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

The Squiggolist

The Squiggolist, Volume 1, Number 4, June 1990

AMSTERDAM Morphisms under control. Because of the uncontrolled growth of different classes of morphisms (in the last half year, paramorphisms, zygomorphisms, anamorphisms, hypomorphisms, and mutumorphisms have been introduced) the universities of Utrecht, Oxford, Groningen, and Nijmegen have called AIM, the Association for the Issue of Morphisms into existence. For a specific class of functions or relations to be considered a morphism by AIM it has to satisfy at least two properties. First, it has to be either a generalisation or a subclass of one of the above named classes of morphisms, and second, there must exist a nice greek prefix which corresponds in a natural way to the class of mor[illegible] under consideration. Applicat[illegible] cognition of classes of morphisms, re- quirements, must be for the Issue of M AA, P.O. Box 407 The Netherlands.

UTRECHT Spy exposed. The Dutch Home Security Office has exposed a russian spy. In these days of relaxation of relations between West and East, the KGB spy H.Z. has infiltrated in the Dutch scientific world. Z. tried to disregulate the scientific production of the computing science departme Utrecht University by posing so called problems. KGB problems are probl are guara eed to have absolutely cal val ever, and which req least t ear to solve. The t of th D. Draaïsma said, "T come just in t eputation res m at mos

AMSTERDAM Abuse of symbols. The Dutch minister of transportation has recently filed a law suit against a group of scientists for plagiarism. As a result of recent scientific discoveries more and more symbols, idently modelled after traffic signs, are ap ing on computer screens and in print. esman said: "Of course we appreci me likes our basic designs, and we ection at all that people like to bols at home with a fountain ale at which copying now s source of confusion to

Tupling and mutumorphisms

Maarten Fokkinga

On induced congruences : an answer

Jaap van der Woude

Balanced binary trees as a partial free algebra

Jeremy Gibbons

Calculation by computer

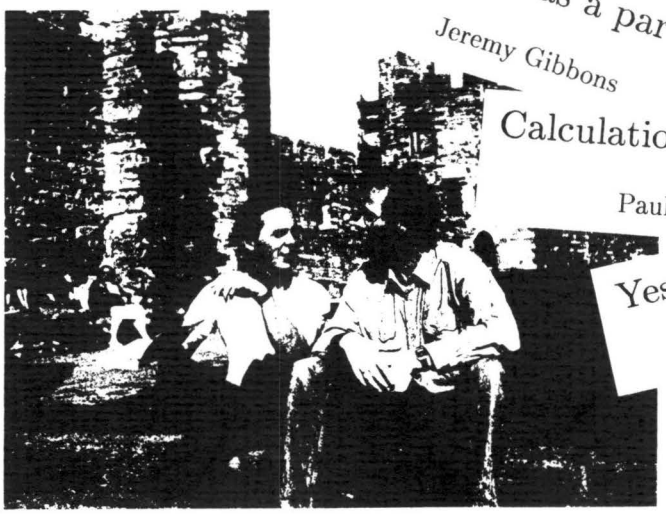
Paul Chisholm

Yes, let's calculate

Jan Kuper

Selector guards

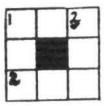
Maarten Fokkinga



Soft arrays

Alan Jeffrey

GRONINGEN Categorical g rolled up. The Committee for Unsqu Activities has dealt the subversive Cat cal Movement another, final, blow. The leader, who has been reported to be fl last year, has been found hiding in the city of Oxford, thereby exposing yet another branch of the Movement of which he is recognised to be the leader. The beaming principal investigator of the CUA told our reporter : "I am positive that we have rolled up this gang consisting solely of categorical perverts."



ıital
$\oplus/ \cdot (f \| g) ** \cdot \mathsf{subs}$
2. $h \cdot (\llbracket h \rrbracket \Uparrow \mathsf{id}) \dagger \cdot j$
vertical
1. $(\llbracket \oplus / \rrbracket) \cdot \mathsf{in} \cdot \llbracket h \rrbracket$
3. $\Upsilon \cdot f * \| g * \cdot j$

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

This is the fourth and last issue of the first volume of the Squiggolist. The contents of all the numbers of the first volume is given on the next page. 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. The Squiggolist has been quite succesful until now: the number of submitted papers keeps growing, and quite a few are reactions upon papers in previous issues of the Squiggolist. I will produce the next issue in September or October, so all papers sent to me before September will be considered for the next isue.

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 of volume 1 of The Squiggolist.

Yes, let's calculate

Jan Kuper
University of Twente

May 10, 1990

In this note we will present an alternative approach to a problem discussed and solved by Richard Bird in two papers, entitled: *Shall we calculate?* and *Shall we calculate–II?* ([1,2]).

We thank Maarten Fokkinga for his stimulating criticism on earlier drafts of this note.

1 The problem

In order to make this note self-contained we start with a description of the problem. Our description is a partial quotation from [1].

> Suppose we want to group the red-headed members of a set of people into classes depending on their sex.
>
> One obvious procedure is first to take the subset of red-headed people and then do the grouping by sex.
>
> Here is another procedure: first classify the original set of people by sex, and then take the red-headed members of each group.
>
> Are these two procedures the same? Imagine that Alice and Diana have red hair, while Bill and Edward do not. The first procedure leads to the single component partition
>
> $$\{\{\text{Alice}, \text{Diana}\}\}$$
>
> while the second procedure, as stated, leads to the set of sets
>
> $$\{\{\text{Alice}, \text{Diana}\}, \emptyset\}$$
>
> It is clear, therefore that we need to amend the second procedure by removing empty sets.

In [1] a general form of the problem is formalized by means of the equation

$$Eq\, f \cdot p\triangleleft \;=\; (\neq\emptyset)\triangleleft \cdot p\triangleleft * \cdot Eq\, f \qquad (1)$$

where p is an arbitrary predicate (above: $p =$ *red-headed*), and $Eq\, f$ is a function that groups a set into equivalence classes under the relation of having the same f-value (where f is some given function. Above: $f =$ *sex*). The task is to define $Eq\, f$ and to prove (1) by calculation.

In [1] and [2] Bird proves equation (1) for two alternative definitions of $Eq\, f$ which we repeat here. In [1] the proof is based on the following definition of $Eq\, f$:

$$\begin{aligned} y \sim_f z &\;\Leftrightarrow\; fy = fz \\ equiv\, f\, x\, y &\;=\; (y\sim_f) \triangleleft x \\ Eq_1\, f\, x &\;=\; (\neq\emptyset) \triangleleft equiv\, f\, x * Dom\, f \end{aligned}$$

where $Dom\, f$ denotes the domain of f.

In [2] the following definition is used:

$$\begin{aligned} Inv\, g\, d &\;=\; \{y \mid gy = d\} \\ q_f u &\;=\; (\#(f * u) = 1) \\ Eq_2\, f &\;=\; \downarrow_\# / \cdot all\, q_f \triangleleft \cdot Inv(\cup/) \end{aligned}$$

Especially the proof in [2] is unexpectedly complicated. It ends with the question: "can it be simplified?".

In the remaining part of this note we will formulate some conceptual considerations which motivate a different approach towards defining $Eq\, f$. We will prove equation (1) using a rather succinct definition of $Eq\, f$ and prove its equivalence with several other definitions, a.o., $Eq_1\, f$. We will not prove (1) for $Eq_2\, f$ directly (as is done in [2]) but instead we prove that $Eq_2\, f$ is equivalent to $Eq_1\, f$. We do not claim that it leads to a shorter proof than presented in [2] (we can not judge that since the proof in [2] is not complete), but we think its elementary steps as well as its line of reasoning are rather simple and we hope that the techniques used will be applicable in other situations as well. We do not give all proofs in detail, partly because of lack of space but mainly because we want to focus on the central ideas in the approach.

2 Some conceptual remarks

In this section we will give some conceptual observations which lead to a simpler definition and proof than given in [1]. The simplification is given in three steps.

