neg X^\cup \sqcup Y))/S^\cup \sqsupseteq P
\end{aligned}$$

So, indeed, I arrived at the candidate $X \sqcap (\neg X^\cup \sqcup Y)$ with only one little rabbit: remove $/S^\cup$ in order to construct a candidate that does *not* depent on S. However, the price is an implication! while an equivalence was needed. It is by no means clear that such an equivalence is possible. To obtain it, it would be nice if

$$P \circ S^\cup \sqcap X^\cup \sqsubseteq P \circ S^\cup \sqcap P \circ (X/S^\cup)^\cup \tag{14}$$

under the assumption put on P in the above calculation. Since in (14) P is only composed with rightconditions and P is contained in a leftcondition by assumption, P may even be assumed to be a leftcondition.
Reintroduction of $/S^\cup$ in the LHS of (14) while enlarging it leaves me no other choice than exploiting $P \sqsubseteq X/S^\cup$, but where? Ah..., in that situation $X^\cup$ has to be removed! so (8) may help. This asks for introduction of a $(P^\cup \circ)$-translation of $X^\cup$, indeed:

$$\begin{aligned}
& P \circ S^\cup \sqcap X^\cup \\
= & \ \{ (11) \ ; (12) \ \} \\
& P \sqcap S^\cup \sqcap P_< \circ X^\cup
\end{aligned}$$

$$
\begin{aligned}
&\sqsubseteq \{\ P_{<} \sqsubseteq \{(13)\}\ P^{\cup} \sqsubseteq (X/S^{\cup})^{\cup}\ \} \\
&\quad P \sqcap S^{\cup} \sqcap (X/S^{\cup})^{\cup} \circ X^{\cup} \\
&= \{\ (8)\ \} \\
&\quad P \sqcap S^{\cup} \sqcap (X/S^{\cup})^{\cup} \\
&= \{\ (11)\ ;\ \text{"wok"}\ \} \\
&\quad P \circ (S \sqcap X/S^{\cup})^{\cup}
\end{aligned}
$$

This proves the wish and so the equation (3) for the candidate $Z = X \sqcap (\neg X^{\cup} \sqcup Y)$.
The heuristics may have shrunk the rabbits, but I think they still are alive.

The candidate may be perfect for the equation (3) (that was why it was constructed the way it is), but that doesn't mean that the challenge is met: is the candidate a preorder? Reflexivity is for free, how about transitivity?
For transitivity the distributed form of the candidate is helpful:

$$(X \sqcap \neg X^{\cup}) \sqcup (X \sqcap Y)$$

The first disjunct is the well-known nonreflexive partial order generated by the preorder X and so it is transitive. The second disjunct is the conjunction of transitive specs, thus transitive; so the cross-compositions remain. It turns out that the cross-compositions are all contained in the first disjunct. The proof thereof and the proof of the transitivity of the first disjunct follow immediately from

(15) Note $(X \sqcap \neg X^{\cup}) \circ X \sqcup X \circ (X \sqcap \neg X^{\cup}) \sqsubseteq X \sqcap \neg X^{\cup}$

Proof.

$$
\begin{aligned}
&(X \sqcap \neg X^{\cup}) \circ X \sqsubseteq X \sqcap \neg X^{\cup} \\
&\Leftarrow \{\ \circ X \text{ is monotonic}\ \} \\
&\quad X \circ X \sqcap \neg X^{\cup} \circ X \sqsubseteq X \sqcap \neg X^{\cup} \\
&\Leftarrow \{\ X \text{ is preorder}\ \} \\
&\quad \neg X^{\cup} \circ X \sqsubseteq \neg X^{\cup} \\
&= \{\ \text{left exchange}\ \} \\
&\quad X^{\cup} \circ X^{\cup} \sqsubseteq X^{\cup} \\
&= \{\ X \text{ is preorder}\ \} \\
&\quad \text{true}
\end{aligned}
$$

□

Transitivity of $X \sqcap \neg X^{\cup}$ follows from (15) since $X \sqcap \neg X^{\cup} \sqsubseteq X$.
Inclusion of the cross-compositions in $X \sqcap \neg X^{\cup}$ follows from (15) via $X \sqcap Y \sqsubseteq X$.

More challenges are anxiously awaited.

4 Comparison

Appreciation of the spec approach depends also on the alternative pointwise treatment of the challenge, but note that the pointwise approach does not cover the general situation.

First the derivation of Z such that

$$(\mathbf{A}\, S :: \lambda Z.S = \lambda Y.(\lambda X.S))$$

The candidate is derived by instantiating doubletons for S in the above, using the fact that

(16) $$aXb \equiv a \in \lambda X.\{a, b\}$$

as follows:

$$\begin{array}{ll}
& aZb \\
\equiv & \{\ (16)\ ;\ \text{instantiation}\ ;\ \text{def.}\ \lambda Y\ \} \\
& a \in \lambda X.\{a,b\} \quad \wedge \quad a \times \lambda X.\{a,b\} \subseteq Y \\
\equiv & \{\ (16)\ ;\ \text{case analysis}\ \} \\
& aXb \quad \wedge \quad (aYb \quad \vee \quad b \notin \lambda X.\{a,b\}) \\
\equiv & \{\ (16)\ \} \\
& aXb \quad \wedge \quad (aYb \quad \vee \quad b \neg X a) \\
\equiv & \{\ \text{calc.}\ \} \\
& a(X \sqcap (Y \sqcup \neg X^{\cup}))b
\end{array}$$