Step 1. Consider from the introductory description the fragment "do the grouping by sex". In general, the result of grouping a set x of people by their sex is a set of two subsets of x: the subset of men and the subset of women. But what, if x consists only of women (men)? We think that in that case too the process of grouping x by sex may yield two subsets of x: x itself and the empty set. Hence, in the example above both procedures may lead to the result

$$\{\{\text{Alice}, \text{Diana}\}, \emptyset\}.$$

Consequently, the conclusion in [1] that "it is clear, therefore that we need to amend the second procedure by removing empty sets", may be replaced by the conclusion that the *first* procedure must be amended by *adding* the empty set. To our opinion, this conclusion is closer to the intuition behind "do the grouping by sex". Hence, equation (1) becomes

$$E\, f \cdot p\triangleleft \;=\; p\triangleleft * \cdot E\, f \tag{2}$$

where $E\, f\, x$ differs from $Eq\, f\, x$ in that it may contain the empty set.

It is immediately clear that – if we want to get rid of the empty set after all – both sides may be filtered by $(\neq\emptyset)$. It is remarkable how much simpler the original proof given in [1] becomes: its length is reduced by about 50%. The only intermediate results used in [1] which remain necessary, are (2) and (7) (these numbers refer to [1]).

Step 2. Consider again the fragment "do the grouping by sex": according to the same point of view as referred to in step 1 there should be exactly one equivalence class of people for every sex. That is to say, there is a bijection between the range of a function f and the set of equivalence classes which results from dividing the domain of f under $\sim_f$. Clearly, this bijection is given by $Inv\, f$. It can be elegantly used in defining $E\, f$:

$$E\, f\, x \;=\; (x\cap \cdot\, Inv\, f) * Rng\, f \tag{3}$$

where $Rng\, f$ denotes the range of f.

In order to prove (2) using this definition we need one intermediate result:

$$p\triangleleft \cdot x\cap \;=\; (p \triangleleft x)\cap \tag{4}$$

Its proof is simple and left to the reader. Now the proof of (2) contains only elementary steps:

$$
\begin{array}{ll}
 & (E\,f \cdot p\triangleleft)x \\
= & \quad \text{definition of composition, } E\,f \\
 & ((p \triangleleft x)\cap \cdot \mathit{Inv}\,f) * \mathit{Rng}\,f \\
= & \quad (4), \text{ associativity of composition} \\
 & (p\triangleleft \cdot x\cap \cdot \mathit{Inv}\,f) * \mathit{Rng}\,f \\
= & \quad *\text{-distributivity} \\
 & p\triangleleft * (x\cap \cdot \mathit{Inv}\,f) * \mathit{Rng}\,f \\
= & \quad \text{definition of composition, } E\,f \\
 & (p\triangleleft * \cdot E\,f)x.
\end{array}
$$

Step 3. After grouping a set of people by their sex, we will know which equivalence class is the class of women (men) not only by its members, but also by the grouping process itself. Hence, the result of the grouping process naturally is an *indexed* set of equivalence classes, i.e., it is a *function* from $\{male, female\}$ to the set of (possibly empty) equivalence classes. The indexed $E\,f$ is written as $Ei\,f$ and its definition is simple:

$$Ei\,f\,x = x\cap \cdot \mathit{Inv}\,f$$

It is possible to reformulate equation (2) for $Ei\,f$. In doing so, *map* must be replaced by *composition*:

$$Ei\,f \cdot p\triangleleft \;=\; (p\triangleleft \cdot) \cdot Ei\,f \tag{5}$$

The proof of (5) follows exactly the same lines as the proof of (2) under step 2.

Clearly, we have

$$E\,f\,x \;=\; Ei\,f\,x * \mathit{Rng}\,f \tag{6}$$

The proof that (2) follows from (5) by using (6) is straightforward and left to the reader.

3 An application to the definition of $Eq\,f$

We return to $Eq\,f$, i.e., the empty set is removed (if present) as proposed by Bird. In the light of the foregoing a straightforward definition of $Eq\,f$ is:

$$Eq\,f \;=\; (\neq\emptyset)\triangleleft \cdot E\,f \tag{7}$$

We prove (2) using this definition:

$$
\begin{aligned}
& Eq\, f \cdot p\triangleleft \\
= & \qquad \text{definition of } Eq\, f \\
& (\neq\emptyset)\triangleleft \cdot E\, f \cdot p\triangleleft \\
= & \qquad \text{cf. Section 2} \\
& (\neq\emptyset)\triangleleft \cdot p\triangleleft * \cdot E\, f \\
= & \qquad gd = d \text{ implies } (\neq d)\triangleleft \cdot g* \;=\; (\neq d)\triangleleft \cdot g* \cdot (\neq d)\triangleleft \\
& \qquad \text{as the reader may check easily} \\
& (\neq\emptyset)\triangleleft \cdot p\triangleleft * \cdot (\neq\emptyset)\triangleleft \cdot E\, f \\
= & \qquad \text{definition of } Eq\, f \\
& (\neq\emptyset)\triangleleft \cdot p\triangleleft * \cdot Eq\, f
\end{aligned}
$$

Definition (3) of Section 2 suggests an alternative definition of $Eq\, f$ in which the intersection $x\cap$ "afterwards" is replaced by the restriction $\restriction x$ "beforehand":

$$Eq\, f\, x \;=\; Inv(f \restriction x) * Rng(f \restriction x) \tag{8}$$

where $f \restriction x$ denotes the *restriction* of the function f to the set x, i.e.,

$$
\begin{aligned}
(f \restriction x)y \;&=\; fy \qquad\quad \text{if } y \in x \\
&=\; \textit{undefined} \quad \text{otherwise}
\end{aligned}
$$

In the remaining part of this note it is understood that $Inv\, f\, d$ is ***undefined*** if $d \notin Rng\, f$, i.e., $Dom \cdot Inv = Rng$. (An alternative choice would be that $Inv\, f\, d = \emptyset$ for $d \notin Rng\, f$.) So here (8) is equivalent to the following concise definition:

$$Eq\, f \;=\; Rng \cdot Inv \cdot f\restriction \tag{9}$$

Finally, we give a definition which is a simpler variant of a definition mentioned by Bird (in [1]):

$$Eq\, f\, x \;=\; equiv\, f\, x \;*\; x \tag{10}$$

We prove that these definitions (7 – 10) are all equivalent to each other and to $Eq_1\, f$. All definitions (except (9) which is equivalent to (8), see above) are touched upon in one calculation. The proof of their equivalence to $Eq_2\, f$ is postponed to the next section.

In the following calculation as well as in the rest of this note, it is understood that $x \subseteq Dom\, f$.

$$
\begin{array}{ll}
 & \mathit{equiv}\ f\ x * x \qquad\qquad [\text{definition (10)}] \\
= & \quad (11), \text{ see below} \\
 & (x\cap \cdot \mathit{Inv}\ f \cdot f) * x \\
= & \quad *\text{-distribution, } f * x = \mathit{Rng}(f \restriction x) \\
 & (x\cap \cdot \mathit{Inv}\ f) * \mathit{Rng}(f \restriction x) \\
= & \quad (12), \text{ see below} \\
 & \mathit{Inv}(f \restriction x) * \mathit{Rng}(f \restriction x) \qquad\qquad [\text{definition (8)}] \\
= & \quad (12) \text{ again} \\
 & (\neq\emptyset) \lhd (x\cap \cdot \mathit{Inv}\ f) * \mathit{Rng}\ f \qquad\qquad [\text{definition (7)}] \\
= & \quad f * \mathit{Dom}\ f = \mathit{Rng}\ f, *\text{-distribution} \\
 & (\neq\emptyset) \lhd (x\cap \cdot \mathit{Inv}\ f \cdot f) * \mathit{Dom}\ f \\
= & \quad (11) \text{ again} \\
 & (\neq\emptyset) \lhd \mathit{equiv}\ f\ x * \mathit{Dom}\ f \qquad\qquad [\mathit{Eq}_1\ f\ x]
\end{array}
$$

The properties used in this proof are:

$$\mathit{equiv}\ f\ x \ = \ x\cap \cdot \mathit{Inv}\ f \cdot f \tag{11}$$

and

$$
\begin{array}{lll}
x \cap (\mathit{Inv}\ f\ d) & = \mathit{Inv}(f \restriction x)d & \text{if } d \in \mathit{Rng}(f \restriction x) \\
 & = \emptyset & \text{if } d \in \mathit{Rng}\ f \text{ and } d \notin \mathit{Rng}(f \restriction x)
\end{array}
\tag{12}
$$

Their correctness may be checked easily.

4 A sketch of the proof of $Eq_1\ f = Eq_2\ f$

In this section we will give an outline of the proof that $Eq_1\ f$ and $Eq_2\ f$ are equivalent. The full proof requires a lot of calculation and may be found [3].

We introduce some abbreviations:

A abbreviates $Eq_1\ f\ x$

B abbreviates $(\mathit{all}\ q_f \lhd \cdot \mathit{Inv}(\cup/))x$

Clearly, $Eq_2\ f\ x = \downarrow_{\#}/B$. Notice that if x is infinite, it is possible that $Eq_2\ f\ x$ is not uniquely determined. In that case f may be such that all $b \in B$ are infinite. Then there may be $b_1, b_2 \in B$ such that $\#b_1 = \#b_2\ (= \infty)$ and yet $b_1 \neq b_2$. Therefore we will suppose that x is finite (for a generalization towards infinite sets, cf. [3]).

Now the proof of $Eq_1\ f = Eq_2\ f$ runs as follows:

$$
\begin{array}{ll}
 & Eq_1\ f\ x \\
= & \quad \text{definition of } A \\
 & A \\
= & \quad \text{immediate} \\
 & \downarrow_{\#}/\{A\} \\
= & \quad \text{if } b \in B \text{ and } b \neq A \text{ then } \#b > \#A, \text{ see (13) below} \\
 & \downarrow_{\#}/(\{A\} \cup (\neq A) \triangleleft B) \\
= & \quad A \in B, \text{ see (14) below} \\
 & \downarrow_{\#}/B \\
= & \quad \text{definition of } B \text{ and } Eq_2 \\
 & Eq_2\ f\ x
\end{array}
$$

We have to fill in two gaps in the proof, i.e., we must prove:

$$b \in B \ \wedge \ b \neq A \ \text{ implies } \ \#b > \#A \tag{13}$$

$$A \in B \tag{14}$$

To prove (14) we must prove two things (left to the reader, cf. [3]):

$$\cup/A \ = \ x$$

$$all\ q_f\ A$$

The proof of (13) is more difficult. It also consists of two parts (suppose $b \in B$):

$$\#b \geq \#A \tag{15}$$

$$\#b = \#A \ \text{ implies } \ b = A \tag{16}$$

In order to prove (15) and (16) we use a function h such that $hu = v$ where $u \in b$, $b \in B$, $v \in A$, and $u \subseteq v$. Then $h * b = A$ and we may use the following basic facts on magnitudes of sets:

$$\#b \geq \#(h * b)$$

$\#b = \#(h * b)$ iff h is injective on b,

$$\text{i.e., iff } Inv(h \restriction b) \cdot (h \restriction b) \ = \ \{\cdot\} \restriction b$$

Notice that the second fact is the "pigeon hole principle" which presupposes the finiteness of b (guaranteed by the finiteness of x).

Now (15) follows immediately from the first basic fact. With respect to (16) we remark that for any $b \in B$

$$\cup/ \cdot Inv(h \upharpoonright b) \;=\; id_A$$

We may reason as follows:

$$\begin{array}{ll} & \#A \;=\; \#b \\ \Leftrightarrow & \quad \text{property of } h \\ & \#(h * b) \;=\; \#b \\ \Leftrightarrow & \quad \text{basic fact, see above} \\ & Inv(h \upharpoonright b) \cdot (h \upharpoonright b) \;=\; \{\cdot\} \upharpoonright b \\ \Rightarrow & \quad \text{Leibniz} \\ & \cup/ \cdot Inv(h \upharpoonright b) \cdot (h \upharpoonright b) \;=\; \cup/ \cdot (\{\cdot\} \upharpoonright b) \\ \Leftrightarrow & \quad \text{property of } h \\ & h \upharpoonright b \;=\; id_b \\ \Rightarrow & \quad \text{immediate} \\ & h * b \;=\; b \\ \Leftrightarrow & \quad \text{property of } h \\ & A \;=\; b \end{array}$$

Except for the precise definition of the function h, this completes the proof of (16). To define h, first define $\hat{f}$ (suppose u is a set such that $\#(f * u) = 1$):

$$\hat{f}u = d \;\text{ iff }\; f * u = \{d\}$$

Finally, we define the function h:

$$h \;=\; Inv(f \upharpoonright x) \cdot \hat{f}$$

The (calculational) proof that h has all the required properties is left to the reader and may be found in [3].

Remark The approach given in this note was motivated by a set theoretical way of looking at the problem. Its initial formulation was by means of logiuqs (a *logiuq* is a symbol of predicate logic such as $\forall$, $\exists$). The main problem was to reformulate it into a calculational format.

References

[1] Bird, R.S., Shall we calculate?, *The Squiggolist*, **1**, nr 1, 1989.

[2] Bird, R.S., Shall we calculate - II?, *The Squiggolist*, **1**, nr 3, 1990.

[3] J. Kuper, *An alternative approach to a problem posed by Richard Bird,* University of Twente, Enschede, 1990.

Calculation by Computer

Paul Chisholm
University of Groningen

Email: paul@cs.rug.nl

In recent years a great deal of effort has been expended on the implementation of systems for interactive proof construction, or proof editors. Most of these systems fall into two main categories: those for handling goal-directed natural deduction or sequent-style logics, and more programming oriented transformation systems. The former category is typified by NuPrl or Isabelle where a goal is reduced to a number of simpler subgoals, and the process iterated until the subgoals become trivial. The latter style systems are concerned with the construction of programs from specifications via correctness preserving transformations (such as CIP-S or Affirm). A common feature of most available systems is the rather poor interface presented to the user, making them difficult and frustrating to use. Certainly, the primitive interfaces can be partly attributed to the fact that the implementors are mainly interested in the use of the formalism for constructing proofs, rather than in the presentation of proofs on a computer terminal. However, the principle on which these systems are built – that any proofs constructed be machine checkable – has a significant effect on how the user interacts with the system. One consequence is that proofs must be either formal[1] or sufficiently detailed that a formal proof can be automatically generated, often forcing the user to pursue uninteresting goals in excruciating detail; and the possible actions open to the user at any point tend to be somewhat limited. Attempts to overcome the complexity of formal proof have concentrated on automating the proof process, but the more work the system does the less control the user has over proof development. Two common problems in this respect are:

[1] In this context, we mean formal in the strict metamathematical sense

- the ability to introduce notational abbreviations is essential for practical use, but the application of some automatic proof procedure may involve unfolding a number of notation instances during manipulation while the result is not folded using suitable notation. The user can, consequently, be presented with a structure which is considerably larger and more complex than that supplied as input, with a corresponding increase in effort required to interpret the result.

- a proof procedure may be unable to completely prove a goal, but can present the user with a number of subgoals whose proof would establish the original goal, but the user may be left mystified as to how the subgoals relate to the original goal. After all, we are not merely interested in having the machine say 'proved', the derivation itself is important.

The cumulative effect of these problems is to discourage all but the most enthusiastic from using such systems.