The derived candidate indeed does the job (for transitivity see the former section):

$$\begin{array}{ll}
& m \in \lambda(X \sqcap (Y \sqcup \neg X^{\cup})).S \\
\equiv & \{\ \text{def.}\ \lambda\ \} \\
& m \in S \quad \wedge \quad m \times S \subseteq X \quad \wedge \quad m \times S \subseteq Y \sqcup \neg X^{\cup} \\
\equiv & \{\ \text{def.}\ \lambda\ ;\ \text{calc.}\ \} \\
& m \in \lambda X.S \quad \wedge \quad (\mathbf{A}\, s : s \in S : mX^{\cup}s \Rightarrow mYs) \\
\equiv & \{\ \text{calc.}\ \} \\
& m \in \lambda X.S \quad \wedge \quad (\mathbf{A}\, s : s \in S \quad \wedge \quad sXm : mYs) \\
\equiv & \{\ m \in \lambda X.S \Rightarrow (sXm \equiv s \in \lambda X.S)\ \} \\
& m \in \lambda X.S \quad \wedge \quad m \times \lambda X.S \subseteq Y \\
\equiv & \{\ \text{def.}\ \lambda\ \} \\
& m \in \lambda Y.(\lambda X.S)
\end{array}$$

This proof is not really shorter than the spec proof, but it is a lot easier if not straightforward. Does this mean that the calculational system of the spec algebra is too extravagant, or is it extra evidence that I lack sufficient adroitness?

Thank you Richard for this interesting challenge. I feel that there is a lot to do to turn the spec calculus into an acceptable alternative for this kind of exercises; these challenges may help. I hope others will make my solution look ridiculous in the near future.

SUMS and DIFFERENTIALS

Roland Backhouse

October 8, 1991

Abstract

An elementary example is given of how the identification of a Galois connection can assist in algorithm derivation.

In the coming months a major activity of the Eindhoven Relational Type Theory Group will be to explore the theory and applications of Galois connections and adjunctions. In order to explain more clearly the sort of objectives I had in mind for such an exploration I sought an appealing example that could be presented within the space of twenty minutes. By good fortune, in connection with a quite different matter, my attention was drawn to the section on "finite calculus" in the book "Constructive Mathematics" by Graham, Knuth and Patashnik. Further reading led me to the construction of the following example which seems to fit my two criteria.

Let f and g denote functions from numbers to reals. Assume that $f.0 = 0$. Define the operators Δ and Σ by

$$\begin{array}{lcl}(\Delta f).x & = & f.(x+1) - f.x \\ (\Sigma g).x & = & \Sigma(y:\ 0 \leq y < x : g.y)\end{array}$$

for all numbers x. Then we have the Galois connection:

$$(1) \qquad f = \Sigma g \quad \equiv \quad \Delta f = g$$

The proof of this identity involves very elementary quantifier calculus and is therefore omitted.

Applying the extensionality axiom (two functions with the same domain are equal if and only if they are equal at all elements of their common domain) we obtain the equivalent but lengthier:

$$(2) \qquad \forall(x ::\ f.x = (\Sigma g).x) \quad \equiv \quad \forall(y ::\ (\Delta f).y = g.y)$$

$f.x$	$(\Delta f).x$
0	0
cx	c
$f.x + g.x$	$(\Delta f).x + (\Delta g).x$
$f.x \times g.x$	$f.x \times (\Delta g).x + g.(x+1) \times (\Delta f).x$

Table 1: Table of Differentials

Let us suppose our goal is to develop a body of rules that enable one to find efficient ways of evaluating finite sums Σg for given function g. This goal may be approached by tackling the easier problem of developing a body of rules to compute differentials Δf and then using the Galois connection (1) to convert the rules to rules about Σ.

To illustrate this idea let us restrict g to the class of polynomial functions. Our goal is thus to develop a little theory that will enable us to compute finite sums of polynomials such as $\Sigma(y:\ 0 \leq y < x:\ y^2 + 3y + 1)$.

We begin our theory development by exploring the *differentials* of polynomials. Since a polynomial function of x is either a constant function, the identity function, the sum of two polynomial functions or the product of two polynomial functions, table 1 suffices to rewrite $(\Delta f).x$ as a polynomial in x for any given polynomial $f.x$ satisfying the assumption $f.0 = 0$. (In the table c denotes an arbitrary constant. Verification of all four statements is straightforward.) We observe that a table of differentials in the finite calculus looks like a table of differentials in the infinite calculus but for the unfortunate form of the product rule. In particular taking derivatives reduces the degree of a polynomial by exactly one.

Ideally we would now like to construct a similar table for Σ. Four entries would be required, one for constants, one for the identity function, one for a sum and one for a product of two polynomials. The Δ entry for products frustrates this particular goal but nevertheless an algorithm for expressing the sum of a polynomial function as a polynomial function can be derived that exploits the above table of differentials. I shall illustrate the algorithm by considering the Σ entry for the identity function.