We are exploring an alternative approach in which fundamental principles are that the system be flexible and easy to use, that proof construction is a syntactic editing process (as opposed to being driven by the rules of a formal logic), and that the user determines at what level of detail a proof is developed; the requirements of formality and correctness being subjugated to these goals. In essence, we want a system to mimic as closely as possible how proofs are developed using pencil and paper. After all, despite the effort that has gone into the implementation of proof editors, pencil and paper is still the first choice for most people; to them, the restrictions imposed by existing systems are simply unacceptable. It is unreasonable to expect any system to exhibit such flexibility, but this lack is offset by the advantages of using a machine – rapid and reliable copying of expressions, automatic application of transformation rules, generation of code for typesetting, etc. The price to be paid for this flexibility is that the burden of ensuring correctness of proof is placed firmly on the user, though we do not regard this as a serious deficiency.

The following proof segment typifies the kind of proof we are primarily interested in: calculational in style, rigorous but not formal.

$$
\begin{array}{ll}
 & \Sigma(i \ : \ 0 \leq i < n+1 \ : \ x^i) \\
= & \quad \{ \text{ range split } \} \\
 & x^0 + \Sigma(i \ : \ 1 \leq i < n+1 \ : \ x^i)
\end{array}
$$

$$
\begin{array}{ll}
= & \{\ x^0 = 1, \text{ dummy substitution}\ \} \\
& 1 + \Sigma(i\ :\ 0 \leq i < n\ :\ x^{i+1}) \\
= & \{\ x^{i+1} = x * x^i, \text{ distribution}\ \} \\
& 1 + x * \Sigma(i\ :\ 0 \leq i < n\ :\ x^i)
\end{array}
$$

This is surely a proof which anyone with a little knowledge of arithmetic and the quantifier calculus would be perfectly satisfied with. Indeed, for some a single step together with the hint "quantifier calculus" would suffice. Now consider the formalisation of this proof segment. We do not know how many formal steps would be required, nor are we interested in carrying out a statistical comparison, but it would be necessary to formalise and prove properties about addition, multiplication, exponentiation, the quantifier calculus in general, and the Σ quantifier in particular. A considerable number of steps would be involved, yet little would be gained other than the machine's seal of approval.

At Groningen we are implementing an interactive proof assistant based on the above considerations. Below we very briefly describe the important features of this system. For those interested in further details, two reports are available:

Calculation by Computer: Overview

gives an overview of the facilities of the system, while

Calculation by Computer

is a detailed system manual. Contact the author for copies of these reports, and send a Sun cartridge if you would like a copy of the system (which is still at an early stage of development and only available for Sun workstations).

At the core of the system is an untyped λ-calculus with abstraction, application, composition, tupling, and tuple mapping as primitives. The user is free to declare new operators and introduce notational abbreviations, while infix binary operators may be arbitrarily sectioned to reduce the need for bound variables.

The style of proof of primary interest to squiggolists is the calculational style displayed in the above proof segment, and the system is geared towards such proof structures, but a form of bottom-up natural deduction proof allowing the formalisation of contextual information is also possible.

The basic method for constructing expressions and proofs is via the familiar structure editing paradigm: one builds up a structure by inserting templates for various forms of expression or proof at selected locations. Structure editing is complemented by various primitive editing actions – such as distribution – for quick and easy manipulation.

Before carrying out any proofs, the user will typically set up an environment to tailor the system to the theory in which they are working. Such an environment contains three components:

- a table of operator declarations. Such operators may be unary pre- or postfix, or infix binary (associative, left associative, or right associative).
- the definition of notational abbreviations. The ability to extend the language of expressions to the problem domain is essential for effective calculation, and we allow user-defined notation which is both parameterised and admits explicit manipulation of bound variables. Notational flexibility is further enhanced by a character set extended with approximately 100 commonly used symbols.
- the definition of transformation rules. A tool is provided allowing the user to install transformation rules, then apply those rules to extend a proof. Such application is performed by selecting the subterm to be transformed, then clicking the mouse over the desired rule in a window displaying rules.

Although proof can be performed using rules, we should emphasise that in many situations such a simple approach is not successful: it is all too easy to become embroiled in minutiae in an effort to be highly rigorous, but consequently lose sight of our goal. It is possible, and, we believe, advisable to perform some (or even all if context so-determines) steps simply by syntactically editing the expressions involved. In fact, much of our implementation effort has gone into ensuring that syntactic editing is quick and easy.

Selector Guards

Maarten M Fokkinga
Centre for Mathematics and Computer Science, Amsterdam
University of Twente, Enschede

The notion of *selector guard* is defined and some of its properties are investigated. A selector guard is a function that may be used as any other function; in the context of a *selector* it behaves as a "guard": it takes care that certain values will not be selected by the selector.

Selector guards are more flexible than constructions with a guard on a fixed position like guarded commands and McCarthy conditionals. Some *filters* may be replaced by mapped selector guards, and this has calculational advantages.

1 Motivation and Origin

Consider the following problem, which is an abstract version of a derivation step in Fokkinga [2]. Let $a \in \mathsf{Nat}$ and $s \in \mathsf{Nat}*$ and suppose s is ascending. It is required to compute

$$\uparrow_{\#}/ \cdot (a< \cdot \mathsf{last})\triangleleft \cdot \mathsf{inits}^+ \cdot s.$$

Function inits^+ yields the set (or bag or whatever) of nonempty initial segments of its argument sequence, and last yields the last element of its argument sequence. No doubt, anyone who understands the formula, knows that the outcome is either s itself, namely if it is nonempty and satisfies $a < \mathsf{last}\, s$, or else $\uparrow_{\#}/\,\emptyset$ (whatever that may be). The problem is: can we generalise this result and prove it calculationally?

My own attempt was as follows. First observe

- Ascendingness of s means that last is $\leq_{\#} \rightarrow \leq$ monotone **on** $\mathsf{inits}^+\, s$
  (i.e., for $x, y \in \mathsf{inits}^+\, s :: x \leq_{\#} y \;\Rightarrow\; \mathsf{last}\, x \leq \mathsf{last}\, y$).
- Section $a<$ is $\leq \rightarrow \Rightarrow$ monotone
  (i.e., for $x, y :: x \leq y \;\Rightarrow\; (a<x \Rightarrow a<y)$).
- Hence $a< \cdot$ last is $\leq_{\#} \rightarrow \Rightarrow$ monotone **on** $\mathsf{inits}^+\, s$.

Now try and prove the following theorem.

(1) Theorem *Suppose p is $\leq \rightarrow \Rightarrow$ monotone* **on** *A. Let $\uparrow$ denote $\uparrow_{\leq}$. Then*

$$\begin{aligned} \uparrow/ \cdot p\triangleleft \cdot A &= (\mathsf{id} \lhd p \rhd \omega^{\bullet}) \cdot \uparrow/ \cdot A \\ &= \uparrow/ \cdot p\triangleleft \cdot \tau \cdot \uparrow/ \cdot A. \end{aligned}$$

Here $\ldots \lhd \ldots \rhd \ldots$ means ... **if** ... **else** ... , and ω abbreviates $\uparrow/\emptyset$. The second right hand side is just a calculational more attractive alternative for the first one. Function τ is the singleton former.

No doubt, the theorem is true. But how do we prove it? One method is to use the Unique Extension Property (or Induction). Thus it is sufficient if we can show that

1. $lhs \cdot \tau = rhs \cdot \tau$ **on** A,

2. both *lhs* and *rhs* are $\cup \to \uparrow$ promotable **on** A,

where *lhs* and *rhs* is the left hand side, respectively right hand side, of the claimed equality without '$\cdot A$'. Part 1 is easy, and the promotability of *lhs* is immediate by the form of the expression. With some case analysis one can also show that *rhs* is $\cup \to \uparrow$ promotable **on** A.

However, I do not like this proof. It is not calculational. Moreover, the right hand sides of the equality of the theorem are ugly. Can't we do better? Yes, we can. Selector guards provide the solution.

2 Definition

A *selector* is a binary function (operation) $\oplus$ satisfying the so-called *selectivity* property: $x \oplus y \in \{x, y\}$. Thus a selector is idempotent. In this note we require selectors to be associative and commutative as well, thus excluding for example operation 'left one' $\ll$. We let $\square$ range over (associative and commutative) selectors.

Let p be a predicate, and $\square$ be a selector. Then we define $p?_\square$ as follows:

$$\begin{array}{lcll} p?_\square\, x & = & x & \text{if } p\, x \\ & = & 1_\square & \text{otherwise.} \end{array}$$

When no confusion can result, we omit the subscript to ?. Function $p?$ is called a *selector guard*, or specifically, a $\square$-*guard*.

As an example of its use, suppose that $1_\square$ is defined to be a zero of f, g and h. Then

$$(f \cdot p?) \;\square\; (g \cdot q?) \;\square\; (h \cdot r?)$$

might also be denoted by $p \to f \,\square\, q \to g \,\square\, r \to h$, although the meanings may differ from each other if not $p \vee q \vee r = \mathsf{true}$. Selector guards are just functions, so expressions like

$$p? \cdot f \qquad \text{and} \qquad p? \cdot f \cdot q? \cdot g \cdot r?$$

make sense as well. This facilitates more freedom in expressing what you want to express.

The requirement that $1_\square$ is a zero of f is not too strong a requirement, and can always be achieved. It simply means that we have to *replace* f by a new function f' defined by $f'\, x =$ if $x = 1_\square$ then $1_\square$ else $f\, x$. So even when f has already been defined for $1_\square$ we can suppose that "$1_\square$ is (locally!) made a zero of a particular occurrence of f".

3 Some Properties

The verification of the following properties poses no problems, and is therefore omitted.

(2) $$p? \cdot f \;=\; f \cdot (p \cdot f)?$$

(3) $$?\ \text{is}\ \ \vee \to \Box\ \text{promotable}$$

(4) $$?\ \text{is}\ \ \wedge \to \circ\ \text{promotable.}$$

In the first one it is assumed that $1_\Box$ is a zero of f (or more precisely, that $1_\Box$ is made a zero of the left most occurrence of f in the right hand side, as described above). The middle one means that $(p \vee q)? = p? \,\Box\, q?$ and $\mathsf{false}^\bullet? = 1_\Box$. The last one means that $(p \wedge q)? = p? \cdot q?$ and $\mathsf{true}^\bullet? = \mathsf{id}$. From these it follows that $p \vee q = p \;\Rightarrow\; p? \,\Box\, q? = p?$ (and similarly for $\wedge$). On account of the injectivity of ? the converse implication is also true. Thus

(5) $$p \vee q = p \;\equiv\; p? \,\Box\, q? = p?$$

(6) $$p \wedge q = p \;\equiv\; p? \cdot q? = p?$$

(Recall also that $p \vee q = p$ equivales $q \Rightarrow p$; the former is calculationally far more attractive. Similarly, $p \wedge q = p$ equivales $p \Rightarrow q$.) Another property is

(7) $$p \text{ is } \Box \to \vee \text{ distributive} \;\Rightarrow\; p? \text{ is } \Box \to \Box \text{ promotable.}$$

This property, in fact, encapsulates the case analysis mentioned in the proof sketch of Theorem (1). Therefore we shall prove it here in full.

Proof Immediately from the definition of ? we have that $p?\, 1_\Box = 1_\Box$, independent of the value of $p\, 1_\Box$. Further, let x, y be arbitrary, and assume without loss of generality that $x \Box y = x$. Observe that $py \Rightarrow px$ since

$$\begin{array}{ll} & py \Rightarrow px \\ \equiv & \quad \text{proposition calculus} \\ & px \vee py = px \\ \equiv & \quad \text{premiss} \\ & p(x \Box y) = px \\ \equiv & \quad \text{assumption } x = x \Box y \\ & px = px \\ \equiv & \quad \text{Leibniz} \\ & \mathsf{true}. \end{array}$$

Now, consider the case that py holds. Then, by the above, px holds too, and:

$$\begin{array}{ll} & p?\,(x \Box y) = p?\, x \,\Box\, p?\, y \\ \equiv & \quad \text{assumption } x = x \Box y \end{array}$$

$$
\begin{array}{ll}
& p?\,x = p?\,x \,\Box\, p?\,y \\
\equiv & \quad \text{truth of } py \text{ and } px \\
& x = x \Box y \\
\equiv & \quad \text{assumption } x = x \Box y \\
& \mathsf{true}.
\end{array}
$$

In case py does not hold, we have

$$
\begin{array}{ll}
& p?\,(x \Box y) = p?\,x \,\Box\, p?\,y \\
\equiv & \quad \text{assumption } x = x \Box y \\
& p?\,x = p?\,x \,\Box\, p?\,y \\
\equiv & \quad \text{case assumption } py = \mathsf{false} \\
& p?\,x = p?\,x \,\Box\, 1_{\Box} \\
\equiv & \quad \text{value } 1_{\Box} \text{ is identity of } \Box \\
& \mathsf{true}.
\end{array}
$$

The converse implication of property (7) holds as well if it is given that $p\,1_{\Box} = \mathsf{false}$:

$$
\begin{array}{l}
p \text{ is } \Box \to \vee \text{ distributive} \quad \equiv \quad p? \text{ is } \Box \to \Box \text{ promotable} \\
\text{provided that } p\,1_{\Box} = \mathsf{false}.
\end{array}
$$

Finally, we mention the law that we will refer to as the **Filter-Guard$*$ exchange**:

$$
(8) \qquad \Box/ \cdot f* \cdot p\triangleleft \;=\; \Box/ \cdot f* \cdot p?*
$$

provided $f\,1_{\Box} = 1_{\Box}$. The proof is easy: by the Unique Extension Property the equality holds if both sides are join $\to \Box$ promotable (which is immediate from their syntactic form) and pre-composed with τ they are equal (which is easy to check, using the proviso $f\,1_{\Box} = 1_{\Box}$).

Since $p?*$ is just a map, it is easier to manipulate than the filter $p\triangleleft$. True enough, several authors, e.g., Jeuring [3], observe that $p\triangleleft = +\!\!+/ \cdot (p\triangleleft \cdot \tau)*$ (since $p\triangleleft$ is join $\to +\!\!+$ promotable), so that

$$
\begin{array}{ll}
& \Box/ \cdot p\triangleleft \\
= & \Box/ \cdot +\!\!+/ \cdot (p\triangleleft \cdot \tau)* \\
= & \Box/ \cdot \Box/* \cdot (p\triangleleft \cdot \tau)* \\
= & \Box/ \cdot (\Box/ \cdot p\triangleleft \cdot \tau)*,
\end{array}
$$

and, indeed, $p? = \Box/ \cdot p\triangleleft \cdot \tau$. But this observation has not suggested to investigate the promotion properties of $p?$ and of ? itself (which is the function $\lambda p{::}\ \Box/ \cdot p\triangleleft \cdot \tau$). In view of the nice promotability properties, $\Box/ \cdot p\triangleleft \cdot \tau$ deserves a notation of its own, which is what we propose in this note.

4 Application

As an application of selector guards we show that the theorem mentioned in the introduction is really a triviality. Throughout the discussion we let $\uparrow$ mean $\uparrow_{\leq}$. First observe that, just by unfolding,

(9) $$p \text{ is } \leq \rightarrow \Rightarrow \text{ monotone } \equiv p \text{ is } \uparrow \rightarrow \vee \text{ distributive.}$$

More generally, any selector $\square$ determines a linear order $\leq_\square$ given by $x \leq_\square y \equiv (x = x \square y)$, and we have $\uparrow_{\leq_\square} / = \square$, so that

(10) $$p \text{ is } \leq_\square \rightarrow \Rightarrow \text{ monotone } \equiv p \text{ is } \square \rightarrow \vee \text{ distributive.}$$

Proof of Theorem (1)

$\uparrow / \cdot p \lhd \cdot A \;=$ right hand side

$\equiv$ express right hand side in terms of selector guards; filter-guard$*$ exchange in left hand side

$\uparrow / \cdot p? * \cdot A \;=\; p? \cdot \uparrow / \cdot A$

$\Leftarrow$ promotion theorem

$p?$ is $\uparrow \rightarrow \uparrow$ promotable **on** A

$\equiv$ property (7)

p is $\uparrow \rightarrow \vee$ distributive **on** A

$\equiv$ observation (9) above

p is $\leq \rightarrow \Rightarrow$ monotone **on** A

$\equiv$ premiss of the theorem

true.