Since taking derivatives reduces the degree of a polynomial by one we conjecture that the sum of the identity function is a quadratic polynomial. The coefficients of that poynomial are calculated as follows:

By construction of a and b:

$$\begin{array}{ll} & \forall(x :: \ ax + bx^2 \ = \ \Sigma(y : \ 0 \leq y < x : \ y)) \\ \equiv & \quad \{ \text{ Galois connection: (2) } \} \\ & \forall(y :: \ \Delta(x \mapsto ax + bx^2).y \ = \ y) \\ \equiv & \quad \{ \text{ differential calculus: table 1 } \} \\ & \forall(y :: \ a + by + b(y+1) \ = \ y) \\ \equiv & \quad \{ \text{ arithmetic } \} \\ & a + b = 0 \ \wedge \ 2b = 1 \\ \equiv & \quad \{ \text{ arithmetic } \} \\ & a = -\frac{1}{2} \ \wedge \ b = \frac{1}{2} \end{array}$$

We have thus established the identity

$$\Sigma(y : \ 0 \leq y < x : \ y) \ = \ -\tfrac{1}{2}x + \tfrac{1}{2}x^2$$

Extrapolating from this four step calculation one can easily see that it embodies an algorithm to express Σg as a polynomial function for any given polynomial function g. The steps in the algorithm are: postulate that Σg is a polynomial function f with degree one higher than g. Compute (symbolically) the coefficients of Δf using the table of differentials. Equate the expressions obtained for the coefficients of f to the corresponding given coefficients of g. In this way one obtains a system of simultaneous equations which is then solved to obtain the coefficients of f.

The point of this little example is to show how one can predict the behaviour of a relatively complicated operator — in this case Σ — by studying the behaviour of its adjoint — in this case Δ.

Some ergonomics of mathematical notation

Roland C. Backhouse

September 30, 1991

Put your hand in your pocket and pull out a number of coins, preferably all of the same denomination. Show them to a friend and ask how many there are. If there are less than five your friend will be able to see instantly how many there are; if there are more than five he will be obliged to count them before giving a reliable response.

I have no doubt that there are many learned articles dealing with this and similar experiments in a proper, scientific way. For me, however, the experiment has a very simple and far-reaching — albeit subjective — significance. The experiment demonstrates to me just how *un*intelligent I and my fellow human beings are. We may try to convince ourselves of our supreme intelligence but the fact remains that we are quite incapable of assimilating or exploiting all but small amounts of information at any one time.

In spite of our inherent stupidity the human race has achieved a very great deal (achievement being quite different from intelligence). Recognition of one's limitations is the first step towards improving one's achievements.

The evolution of mathematical notation has been of fundamental importance to the development of science. Because it is both concise and precise, mathematical notation helps to simplify concepts to a level at which we can begin to understand them and to overcome our tendency to woolly and disorderly thinking. However, whether it is used well or badly can make all the difference between whether mathematical notation makes molehills out of mountains or mountains out of molehills. The ergonomics of mathematical notation is a little-discussed but vital aspect of creative mathematics.

A non-mathematical example may be the best introduction to the sort of points I want to make. If, when doing a crossword puzzle, I suspect that one of the answers is an anagram of some phrase then I write the letters of the

phrase in a circle. This notational trick is enormously helpful in enabling the eye to see different permutations of letters. Compactness of the notation is highly significant: a computer-generated listing of all permutations of a given set of letters may be a more reliable way of discovering all the anagrams but is decidedly less effective. Computer-generated lists are just not for human consumption!

Recognising human characteristics is important to the design of good notation. One of the first rules that one should learn about mathematical notation is that the precedence chosen for a binary operator should determine the size of the symbol used to denote that operator, the higher the precedence the smaller the symbol. This is because small symbols "pull" their neighbours together thus suggesting a grouping of the symbols. For example, in the expression

$$a + b.c + d$$

one naturally sees the sequence $b.c$ as a group because the variables b and c are close together.

Note that the size of a symbol should also include the amount of white space around it. Text produced by a typewriter illustrates this well. The expression

```
a+b.c+d
```

has been printed in teletype mode, i.e. in such a way that each symbol has exactly the same width. The intention may be that the dot has higher precedence than plus but one must work very hard in order to read the expression in that way.

The principle underlying the precedence rule is that mathematical notation should suggest relevant groupings of symbols, or at least not be biased to specific groupings. For example, if $\oplus$ is an associative operator then one should denote its application using infix notation; for then in an expression like

$$a \oplus b \oplus c \oplus d$$

one can choose at will whether to continue the calculation by manipulating $a \oplus b$, $b \oplus c$, or $c \oplus d$. In contrast, if Polish notation is used the expression above could be written in five different ways

$$\oplus(\oplus(\oplus(a, b), c), d)$$

$$\oplus(\oplus(a,b),\oplus(c,d))$$
$$\oplus(\oplus(a,\oplus(b,c)),d)$$
$$\oplus(a,\oplus(\oplus(b,c),d))$$
$$\oplus(a,\oplus(b,\oplus(c,d)))$$

each of which is biased to particular groupings of the arguments.