In this series of proof steps, the most inventive one is the first — introducing the selector guards. The most complicated one is the proof of property (7) which we have given in full above. The remaining steps are trivial indeed.

Discussion Since monotonicity is a special instance of distributivity, in the sense of (10), and since —thus far— all calculational laws refer to the notion of distributivity rather than monotonicity, it is to be expected that looking for distributivity properties right from the beginning is at least as easy as sometimes looking for monotonicity properties. Thus we are led to reformulate Theorem (1) in terms of distributivity properties of p. Moreover, now that we have introduced selector guards, and know that they are at least as easy to manipulate as filters, we use these in the formulation too. Generalising a bit, we come to the following triviality:

(11) Theorem *Suppose f and $p?$ are $\Box \to \Box$ promotable (where we know that for the latter it suffices if p is $\Box \to \vee$ distributive). Then*

$$\Box/ \cdot f* \cdot p?* \;=\; f \cdot p? \cdot \Box/.$$

Obviously, this claim —although it is true— is not worth the status of a Theorem. It is merely an application of the Promotion Theorem.

5 Conclusion

The idea of "guard" is not new. As early as 1972 De Bakker and De Roever [1] use partial identity relations $p?'$, i.e., $p?' \subseteq \mathbf{I}$, to *describe* various programming language constructs (like conditionals and repetitions). Recently, Malcolm [4] uses this very construct in the *derivation* of a program: the guard is used to record context information. However, as Peter de Bruin has remarked, the *relational* guard $p?'$ can not be used in a filter-guard$*$ exchange, since with the relational interpretation we have $p?'* = (\mathsf{all}\, p)?'$ and this is certainly not a valid equation for our $p?*$. For example,

$$\begin{array}{ll} \text{for } R = (\Box/ \cdot (0=)?'*) & \text{we have NOT } \; \emptyset\, R\, [0,1] \\ \text{for } f = (\Box/ \cdot (0=)?*) & \text{we DO have } \; \emptyset = f\, [0,1]. \end{array}$$

Meertens [5] has introduced a notion of *functional* guard $p?$ but only for the indeterminate choice selector $[]$. The novelty of our concept is its relation to a particular selector, so that in particular the filter-guard$*$ exchange law is valid.

References

[1] J.W. de Bakker and W.P. de Roever. *A Calculus For Recursive Program Schemes.* Technical Report MR 131/72, Mathematical Centre, Amsterdam, February 1972.

[2] M.M. Fokkinga. A BM style derivation of the lexico least upperbound of pattern e in subs x. March 1990. CWI, Amsterdam.

[3] J.T. Jeuring. *Algorithms from Theorems.* Technical Report RUU-CS-90-3, Utrecht University, Dept of Computer Science, January 1990. Also Report CS-R9006, CWI, Amsterdam, Febr 1990.

[4] G. Malcolm. Squiggoling in context. *The Squiggolist*, 1(3):14th – 19th, 1990.

[5] L. Meertens. Algorithmics — towards programming as a mathematical activity. In J.W. de Bakker and J.C. van Vliet, editors, *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334, North-Holland, 1986.

Soft arrays

Alan Jeffrey

Programming Research Group, Oxford

In *A Theory of Arrays for Program Derivation* (Chris Wright, Oxford University, 1988), arrays are defined as an initial algebra. Whereas $+\!\!+$-lists are the initial monoid, arrays are the initial *binoid*. A binoid on α is an algebra $(X, \oplus, \otimes, f)$ with carrier set X, embedding $f : \alpha \to X$ and associative operators $\oplus$ and $\otimes$ which *abide*, that is:

$$(w \oplus x) \otimes (y \oplus z) \quad = \quad (w \otimes y) \oplus (x \otimes z)$$

For example, $(\mathbf{N}, +, +, K_1)$ and $([\alpha], +\!\!+, \ll, [\cdot])$ are both binoids on α, and the algebra of arrays $((\alpha), \ominus, \oplus, (\cdot))$ is the initial binoid.

Unfortunately, $\ominus$ and $\oplus$ are partial operators, and so we need some rather messy constraints that say when $x \ominus y$ and $x \oplus y$ are defined. If we define *width* to be the homomorphism from $((\alpha), \ominus, \oplus, (\cdot))$ to $(\mathbf{N}, \bullet, +, K_1)$ then $x \ominus y$ is defined iff *width* x = *width* y, and similarly for $\oplus$.

All this is rather ugly, so I wondered what would happen if we made $\ominus$ and $\oplus$ total operators. So, define a *total binoid* on α to be a binoid on α, $(X, \oplus, \otimes, f)$, where $\oplus$ and $\otimes$ are total. Then define $(\langle\alpha\rangle, \multimap, \Phi, \langle\cdot\rangle)$ to be the initial total binoid on α. We will call this algebra *soft arrays*, for reasons which should become apparent.

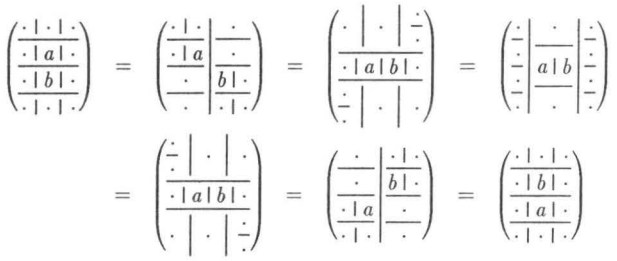
Unfortunately, soft arrays aren't the same as arrays. Consider the soft array

$$\left(\begin{array}{c|c|c} \cdot & \cdot & \cdot \\ \hline \cdot & a & \cdot \\ \hline \cdot & b & \cdot \\ \hline \cdot & \cdot & \cdot \end{array}\right)$$

where '$\cdot$' just represents an element whose value we don't care about. This is just a pictorial representation of

$$(\langle\cdot\rangle \Phi \langle\cdot\rangle \Phi \langle\cdot\rangle) \multimap (\langle\cdot\rangle \Phi \langle a\rangle \Phi \langle\cdot\rangle) \multimap (\langle\cdot\rangle \Phi \langle b\rangle \Phi \langle\cdot\rangle) \multimap (\langle\cdot\rangle \Phi \langle\cdot\rangle \Phi \langle\cdot\rangle)$$

Then using associativity and abiding of $\multimap$ and $\blacklozenge$ we can show



In fact in any rectangular soft array, we can move around any elements that aren't on the border. However, the borders remain fixed—this can be shown by defining

- *top* is the homomorphism to $([\alpha], \ll, +\!\!+, [\cdot])$,
- *bottom* is the homomorphism to $([\alpha], \gg, +\!\!+, [\cdot])$,
- *left* is the homomorphism to $([\alpha], +\!\!+, \ll, [\cdot])$,
- *right* is the homomorphism to $([\alpha], +\!\!+, \gg, [\cdot])$.

As *top*, *bottom*, *left* and *right* are homomorphisms, they are well-defined, so the borders of any soft array are fixed. Similarly, *bagify* is the homomorphism to $(\wr\alpha\int, \uplus, \uplus, \wr\cdot\int)$, so the bag of elements a soft array contains is fixed.

So, any rectangular soft array can be defined by its borders and its elements. Unfortunately, this is not the case for soft arrays in general, as

and

aren't the same, but have the same borders and elements. In fact, I don't know what the intuition for soft arrays is, or any uses for them. Just about the only thing we know is that they aren't arrays, so arrays will have to stay partial.

On induced congruences: an answer

20-02-90

Jaap van der Woude

In [0] relational homomorphisms and congruence relations on $\mathcal{T}$ were shown to be dependent. To be more accurate only one direction was shown, the other was left as an "open question". I'll give an affirmative answer.
I adopt the notation and "statement numbering" from [0].