The advantages of infix notation for associative operators are not so striking because they are very familiar. A less familiar example is provided by so-called "abide" laws. Two binary operators $\otimes$ and $\oslash$ are said to abide with each other if for all u,v,w and x

$$(u \otimes v) \oslash (w \otimes x) = (u \oslash w) \otimes (v \oslash x)$$

Written as above the law seems hideously complex; a two-dimensional notation reveals the true nature of such laws. The name "abide" signifies that the operators can be written *ab*ove or bes*ide* each other as shown below

$$\begin{array}{ccc} u & \otimes & v \\ & \oslash & \\ w & \otimes & x \end{array} \quad = \quad \begin{array}{ccc} u & & v \\ \oslash & \otimes & \oslash \\ w & & x \end{array}$$

A standard example of an abide law is provided by multiplication and division in real arithmetic. (Replace "$\otimes$" by "$\times$" and "$\oslash$" by "/".) The validity of this law is the only justification I know for why the operands in a division are written one on top of the other. Take, for example,

$$\frac{u \cdot v}{w \cdot x}$$

Because the arguments are pulled together the eye is more readily encouraged to spot different groupings of the operands — $u \cdot v$, $\frac{u}{w}$, $w \cdot x$, $\frac{v}{x}$ and, since multiplication is commutative, $\frac{u}{x}$ or $\frac{v}{w}$.

Aside Abide laws abound in mathematics, sometimes being called interchange laws. However, they don't seem to be well known. One example occurs in boolean algebra: Suppose $p \ldots u$ are booleans and define

$$p\langle q\rangle r \quad \equiv \quad \texttt{if } q \texttt{ then } p \texttt{ else } r.$$

Then

$$\begin{array}{cccc} p & \langle q \rangle & r & \qquad\qquad p \qquad\quad r \\ & \langle s \rangle & & = \quad \langle s \rangle \quad \langle q \rangle \quad \langle s \rangle \\ t & \langle q \rangle & u & \qquad\quad t \qquad\quad u \end{array}$$

End of Aside

(Readers of The Squigolist will know that the term "abide law" was coined by Richard Bird and that the above example of such a law is due to Tony Hoare.)

Subscripts and superscripts are probably the most abused elements of mathematical notation. Because they are smaller than the symbols around them they are easily overlooked. Just like the small print in legal documents this can be deliberately used to deceive the reader, or it can be used to suppress details that are only relevant in exceptional circumstances. Very occasionally deception can be beneficial! Suppose a given function distributes over a given binary operator. Denoting the function by C, the operator by $\times$ and function application by an infix dot, distributivity can be expressed syntactically by

$$C.(X \times Y) = C.X \times C.Y$$

for all X and Y. An alternative denotation is obtained by choosing c to denote the function and using superscripting to denote function application. We then obtain

$$(X \times Y)^c = X^c \times Y^c$$

for all X and Y. (To emphasise my point about the size of superscripts I have used capital letters for the dummies.) What is the essential difference between the two notations? Well, compare $C.X \times C.Y$ with $X^c \times Y^c$. In the former "X" and "Y" are relatively far apart, in the latter "X" and "Y" have been pulled together by the relative size of the dummies and the superscript. In the latter, therefore, the intention is that the eye is tricked into overlooking the superscript and grouping together X and Y. The notation avoids the need to consciously remember the distributivity law.

Now you know why the "Eindhoven School" insists on beautiful handwriting: clear handwriting, paying attention to the ergonomics of mathematical notation, pays dividends whereas bad handwriting can often deceive you into making mistakes. And those computer algebra systems that are currently all the rage? How anyone can begin to do creative mathematics with an input-output system that is hardly better than that of a teletype is beyond me!

On Demonic Composition

Roland Backhouse

October 28, 1991

Demonic composition has been the subject of at least two recent articles [1, 2]. In both these papers a pointwise definition is given from which, by magic, a — rather ugly — point-free definition is plucked out of the blue. Subsequently, proofs are given of the associativity of the so-defined operator.

Neither proof provided to my mind any justification for point-free reasoning. Rather the opposite: Berghammer's proof [1] struck me as a "machine-code proof" in that the definition of demonic composition was used to expand the two possible ways of composing three specs into — inevitably very long — expressions involving the primitive operators of the plat calculus. These expressions were then laboriously proved to be equal. Van der Woude's proof [2] used exactly the same strategy but was a little better in that his primitives were a little less primitive than Berghammer's and consequently the expressions he had to manipulate were a little shorter. To be fair to van der Woude he made his own dissatisfaction with the proofs explicit by entitling the paper "Free style Specwrestling" and concluding with the words: "Frustating is still the fact that the truth of the statements in the tasks is easily seen with a pointwise interpretation but a rigorous proof on the spec level is still unappealing." In this note I want to take up the challenge of constructing an appealing point-free proof of the associativity of demonic composition.

The task according to van der Woude [2] begins as follows: "Define the demonic composition $R; S$ as the usual composition, provided that it is only defined in states x such that R is defined on all the S-results Sx."

Interpreting this specification literally we are required to specify $R; S$ formally via two clauses: The first clause states that it is "the usual composition" but with a restricted right domain. I.e.

$$(1) \qquad R;S \;=\; R \circ S \circ R\&S$$

where

(2) $\quad monotype.(R\&S)$

Thus $R\&S$ is the "restriction" on the right domain.

The second clause states that the said restriction should include only those "states x such that R is defined on all the S-results Sx." Replacing "states x" by "monotypes B", we formulate this second clause as the requirement that $R\&S$ satisfy the specification:

(3) $\quad A :: \quad \forall(B : monotype.B : A \sqsupseteq B \equiv R\rangle \sqsupseteq (S \circ B)\langle)$

1 Preliminary Analysis

Before embarking on the task of proving that demonic composition is indeed associative let us examine the more elementary consequences.

We begin with the conjunction of (1) and (2). An immediate consequence — at least immediate to the experienced "speculist" — is

(4) $\quad (R;S)\rangle = (R \circ S)\rangle \circ R\&S$

whence also

(5) $\quad R;S = R \circ S \circ (R;S)\rangle$

For the proofs of (4) and (5) we appeal to a slightly more general lemma and its corollary, specifically:

Lemma 6 For all specs S and monotypes A,

$$(S \circ A)\rangle = S\rangle \circ A$$

Proof

$$\begin{array}{ll} & (S \circ A)\rangle \\ = & \quad \{ \text{ domains } \} \\ & (S\rangle \circ A)\rangle \\ = & \quad \{ S\rangle \text{ and } A \text{ are both monotypes. Hence, so is } S\rangle \circ A \} \\ & (S\rangle \circ A) \end{array}$$

□

Corollary 7

$$\exists(A : \mathit{monotype}.A : R = S \circ A) \quad \equiv \quad R = S \circ R{>}$$

Proof Follows-from is obvious. For the implication we have:

$$\begin{array}{ll} & R = S \circ A \\ \equiv & \quad \{\ S = S \circ S{>}\ \} \\ & R = S \circ A = S \circ S{>} \circ A \\ \Rightarrow & \quad \{\ \text{lemma 6}\ \} \\ & R = S \circ R{>} \end{array}$$

□

(Lemmas that have two-step proofs may not be "immediate" to the novice but surely are to the more practised.) Equation (4) is an instance of lemma 6 in which S is instantiated to $R \circ S$ and A to $R\&S$. Similarly, (5) is an application of corollary 7 in which the same assignments are made to S and A, and R is instantiated to $R; S$.

Now we consider (3). The immediate question is whether there is a solution to the specification. To see that this is indeed the case — at a glance — we observe that both the functions $<$ and $S\circ$ are universally $\sqcup$-junctive, hence so is their composition and thus

$$(8) \qquad \sqcup(B : \mathit{monotype}.B \wedge R{>} \sqsupseteq (S \circ B){<} : B)$$

solves (3). As a function of R it is also obviously a monotype transformer (i.e. a function mapping monotypes to monotypes), and we may conclude that the binary operator & does indeed exist.

Knowing this formula is however of little help in any calculations involving $R\&S$ since the inevitable first step in any such calculation will be to return to (3). More progress can be made if one is aware that being universally $\sqcup$-junctive is equivalent to having a certain sort of adjoint. Note that the requirement on $R\&S$ — for all monotypes B and all specs R and S,

$$(9) \qquad R\&S \sqsupseteq B \quad \equiv \quad R{>} \sqsupseteq (S \circ B){<}$$

— is almost a Galois connection between the function $(R \mapsto R\&S)$ and the function $(B \mapsto (S \circ B){<})$. That it is not so can be solely attributed to the occurrence of the right domain operator on the right side of (9).

We can dismiss this obstacle by noting that, for all monotypes A we have $A{>} = A$ and, in particular, $(R{>}){>} = R{>}$. Consequently,

$$(10) \qquad R\&S \;=\; R{>}\&S$$

where, for all monotypes A and B,

$$(11) \qquad A\&S \sqsupseteq B \;\equiv\; A \sqsupseteq (S \circ B){<}$$

Property (10) tells us that the left operand of & may always, without loss of generality, be assumed to be a monotype. Property (11) says that — with the said assumption — the function $\&S$ is adjoint to the function $(B \mapsto (S \circ B){<})$. I.e. in the domain of monotypes there is a Galois connection between $\&S$ and the composition of the two functions $<$ and $S\circ$.

The recognition of a Galois connection is a very crucial observation and unleashes a welcome gush of properties. In order to proceed more quickly to our main task we limit attention to those that prove to be directly relevant. There are just two. The first is the cancellation property:

$$(12) \qquad A \;\sqsupseteq\; (S \circ A\&S){<}$$

Equivalently,

$$(13) \qquad A \circ S \circ A\&S \;=\; S \circ A\&S$$

Comparing (13) with (4) the experienced speculist should spot that

$$(14) \qquad (R;S){>} \;=\; S{>} \circ R\&S$$

which proves to be a crucial lemma in the proof of associativity. The four-step proof of (14) follows:

$$\begin{array}{ll} & (R;S){>} \\ = & \quad \{\ (1)\ \} \\ & (R \circ S \circ R\&S){>} \\ = & \quad \{\ \text{domains}\ \} \\ & (R{>} \circ S \circ R\&S){>} \\ = & \quad \{\ (10),\ (13)\ \} \\ & (S \circ R\&S){>} \\ = & \quad \{\ \text{lemma 6, } R\&S \text{ is a monotype}\ \} \\ & S{>} \circ R\&S \end{array}$$

The second is that the monotype transformer $\&S$ is universally $\sqcap$-junctive.