The result in [0], referred to in the above, was:
For a relational homomorphism R
- Rt is reflexive, transitive and F-invariant (7,9,12);
- $Rt \cap Rt^{\smile}$ is a congruence relation (14).

The open question was:
Can every congruence relation on $\mathcal{T}$ be expressed as $Rt \cap Rt^{\smile}$ for some relational homomorphism R?
It turns out that a congruence relation on $\mathcal{T}$ can even be expressed as Rt. The symmetry may even be dropped!
In the following lemmata I address reflexivity and transitivity separate from F-invariance:

15 **Lemma** Let S be a relation on $\mathcal{T}$. Then

$$St = S \equiv S \supseteq I \wedge S \supseteq S \circ S$$

Proof
$St = S$
$\Rightarrow$ {7} {by 5: $S \supseteq S \circ St$}
$S \supseteq I \wedge S \supseteq S \circ S$
$\Rightarrow$ {monotonicity} {5}
$S \circ St \supseteq St \wedge St \supseteq S$
$\Rightarrow$ {5; transitivity}
$S \supseteq St \wedge St \supseteq S$
$=$ {calc.}
$St = S$

□

16 Lemma Let S be reflexive and transitive. Then

$$\tau \in S \leftarrow S^F \equiv S = \mathbb{C}S \circ \tau \mathbb{D}$$

Proof
$$\begin{aligned}
& S = \mathbb{C}S \circ \tau \mathbb{D} \\
&= \{1\} \\
& S \circ \tau = S \circ \tau \circ S^F \\
&\Rightarrow \{S \supseteq I\} \\
& S \circ \tau \supseteq \tau \circ S^F \\
&\Rightarrow \{\text{monotonicity}; S \supseteq S \circ S\} \\
& S \circ \tau \supseteq S \circ \tau \circ S^F \\
&= \{I \subseteq S, \text{ so } I \subseteq S^F, \text{ hence } S \circ \tau \subseteq S \circ \tau \circ S^F\} \\
& S \circ \tau = S \circ \tau \circ S^F
\end{aligned}$$

The lemma follows from 2 and the definition of $\in$. □

A combination of 15 and 16 answers the open question affirmatively. The result is even somewhat stronger.

17 Theorem Let S be a relation on $\mathcal{T}$. Then

$$S \supseteq I \wedge S \supseteq S \circ S \wedge \tau \in S \leftarrow S^F \equiv S = \mathbb{C}S \circ \tau \mathbb{D}\dagger$$

Proof
$$\begin{aligned}
& S \supseteq I \wedge S \supseteq S \circ S \wedge \tau \in S \leftarrow S^F \\
&= \{15, 16\} \\
& S\dagger = S \wedge S = \mathbb{C}S \circ \tau \mathbb{D} \\
&\Rightarrow \{\text{calc.}\} \\
& S = \mathbb{C}S \circ \tau \mathbb{D}\dagger \\
&\Rightarrow \{7, 9, 12 \text{ with } R := \mathbb{C}S \circ \tau \mathbb{D}\} \\
& S \supseteq I \wedge S \supseteq S \circ S \wedge \tau \in S \leftarrow S^F
\end{aligned}$$

□

[0] Backhouse R. and G. Malcolm, On induced congruences
The Squiggolist 1.3

Balanced Binary Trees as a (Partial) Free Algebra

Jeremy Gibbons

1 A little bit of type theory

In [BCMS89], Backhouse *et al.* give two classifications of types.

- A type can be a *free* type, or it can be a *congruence* type. Free types are initial algebras. There are no algebraic laws unifying terms — there is an injection between 'terms' which can be built from the constructors and 'elements' in the algebra. Peano numbers and 'cons' lists are examples of free types. Congruence types, on the other hand, have some additional laws unifying terms; the mapping from terms to elements is surjective but no longer injective. For example, the type of 'cat' lists or monoids is a congruence type.

- A type can be *constrained* (also called a *subset* type), or it can be *unconstrained*. Backhouse has described a subset type as a type

  > in which elements of the free type are restricted to those satisfying some given property

  though presumably one can also have a subset congruence type. (It is a pity that subset types are not quite those in which the mapping from terms to elements is no longer surjective.) The type of even naturals is a subset type of the type of naturals; they are the natural numbers which enjoy the property of being even.

In constructive type theory, the context of [BCMS89], all functions and constructors are total, but in a context which allows partial types we can differentiate further:

- (the constructors of) a type may be *partial* or *total*

The difference between a partial free and a subset free type, and indeed the property that makes subset types difficult to work with and algebraically undesirable, is that the subset type doesn't have its own constructors; an element of a subset type is not necessarily constructed from smaller elements of the same type. In some cases, it is possible to reformulate a subset type as a non-subset type; this immediately makes it easier to work with, since we can operate directly in the algebra of interest. Take, for example, the type of even (natural) numbers used above. The even naturals are isomorphic to the (even and odd) naturals; they can (naturally!) be constructed Peano-style from the constructors zero and addtwo, which form a much simpler basis.

2 Balanced trees

We turn now to our area of real interest, trees. A *balanced tree* is a (not necessarily binary) tree such that the depths of any two of its immediate subtrees differ by no more than 1. Equivalently, its depth is no more than 1 greater than its 'proximity', the *shortest* distance from the root to a leaf.

We use leaf-labelled binary trees for a concrete example to discuss in this note, but the following applies equally well to any kind of tree. As a subset type, the type $\mathsf{baltree}\,\alpha$ of balanced binary trees is given by

$$\begin{aligned} \triangle &: \alpha \to \mathsf{baltree}\,\alpha \\ \pm &: (\mathsf{baltree}\,\alpha \times \mathsf{baltree}\,\alpha) \to \mathsf{baltree}\,\alpha \end{aligned}$$

constrained such that the condition $\mathsf{all\,balanced} \cdot \mathsf{subtrees}$ is satisfied, where

$$\mathsf{balanced\,t} \;=\; \mathsf{depth\,t} - \mathsf{proximity\,t} \leqslant 1$$

and subtrees is as you might expect.

Now, consider the further restricted type $\mathsf{baltree}_n\,\alpha$, 'balanced binary trees of depth n'. These are trees t such that

$$\mathsf{all\,balanced\,(subtrees\,t)} \wedge \mathsf{depth\,t} = n$$

You could think of these trees as forming another subset type of the type of binary trees. Alternatively, you could think of them as being constructed *without any constraints* from the operators

$$\begin{aligned} \triangle &: \alpha \to \mathsf{baltree}_0\,\alpha \\ \pm &: (\mathsf{baltree}_n\,\alpha \times \mathsf{baltree}_{n+1}\,\alpha) \to \mathsf{baltree}_{n+2}\,\alpha \\ \pm &: (\mathsf{baltree}_n\,\alpha \times \mathsf{baltree}_n\,\alpha) \to \mathsf{baltree}_{n+1}\,\alpha \\ \pm &: (\mathsf{baltree}_{n+1}\,\alpha \times \mathsf{baltree}_n\,\alpha) \to \mathsf{baltree}_{n+2}\,\alpha \end{aligned}$$

The equations for $\pm$ can in turn be interpreted in two ways. Either they specify a single partial constructor, which combines only compatible subtrees, or they specify

a (countably infinite) family of total constructors, constructing left-, not- and right-squint balanced trees for each natural height. Either way, the disjoint union ' $\mathrm{baltree}_n$ for all natural n ' is isomorphic to the type baltree , and is a free type (either partial, or with an infinite basis).

The important point is that balanced trees can be constructed from smaller balanced trees.

3 Coda

What I would like to know is

- Is this well known? In [BBMS89], height balanced trees are given as the example of a subset type, which suggests that this is not the case.
- Is it at all useful?
- What are the ramifications of making the constructor partial? Or of having a type with a (countably) infinite basis?
- For how many more constrained types can this be done?

References

[BBMS89] Roland Backhouse, Richard Bird, Lambert Meertens, and Mary Sheeran. Constructive algorithmics. Proposal for a collaborative project, October 1989.

[BCMS89] Roland Backhouse, Paul Chisholm, Grant Malcolm, and Erik Saaman. Do-it-yourself type theory. *Formal Aspects of Computing*, 1(1):19–84, 1989.

Tupling and Mutumorphisms

Maarten M Fokkinga, CWI, Amsterdam

We prove, in a very general setting, two *folklore* theorems about tupling, one of which involves *mutumorphisms*, which generalise both cata-, para- and Malcolm's zygomorphisms, and have useful properties for calculation.

* * *

Throughout this note we let F be an arbitrary functor, and we let $(L, \mathsf{in}) = \mu(\mathsf{F})$, the initial F-algebra with constructor $\mathsf{in} \in L \leftarrow L\mathsf{F}$. An F-catamorphism is denoted $(\!|\phi|\!)$ where $\phi \in A \leftarrow A\mathsf{F}$; it is the unique function $f \in A \leftarrow L$ satisfying $\phi \cdot f\mathsf{F} = f \cdot \mathsf{in}$. The tupling $f \Uparrow g$ of f and g is defined by $(f \Uparrow g)x = (fx, gx)$. Further, $(f \parallel g)(x,y) = (fx, gy)$, and $\ll(x,y) = x$ and $\gg(x,y) = y$.

(1) Theorem $\qquad (\!|\phi|\!) \Uparrow (\!|\psi|\!) \;=\; (\!|\phi \parallel \psi \cdot \ll\mathsf{F} \Uparrow \gg\mathsf{F}|\!)$.

Proof An easy calculation, using the catamorphism property in the first step:

$$\begin{array}{ll} & (\!|\phi|\!) \Uparrow (\!|\psi|\!) \cdot \mathsf{in} \\ = & (\phi \cdot (\!|\phi|\!)\mathsf{F}) \Uparrow (\psi \cdot (\!|\psi|\!)\mathsf{F}) \\ = & \phi \parallel \psi \cdot (\!|\phi|\!)\mathsf{F} \Uparrow (\!|\psi|\!)\mathsf{F} \\ = & \phi \parallel \psi \cdot (\ll \cdot (\!|\phi|\!) \Uparrow (\!|\psi|\!))\mathsf{F} \Uparrow (\gg \cdot (\!|\phi|\!) \Uparrow (\!|\psi|\!))\mathsf{F} \\ = & \phi \parallel \psi \cdot \ll\mathsf{F} \Uparrow \gg\mathsf{F} \cdot ((\!|\phi|\!) \Uparrow (\!|\psi|\!))\mathsf{F} \end{array}$$

Hence, by the catamorphism property, the equation of the theorem holds.

* * *

We say that f *is* F*-catamorphic modulo* g if for some ϕ

$$\phi \cdot (f \Uparrow g)\mathsf{F} \;=\; f \cdot \mathsf{in}$$

which we denote by $f : \phi \xleftarrow{\mathsf{F}} \mathsf{in}$ **mod** g. Similarly, $g : \psi \xleftarrow{\mathsf{F}} \mathsf{in}$ **mod'** f means that $\psi \cdot (f \Uparrow g)\mathsf{F} \;=\; g \cdot \mathsf{in}$. We call functions like f and g above *mutumorphisms*, since their essential property is that they are defined *mutu*ally recursive by induction on the structure (*morphe*) of the data of the initial data type L.

(2) Theorem *Suppose* f *and* g *are* F*-catamorphic modulo each other:*

$$f : \phi \xleftarrow{\mathsf{F}} \mathsf{in} \textbf{ mod } g \qquad \wedge \qquad g : \psi \xleftarrow{\mathsf{F}} \mathsf{in} \textbf{ mod' } f\,.$$

Then $f \Uparrow g$ is a F-catamorphism; in particular we have:

$$f \Uparrow g = (\!|\phi \Uparrow \psi|\!) \qquad f = \ll \cdot (\!|\phi \Uparrow \psi|\!) \qquad g = \gg \cdot (\!|\phi \Uparrow \psi|\!) .$$

Proof An easy calculation, using the catamorphism property in the first step:

$$\begin{array}{ll} & f \Uparrow g = (\!|\phi \Uparrow \psi|\!) \\ = & \phi \Uparrow \psi \cdot (f \Uparrow g)\mathsf{F} = f \Uparrow g \cdot \mathsf{in} \\ = & \phi \cdot (f \Uparrow g)\mathsf{F} = f \cdot \mathsf{in} \quad \wedge \quad \psi \cdot (f \Uparrow g)\mathsf{F} = g \cdot \mathsf{in} \\ = & \mathsf{true} \end{array}$$

So, any mutumorphism can be expressed as a catamorphism followed by a projection. The following facts, observations and corollaries can be proved very easily from the above theorem and from what we already know for catamorphisms, which is sufficient reason for us to omit the formal proofs.

First we have that proper catamorphisms are mutumorphisms as well:

$$(3) \qquad g : \psi \xleftarrow{\mathsf{F}} \mathsf{in} \quad \Rightarrow \quad g : (\psi \cdot \gg\mathsf{F}) \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod}' \ f .$$

Using this we find that mutumorphisms generalise paramorphisms in that any paramorphism is a mutumorphism as well:

$$(4) \qquad [\![\phi]\!] = \ll \cdot (\!|\phi \Uparrow (\mathsf{in} \cdot \gg\mathsf{F})|\!) \qquad \text{i.e.,} \quad [\![\phi]\!] : \phi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod} \ \mathsf{id}(= (\!|\mathsf{in}|\!)) .$$

The same holds with respect to the *zygomorphisms* developed by Malcolm in his forthcoming thesis:

$$(5) \qquad (\phi, \chi)^{\natural} = \ll \cdot (\!|\phi \Uparrow (\chi \cdot \gg\mathsf{F})|\!) \qquad \text{i.e.,} \quad (\phi, \chi)^{\natural} : \phi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod} \ (\!|\chi|\!) .$$

Tupled mutumorphisms are useful in program derivation by calculation, since they have the following properties. (Notice that by the type and combinator calculus we have that any $(\!|\phi \Uparrow \psi|\!)$ equals a tupling $f \Uparrow g$ for some f and g.)

First, the catamorphism property and the theorem gives us MUTUMORPHISM:

$$(6) \qquad f \Uparrow g = (\!|\phi \Uparrow \psi|\!) \quad \equiv \quad \begin{array}{l} \phi \cdot (f \Uparrow g)\mathsf{F} = f \cdot \mathsf{in} \\ \psi \cdot (f \Uparrow g)\mathsf{F} = g \cdot \mathsf{in} \end{array} .$$

Then we have the UNIQUE EXTENSION PROPERTY for tupled mutumorphisms:

$$(7) \qquad \begin{array}{l} f = h \\ g = j \end{array} \quad \Leftarrow \quad \begin{array}{ll} f : \phi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod} \ g & h : \phi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod} \ j \\ g : \psi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod}' \ f & j : \psi \xleftarrow{\mathsf{F}} \mathsf{in} \ \mathbf{mod}' \ h \end{array} .$$

Finally, the PROMOTION THEOREM for tupled mutumorphisms:

$$(8) \qquad (\!|\phi \Uparrow \psi|\!) = f \Uparrow g \cdot (\!|\zeta \Uparrow \eta|\!) \quad \Leftarrow \quad \begin{array}{l} f : \phi \xleftarrow{\mathsf{F}} \zeta \Uparrow \eta \ \mathbf{mod} \ g \\ g : \psi \xleftarrow{\mathsf{F}} \zeta \Uparrow \eta \ \mathbf{mod}' \ f \end{array} ,$$

and the 'follows from' is an 'equivales' if $(\!|\zeta \Uparrow \eta|\!)$ has a right inverse (i.e., is surjective).