Since, however, for monotypes the $\sqcap$ operator coincides with composition the monotype transformer $\&S$ is universally composition-junctive and, more particularly, for all monotypes A and B,

$$(15) \qquad (A \circ B)\&S \;=\; A\&S \circ B\&S$$

2 The Proof of Associativity

Now let us turn to the task in hand — proving that demonic composition is associative. We consider the two terms $R;(S;T)$ and $(R;S);T$, and expand each using (1) very cautiously in order not to allow the formulae to grow too big. First, we obtain

$$\begin{array}{ll} & R;(S;T) \\ = & \quad \{\ (1)\ \} \\ & R \circ (S;T) \circ R\&(S;T) \\ = & \quad \{\ (1)\ \} \\ & R \circ S \circ T \circ S\&T \circ R\&(S;T) \end{array}$$

(Note that the outermost occurrence of ";" has been expanded first. Expanding the innermost occurrence leads to a larger formula.)

This is a pleasing result becauses it expresses $R;(S;T)$ in terms of a restriction on the right domain of $R \circ S \circ T$. Now for the other term:

$$\begin{array}{ll} & (R;S);T \\ = & \quad \{\ (1)\ \} \\ & (R;S) \circ T \circ (R;S)\&T \\ = & \quad \{\ \text{Applying (1) for a second time would introduce an undesirable restriction on the } \textit{left} \text{ domain of } T, \text{ not on the right. We search around for something more suitable. Aiming for (13) we apply (5)}\ \} \\ & R \circ S \circ (R;S)\!> \circ T \circ (R;S)\&T \\ = & \quad \{\ (10),\ (13),\ R,S := (R;S),T\ \} \\ & R \circ S \circ T \circ (R;S)\&T \end{array}$$

Thus $(R;S);T$ has also been expressed in terms of a restriction on the right domain of $R \circ S \circ T$ and we can infer that

$$
\begin{array}{lcl}
R;(S;T) & = & (R;S);T \\
\Leftarrow & & \\
S\&T \circ R\&(S;T) & = & (R;S)\&T
\end{array}
$$

The reader will undoubtedly have observed that only limited use of (3) has been used. The cancellation property (12) has been used but nowhere have we used the fact that $R\&S$ is the *limit* of a set of monotypes. This element of the specification will figure highly in the final proof obligation which is to show that

$$(16) \qquad S\&T \circ R\&(S;T) \;=\; (R;S)\&T$$

Demonic composition is still present in both the left and right sides of this equation. Let us try to remove it using the adjointness of the & operator. We choose to begin with the right side of (16), this choice being made because the demonic composition appears in the left argument of the & operator and we know so much more about the behaviour of that operator with respect to its left operand than with respect to its right operand.

$$
\begin{array}{ll}
 & (R;S)\&T \\
= & \quad \{\ (10)\ \} \\
 & (R;S)\!>\!\&T \\
= & \quad \{\ (14)\ \} \\
 & (S\!>\circ R\&S)\&T \\
= & \quad \{\ (15)\ \} \\
 & S\!>\!\&T \circ (R\&S)\&T \\
= & \quad \{\ (10)\ \} \\
 & S\&T \circ (R\&S)\&T
\end{array}
$$

Summarising,

$$(17) \qquad (R;S)\&T \;=\; S\&T \circ (R\&S)\&T$$

The right side of (17) is very close to the left side of (16). Only the terms $R\&(S;T)$ and $(R\&S)\&T$ differ. We now try to eliminate the demonic composition appearing in the former and simultaneously prove its equality to the latter. (It turns out that they are not equal but that is our proof strategy nevertheless.)

Since the demonic composition appears in the second operand we have little choice but to apply (9). We have, for all monotypes B,

$$
\begin{array}{ll}
 & R\&(S;T) \sqsupseteq B \\
\equiv & \quad \{ (9) \} \\
 & R\rangle \sqsupseteq ((S;T) \circ B)\langle \\
\equiv & \quad \{ (1) \} \\
 & R\rangle \sqsupseteq (S \circ T \circ S\&T \circ B)\langle \\
\equiv & \quad \{ \bullet \text{ assume } S\&T \sqsupseteq B, \text{ monotypes} \} \\
 & R\rangle \sqsupseteq (S \circ T \circ B)\langle \\
\equiv & \quad \{ \text{domains} \} \\
 & R\rangle \sqsupseteq (S \circ (T \circ B)\langle)\langle \\
\equiv & \quad \{ (9) \} \\
 & R\&S \sqsupseteq (T \circ B)\langle \\
\equiv & \quad \{ (9) \} \\
 & (R\&S)\&T \sqsupseteq B
\end{array}
$$

We have thus established that

$$
\begin{array}{ll}
 & S\&T \sqsupseteq B \ \wedge \ R\&(S;T) \sqsupseteq B \\
\equiv & \\
 & S\&T \sqsupseteq B \ \wedge \ (R\&S)\&T \sqsupseteq B
\end{array}
$$

from which it follows (since all inclusions are between monotypes) that

(18) $\quad S\&T \circ R\&(S;T) \ = \ S\&T \circ (R\&S)\&T$

Combining (17) and (18) we have established (16) and our task is complete.

3 Discussion

Our concern here has not been to *establish* a mathematical theorem — that demonic composition is associative has been known for decades — but with economy and elegance of calculation. Writing the note was prompted by discontent with the only two proofs that I know of using the axiomatic relational calculus. In this section I want to take the opportunity to compare those proofs with that given here in order to clarify my criticisms and to reinforce the lessons that I believe can be learnt form this exercise.

Both van der Woude and Berghammer use explicit, quantifier-free formulae for $R;S$ which, I complained earlier, are plucked out of the blue. Before embarking on the discussion of relative merits it is worthwhile to see how one might derive those formulae. The task is thus to derive a definition of R,S that fulfills the specification as given by (1), (2) and (3). (The formula (8) is inadequate because it is not quantifier-free.)

The calculation amounts to finding a closed form for $R\&S$ which is then substituted in (1). In broad terms the calculation of $R\&S$ consists of three steps. In the first step we reduce the problem to the calculation of $A\&S$, for monotype A, using (10). In the second step the adjoints of $<$ and $S\circ$ are composed, giving a solution to (3) but one that does not map monotypes to monotypes. In the final step the latter complication is overcome. In detail the calculation is as follows, beginning with the second and third steps. For all monotypes A and B and all specs S, we have:

$$
\begin{array}{ll}
 & A \;\sqsupseteq\; (S\circ B)< \\
\equiv & \quad \{ \text{ adjoint of } < \} \\
 & A\circ\top\top \;\sqsupseteq\; S\circ B \\
\equiv & \quad \{ \text{ adjoint of } S\circ \} \\
 & S\backslash(A\circ\top\top) \;\sqsupseteq\; B \\
\equiv & \quad \{ \text{ An adjoint has been found but is not} \\
 & \quad\ \text{a monotype transformer. We reintroduce } < \\
 & \quad\ \text{to obtain a monotype.} \\
 & \quad\ \mathit{leftcondition}.(U\backslash V) \Leftarrow \mathit{leftcondition}.V, \text{ domains } \} \\
 & (S\backslash(A\circ\top\top))<\circ\top\top \;\sqsupseteq\; B \\
\equiv & \quad \{ B = B<, \text{ adjoint of } < \} \\
 & (S\backslash(A\circ\top\top))< \;\sqsupseteq\; B
\end{array}
$$

Comparing with (11) the appropriate definition of $\&S$ for monotype A is

(19) $\quad A\&S \;=\; (S\backslash(A\circ\top\top))<$

and for arbitrary spec R (see (10))

(20) $\quad R\&S \;=\; (S\backslash(R>\circ\top\top))<$

Finally, by substitution in (1), we have

(21) $\quad R;S \;=\; R\circ S\circ(S\backslash(R>\circ\top\top))<$

The right side of (21) is an ugly formula, and direct manipulation of it is strongly discouraged. Some simplification is possible although the gain is marginal. The (equivalent) formulae used by van der Woude [2] and Berghammer [1] were, respectively,

$$(22) \qquad R;S \;=\; (R \circ S) \sqcap (\top\!\!\top \circ R)/S\cup$$

and

$$(23) \qquad R;S \;=\; (R \circ S) \sqcap \neg(\neg(\top\!\!\top \circ R) \circ S)$$

The main difference between the proof that I have presented here and those of Berghammer and van der Woude is the ubiquitous use of monotypes and the domain operators instead of right conditions/vectors. ("Right condition" is the term used by van der Woude, "vector" is the term used by Berghammer. Their meaning is the same, namely, spec R is a right condition/vector iff $R = \top\!\!\top \circ R$.)

The choice of which to use is difficult because right domains and right conditions are intimately connected — indeed Galois connected. Specifically, for all monotypes A and specs R we have

$$(24) \qquad A \sqsupseteq R\!> \;\equiv\; \top\!\!\top \circ A \sqsupseteq R$$

(which result is due to van der Woude). Moreover, this is a somewhat special Galois connection in that $(\top\!\!\top \circ A)\!>$ is *equal* to A, whereas the existence of a Galois connection predicts only an inclusion between the two.

The principle argument for the use of right conditions is that they are closed under negation whereas monotypes are not. Against that must be weighed the fact that intersection coincides with composition for monotypes and the domain operators both have adjoints for arbitrary specs rather than just for monotypes.

In my own mind I have no doubt that the emphasis on right (and left) conditions is misplaced. Those who work with them have to clutter their brains with ugly distributivity laws such as

$$(25) \qquad R \circ (S \circ \top\!\!\top \sqcap T) \;=\; (R \sqcap \top\!\!\top \circ S\cup) \circ T$$

whereas for the user of the domain operators it suffices to be aware that composition in the expression

$$(26) \qquad R \circ S\!< \circ T$$

is associative. That right conditions are closed under negation is, in my experience, rarely relevant. Indeed, one reason for preferring van der Woude's proof to Berghammer's is that negation has almost been eliminated. (In fact, by using Dedekind's rule it could have been eliminated altogether.)

The ugliness of the equational properties of right conditions is illustrative, I believe, of a more general phenomenon. When two functions are connected by a Galois connection it is often the case that one has very amenable algebraic properties whereas the other is much more difficult to work with. Commonly also one function is well-known, the other not (for example, think of composition and factors). From a calculational viewpoint a commendable heuristic would thus seem to be to limit explicit use of the algebraic properties of the "ugly sister" as much as possible by appealing instead to the Galois connection combined with the properties of its more beautiful partner. That is the tactic that has been adopted above.

A final remark on the length of the paper: I have been somewhat verbose in the presentation of the proofs in order to properly explain the important considerations at each step. Bearing this in mind, I believe that the length of the calculations compares favourably with those of van der Woude and Berghammer.

References

[1] R. Berghammer. Relational specification of data types and programs. Bericht Nr. 9109, Universität der Bundeswehr München, Fakultät für Informatik, September 1991.

[2] Jaap van der Woude. Free style specwrestling: Demonic composition and choice. In *Lambert Meertens, CWI, Liber Amicorum, 1966-1991.* Stichting Mathematisch Centrum, Amsterdam, January 1991.

How to compute $y - x$ or why overloading is a minus

Eerke Boiten

It is required to write a Squiggol function that computes, given the pair (x, y), the value $y - x$. Let us call this function $\ominus$. A first definition of $\ominus$ might be:

$$\ominus\,(x, y) = y - x \tag{1}$$

So far, so good – but this definition uses *dummies*! Heresy!

We now give two derivations of $\ominus$, one top-down, one bottom-up.

Top-down development

A careful look at specification (1) leads us to consider two subproblems: first exchange x and y, and then apply ordinary $-$. Thus we have (using known solutions for the subproblems)

$$\ominus = - \cdot (\gg, \ll)$$

which, using $\oplus \cdot (\gg, \ll) = \tilde{\oplus}$ can be written as:

$$\ominus = \tilde{-}$$

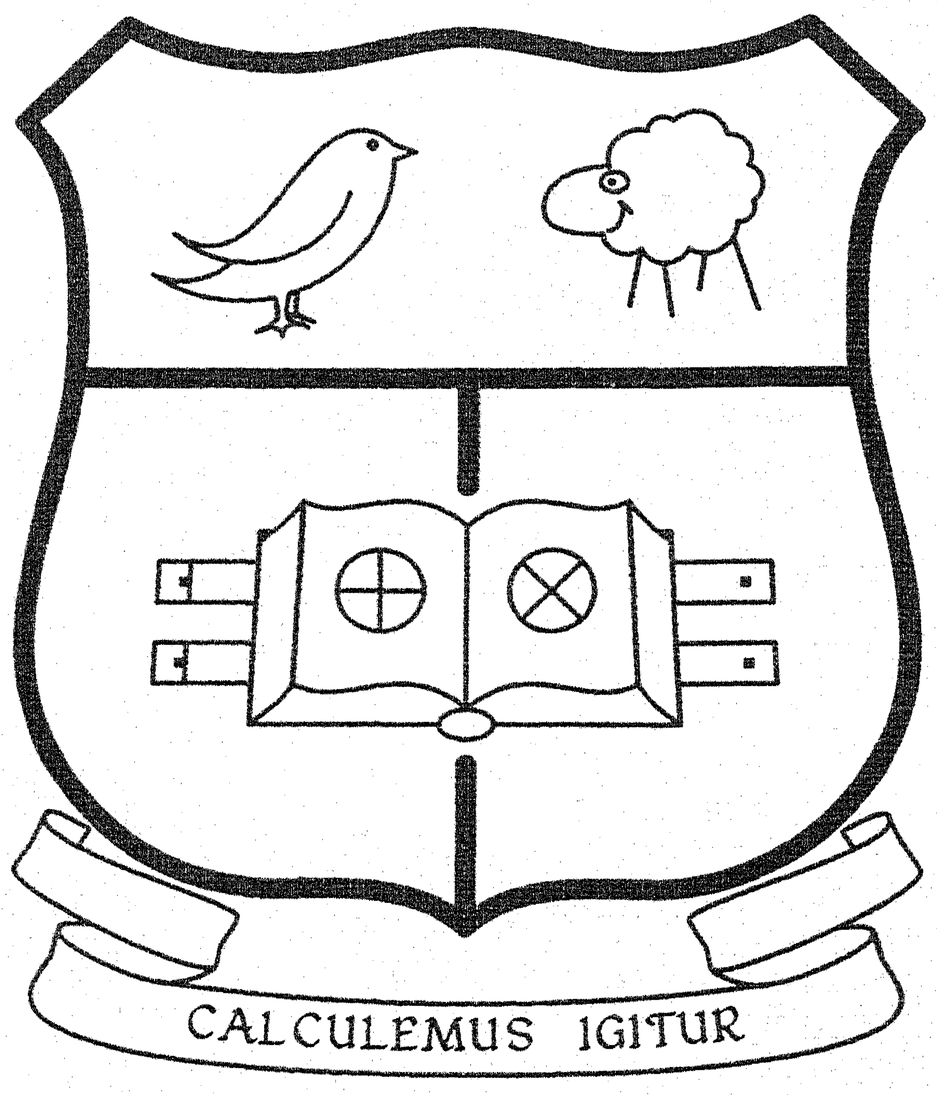
Bottom-up development

Let us just see what we *can* do. If we just apply $-$ to (x, y), we get $x - y$. This is not what we need, but fortunately we have the unary operator $-$, which turns $x - y$ into $y - x$. Altogether, we have:

$$\ominus = - \cdot -$$

Combining the results of these two developments, we get a concise formulation of a well-known law of arithmetic:

$$\tilde{-} = - \cdot -$$


CALCULEMUS IGITUR